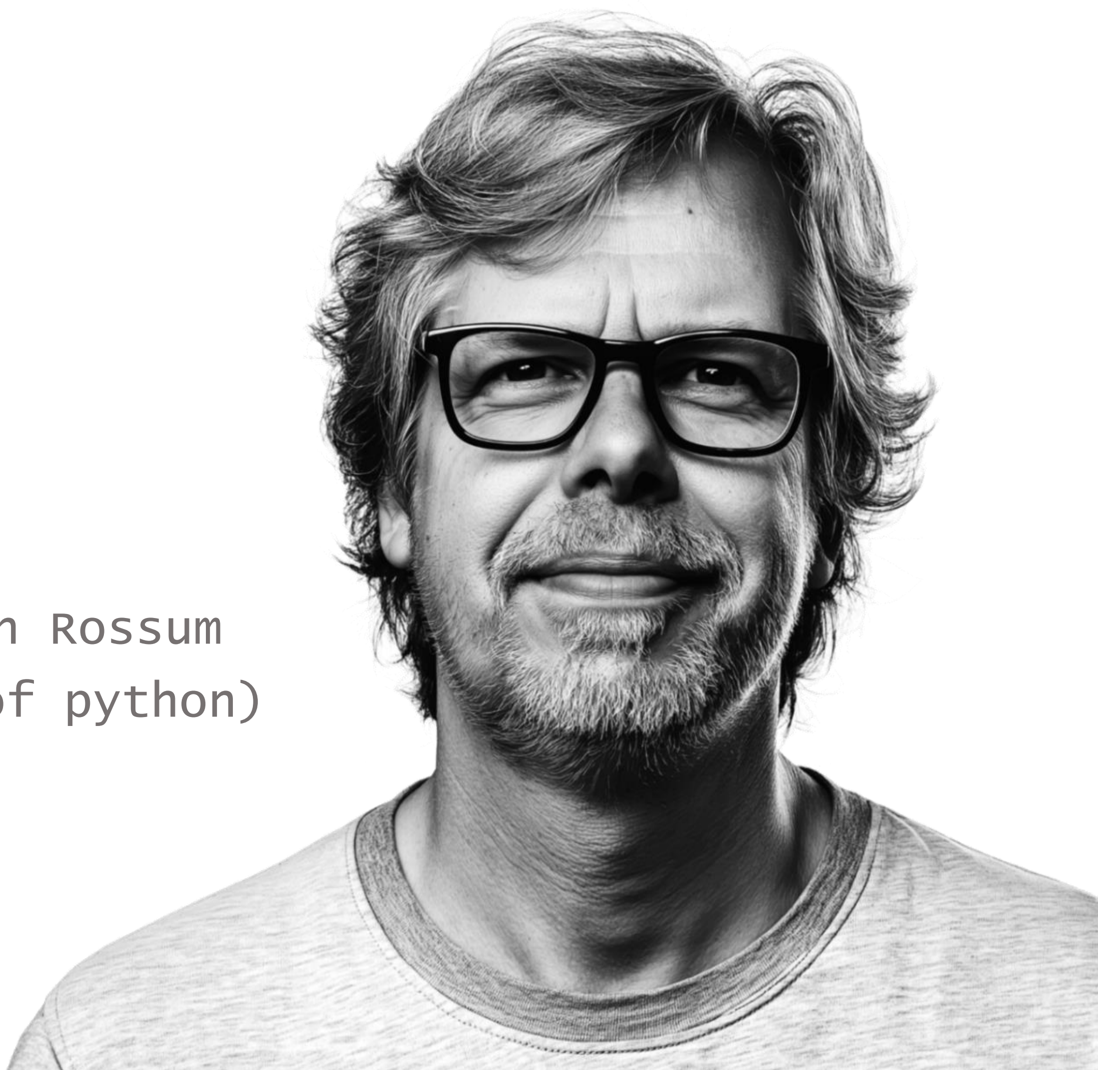
Talk is cheap.
Show me the code.
~ Linus Torvalds



Guido Van Rossum
(Founder of python)

ictfromabc.com

Contents

Python introduction 2
Modularization 4
Structure charts 7
Programming languages 9
Programming paradigms 11
Source code vs object code 14
Language translators 16
Linkers 19
Loaders 20
Integrated Developments Environment 21
Python basics 29
Operators 35
Variable Types 47
Getting input 53
Casting 54
Control structures 58
Data structures 59
Conditional Statements 60
Repetition 99
List Comprehension 114
Functions 115
Recursive functions 119
Global and Local variables 121
Number conversion and built-in functions 123
Error handling 129
File operations 130
Python database connection 136
Sorting and searching algorithms 143
Flowcharts and Pseudocodes 158

What is Python?
Python යනු කුමක්ද?

Python is a high-level, interpreted programming language.
Python යනු ඉහල මට්ටමේ, අර්ථවින්‍යාසක භාවිතා කරන ක්‍රමලේඛණ භාශාවකි.

It was created by Dutch Programmer Guido van Rossum and released in 1991.
1991 දී ලන්දේසි ජාතික Guido van Rossum විසින් නිර්මාණය කරන ලදී

Python is used in a wide range of applications, from web development, game development, Artificial Intelligence, machine learning, scientific computing, academic research, and more.
වෙබ් අඩවි සංවර්ධනය, පරිගණක ක්‍රීඩා සංවර්ධනය, කෘතිම බුද්ධිය, යාන්ත්‍ර ඉගෙනීම, විද්‍යාත්මක පරිගණනය, අධ්‍යාපන පර්යේෂණ ආදී විවිධ ක්ෂේත්‍ර ගණනාවක යෙදීම් සඳහා Python භාවිතා වේ

Python supports multiple programming paradigms including procedural, object-oriented, and functional programming.
Python ක්‍රියා පටිපාටි, වස්තු-නැඹුරු සහ ක්‍රියාකාරී ක්‍රමලේඛනය ඇතුළු බහු ක්‍රමලේඛන සුසමාදර්ශයන්ට සහය දක්වයි.

It emphasizes readability and ease of use, with a syntax that allows programmers to express concepts in fewer lines of code than might be possible in other languages.
ක්‍රමලේඛකයින්ට වෙනත් භාෂාවලින් ලිවිය හැකි ප්‍රමාණයට වඩා අඩු කේත පේළි ගණනකින් සංකල්ප ප්‍රකාශ කිරීමට ඉඩ සලසන . එකක් සමඟින් එය කේතය කියවීමේ හැකියාව සහ භාවිතයේ පහසුව ලබාදේ.

Python supports most of the operating systems.
Python බොහෝ මෙහෙයුම් පද්ධති සමග සහය දක්වයි.

How python differs from other languages?
පයිතන් වෙනත් භාෂාවලින් වෙනස් වන්නේ කෙසේද?

Simplicity and Readability
Python emphasizes clean, easy-to-read code with a simple syntax. It often uses natural language-like expressions, which reduces the learning curve, especially compared to more complex languages like C++ or Java.
Python සරල එකක් සහිත පිරිසිදු, කියවීමට පහසු කේතයක් විශේෂණය කරයි. එය බොහෝ විට මිනිස් භාෂා වලට සමාන ප්‍රකාශන භාවිතා කරයි, විශේෂයෙන් C++ හෝ Java වැනි වඩාත් සංකීර්ණ භාෂා සමඟ සසඳන විට එමගින් ඉගෙනීමේ වක්‍රය අඩු කරයි.

Dynamically Typed
Python is dynamically typed, meaning you don’t need to declare variable types explicitly.
Python ගතිකව ටයිප් කර ඇත, එනම් ඔබට විචල්‍ය වර්ග ප්‍රකාශ කිරීමට අවශ්‍ය වන්නේ නැත.

Interpreted Language
Python is an interpreted language. It means the code is executed line-by-line by the interpreter at runtime.
Python යනු අර්ථකථනය කරන ලද භාෂාවකි, එනම් කේතය ධාවන වේලාවේදී අර්ථවින්‍යාසකය විසින් පේළියෙන් පේළිය ක්‍රියාත්මක කරයි.

Cross-Platform
Python is cross-platform. You can write Python code on one operating system and run it on another with little or no modification.
Python cross-platform වේ. ඔබට එක් මෙහෙයුම් පද්ධතියක ඡහඑයදබ කේතය ලිවිය හැකි අතර එය කුඩා හෝ වෙනස් කිරීමකින් තොරව තවත් හෙයුම් පද්ධතියක ධාවනය කළ හැක.

Problem-Solving Process
ගැටළු විසඳීමේ ක්‍රියාවලිය

1. Understand the Problem
ගැටලුව තේරුම් ගැනීම

Clarify the problem by thoroughly understanding what is being asked and ensuring all aspects of the issue are clear.
අසන දේ හොඳින් අවබෝධ කර ගැනීමෙන් සහ ගැටලුවේ සියලු පැති පැහැදිලි බව සහතික කිරීමෙන් ගැටලුව පැහැදිලි කරන්න.

Identify inputs and outputs: What are the inputs, and what should the outputs look like?
යෙදවුම් සහ ප්‍රතිදානයන් හඳුනා ගන්න: යෙදවුම් මොනවාද, සහ ප්‍රතිදානයන් කෙබඳු විය යුතුද?

Consider any limitations, such as time, memory, or legal constraints.
කාලය, මතකය, හෝ නීතිමය සීමාවන් වැනි ඕනෑම සීමාවක් සලකා බලන්න.

Root cause analysis
මූල හේතුව විශ්ලේෂණය

Identify the underlying issue that needs solving, not just the symptoms.
මතු පිට ලක්ෂණ පමණක් නොව, විසඳිය යුතු යටින් පවතින ගැටළුව හඳුනා ගන්න.

2. Break Down the Problem

Divide into smaller components .
කුඩා කොටස් වලට බෙදන්න.

Start with the simplest case, and build up from there.
සරලම කොටසින් ආරම්භ කරන්න, එතැන් සිට ගොඩනඟන්න.

Work on the most critical or time-sensitive parts first.
වඩාත්ම හදිසි හෝ කාලයට සංවේදී කොටස් මත පළමුව වැඩ කරන්න.

3. Explore Solutions
විසඳුම් ගවේෂණය කරන්න

Consider different approaches.
විවිධ ප්‍රවේශයන් සලකා බලන්න.

Analyze the pros and cons of each approach in terms of complexity, time, cost, and scalability.
සංකීර්ණත්වය, කාලය, පිරිවැය සහ පරිමාණය අනුව එක් එක් ප්‍රවේශයේ වාසි සහ අවාසි විශ්ලේෂණය කරන්න.

Look for similar problems you've solved before, and adapt those techniques.
ඔබ මීට පෙර විසඳා ඇති සමාන ගැටළු සොයන්න, සහ එම ශිල්පීය ක්‍රම අනුගත කරන්න.

4. Develop a Plan
සැලැස්මක් සකස් කරන්න

Create a step-by-step approach (Write down an outline or pseudocode.)
පියවරෙන් පියවර ක්‍රියවලියක් සාදන්න (දළ සටහනක් හෝ ව්‍යාජ කේතයක් ලියන්න.)

Ensure your plan covers all cases, edge cases, and scenarios.
ඔබගේ සැලසුම මගින් සියලුම අවස්ථා ආවරණය වන බවට සහතික වන්න.

5. Implement the Solution
විසඳුම ක්‍රියාත්මක කරන්න.

Implement the steps of the plan.
සැලැස්මේ පියවර ක්‍රියාත්මක කරන්න.

Test frequently during development to ensure each part works as expected.
එක් එක් කොටස අපේක්ෂිත පරිදි ක්‍රියා කරන බව සහතික කිරීම සඳහා සංවර්ධනය අතරතුර නිතර පරීක්ෂා කරන්න.

If something feels convoluted or inefficient, refactor to improve readability or performance.
යමක් ව්‍යාකූල හෝ අකාර්යක්ෂම බවක් හැඟේ නම්, කියවීමේ හැකියාව හෝ කාර්ය සාධනය වැඩි දියුණු කිරීම සඳහා නැවත සකස් කරන්න.

6. Test the Solution
විසඳුම පරීක්ෂා කරන්න

Test the solution with a variety of inputs, including edge cases and extreme values.
සියලුම අවස්ථා සහ ආන්තික අගයන් ඇතුළුව විවිධ යෙදවුම් සමඟ විසඳුම පරීක්ෂා කරන්න.

Compare expected vs. actual output, Ensure the output matches expectations.
අපේක්ෂිත ප්‍රතිදානයට එදිරිව සත්‍ය ප්‍රතිදානය සසඳන්න, ප්‍රතිදානය අපේක්ෂාවන්ට ගැළපෙන බවට සහතික වන්න.
If the solution is too slow or memory-heavy, go back and optimize it.
විසඳුම ඉතා මන්දගාමී හෝ memory-heavy නම්, ආපසු ගොස් එය ප්‍රශස්ත කරන්න.

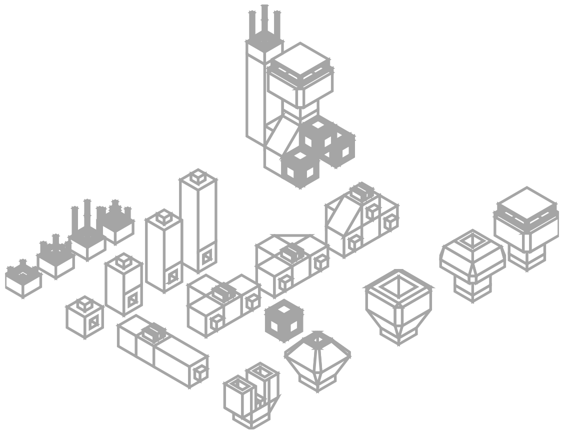
7. Review and Reflect
සමාලෝචනය

Evaluate whether the solution completely addressed the problem and identify any potential oversights.
විසඳුම සම්පූර්ණයෙන්ම ගැටලුව විසඳා තිබේද යන්න ඇගයීම සහ කිසියම් විභව අධීක්ෂණයක් හඳුනා ගන්න.

Evaluate if there is a more efficient or improved approach for addressing similar problems in the future.
අනාගතයේදී සමාන ගැටළු විසඳීම සඳහා වඩාත් කාර්යක්ෂම හෝ වැඩිදියුණු කළ ප්‍රවේශයක් තිබේදැයි තක්සේරු කරන්න.

Modularization
මොඩියුලරිකරණය

Modularization is a design principle that involves breaking down a complex problem into smaller, manageable, and self-contained modules or components
මොඩියුලීකරණය යනු සංකීර්ණ ගැටලුවක් කුඩා, කළමනාකරණය කළ හැකි සහ ස්වයං අන්තර්ගත මොඩියුල හෝ සංරචක බවට බිඳ දැමීම ඇතුළත් වන සැලසුම් මූලධර්මයකි.



Each module is designed to handle a specific aspect of the overall functionality, which promotes efficiency, reusability, and easier maintenance.
සෑම මොඩියුලයක්ම කාර්යක්ෂමතාව, නැවත භාවිතා කිරීමේ හැකියාව සහ පහසු නඩත්තුව ප්‍රවර්ධනය කරන සමස්ත ක්‍රියාකාරිත්වයේ නිශ්චිත අංගයක් හැසිරවීමට සැලසුම් කර ඇත.

Key Concepts of Modularization
මොඩියුලරිකරණයේ ප්‍රධාන සංකල්ප

Each module focuses on a distinct functionality or task, minimizing dependencies between them.
සෑම මොඩියුලයක්ම එකිනෙකට වෙනස් ක්‍රියාකාරිත්වයක් හෝ කාර්යයක් කෙරෙහි අවධානය යොමු කරයි, ඒවා අතර පරායත්තතා අවම කරයි.

Once a module is created, it can be reused across different parts of a system or even in different projects.
මොඩියුලයක් නිර්මාණය කළ පසු, එය පද්ධතියක විවිධ කොටස් හරහා හෝ විවිධ ව්‍යාපෘතිවල පවා නැවත භාවිතා කළ හැක.

Modules encapsulate their data and behavior, meaning they expose only what is necessary for other modules to interact with them.
මොඩියුල ඔවුන්ගේ දත්ත සහ හැසිරීම් සංග්‍රහ කරයි, එයින් අදහස් කරන්නේ ඒවා වෙනත් මොඩියුලවලට ඔවුන් සමඟ අන්තර් ක්‍රියා කිරීමට අවශ්‍ය දේ පමණක් හෙළි කරයි.

Modularity makes it easier to update or replace specific components without affecting the entire system.
මොඩියුලකරණය මගින් සම්පූර්ණ පද්ධතියට බලපෑම් නොකර නිශ්චිත සංරචක යාවත්කාලීන කිරීම හෝ ප්‍රතිස්ථාපනය කිරීම පහසු කරයි.

As systems grow, new features or functionalities can be added by creating new modules without overhauling the existing codebase.
පද්ධති වර්ධනය වන විට, පවතින කේත පදනම ප්‍රතිසංස්කරණය නොකර නව මොඩියුල නිර්මාණය කිරීමෙන් නව විශේෂාංග හෝ ක්‍රියාකාරීත්වය එක් කළ හැක.

Testing is simplified since each module can be tested independently.
එක් එක් මොඩියුලය ස්වාධීනව පරීක්ෂා කළ හැකි බැවින් පරීක්ෂණය කිරීම සරල වේ.

Top-Down Design
මුදුන් බිම් සැලැස්ම

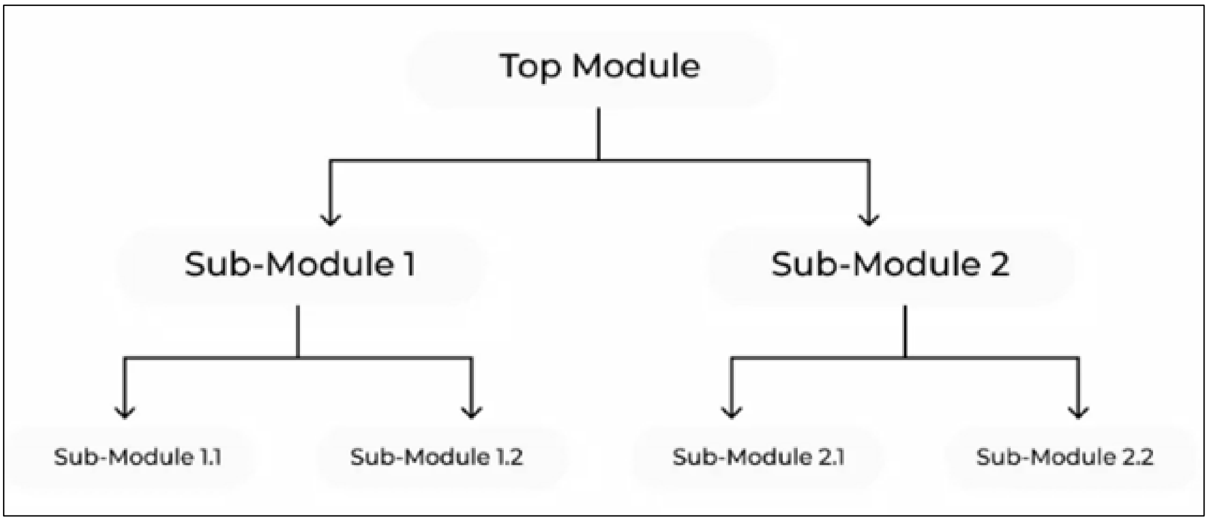
Top-down design is a systematic approach to problem-solving that starts with a high-level overview of a system or problem and progressively breaks it down into smaller, more detailed components or subproblems.
මුදුන් බිම් සැලැස්ම යනු පද්ධතියක් හෝ ගැටලුවක් පිළිබඳ ඉහළ මට්ටමේ දළ විශ්ලේෂණයකින් ආරම්භ වන ගැටළු විසඳීම සඳහා ක්‍රමානුකූල ප්‍රවේශයක් වන අතර එය ක්‍රමානුකූලව කුඩා, වඩාත් සවිස්තරාත්මක සංරචක හෝ උප ගැටළු වලට බිඳ දමයි.

Steps

Clearly articulate the main objective of the system or project.
පද්ධතියේ හෝ ව්‍යාපෘතියේ ප්‍රධාන අරමුණ පැහැදිලිව ප්‍රකාශ කරන්න.

Break the overall goal into its main functional components or modules.
සමස්ත ඉලක්කය එහි ප්‍රධාන ක්‍රියාකාරී සංරචක හෝ මොඩියුලවලට බිඳ දමන්න.

For each major component, further break it down into smaller subcomponents or tasks.
සෑම ප්‍රධාන සංරචකයක් සඳහාම, එය තවදුරටත් කුඩා උප සංරචක හෝ කාර්යයන් බවට බිඳ දමන්න.



Determine how each component interacts with others, outlining their relationships and dependencies.
එක් එක් සංරචක අන් අය සමඟ අන්තර්ක්‍රියා කරන ආකාරය, ඒවායේ සම්බන්ධතා සහ යැපීම් ගෙනහැර දක්වන්න.

Once the hierarchy is established, begin detailing each component's functionality, inputs, and outputs.
ධුරාවලිය ස්ථාපිත වූ පසු, එක් එක් සංරචකයේ ක්‍රියාකාරීත්වය, යෙදවුම් සහ ප්‍රතිදානයන් විස්තර කිරීම ආරම්භ කරන්න.

Analyze the structure for completeness and coherence, making adjustments as necessary.
සම්පූර්ණත්වය සහ සමෝධානය සඳහා ව්‍යුහය විශ්ලේෂණය කරන්න, අවශ්‍ය පරිදි ගැලපීම් සිදු කරන්න.

Stepwise Refinement
පියවරාකාර පිරිපහදු ක්‍රමවේදය

Stepwise Refinement is a problem-solving technique that involves breaking down a complex task into smaller, simpler steps or sub-tasks until each is clear and manageable for implementation.
පියවරාකාර පිරිපහදු ක්‍රමවේදය යනු එක් එක් ක්‍රියාවට නැංවීම සඳහා පැහැදිලි සහ කළමනාකරණය කළ හැකි වන තෙක් සංකීර්ණ කාර්යයක් කුඩා, සරල පියවර හෝ උප-කාර්යයන් බවට බිඳ දැමීම ඇතුළත් වන ගැටළු විසඳීමේ තාක්ෂණයකි.

Steps

Define the primary task or problem that needs to be solved.
විසඳිය යුතු මූලික කාර්යය හෝ ගැටලුව නිර්වචනය කරන්න.

Divide the main task into smaller sub-tasks or steps that are more specific.
ප්‍රධාන කාර්යය කුඩා උප කාර්යයන් හෝ වඩාත් නිශ්චිත පියවර වලට බෙදන්න.

For each sub-task, further break it down into even smaller, manageable tasks until they can be easily understood and implemented.
එක් එක් උප කර්තව්‍යය සඳහා, ඒවා පහසුවෙන් තේරුම් ගෙන ක්‍රියාත්මක කළ හැකි වන තෙක් එය ඊටත් වඩා කුඩා, කළමනා කළ හැකි කාර්යයන් බවට එය තවදුරටත් බිඳ දමන්න.

Clearly describe what needs to be done for each small task, including inputs, outputs, and processes.
යෙදවුම්, ප්‍රතිදාන සහ ක්‍රියාවලි ඇතුළුව එක් එක් කුඩා කාර්යය සඳහා කළ යුතු දේ පැහැදිලිව විස්තර කරන්න.

Implement each small task individually, testing as you go to ensure correctness and functionality.
සෑම කුඩා කාර්යයක්ම තනි තනිව ක්‍රියාත්මක කරන්න, නිවැරදි බව සහ ක්‍රියාකාරීත්වය සහතික කිරීමට ඔබ යන විට පරීක්ෂා කරන්න.

If any task is found to be too complex or unclear, repeat the refinement process to break it down further.
කිසියම් කාර්යයක් ඉතා සංකීර්ණ හෝ නොපැහැදිලි බව සොයා ගතහොත්, එය තවදුරටත් බිඳ දැමීම සඳහා පිරිපහදු කිරීමේ ක්‍රියාවලිය නැවත කරන්න.

Structure charts
ව්‍යුහ සටහන්

Structure charts are a visual representation of a system's architecture, illustrating the relationships and hierarchies between different components, modules, or functions within a software application.
ව්‍යුහ සටහන් යනු මෘදුකාංග යෙදුමක් තුළ විවිධ සංරචක, මොඩියුල හෝ කාර්යයන් අතර සම්බන්ධතා සහ ධූරාවලිය නිදර්ශනය කරන පද්ධතියේ ගෘහ නිර්මාණ ශිල්පයේ දෘශ්‍ය නිරූපණයකි.

Types of Structure Charts
ව්‍යුහ සටහන් වර්ග

1.Transaction Structure Chart
ගනුදෙනු ව්‍යුහ සටහන

A Transaction Structure Chart routes control to different modules based on user input, ideal for systems with multiple independent operations.
ගනුදෙනු ව්‍යුහ සටහනක් පරිශීලක ආදානය මත පදනම්ව විවිධ මොඩියුල වෙත පාලනය ලැබීම දක්වයි, බහු ස්වාධීන මෙහෙයුම් සහිත පද්ධති සඳහා වඩාත් සුදුසුය.

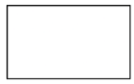
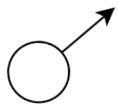
2. Transform Structure Chart
පරිවර්තන ව්‍යුහ සටහන

A Transform Structure Chart shows sequential data processing steps, used in systems with linear workflows.
පරිවර්තන ව්‍යුහ සටහනක් රේඛීය කාර්ය ප්‍රවාහයන් සහිත පද්ධතිවල පිහිටුනු අනුවාද ක්‍රම, දත්ත සැකසුම් පියවර පෙළගැසීමේ ක්‍රමය ප්‍රදර්ශනය කරයි.

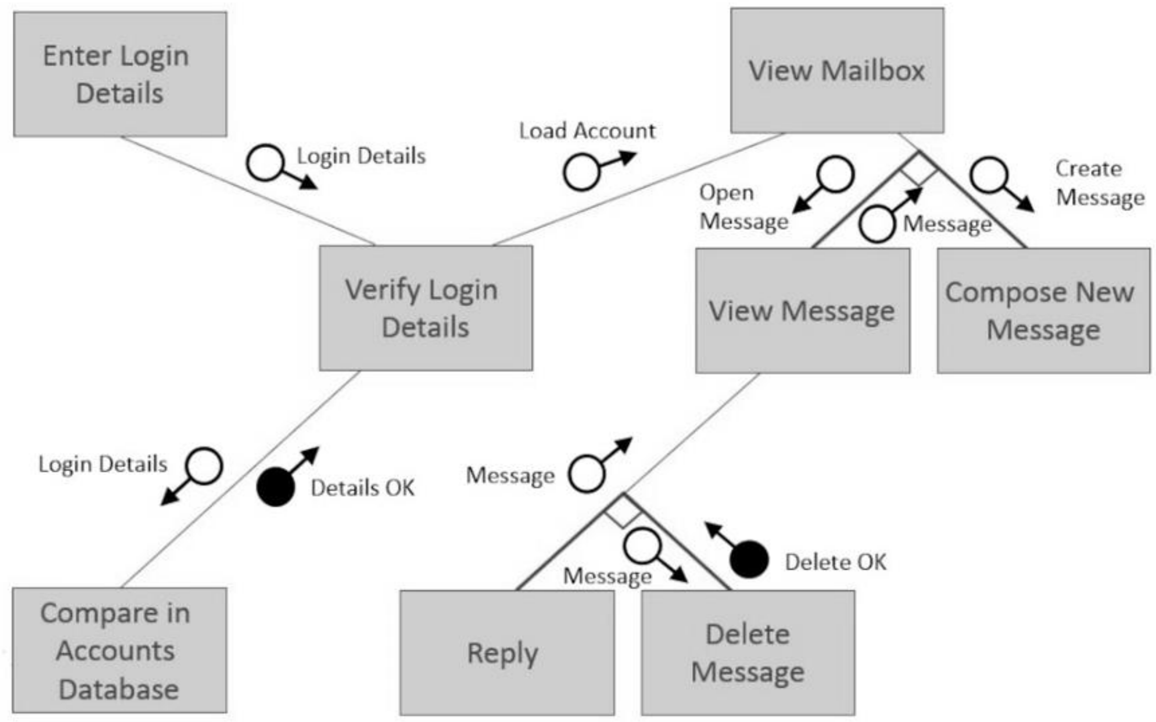
3. Hybrid Structure Chart
දෙමුහුන් ව්‍යුහ සටහන

A Hybrid Structure Chart combines decision-making and sequential processing, suited for complex systems.
දෙමුහුන් ව්‍යුහ ප්‍රස්ථාරයක් සංකීර්ණ පද්ධති සඳහා සුදුසු තීරණ ගැනීමේ සහ අනුක්‍රමික සැකසුම් ඒකාබද්ධ කරයි.

Shapes used in structure charts
ව්‍යුහ සටහන්වල භාවිතා වන හැඩතල

A module or a process in the system. මොඩියුල අතර පාලන ප්‍රවාහය හෝ අන්තර්ක්‍රියා.	
The flow of control or interaction between modules. ක්‍රියාවලි අතර දත්ත ගලායාම පෙන්නුම් කරයි	
A decision or condition in the flow of control. පාලන ප්‍රවාහයේ තීරණයක් හෝ කොන්දේසියක්.	
Represents a repetition and highlight that a process can occur multiple times පුනරාවර්තනයක් නියෝජනය කිරීමට සහ ක්‍රියාවලියක් කිහිප වතාවක් සිදුවිය හැකි බව ඉස්මතු කිරීමට භාවිතා කරයි	
Indicates the flow of data between processes ක්‍රියාවලි අතර දත්ත ගලායාම පෙන්නුම් කරයි	
Indicate that a criteria has been met, Providing confirmation for the system to proceed පද්ධතිය ඉදිරියට යාමට තහවුරු කිරීමක් ලබා දෙමින් නිර්ණායකයක් සපුරා ඇති බව දක්වයි	

Example: The following Structure Chart outlines the use of an Email Server



Steps to Create a Structure Chart
ව්‍යුහ සටහනක් සෑදීමට පියවර

Break down the system into major functions or processes.
පද්ධතිය ප්‍රධාන කාර්යයන් හෝ ක්‍රියාවලි වලට බිඳ දමන්න.

Decompose each function into smaller sub-functions or modules, each performing a distinct task.
එක් එක් ශ්‍රිතය කුඩා උප ශ්‍රිත හෝ මොඩියුල වලට වියෝජනය කරන්න, ඒ සෑම එකක්ම වෙනම කාර්යයක් ඉටු කරයි.

Specify how each module interacts with others and draw the necessary control and data flow between them.
එක් එක් මොඩියුලය අන් අය සමඟ අන්තර්ක්‍රියා කරන ආකාරය සඳහන් කර ඒවා අතර අවශ්‍ය පාලනය සහ දත්ත ප්‍රවාහය අඳින්න.

Arrange the modules in a hierarchical structure, with the most controlling modules at the top.
ඉහළම පාලන මොඩියුල සමඟ මොඩියුල ධූරාවලි ව්‍යුහයක් තුළ සකසන්න.

Programming Languages
ක්‍රමලේඛන භාෂා

Programming languages are formal systems of communication used to instruct computers on how to perform specific tasks.
ක්‍රමලේඛන භාෂා යනු නිශ්චිත කාර්යයන් ඉටු කරන ආකාරය පිළිබඳව පරිගණකවලට උපදෙස් දීමට භාවිතා කරන විධිමත් සන්නිවේදන පද්ධති වේ.

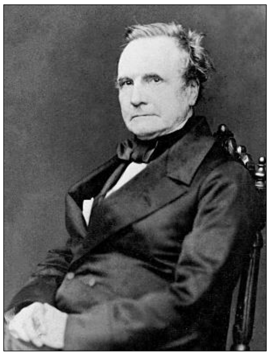
Evolution of programming
ක්‍රමලේඛනකරණයේ පරිණාමය

The title of "first programmer" is often attributed to Ada Lovelace, an English mathematician and writer, due to her work in the mid-19th century.
"පළමු ක්‍රමලේඛකයා" යන නාමය බොහෝ විට 19 වන සියවස මැද භාගයේදී ඇය කළ කාර්යය නිසා ඉංග්‍රීසි ගණිතඥයෙකු හා ලේඛිකාවක් වූ ඒඩා ලව්ලේස්ට ආරෝපණය කරනු ලැබේ.

She collaborated with Charles Babbage on his design for an early mechanical general-purpose computer known as the Analytical Engine.
ඇය චාල්ස් බැබේජ් සමඟ එක්ව විශ්ලේෂණාත්මක එන්ජිම (Analytical Engine) ලෙස හැඳින්වෙන මුල් යාන්ත්‍රික සාමාන්‍ය කාර්ය පරිගණකයක් සඳහා ඔහුගේ සැලසුම සඳහා සහයෝගයෙන් කටයුතු කළාය.

Ada Lovelace

Charles Babbage



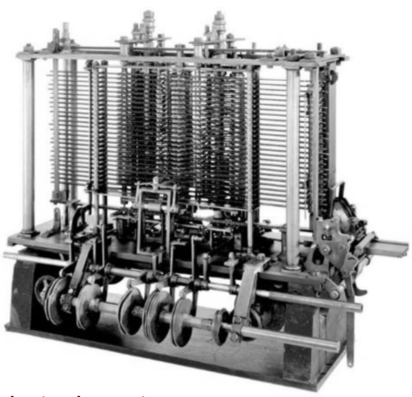
1. Machine Code (1940s-1950s)
යන්ත්‍ර කේතය

The first "language" was machine code, consisting of binary instructions (0s and 1s) directly understood by computers.
පළමු "භාෂාව" යනු පරිගණක මගින් සෘජුවම අවබෝධ කරගත් ද්විමය උපදෙස් (0 සහ 1) වලින් සමන්විත යන්ත්‍ර කේතයයි.

Machine code is specific to the hardware architecture of the computer (Ex. x86, ARM) and difficult for humans to write and read.
යන්ත්‍ර කේතය පරිගණකයේ දෘඩාංග ගෘහ නිර්මාණ ශිල්පයට විශේෂිත වන අතර (උදා: .86, ARM) මිනිසුන්ට ලිවීමට සහ කියවීමට අපහසු වේ.

Example:

```
00100101 11010011
00100100 11010100
10001010 01001001 11110000
01000100 01010100
01001000 10100111 10100011
11100101 10101011 00000010
00101001
11010101
11010100 10101000
10010001 01000100
```



Analytical Engine

2. Assembly Language **(1950s)**
ඇසෙම්බ්ලි භාෂාව

Assembly language provided a small level of abstraction over machine code, using mnemonics instead of binary to represent operations.
ඇසෙම්බ්ලි භාෂාව මඟින් යන්ත්‍ර කේතය හරහා කුඩා මට්ටමේ වියුක්ත කිරීමක් සපයන ලදී, මෙහෙයුම් නියෝජනය කිරීම සඳහා ද්විමය වෙනුවට සිහිවටන භාවිතා කරයි.

While it is hardware-specific, assembly is easier to read than machine code but still requires a deep understanding of the computer's architecture.
එය දෘඩාංග විශේෂිත වුවත්, එකලස් කිරීම යන්ත්‍ර කේතයට වඩා කියවීමට පහසු වන නමුත් පරිගණකයේ ගෘහ නිර්මාණ ශිල්පය පිළිබඳ ගැඹුරු අවබෝධයක් අවශ්‍ය වේ.

3. Early High-Level Languages
මුල් උසස් මට්ටමේ භාෂා
(1950s-1960s)

High-level languages abstracted away from machine details, allowing programs to be written more naturally using mathematical notation or symbolic expressions are known as High level languages.
ගණිතමය අංකනය හෝ සංකේතාත්මක ප්‍රකාශන භාවිතයෙන් වැඩසටහන් වඩාත් ස්වභාවිකව ලිවීමට ඉඩ සලසමින් යන්ත්‍ර විස්තර වලින් ඈත් වූ ඉහළ මට්ටමේ භාෂා වේ.

These languages made programming accessible to more people, allowing complex tasks to be written more concisely.
මෙම භාෂා මඟින් ක්‍රමලේඛනය වැඩි පිරිසකට ප්‍රවේශ විය හැකි අතර, සංකීර්ණ කාර්යයන් වඩාත් සංක්ෂිප්තව ලිවීමට ඉඩ සලසයි.

Examples
FORTRAN (1957), COBOL (1959), LISP (1958), ALGOL (1958)

4. Structured Programming **(1970s)**
ව්‍යුහගත ක්‍රමලේඛනකරණය

A paradigm that emphasized a clear, modular approach to code, making it easier to understand, maintain, and debug.
කේතය සඳහා පැහැදිලි, මොඩියුලර් ප්‍රවේශයක් අවධාරනය කරන ලද සුසමාදර්ශයක්, එය තේරුම් ගැනීමට, නඩත්තු කිරීමට සහ දෝෂහරණය කිරීමට පහසු කරයි.

Encouraged the use of control structures (loops, conditionals) and functions, helping programmers manage larger and more complex software.
පාලන ව්‍යුහයන් (ලූප, කොන්දේසි) සහ ශ්‍රිත භාවිතය දිරිමත් කරන ලද අතර, ක්‍රමලේඛකයින්ට විශාල සහ වඩාත් සංකීර්ණ මෘදුකාංග කළමනාකරණය කිරීමට උපකාර කරයි.

Examples
C (1972), Pascal (1970)

5. Object-Oriented Programming
වස්තු නැඹුරු ක්‍රමලේඛනකරණය
(1980s-1990s)

Object-oriented programming (OOP) introduced the concept of organizing code into "objects", self-contained units that contain both data and the methods that operate on the data.
වස්තු-නැඹුරු ක්‍රමලේඛනය (OOP) විසින් කේතය සංවිධානය කිරීමේ සංකල්පය "වස්තු" බවට හඳුන්වා දෙන ලදී. දත්ත සහ දත්ත මත ක්‍රියාත්මක වන ක්‍රම යන දෙකම අඩංගු ස්වයං අන්තර්ගත ඒකක මෙම වස්තු තුල ඇතුලත් වේ.

OOP allowed better modeling of real-world problems, promoting code reuse, encapsulation, inheritance, and polymorphism.
OOP විසින් තථ්‍ය-ලෝක ගැටලු වඩා හොඳ ආකෘති නිර්මාණය කිරීමට, කේත නැවත භාවිතය ප්‍රවර්ධනය කිරීමට, සංවෘත කිරීමට, උරුමයට සහ බහුරූපතාවට ඉඩ ලබා දී ඇත .

Examples
Smalltalk (1980), C++ (1985), Java (1995

6. Scripting and Web Development
ස්ක්‍රිප්ටින් සහ වෙබ් සංවර්ධනය
(1990s-2000s)

As the web gained prominence, scripting languages became essential for creating dynamic, interactive content and automating tasks.
වෙබ්, ප්‍රමුඛත්වය ලබා ගත් විට, ගතික, අන්තර්ක්‍රියාකාරී අන්තර්ගතයන් සහ ස්වයංක්‍රීය කාර්යයන් නිර්මාණය කිරීම සඳහා ස්ක්‍රිප්ටින් භාෂා අත්‍යවශ්‍ය විය.

These languages made web development easier, enabling the growth of dynamic, data-driven websites and applications.
ගතික, දත්ත මත පදනම් වූ වෙබ් අඩවි සහ යෙදුම්වල වර්ධනය සක්‍රීය කරමින් මෙම භාෂා වෙබ් සංවර්ධනය පහසු කළේය.

Examples
JavaScript (1995), PHP (1995), Python (1991), Ruby (1995)

7. The Rise of Multi-Paradigm and Modern Languages (2000s-2020s)
බහු සුසමාදර්ශ සහ නවීන භාෂා වල ආරම්භය

Modern languages often support multiple paradigms, combining the best features from procedural, object-oriented, and functional programming.
ක්‍රියාපටිපාටික, වස්තු-නැඹුරු සහ ක්‍රියාකාරී ක්‍රමලේඛන වලින් හොඳම විශේෂාංග ඒකාබද්ධ කරමින් නවීන භාෂා බොහෝ විට බහු සුසමාදර්ශ සඳහා සහය දක්වයි.

These languages reflect a shift towards safety, concurrency, and developer productivity, balancing ease of use with performance.
මෙම භාෂා ආරක්ෂාව, සමගාමීත්වය සහ සංවර්ධක ඵලදායිතාව වෙත නැඹුරුවීමක් පිළිබිඹු කරයි, කාර්ය සාධනය සමඟ භාවිතයේ පහසුව තුලනය කරයි.
Examples
C# (2000), Scala (2003), Go (2009), Rust (2010), Kotlin (2011)

Programming Paradigms
ක්‍රමලේඛන සුසමාදර්ශ

A programming paradigm is a fundamental style or approach to programming that defines how code is organized, structured, and executed.
ක්‍රමලේඛන සුසමාදර්ශයක් යනු කේත සංවිධිත, ව්‍යුහගත සහ ක්‍රියාත්මක වන ආකාරය නිර්වචනය කරන ක්‍රමලේඛනයට මූලික ශෛලියක් හෝ ප්‍රවේශයකි.

It represents a particular way of thinking about how to solve problems and structure programs.
එය ගැටළු විසඳීමට සහ වැඩසටහන් ව්‍යුහගත කරන්නේ කෙසේද යන්න පිළිබඳ විශේෂිත චින්තනයක් නියෝජනය කරයි.

Types of Programming Paradigms
ක්‍රමලේඛන සුසමාදර්ශ වර්ග:

1. Declarative Programming
ප්‍රකාශාත්මක ක්‍රමලේඛකරණය

A programming paradigm that focuses on specifying what the program should accomplish without detailing how to achieve it.
වැඩසටහන ඉටු කළ යුතු දේ සඳහන් කිරීම කෙරෙහි අවධානය යොමු කරන නමුත් එය සිදු කරන්නේ කෙසේද යන්න විස්තර නොකරන ක්‍රමලේඛන සුසමාදර්ෂයකි.

Characteristics
ලක්ෂණ

Emphasizes expressing the desired outcome, abstracting the implementation details.
ක්‍රියාත්මක කිරීමේ විස්තර සාරාංශ කරමින් අපේක්ෂිත ප්‍රතිඵලය ප්‍රකාශ කිරීම අවධාරණය කරයි.

Often results in more readable and maintainable code.
බොහෝ විට වඩාත් කියවිය හැකි සහ නඩත්තු කළ හැකි කේතයක් ඇති කරයි.

Examples

- **SQL**
  A query like `SELECT * FROM Users WHERE age > 30` specifies what data to retrieve without detailing the retrieval process.
  `SELECT * FROM Users WHERE age > 30` වැනි විමසුමක් ලබා ගැනීමේ ක්‍රියාවලිය විස්තර නොකර ලබා ගත යුතු දත්ත මොනවාදැයි සඳහන් කරයි.

- **HTML**
  Defines the structure of web pages by specifying elements without dictating how they should be rendered.
  ඒවා විදැහුම් කළ යුතු ආකාරය නියම නොකර මූලද්‍රව්‍ය සඳහන් කිරීමෙන් වෙබ් පිටු ව්‍යුහය නිර්වචනය කරයි.

2. Imperative Programming
විධානාත්මක ක්‍රමලේඛකරණය

A programming paradigm that involves writing code to explicitly define how to perform tasks through sequences of commands.
විධාන අනුපිළිවෙලවල් හරහා කාර්යයන් ඉටු කරන ආකාරය පැහැදිලිව නිර්වචනය කිරීම සඳහා කේතය ලිවීම ඇතුළත් ක්‍රමලේඛන සුසමාදර්ශයකි.

Characteristics
ලක්ෂණ

Centers on detailing the steps to achieve a result.
ප්‍රතිඵලයක් ළඟා කර ගැනීමට අවශ්‍ය පියවර ගැන විස්තර කිරීම මූලික වී ඇත.

Involves managing program state through variable assignments and control structures.
විචල්‍ය පැවරුම් සහ පාලන ව්‍යුහයන් හරහා වැඩසටහන් තත්ත්වය කළමනාකරණය කිරීම සම්බන්ධ වේ.

Ex. C, C++, Java, Python, JavaScript, Ruby, Go, FORTRAN, Swift

3. Object-Oriented Programming (OOP)
වස්තු-නැඹුරු ක්‍රමලේඛකරණය

A programming paradigm that organizes software design around objects, which are instances of classes that encapsulate both data (attributes) and behaviors (methods).
වස්තු වටා මෘදුකාංග නිර්මාණය සංවිධානය කරන ක්‍රමලේඛන සුසමාදර්ශයක්, එනම් දත්ත (ගුණාංග) සහ හැසිරීම් (ක්‍රම) යන දෙකම ඇතුළත් වන පන්තිවල අවස්ථා වේ.

Classes

A class is a blueprint for creating objects.
පන්තියක් යනු වස්තූන් නිර්මාණය කිරීම සඳහා වූ සැලැස්මකි.

It defines the properties (also called attributes or fields) and behaviors (also called methods or functions) that the objects created from the class will have.
එමගින් පන්තියෙන් සාදන ලද වස්තූන් සතු ගුණාංග (උපලක්ෂණ හෝ ක්ෂේත්‍ර ලෙසද හැඳින්වේ) සහ හැසිරීම් (ක්‍රම හෝ ශ්‍රිත ලෙසද හැඳින්වේ) නිර්වචනය කරයි.

Objects

An object is an instance of a class.
වස්තුවක් යනු පන්තියක එක් අවස්ථාවකි.

Objects are actual entities that hold real data, and they can interact with one another.
වස්තු යනු සත්‍ය දත්ත රඳවා තබා ගන්නා සත්‍ය භූතාර්ත වන අතර ඒවාට එකිනෙකා සමඟ අන්තර් ක්‍රියා කළ හැකිය.

If the class is the blueprint, the object is the actual house built from that blueprint.
පන්තිය සැලසුම නම්, වස්තුව එම සැලැස්මෙන් ගොඩනඟන ලද සැබෑ නිවසයි.

Ex. Java, C++, C#, Python, PHP, JavaScript, Kotlin, Smalltalk, Visual basic .NET(VB NET), TypeScript

ictfromabc.com

Game Characters

Class: `Character`
Attributes:
role, weapon, health, speed, alignment
Behaviors:
attack, patrol, defend, interact.

Objects

Main Player (Object of `Character` class):
Role = Player, Weapon = Rifle, Health = High, Speed = Fast, Alignment = Friendly.
Behaviors: Attack enemies, complete objectives.

Soldier (Object of `Character` class):
Role = Enemy, Weapon = Gun, Health = Medium, Speed = Moderate, Alignment = Hostile.
Behaviors: Patrol, attack the player.

Civilian (Object of `Character` class):
Role = Non-combatant, Weapon = None, Health = Low, Speed = Slow, Alignment = Neutral.
Behaviors: Walk, talk, flee danger.

Pedestrian (Object of `Character` class):
Role = Bystander, Weapon = None, Health = Low, Speed = Moderate, Alignment = Neutral.
Behaviors: Wander aimlessly.

If you replace a Soldier's file with a Civilian's, the soldier loses their weapon, stop patrolling, and start behaving passively. Similarly, the soldier now flees when they see the player instead of attacking.
සොල්දාදුවෙකුට අදාල ගොනුවක් සිවිලියන් කෙනෙකුගේ ගොනුවක් සමඟ ප්‍රතිස්ථාපනය කළහොත්, සොල්දාදුවාට ඔහුගේ ආයුධය අහිමි වේ, මුර කිරීම නතර කර නිෂ්ක්‍රීයව හැසිරීමට පටන් ගනී. එමෙන්ම, සොල්දාදුවා දැන් පහර දෙනවා වෙනුවට player ව දුටු විට පලා යයි.

Cars in a Game

Class: `Car`
Attributes:
color, model, speed, fuel type, durability.
Behaviors:
drive, brake, refuel, drift.

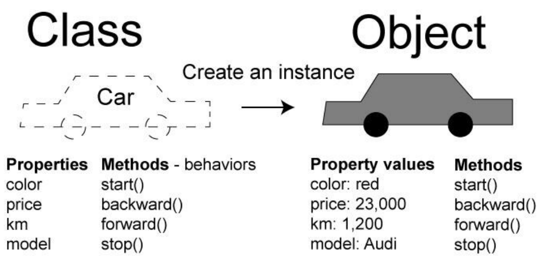
Objects

Sports Car (Object of Car class):
Color = Red, Model = Ferrari, Speed = 200 mph, Fuel Type = Gasoline, Durability = Low.
Behaviors: High speed, quick acceleration.

Truck (Object of Car class):
Color = Blue, Model = Freightliner, Speed = 80 mph, Fuel Type = Diesel, Durability = Very High.
Behaviors: Carry heavy loads.

Police Car (Object of Car class):
Color = White, Model = Dodge Charger, Speed = 150 mph, Fuel Type = Gasoline, Durability = Medium.
Behaviors: Pursue suspects, use sirens.

If you take a Sports Car's file and replace it to a Truck's, the car would become slower, gain more durability, and lose its ability to accelerate quickly.
ගොනුවක් ගෙන එය ට්‍රක් රථයක ගොනුවක් සමග ප්‍රතිස්ථාපනය කළහොත්, මෝටර් රථය මන්දගාමී වේ, වැඩි කල්පැවැත්මක් සහ ඉක්මනින් ත්වරණය කිරීමේ හැකියාව නැති වේ.



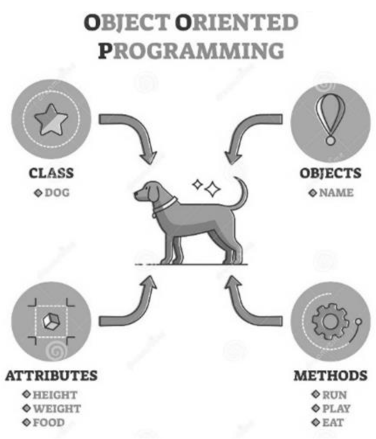
Characteristics
ලක්ෂණ

Encapsulation
Bundles data and behavior within objects, promoting modularity.
දත්ත හා හැසිරීම් වස්තු තුළ අන්තර්ගත කරමින් මොඩියුලර් නිර්මාණයක් ප්‍රවර්ධනය කරයි.

Inheritance
Allows new classes to be derived from existing ones, facilitating code reuse.
නව පන්ති පවතින ඒවායින් ව්‍යුත්පන්න කිරීමට ඉඩ සලසයි, කේත නැවත භාවිතයට පහසුකම් සපයයි.

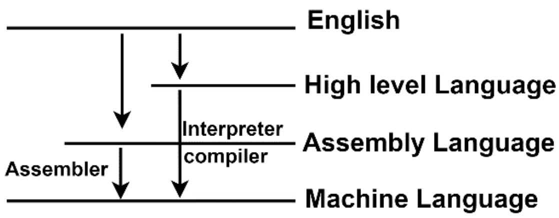
Polymorphism
Enables objects of different types to be treated through a common interface.
විවිධ වර්ගවල වස්තු සාමාන්‍ය අතුරු මුහුණතක් මඟින් සලකා බැලීමට හැකියාව සලසයි.

Abstraction
Hides complex implementation details, showing only necessary parts of functionality.
සංකීර්ණ ක්‍රියාවලිය සැඟවී, අවශ්‍ය කොටස් පමණක් පෙන්වයි.



The need of translation of programs
වැඩසටහන් පරිවර්තනය කිරීමේ අවශ්‍යතාවය

Programs need to be converted into a language the computer understands, binary machine code, because computers can only execute instructions as 1s and 0s.
පරිගණක වලට උපදෙස් ක්‍රියාත්මක කළ හැක්කේ 1 සහ 0 ලෙස පමණක් නිසා වැඩසටහන් පරිගණකයට තේරෙන භාෂාවකට, එනම් ද්විමය යන්ත්‍ර කේතයක් බවට පරිවර්තනය කළ යුතුය.



Source Code
මූලාශ්‍ර කේතය

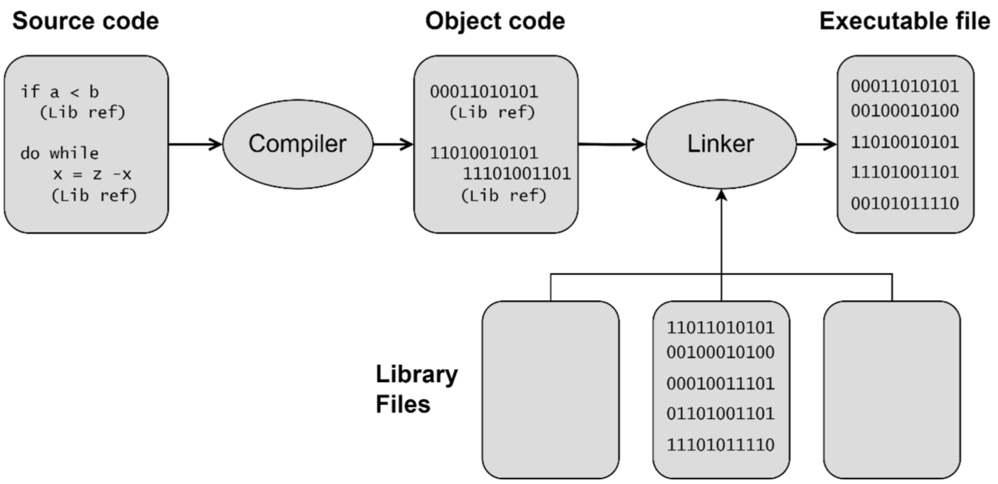
Source code is the human-readable set of instructions written in a programming language.
මූලාශ්‍ර කේතය යනු ක්‍රමලේඛන භාෂාවකින් ලියා ඇති මිනිසුන්ට කියවිය හැකි උපදෙස් මාලාවකි.

Object Code
වස්තු කේතය

Object code is the machine-readable version of the source code generated by a compiler or assembler.
වස්තු කේතය යනු සම්පාදකයෙකු හෝ එකලස් කරන්නෙකු විසින් ජනනය කරන ලද මූලාශ්‍ර කේතයේ යන්ත්‍රයෙන් කියවිය හැකි අනුවාදයයි.

Object code is often not human-readable and is typically stored in files with extensions like .o or .obj.
වස්තු කේතය බොහෝ විට මිනිසුන්ට කියවිය නොහැකි අතර සාමාන්‍යයෙන් .o හෝ .obj වැනි දිගු සහිත ගොනු වල ගබඩා වේ.

This code may be linked with other object code files and libraries.
මෙම කේත වෙනත් වස්තු කේත ගොනු සහ libraries සමඟ සම්බන්ධ කළ හැක.



Comparison

Feature	**Source Code**	**Object Code**
Definition අර්ථ දැක්වීම	Human-readable instructions written in a programming language. ක්‍රමලේඛන භාෂාවකින් ලියා ඇති මිනිසුන්ට කියවිය හැකි උපදෙස්.	Machine-readable code generated from the source code. යන්ත්‍රයෙන් කියවිය හැකි කේතය මූලාශ්‍ර කේතයෙන් ජනනය වේ.
Format ආකෘතිය	Typically plain text files සාමාන්‍යයෙන් සරල පෙළ ගොනු ආකෘතිකරණය පවතී (Ex. .c, .py, .java).	Binary format (often in .o or .obj files). ද්විමය ආකෘතිය (බොහෝ විට .o හෝ .obj ගොනු වල) පවතී
Readability කියවීමේ හැකියාව	Easy to read and understand by humans. මිනිසුන්ට කියවීමට සහ තේරුම් ගැනීමට පහසුය.	Difficult for humans to read; meant for machines. මිනිසුන්ට කියවීමට අපහසුය; යන්ත්‍ර සඳහා වේ.
Compilation සම්පාදනය	Needs to be compiled or interpreted to be executed. ක්‍රියාත්මක කිරීම සඳහා සම්පාදනය කිරීම හෝ අර්ථකථනය කිරීම අවශ්‍ය වේ.	Already compiled; can be executed directly by a machine (with linking). දැනටමත් සම්පාදනය කර ඇත; යන්ත්‍රයක් මගින් සෘජුවම ක්‍රියාත්මක කළ හැක (සබැඳි කිරීම සමඟ).
Modification වෙනස් කිරීම	Can be easily modified and edited by programmers. ක්‍රමලේඛකයින් හට පහසුවෙන් වෙනස් කර සංස්කරණය කළ හැක.	Difficult to modify without disassembling. විසුරුවා හැරීමකින් තොරව වෙනස් කිරීම අපහසුය.
Use in Development සංවර්ධනයේදී භාවිතය	Used during the development phase for writing programs. වැඩසටහන් ලිවීම සඳහා සංවර්ධන අවධියේදී භාවිතා වේ.	Used during the linking phase and execution of programs වැඩසටහන් සම්බන්ධ කිරීමේ අදියර සහ ක්‍රියාත්මක කිරීමේදී භාවිතා වේ.
Example	print("Hello, World!") in Python. පයිතන් හි print("Hello, World!")	Binary representation of the compiled code (Ex. 110100101...). සම්පාදනය කරන ලද කේතයේ ද්විමය නිරූපණයයි(Ex. 110100101...).

Language translators
භාෂා පරිවර්තකයන්

Programs that convert programs written in high-level programming languages into machine code are known as language translators.
ඉහළ මට්ටමේ ක්‍රමලේඛන භාෂාවෙන් ලියන ලද වැඩසටහන් යන්ත්‍ර කේතයක් බවට පරිවර්තනය කරන වැඩසටහන් භාෂා පරිවර්තක ලෙස හැඳින්වේ.

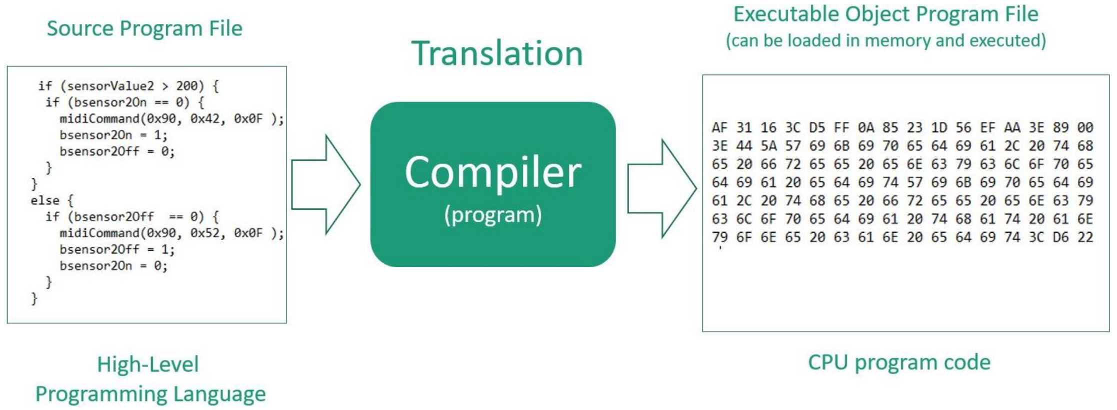
Additionally, they can translate code between different programming languages.
මීට අමතරව, ඔවුන්ට විවිධ ක්‍රමලේඛන භාෂා අතර කේත පරිවර්තනය කළ හැකිය.

Types of Language Translators
භාෂා පරිවර්තක වර්ග

1. Compilers
සම්පාදකයන්

A compiler translates source code written in a high-level programming language (Ex. C, C++, Java, python) into machine code.
සම්පාදකයක් ඉහළ මට්ටමේ ක්‍රමලේඛන භාෂාවකින් (උදා: C, C++, Java, python) ලියා ඇති මූල කේතය යන්ත්‍ර කේතයට පරිවර්තනය කරයි.

This translation occurs before the program runs, producing an executable file.
ක්‍රියාත්මක කළ හැකි ගොනුවක් නිෂ්පාදනය කරමින් වැඩසටහන ක්‍රියාත්මක වීමට පෙර මෙම පරිවර්තනය සිදු වේ.



Process
ක්‍රියාවලිය

Lexical Analysis
Scans the source code and breaks it down into tokens.
ලෙක්සිකල් විශ්ලේෂණය: මූලාශ්‍ර කේතය පරිලෝකනය කර එය ටෝකන වලට කඩා දමයි.

Syntax Analysis
Organizes tokens according to the grammatical rules of the language, forming a parse tree.
වාක්‍ය ඛණ්ඩ විශ්ලේෂණය: භාෂාවේ ව්‍යාකරණ රීති අනුව ටෝකන සංවිධානය කරයි, එමගින් විග්‍රහ ගසක් (parse tree) සාදයි.

Semantic Analysis
Checks for logical errors and verifies the operations' correctness.
අර්ථ විශ්ලේෂණ: තාර්කික දෝෂ සඳහා පරීක්ෂා කර මෙහෙයුම් වල නිවැරදි බව තහවුරු කරයි.

Code Generation
Converts the parse tree into machine code.
කේත උත්පාදනය: විග්‍රහ කරන ගස (parse tree) යන්ත්‍ර කේතය බවට පරිවර්තනය කරයි.

Optimization
Improves the efficiency of the generated code
ප්‍රශස්තකරණය: ජනනය කරන ලද කේතයේ කාර්යක්ෂමතාව වැඩි දියුණු කරයි

ictfromabc.com

Advantages

- Compiled code typically executes faster than interpreted code due to direct translation into machine code.
  සම්පාදනය කරන ලද කේතය සාමාන්‍යයෙන් යන්ත්‍ර කේතයට සෘජු පරිවර්තනය හේතුවෙන් අර්ථකථනය කරන ලද කේතයට වඩා වේගයෙන් ක්‍රියාත්මක වේ.

- Compilers catch many errors at compile time, allowing developers to fix issues before runtime.
  සංවර්ධකයින්ට ධාවන කාලයට පෙර ගැටළු නිරාකරණය කිරීමට ඉඩ සලසමින් සම්පාදකයින් සම්පාදනය කරන වේලාවේදී බොහෝ දෝෂ හඳුනා ගනී.

- Compilers can optimize code to improve performance and reduce resource usage.
  සම්පාදකයින්ට කාර්ය සාධනය වැඩි දියුණු කිරීමට සහ සම්පත් භාවිතය අඩු කිරීමට කේතය ප්‍රශස්ත කළ හැක.

- Many compiled languages enforce static typing, preventing type-related runtime errors.
  බොහෝ සම්පාදනය කරන ලද භාෂා ස්ථිතික ටයිප් කිරීම බලාත්මක කරයි, වර්ගයට අදාළ ධාවන කාල දෝෂ වළක්වයි.

Disadvantages

- Compiling large projects can take considerable time.
  විශාල ව්‍යාපෘති සම්පාදනය කිරීම සඳහා සැලකිය යුතු කාලයක් ගත විය හැක

- Compilers may not support incremental development as easily as interpreters.
  අර්ථවින්‍යාසයන් තරම් පහසුවෙන් වර්ධක සංවර්ධනය සඳහා සම්පාදකයන් සහාය නොදක්වයි.

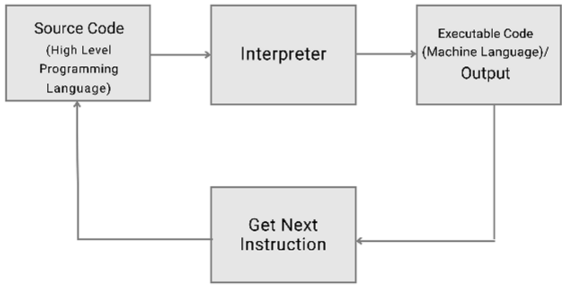
- Compiled programs are often specific to platforms, requiring separate compilations for different operating systems.
  සම්පාදනය කරන ලද වැඩසටහන් බොහෝ විට වේදිකා සඳහා විශේෂිත වේ, විවිධ මෙහෙයුම් පද්ධති සඳහා වෙන වෙනම සම්පාදනය කිරීම අවශ්‍ය වේ.

2. Interpreters

අර්ථවින්‍යාස

An interpreter directly executes the instructions in the source code without compiling it into machine code first.
අර්ථවින්‍යාසය ප්‍රභව කේතයේ ඇති උපදෙස් පළමුව යන්ත්‍ර කේතයට සම්පාදනය නොකර කෙලින්ම ක්‍රියාත්මක කරයි.

It translates high-level code into machine code line-by-line at runtime.
එය ක්‍රියාත්මක වන විට ඉහළ මට්ටමේ කේතය යන්ත්‍ර කේත පේළියෙන් පේළියට පරිවර්තනය කරයි.



Advantages

- Provides immediate feedback, making it ideal for interactive environments and scripting.
  ක්ෂණික ප්‍රතිපෝෂණ සපයයි, අන්තර්ක්‍රියාකාරී පරිසරයන් සහ ස්ක්‍රිප්ටින් සඳහා එය වඩාත් සුදුසු වේ.

- Allows for real-time testing and debugging, pinpointing issues as they arise.
  තත්‍ය කාලීන පරීක්ෂණයට සහ දෝෂහරණයට, ගැටලු මතුවන විට ඒවා හඳුනා ගැනීමට ඉඩ සලසයි.

Disadvantages

- Interpreted code runs slower than compiled code due to runtime translation.
  පරිවර්ථනය කරන ලද කේතය ධාවන කාල පරිවර්තන හේතුවෙන් සම්පාදනය කරන ලද කේතයට වඩා සෙමින් ධාවනය වේ.

- Errors are caught only during execution, potentially leading to unhandled exceptions.
  දෝෂ හසුවන්නේ ක්‍රියාත්මක කිරීමේදී පමණක් වන අතර, එය හසුරුවා නොගත් ව්‍යතිරේකවලට තුඩු දිය හැකිය.

Assemblers

An assembler translates assembly language (a low-level programming language) into machine code.
Assembler එකක් මගින් assembly භාෂාව (පහළ මට්ටමේ ක්‍රමලේඛන භාෂාවක්) යන්ත්‍ර කේතයට පරිවර්තනය කරයි.

Assembly language is closely related to machine code, making this translation relatively straightforward.
එකලස් කිරීමේ භාෂාව යන්ත්‍ර කේතයට සමීපව සම්බන්ධ වන අතර, මෙම පරිවර්තනය සාපේක්ෂව සරල කරයි.

Advantages

- Allows for efficient translation with minimal overhead.
  අවම පොදු කාර්ය සමග කාර්යක්ෂම පරිවර්තනය සඳහා ඉඩ ලබා දේ.

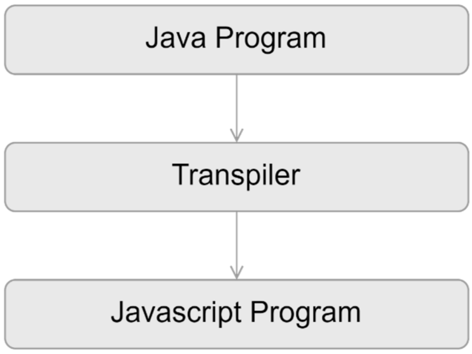
Disadvantages

- Writing in assembly can be tedious and requires detailed hardware knowledge.
  එකලස් කිරීමේදී ලිවීම වෙහෙසකර විය හැකි අතර සවිස්තරාත්මක දෘඪාංග දැනුමක් අවශ්‍ය වේ.

Transpilers
සම්ප්‍රේෂකය
(Source-to-Source Compilers)

A transpiler converts code from one high-level programming language to another high-level programming language.
සම්ප්‍රේෂකයක් එක් ඉහළ මට්ටමේ ක්‍රමලේඛන භාෂාවක සිට තවත් ඉහළ මට්ටමේ ක්‍රමලේඛන භාෂාවකට කේතය පරිවර්තනය කරයි.

It preserves the program's semantics while changing its syntax or structure.
එය එහි වාක්‍ය ඛණ්ඩය හෝ ව්‍යුහය වෙනස් කරන අතරතුර වැඩසටහනේ අර්ථ ශාස්ත්‍රය ආරක්ෂා කරයි.



Advantages

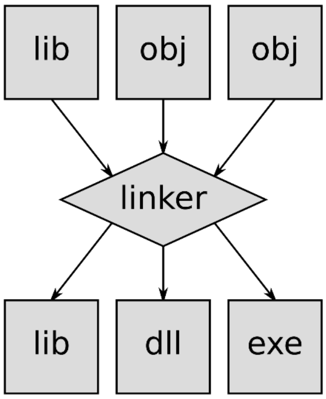
- Enables the use of modern language features while maintaining compatibility with older environments.
  පැරණි පරිසරයන් සමඟ ගැළපීම පවත්වා ගනිමින් නවීන භාෂා විශේෂාංග භාවිතය සබල කරයි.

Disadvantages

- May not perform extensive optimizations, resulting in less efficient generated code.
  වැඩිදුර ප්‍රශස්තකරණයන් සිදු නොකළ හැකිය, එහි ප්‍රතිඵලයක් ලෙස අඩු කාර්යක්ෂමතාවයෙන් උත්පාදනය වූ කේත ඇති වේ.

Linkers
සන්ධාරක

A linker combines multiple object code files and libraries into a single executable program.
සන්ධාරකයක් බහු වස්තු කේත ගොනු සහ libraries එක් ක්‍රියාත්මක කළ හැකි වැඩසටහනකට ඒකාබද්ධ කරයි.



Its primary role is to resolve references between different object files and ensure that all necessary components are correctly linked together to produce a functioning application.
එහි මූලික කාර්යභාරය වන්නේ විවිධ වස්තු ගොනු අතර යොමු නිරාකරණය කිරීම සහ ක්‍රියාකාරී යෙදුමක් නිෂ්පාදනය කිරීම සඳහා අවශ්‍ය සියලුම සංරචක නිවැරදිව එකට සම්බන්ධ කර ඇති බව සහතික කිරීමයි.

Key Functions of a Linker
සම්බන්ධකයේ ප්‍රධාන කාර්යයන්

Symbol Resolution
When one object file refers to a function or variable defined in another object file, the linker ensures that these references point to the correct addresses in the final executable.
එක් වස්තු ගොනුවක් වෙනත් වස්තු ගොනුවක අර්ථ දක්වා ඇති ශ්‍රිතයක් හෝ විචල්‍යයක් ගැන සඳහන් කරන විට, මෙම යොමු කිරීම් අවසාන ක්‍රියාත්මක කළ හැකි නිවැරදි ලිපින වෙත යොමු වන බව සම්බන්ධකය සහතික කරයි.

Address Binding
The linker assigns final memory addresses to the various components of the program.
සම්බන්ධකය විසින් වැඩසටහනේ විවිධ සංරචක සඳහා අවසාන මතක ලිපින පවරයි.

Combining Object Files
The linker takes multiple object files generated by compilers and combines them into a single executable.
සම්බන්ධකය විසින් සම්පාදක මගින් ජනනය කරන ලද බහු වස්තු ගොනු ගෙන ඒවා තනි ක්‍රියාත්මක කළ හැකි බවට ඒකාබද්ධ කරයි.

Library Linking
The linker can include external libraries (static or dynamic) in the executable.
සම්බන්ධකයට ක්‍රියාත්මක කළ කේතය හැකි තුළ බාහිර libraries (ස්ථිතික හෝ ගතික) ඇතුළත් කළ හැක.

Relocation
The linker adjusts the addresses of code and data in the object files so that they fit into the final program structure.
සම්බන්ධකය විසින් වස්තු ගොනු තුළ ඇති කේත සහ දත්තවල ලිපින අවසාන වැඩසටහන් ව්‍යුහයට ගැලපෙන පරිදි සකස් කරයි.

Loaders
ප්‍රවේශක

The Loader is responsible for loading programs and their dependencies into memory so they can be executed.
වැඩසටහන් සහ ඒවායේ පරායත්තතා මතකයට පැටවීම සඳහා ප්‍රවේශකය වගකිව යුතු බැවින් ඒවා ක්‍රියාත්මක කළ හැක.

Key Functions of a Loader
ප්‍රවේශකයක ප්‍රධාන කාර්යයන්

Loading Programs into Memory

The primary role of the loader is to read the executable files produced by a compiler or interpreter and load them into the main memory (RAM) for execution.
ප්‍රවේශකයේ මූලික කාර්යභාරය වන්නේ සම්පාදකයෙකු හෝ පරිවර්තකයෙකු විසින් නිපදවන ක්‍රියාත්මක කළ හැකි ගොනු කියවා ඒවා ක්‍රියාත්මක කිරීම සඳහා ප්‍රධාන මතකයට (RAM) පැටවීමයි.

Relocation

If the program is not loaded at its preferred memory address, the loader adjusts addresses in the program to point to the correct locations in memory.
වැඩසටහන එහි කැමති මතක ලිපිනයේ නොමැති නම්, ප්‍රවේශකය මතකයේ නිවැරදි ස්ථාන වෙත යොමු කිරීම සඳහා වැඩසටහනේ ලිපින සකස් කරයි

Linking

In cases where programs consist of multiple modules or dependencies, the loader is responsible for linking these components together.
වැඩසටහන් බහු මොඩියුල හෝ පරායත්තයන්ගෙන් සමන්විත වන අවස්ථා වලදී, මෙම සංරචක එකට සම්බන්ධ කිරීම සඳහා ප්‍රවේශකය වගකිව යුතුය.

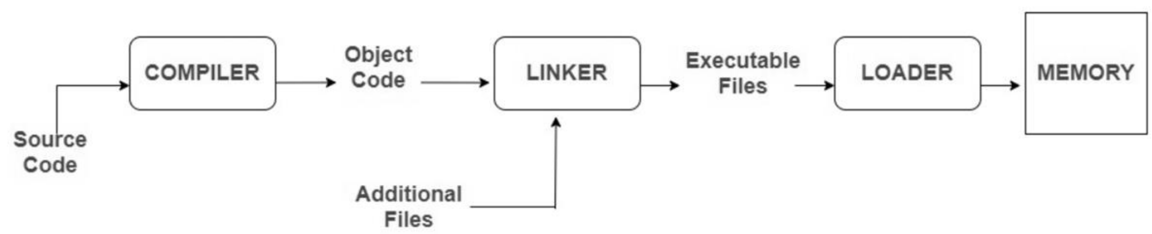
Initialization

After loading the program into memory, the loader initializes the runtime environment.
වැඩසටහන මතකයට පැටවීමෙන් පසු, ප්‍රවේශකය ධාවන කාල පරිසරය ආරම්භ කරයි

Dynamic Loading

In dynamic interpreting environments, loaders can facilitate dynamic loading of modules or libraries.
ගතික පරිවර්ථන පරිසරයන්හිදී, ප්‍රවේශකවල මොඩියුල හෝ libraries වල ලෝඩ් කිරීම පහසු කළ හැක.

This means that certain parts of the program can be loaded and linked at runtime rather than at startup.
මෙයින් අදහස් කරන්නේ වැඩසටහනේ සමහර කොටස් ආරම්භයේදී නොව ධාවන වේලාවේදී පූරණය කර සම්බන්ධ කළ හැකි බවයි.



***IMPORTANT**

Integrated Development Environments
සමෝධානිත සංවර්ධන පරිසර
(IDE)

An IDE is a comprehensive software suite that brings together essential tools used in programming, such as text editors, compilers, and debuggers, into a single interface.
IDE එකක් යනු ක්‍රමලේඛනයේදී භාවිතා වන පෙළ සංස්කාරක, සම්පාදක, සහ නිදොස්කාරක වැනි අත්‍යවශ්‍ය මෙවලම් එක් අතුරු මුහුණතකට ගෙන එන විස්තීර්ණ මෘදුකාංග කට්ටලයකි.

Examples of IDEs
Pycharm, Sublime text, Visual Studio code, Atom, Python IDLE

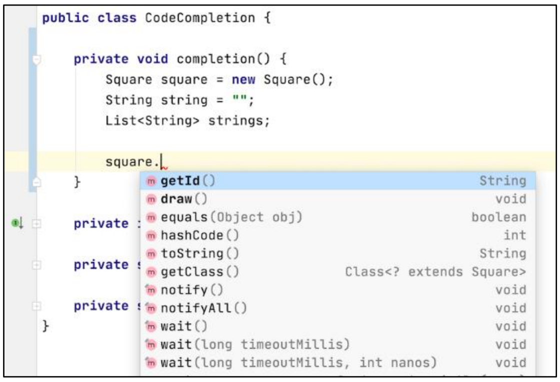
PyCharm | VS code | Sublime text | Atom

Why do we need an IDE?
අපට IDE එකක් අවශ්‍ය වන්නේ ඇයි?

1. Code Writing and Editing

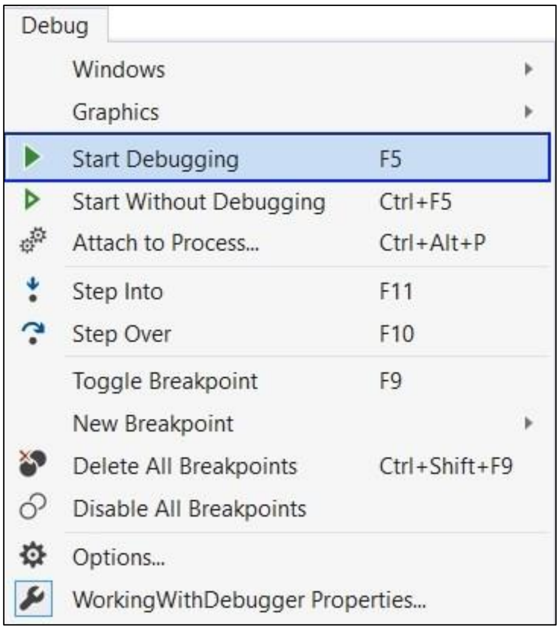
IDEs provide advanced text editors specifically designed for coding.
IDEs විශේෂයෙන් කේතීකරණය සඳහා නිර්මාණය කර ඇති උසස් පෙළ සංස්කාරක සපයයි.

They support features like syntax highlighting, code completion, and automatic formatting, which improve readability and reduce errors during code writing.
ඒවා කේත ඛණ්ඩ උද්දීපනය, කේත සම්පූර්ණ කිරීම සහ ස්වයංක්‍රීය හැඩතල ගැන්වීම වැනි විශේෂාංග සඳහා සහය දක්වයි, එය කියවීමේ හැකියාව වැඩි දියුණු කරන අතර කේත ලිවීමේදී දෝෂ අඩු කරයි.



2. Code Debugging

One of the most critical aspects of software development is debugging.
මෘදුකාංග සංවර්ධනයේ වඩාත්ම තීරණාත්මක අංගයක් වන්නේ නිදොස්කරණයයි.

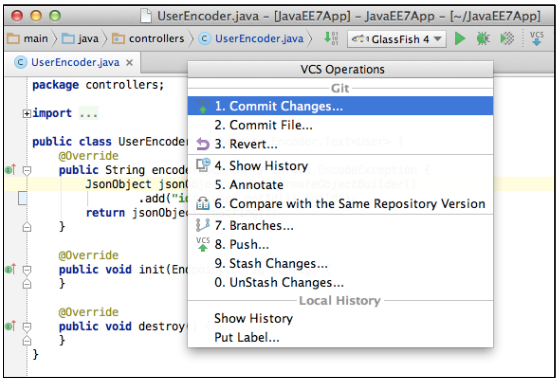


IDEs come with built-in debuggers that allow developers to identify and fix issues in their code efficiently.
IDEs පැමිණෙන්නේ සංවර්ධකයින්ට ඔවුන්ගේ කේතයේ ගැටළු කාර්යක්ෂමව හඳුනා ගැනීමට සහ විසඳීමට ඉඩ සලසන පෙරනිමි නිදොස්කරණයන් සමඟිනි.

3. Project Management

Modern IDEs come with project management features that allow developers to organize code files, libraries, and resources.
සංවර්ධකයින්ට කේත ගොනු, libraries සහ සම්පත් සංවිධානය කිරීමට ඉඩ සලසන ව්‍යාපෘති කළමනාකරණ විශේෂාංග සමඟ නවීන IDE පැමිණේ.

They support version control systems like Git, helping developers collaborate on projects, track changes, and manage different versions of the software.
ඒවා මගින් Git වැනි අනුවාද පාලන පද්ධති සඳහා සහය දක්වයි, සංවර්ධකයින්ට ව්‍යාපෘති සමඟ සහයෝගයෙන් කටයුතු කිරීමට, වෙනස්කම් නිරීක්ෂණය කිරීමට සහ මෘදුකාංගයේ විවිධ අනුවාද ඵලදායී ලෙස කළමනාකරණය කිරීමට උපකාර කරයි.



4. Code Compilation and Execution

IDEs streamline the process of compiling and executing code.
IDEs කේත සම්පාදනය කිරීමේ සහ ක්‍රියාත්මක කිරීමේ ක්‍රියාවලිය විධිමත් කරයි.

They often offer one-click options to build, compile, and run projects, which eliminates the need to use command-line tools manually.
ඒවා බොහෝ විට ව්‍යාපෘති ගොඩනැගීමට, සම්පාදනය කිරීමට සහ ක්‍රියාත්මක කිරීමට විකල්ප ඉදිරිපත් කරයි, එමඟින් විධාන පේළිය හස්තමය වශයෙන් භාවිතා කිරීමේ අවශ්‍යතාවය ඉවත් කරයි

5. Automation

IDEs can automate repetitive tasks like compiling code, running tests, or deploying applications.
IDE වලට කේතය සම්පාදනය කිරීම, පරීක්ෂණ ධාවනය කිරීම හෝ යෙදුම් ස්ථාපනය වැනි පුනරාවර්තන කාර්යයන් ස්වයංක්‍රීය කළ හැක.

6. Cross-Language Support

Many IDEs support multiple programming languages and frameworks, allowing developers to work on different types of projects in one environment.
බොහෝ IDEs බහු ක්‍රමලේඛන භාෂා සහ රාමු සඳහා සහය දක්වයි, සංවර්ධකයින්ට එක් පරිසරයක විවිධ වර්ගයේ ව්‍යාපෘතිවල වැඩ කිරීමට ඉඩ සලසයි.

7. Collaboration

Some modern IDEs are integrated with cloud-based tools that allow real-time collaboration among team members, enabling pair programming or code reviews from remote locations.
සමහර නවීන IDEs, කණ්ඩායම් සාමාජිකයින් අතර තත්‍ය කාලීන සහයෝගීතාවයට ඉඩ සලසන, දුරස්ථ ස්ථාන වල සිට යුගල ක්‍රමලේඛනය හෝ කේත සමාලෝචන සක්‍රීය කරන වලාකුළු මත පදනම් වූ මෙවලම් සමඟ ඒකාබද්ධ කර ඇත.

8. Customization

IDEs are highly customizable, allowing developers to tailor the environment according to their workflow.
IDEs ඉතා අභිරුචිකරණය කළ හැකි අතර, සංවර්ධකයින්ට ඔවුන්ගේ කාර්ය ප්‍රවාහයට අනුව ගැලපෙන පරිසරය සකස් කිරීමට ඉඩ සලසයි.

Python IDLE
(Integrated Development and Learning Environment)

Python's default IDLE (Integrated Development and Learning Environment) is a simple and user-friendly environment that comes bundled with Python, designed primarily for beginners to write, test, and debug Python code.
Python හි පෙරනිමි IDLE එක යනු සරල සහ පරිශීලක-හිතකාමී පරිසරයක් වන අතර එය පයිතන් සමඟ එකතු වී, මූලික වශයෙන් ආරම්භකයින් සඳහා පයිතන් කේත ලිවීමට, පරීක්ෂා කිරීමට සහ නිදොස් කිරීමට නිර්මාණය කර ඇත

Characteristics of IDLE
IDLE එකක ලක්ෂණ

Text Editor
It provides a basic text editor with syntax highlighting, auto-completion, and indentation support.
එය සින්ටැක්ස් උද්දීපනය, ස්වයංක්‍රියව සම්පූර්ණ කිරීම සහ ඉන්ඩෙන්ටේෂන් සහාය සහිත මූලික පෙළ සංස්කාරකයක් සපයයි.

Interactive Shell
The interactive shell allows you to write Python code line by line, execute it immediately, and see the output, making it perfect for testing small code snippets.
අන්තර්ක්‍රියාකාරී shell එක මඟින් ඔබට කුඩා කේත කොටස් පරීක්ෂා කිරීම සඳහා හොඳින් සහාය දක්වමින් Python කේත පේළියෙන් පේළිය ලිවීමට, එය වහාම ක්‍රියාත්මක කිරීමට සහ ප්‍රතිදානය බැලීමට ඉඩ සලසයි.

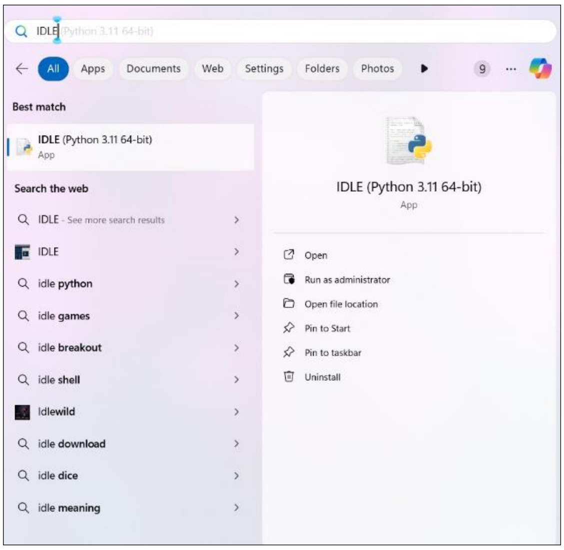
Cross-platform
Since IDLE comes packaged with Python, it works across all major operating systems, including Windows, macOS, and Linux.
IDLE පයිතන් සමඟ සම්බන්ධ කර ඇති බැවින්, එය Windows, macOS& සහ Linux ඇතුළු සියලුම ප්‍රධාන මෙහෙයුම් පද්ධති සමඟ ක්‍රියා කරයි

Debugger
IDLE includes a built-in debugger with features like stepping through code, breakpoints, and variable inspection, though it's not as advanced as other IDEs.
IDLE අනෙකුත් IDE තරම් දියුණු නොවුවත්, කේතය හරහා පියවර තැබීම, breakpoints සහ විචල්‍ය පරීක්ෂාව වැනි විශේෂාංග සහිත පෙරනිමි නිදොස්කාරකයක් ඇතුළත් වේ.

How to use IDLE
IDLE භාවිතා කරන ආකාරය

1. Opening IDLE

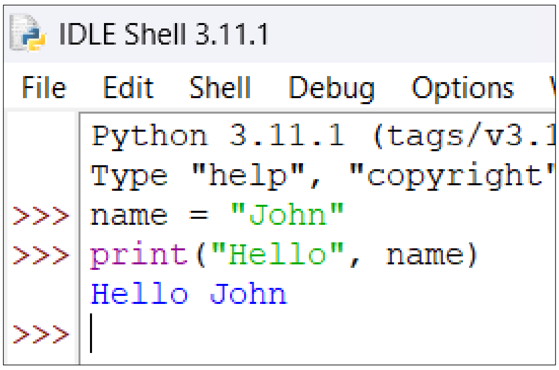
On most systems, IDLE comes pre-installed with Python. Open it by searching for "IDLE" in your system's search bar.
බොහෝ පද්ධතිවල, IDLE එක පයිතන් සමඟ පූර්ව ස්ථාපනය කර ඇත. ඔබගේ පද්ධතියේ සෙවුම් තීරුවේ "IDLE" ලෙස සොයා එය විවෘත කරන්න.



2. Writing Code in the Python Shell

IDLE starts in the Python shell, which is an interactive mode. Here, you can type and run Python commands one by one.
IDLE ආරම්භ වන්නේ පයිතන් shell එකෙන් වන අතර එය අන්තර්ක්‍රියාකාරී මාදිලියකි. මෙහිදී ඔබට පයිතන් විධාන එකින් එක ටයිප් කර ක්‍රියාත්මක කළ හැක.

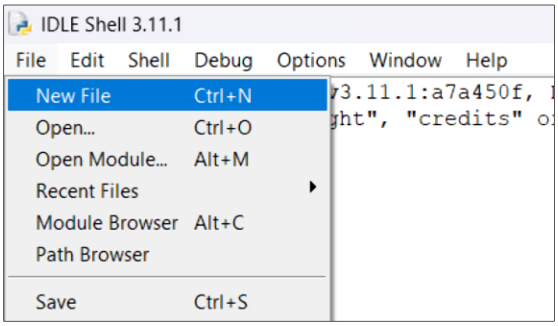
You can write the code and hit Enter to see the output immediately.
ඔබට කේතය ලියා ප්‍රතිදානය එසැනින් බැලීමට Enter එබිය හැක.



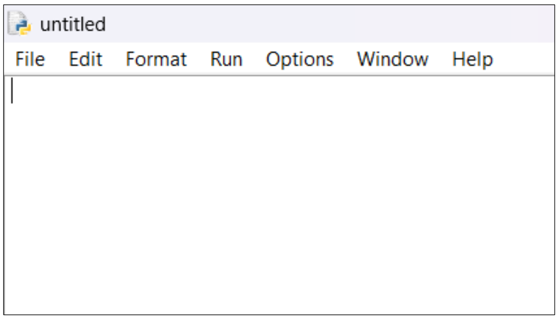
3. Creating a New Script File

To write more complex code, you need to open a script file instead of using the interactive shell.
වඩාත් සංකීර්ණ කේතයක් ලිවීමට, අන්තර්ක්‍රියාකාරී එක භාවිතා කිරීම වෙනුවට කේත ගොනුවක් විවෘත කළ යුතුය.

Go to the File menu and select New File.
ගොනු මෙනුව වෙත ගොස් නව ගොනුව තෝරන්න.

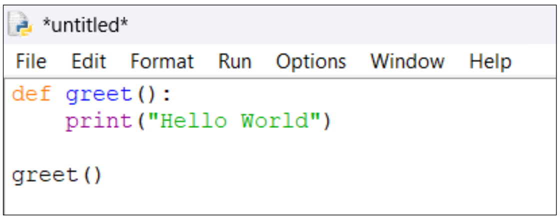


This opens a blank editor window.
මෙය හිස් සංස්කාරක කවුළුවක් විවෘත කරයි.



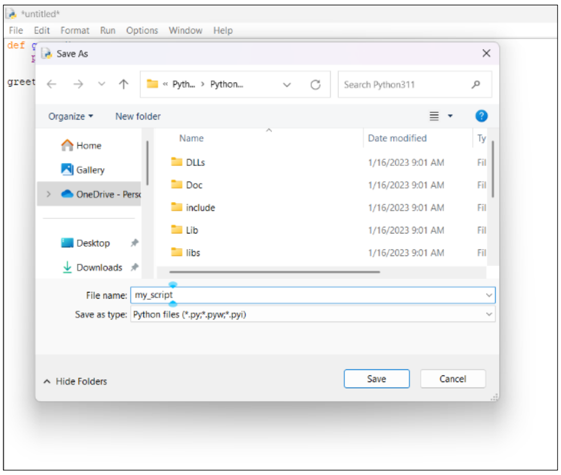
4. Writing Code in the Script File

In the new editor window, you can write your code like you would in any text editor.
නව සංස්කාරක කවුළුව තුළ, ඔබට ඕනෑම පෙළ සංස්කාරකයක මෙන් ඔබේ කේතය ලිවිය හැකිය.



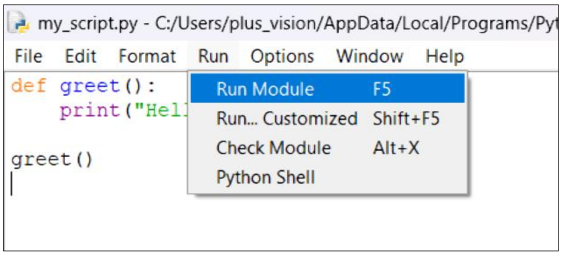
5. Saving Your File

To save your file, go to File > Save or press Ctrl + S (Cmd + S on macOS).
ඔබගේ ගොනුව සුරැකීමට, File > Save වෙත යන්න. හෝ Ctrl + S (macOS හි Cmd + S) ඔබන්න.

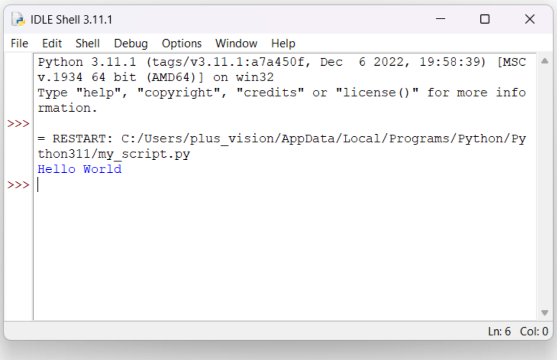


6. Running Your Python Code

Once your code is saved, you can run it by going to Run > Run Module or pressing F5.
ඔබගේ කේතය සුරකින ලද පසු, ඔබට එය Run > Run Module වෙත යාමෙන් හෝ F5එබීමෙන් එය ක්‍රියාත්මක කළ හැක.



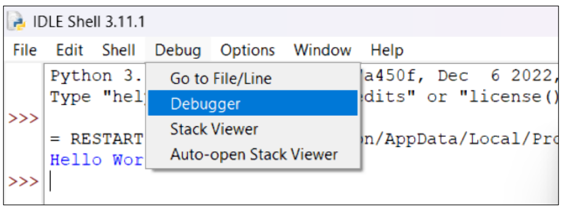
The output will appear in the Python Shell window.
ප්‍රතිදානය පයිතන් Shell කවුළුවෙහි දිස්වනු ඇත.



7. Debugging

IDLE also has a basic debugger to help find errors in your code.
IDLE සතුව ඔබේ කේතයේ දෝෂ සෙවීමට මූලික නිදොස්කරණයක් ද ඇත.

You can access it via Debug > Debugger.
ඔබට Debug > Debugger හරහා එයට ප්‍රවේශ විය හැක.



How to debug
නිදොස්කරණය කරන ආකාරය

Types of Bugs
දෝෂ වර්ග

1. **Syntax Errors:** Mistakes in the code structure or language rules, such as missing parentheses, incorrect indentation, or misspelled keywords.
   කේත ව්‍යුහයේ හෝ භාෂා රීතිවල වැරදි වේ, එනම් වරහන් නොමැති වීම, වැරදි , හෝ වැරදි අක්ෂර වින්‍යාස වචන වැනි.

2. **Runtime Errors:** Errors that occur while the program is running, usually due to illegal operations, like dividing by zero or accessing an unavailable file.
   සාමාන්‍යයෙන් ශුන්‍යයෙන් බෙදීම හෝ නොපවතින ගොනුවකට ප්‍රවේශ වීම වැනි රීති විරෝධී මෙහෙයුම් හේතුවෙන්, වැඩසටහන ක්‍රියාත්මක වන විට සිදුවන දෝෂ.

For further reference

Examples

- **NameError**: Using an undefined variable.
  නිර්වචනය නොකළ විචල්‍යයක් භාවිතා කිරීම.

- **TypeError**: Performing an invalid operation on a data type.
  දත්ත වර්ගයක වලංගු නොවන මෙහෙයුමක් සිදු කිරීම.

- **ValueError**: Incorrect value for a function's argument.
  ශ්‍රිතයක තර්කය සඳහා වැරදි අගයක්

- **IndexError**: Accessing an invalid index of a list or similar object.
  ලැයිස්තුවක හෝ ඒ හා සමාන වස්තුවක වලංගු නොවන දර්ශකයක් වෙත ප්‍රවේශ වීම.

- **KeyError**: Accessing a non-existent key in a dictionary.
Dictionary එකක නොපවතින යතුරක් වෙත ප්‍රවේශ වීම.

- **AttributeError**: Trying to access an undefined attribute of an object.
වස්තුවක නිර්වචනය නොකළ ගුණාංගයකට ප්‍රවේශ වීමට උත්සාහ කිරීම.

- **ZeroDivisionError**: Dividing by zero.
බිංදුවෙන් බෙදීම.

- **FileNotFoundError**: Trying to open a file that doesn't exist.
නොපවතින ගොනුවක් විවෘත කිරීමට උත්සාහ කිරීම.

- **ModuleNotFoundError**: Importing a module that isn't available.
නොමැති මොඩියුලයක් ආයාත කිරීම.

- **EOFError**: Input ends unexpectedly, often with the `input()` function.
ආදානය අනපේක්ෂිත ලෙස අවසන් වේ, බොහෝ විට `input()` ශ්‍රිතය සමඟ.

- **MemoryError**: Running out of memory for operations.
මෙහෙයුම් සඳහා මතකය අවසන් වීම.

- **RecursionError**: Exceeding the maximum recursion depth.
උපරිම පුනරාවර්තන ගැඹුර ඉක්මවා යාම.

- **OverflowError**: Numerical result exceeds the limits of the data type.
සංඛ්‍යාත්මක ප්‍රතිඵලය දත්ත වර්ගයේ සීමාවන් ඉක්මවයි.

- **UnicodeEncodeError**
UnicodeDecodeError:
Issues with encoding or decoding text.
පෙළ කේතනය කිරීම හෝ විකේතනය කිරීම සම්බන්ධ ගැටළු.

- **KeyboardInterrupt**: Manually interrupted by the user, typically with Ctrl+C.
සාමාන්‍යයෙන් Ctrl+C සමඟින්, පරිශීලකයා විසින් අතින් බාධා කරනු ලැබේ

- **SystemExit**: Triggered when calling sys.exit() to exit the program.
වැඩසටහනෙන් පිටවීමට sys.exit() ඇමතීමේදී ක්‍රියාරම්භ කරන ලදී.

- **TimeoutError**: Operation times out, common in networking.
මෙහෙයුම් කාලය අවසන්, ජාලකරණයේදී පොදු වේ.

- **BrokenPipeError**: Writing to a closed connection.
සංවෘත සම්බන්ධතාවයකට ලිවීම.

- **StopIteration**: Iterator has no more items.
පුනරාවර්තකයට තවත් අයිතම නොමැත.

3. **Logical Errors:** Errors that occur when the program doesn't perform the intended task. These are often the most difficult to detect since the code runs without crashing, but it produces the wrong result.
වැඩසටහන අපේක්ෂිත කාර්යය ඉටු නොකරන විට සිදුවන දෝෂ වේ. කේතය බිඳවැටීමකින් තොරව ක්‍රියාත්මක වන බැවින් මේවා බොහෝ විට හඳුනා ගැනීමට අපහසු වේ, නමුත් එය වැරදි ප්‍රතිඵලයක් ලබා දෙයි.

For further reference

Examples:

- **Incorrect calculations**: Using the wrong formula or arithmetic.
වැරදි සූත්‍රය හෝ අංක ගණිතය භාවිතා කිරීම.

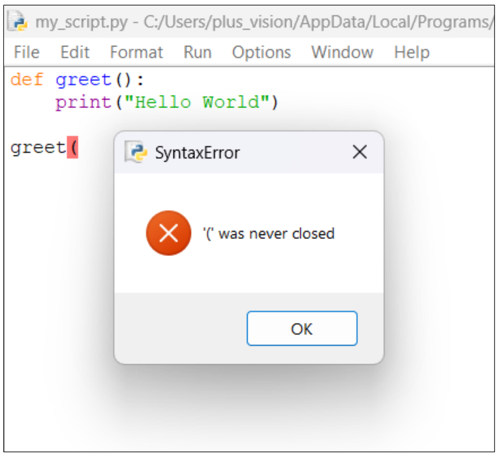
- **Off-by-one errors**: Misconfiguring a loop range, resulting in extra or missing iterations.
ලූප පරාසයක් වැරදි ලෙස වින්‍යාස කිරීම, අමතර හෝ නැති වූ පුනරාවර්තන ඇති කරයි.

- **Infinite loops**: Creating loops that don't terminate due to incorrect conditions.
වැරදි කොන්දේසි හේතුවෙන් අවසන් නොවන ලූප නිර්මාණය කිරීම.

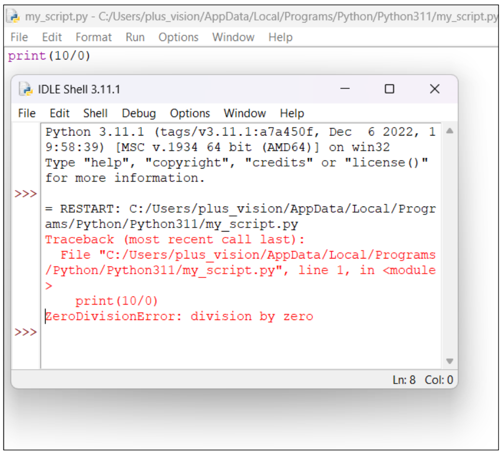
- **Misplaced conditional checks**: Checking conditions in the wrong order or context, which can lead to unexpected behavior.
අනපේක්ෂිත හැසිරීම් වලට තුඩු දිය හැකි වැරදි පිළිවෙලට හෝ සන්දර්භය තුළ කොන්දේසි පරීක්ෂා කිරීම.

- **Incorrectly ordered operations**: Mixing up operations so that calculations are performed in the wrong order.
ගණනය කිරීම් වැරදි අනුපිළිවෙලකට සිදු වන පරිදි මෙහෙයුම් මිශ්‍ර කිරීම.

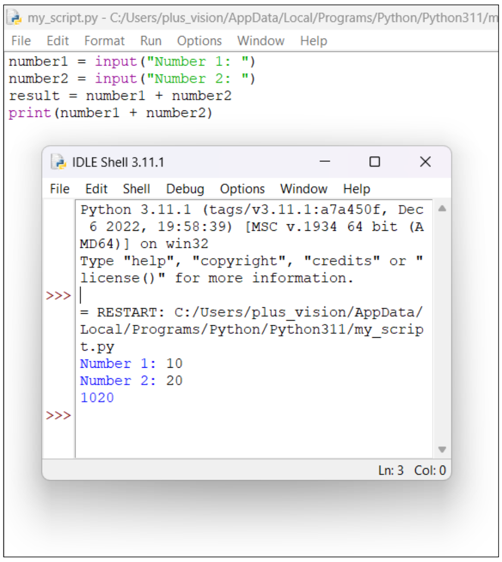
SyntaxError



RuntimeError



LogicalError



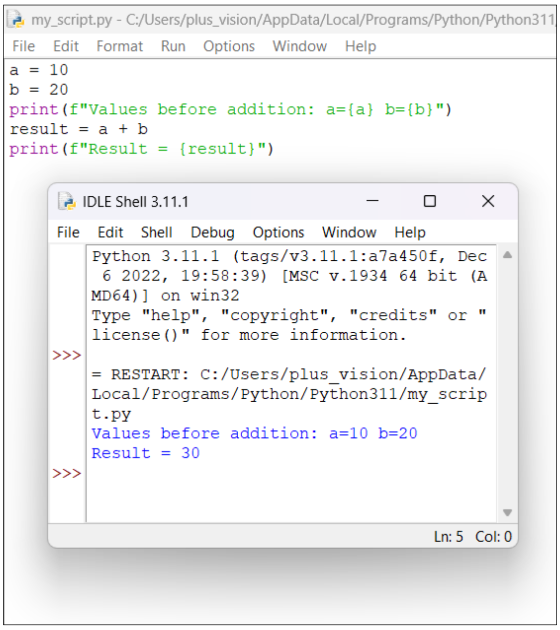
Debugging Techniques
දෝෂ නිරාකරණ ශිල්පීය ක්‍රම

1. Manual Code Inspection

Carefully reviewing the code for mistakes, logic errors, or incorrect assumptions can often reveal the source of the problem.
වැරදි, තාර්කික දෝෂ හෝ වැරදි උපකල්පන සඳහා කේතය ප්‍රවේශමෙන් සමාලෝචනය කිරීම බොහෝ විට ගැටලුවේ මූලාශ්‍රය හෙළිදරව් කළ හැකිය.

2. Print Statements

One of the simplest ways to debug is by inserting `print` statements in various parts of the code to track the program's flow and the values of variables at different stages.
දෝෂ නිරාකරණය කිරීමට ඇති සරලම ක්‍රමයක් වන්නේ වැඩසටහනේ ප්‍රවාහය සහ විවිධ අවධීන්හි විචල්‍යවල අගයන් නිරීක්ෂණය කිරීම සඳහා කේතයේ විවිධ කොටස්වල `print` ප්‍රකාශ ඇතුළත් කිරීමයි.

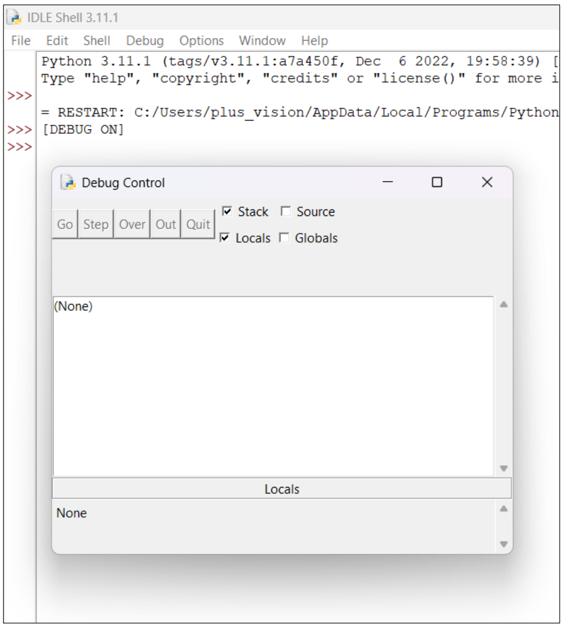


3. Using Debuggers

- **Breakpoints**
  You can set a breakpoint in your code to pause execution at a specific line. Once the program pauses, you can inspect the variables, step through the code line by line, and observe how the values change.
  නිශ්චිත පේළියක ක්‍රියාත්මක කිරීම විරාම කිරීමට ඔබට කේතයේ breakpoint එකක් සැකසිය හැක. වැඩසටහන නතර වූ පසු, ඔබට විචල්‍යයන් පරීක්ෂා කර, කේත පේළියෙන් පේළිය හරහා ගොස් අගයන් වෙනස් වන ආකාරය නිරීක්ෂණය කළ හැකිය.

In IDLE, you can turn on the debugger by navigating to Debug > Debugger in the menu.
IDLE හි, මෙනුවේ Debug > Debugger වෙත සංචාලනය කිරීමෙන් නිදොස්කාරකය සක්‍රිය කළ හැක.

The debugger will display buttons for actions like Step, Go, Over, and Out.
නිදොස්කාරකය Step, Go, Over සහ Out වැනි ක්‍රියා සඳහා බොත්තම් පෙන්වයි.



`print()` function

Used to output something to the standard output device (the screen)
පරිගණකයේ සම්මත ප්‍රතිදාන උපාංගයට (තිරය මතට) යමක් පෙන්වීම සඳහා භාවිතා කරයි

Syntax

```
print(*objects, sep=' ', end='\n'
file=None flush=False)
```

Arguments

- **`sep`** `(default: " ")`:

String to separate objects.
වස්තු වෙන් කිරීමේ String එක අර්ථ දක්වයි.

```
txt1 = "Hello"
txt2 = "World"

print(txt1, txt2, sep = '-')
Hello-World
```

- **`end`** `(default: "\n")`:

String appended after the last object.
අවසාන වස්තුවට පසුව පෙන් විය යුතු String එක අර්ථ දක්වයි.

```
txt1 = "Hello"
txt2 = "World"

print(txt1, end = '$')
print(txt2)
Hello$World
```

- **`file`** (`default: sys.stdout`):

Specifies where print sends output, such as the screen, a file that can handle text.
තිරය, ගොනුවක් වැනි මුද්‍රණය මඟින් ප්‍රතිදානය යවන ස්ථානය සඳහන් කරයි.

By default, print sends output to `sys.stdout`, which is usually the screen or terminal window where the program is running.
පෙරනිමියෙන්, මුද්‍රණය මඟින් ප්‍රතිදානය `sys.stdout` වෙත යවයි, එය සාමාන්‍යයෙන් වැඩසටහන ක්‍රියාත්මක වන තිරය හෝ පර්යන්ත කවුළුව වේ.

- **`flush`** `(default: False)`:

Controls how quickly output appears.
ප්‍රතිදානය දිස්වන වේගය පාලනය කරයි.

If False: Output may be delayed to improve performance.
False නම් කාර්ය සාධනය වැඩි දියුණු කිරීම සඳහා ප්‍රතිදානය ප්‍රමාද විය හැක.

If True: Output appears instantly, which is useful for real-time updates.
True නම් ප්‍රතිදානය ක්ෂණිකව දිස්වේ, එය තත්‍ය කාලීන යාවත්කාලීන සඳහා ප්‍රයෝජනවත් වේ.

Examples.

```
print("Hello") # strings
Hello

print(100) # values
100

x = 11 # variables
print(x)
11
```

What is a string
String එකක් යනු කුමක්ද

A string is made up of zero or more unicode characters.
string එකක් සෑදී ඇත්තේ යුනිකේත සංකේත කිහිපයකිනි

They are enclosed in either single quotation marks, double quotation marks, or triple quotation marks
ඒවා තනි උද්‍ධෘත පාඨ, ද්වී උද්‍ධෘත පාඨ හෝ උද්‍ධෘත පාඨ ත්‍රිත්ව තුල ඇතුලත් කර පවතී

Single line strings
තනි පේලියේ strings

```
1. "Helloworld"
2. 'Helloworld'
```

Multiline Strings are written using triple quotation marks.
පේලි කිහිපයක strings උද්‍ධෘත පාඨ ත්‍රිත්ව තුල ඇතුලත් කර ලියයි

1.
```
"""This is a
   mutliline
   string"""
```

2.
```
'''Helloworld
   This is a
   multiline string'''
```

Strings will later be discussed further.
තන්තු පිලිබද පසුව තවදුරටත් සාකච්ඡා කරනු ඇත.

We can insert a string into the `print()` function to display itself on the screen.
`print()` ශ්‍රිතයට string එකක් ඇතුලත් කිරීම මගින් එය තිරය මත පෙන්විය හැකිය

Printing strings using `print()`

```
print("Helloworld")
Helloworld
print('Helloworld')
Helloworld
```

Double quoted text is used to display single quotes in a string.
තනි උද්ධෘත පාඨ ප්‍රතිදානය සඳහා ද්වී උද්ධෘත පාඨ භාවිතා කල හැක

```
print ("Ravindu's pen")
Ravindu's pen
```

Single quoted text is used to display double quotes in a string.
ද්වී උද්ධෘත පාඨ ප්‍රතිදානය සඳහා තනි උද්ධෘත පාඨ භාවිතා කල හැක

```
print('Ravindu says "hello"')
Ravindu says "hello"
```

Triple single or double quotes are used to display multi line strings
පේලි කිහිපයක strings පෙන්වීමට තනි හෝ ද්වී උද්ධෘත පාඨ තුන බැගින් භාවිතා වේ

```
print (''' Hello
World ''')
Hello
print (""" Hello
World""")
Hello
World
```

Strings containing both single and double quotes can be displayed by quoting them with triple single or triple double quotes.
තනි සහ ද්වී උද්ධෘත පාඨ දෙකම අඩංගු strings ප්‍රතිදානය සඳහා තනි හෝ ද්වී උද්ධෘත පාඨ තුන බැගින් භාවිතා වේ

```
print("""There's a "PEN" """)
There's a "PEN"
print('''There's a "PEN" ''')
There's a "PEN"
```

Escape sequences

These are a combination of specific characters used with backslash.
පසුපස ඇල ඉර සමග එකට යෙදෙන නිශ්චිත සංකේත සංයෝජන සමූහයකි

They are used to insert special characters into a string.
String එකක් තුල ට විශේෂ සංකේත ඇතුලත් කිරීම සඳහා භාවිතා කරයි.

The backslash \ character alerts the computer to exclude the letter immediately next to it as it is normally considered.
තමාට පසුව පවතින අනුලක්ශණය සාමාන්‍ය භාවිතයේදී භාවිතා කරන ආකාරයට නොසලකන ලෙස පරිගණකය දැනුවත් කිරීම \ මගින් සිදුවේ

1. `\n` newline
2. `\t` tab
3. `\'` single quote
4. `\''` double quote
5. `\\` backslash
6. `\t` tab
7. `\b` backspace
8. `\f` form feed
9. `\000` octal value
10. `\xhh` hex value

\t is used to keep horizontal tab spaces
තිරස් tab spaces යෙදීමට \t භාවිතා කරයි

```
print ('hello\tworld')
```
```
hello	world
```

\n is used to insert a new line
නව පේලියක් යෙදීම සඳහා \n භාවිතා කරයි

```
print ('hello\nworld')
```
```
hello
world
```

Double backslashes can be used to print a backslash to the output
ප්‍රතිදානය තුල \ මුද්‍රණය කිරීමට \\ භාවිතා කරයි

```
print('hello\\world')
```
```
hello\world
```

Also used to print single or double quotation marks on the output.
තිරය මත තනි හෝ ද්වි උද්ධෘත පාඨ මුද්‍රණය කිරීමටද භාවිතා කරයි

```
print ('Ravindu\'s pen')
```
```
Ravindu's pen
```
```
print ("People said \"hello\"")
```
```
People said "hello"
```

\\ is used to insert a literal backslash (\) in a string
String එකක් තුළ backslash ඇතුළු කිරීමට භාවිතා කරයි

```
print('c:\\windows')
```
```
C:\windows
```

\f represents a form feed, which traditionally moves to a new page in printed documents but usually acts as whitespace or has no visible effect in modern environments.
\f පෝරම සංග්‍රහයක් (form feed) නියෝජනය කරයි, එය සම්ප්‍රදායිකව මුද්‍රිත ලේඛනවල නව පිටුවකට මාරු වන නමුත් සාමාන්‍යයෙන් හිස්තැනක් ලෙස ක්‍රියා කරයි හෝ නවීන පරිසරයන්හි දෘශ්‍යමාන බලපෑමක් නැත.

\000 is used to represent characters by their octal values
\000 අක්ෂර අෂ්ටක අගයන් මගින් නිරූපණය කිරීමට භාවිතා කරයි

```
print ('\101')
```
```
A
```

\xhh escape sequence is used to insert characters by their hexadecimal values.
\xhh භාවිතා කරනුයේ අක්ෂර ඒවායේ ෂඩාස්‍ර අගයන් ඇතුළත් කිරීමටය.

```
print ('\x41')
```
```
A
```

Raw strings

A raw string in Python treats backslashes as literal characters.
Python හි raw string එකක් මගින් backslashes වලට වචනාර්ථ අනුලක්ෂණ ලෙස සලකයි.
Python still interprets **escaped quotes** (like \" or \')

```
print (r'hello\tworld')
hello\tworld
```

```
print (R'hello\nworld')
hello\nworld
```

```
print(R'ravindu\tharindu')
ravindu\tharindu
```

Useful for file paths, as paths with backslashes can be specified without escaping.
ගොනු මාර්ග සඳහා ප්‍රයෝජනවත් වේ, මන්ද බැක්ස්ලෑෂ් සහිත මාර්ග Escape වීමකින් තොරව නියම කළ හැක.

```
print(r'C:\newfolder\newfile.txt')
C:\newfolder\newfile.txt
```

Python still interprets **escaped quotes** (like \" or \')
පයිතන් තවමත් **escaped quotes** අර්ථකථනය කරයි (\" , \' වැනි)

```
print(r"This is a raw string with
an escaped quote: \"")
This is a raw string with an escaped
quote: "
```

A raw string can't end with an unescaped backslash because it would escape the closing quote, causing a syntax error.
raw string එකක් ගැලවී නොගිය unescaped backslash එකකින් අවසන් කළ නොහැක, මන්ද එය වාක්‍ය ඛණ්ඩ දෝෂයක් ඇති කරමින් වසා දැමීමේ උද්ධෘතයෙන් escape වනු ඇත.

If you need a backslash at the end of a raw string, you can either:
ඔබට raw string එකක් අවසානයේ backslashes අවශ්‍ය නම්, :

- Add a double backslash \\.
  ද්විත්ව backslashes එක් කරන්න

```
print(r"Ends with a backslash\\")
Ends with a backslash\
```

Below example will generate a syntax error
පහත කේතය මගින් දෝෂයක් ලබා දෙනු ඇත

```
print(r"Ends with a backslash\\")
#syntax error
```

- Use string concatenation.
  string සංයුක්ත කිරීම භාවිතා කරන්න

```
print(r"Ends with a
backslash"+"//")
Ends with a backslash\
```

Comments in python
Python භාෂාවේ විවරණ

Code lines that the computer should neglect are known as comments.
පරිගණකය නොසලකා හැරිය යුතු කේත පේළි විවරණ ලෙස හැඳින්වේ.

Uses of comments
විවරණ වල භාවිත

- To improve self-readability of the code
  කේතයක ස්වයං-කියවීමේ හැකියාව වර්ධනය කිරීම සඳහා
- For the easy reference of third parties
  තුන්වන පාර්ශවයක පහසු අධ්‍යනය සඳහා
- Fo temporarily neglect code sectors
  කේත කොටස් තාවකාලිකව නොසලකාහැරීම සඳහා

Comments are started using a hashtags # in python language.
Python භාෂාවේදී hashtag # භාවිතා කරමින් විවරණයක් ලියනු ලබයි

```
# this is a comment
```

Python does not have a way of indicating multi line comments.
Python භාෂාවේදී පේළි කිහිපයක විවරණ යෙදීමේ හැකියාවක් නොපවතී

We must put a hashtag at the beginning of each line to put multi line comments.
එමනිසා විවරණයේ සෑම පේළියක්ම ආරම්භයේදී නැවත hashtag යෙදිය යුතුය

Since python ignores string literals that are not assigned to a variable, we can also add a multi-line string (using triple quotes) to place a comment inside of it
කේතය තුල පවතින නමුත් විචල්‍යකට ආදේශ කර නොමැති string වර්ගයේ දත්ත python භාෂාවෙන් නොසලකාහරින නිසා උද්ධෘත පාඨ 3ක් සහිත string එකක් ලෙස පේළි කිහිපයක විවරණයක් කේතය තුල භාවිතා කිරීමටද හැකියාව පවතී

```
'''This
   is a multiline
   comment'''
```

```
"""This
   is a multiline
   comment"""
```

Best Practices

- Keep comments brief and focused on the code's purpose.
  අදහස් කෙටියෙන් තබාගෙන කේතයේ අරමුණ කෙරෙහි අවධානය යොමු කරන්න.
- Don't restate obvious functionality.
  පැහැදිලි ක්‍රියාකාරීත්වය නැවත සඳහන් නොකරන්න.
- Update comments as code evolves.
  කේතය පරිණාමය වන විට comments යාවත්කාලීන කරන්න.
- Use docstrings for functions and classes.
  කාර්යයන් (functions) සහ පන්ති (classes) සඳහා docstrings භාවිතා කරන්න.

NOTE: Docstrings
String literals that describe the purpose of a function, method, class, or module.
ශ්‍රිතයක, ක්‍රමයක, පන්තියක හෝ මොඩියුලයක අරමුණ විස්තර කරන Docstrings වචනාර්ථය.

Placed right after the definition, enclosed in triple quotes (""").
නිර්වචනයට පසුව ස්ථානගත කර ඇත, ත්‍රිත්ව උද්ධෘතවල (""") අමුණා ඇත.

```
def add(x, y):
    """
    Adds two numbers.
    Args:
        x: The first number.
        y: The second number.
    Returns:
        The sum of x and y.
    """
    return x + y
```

`help()` function

The `help()` function is used to display documentation about objects, modules, functions, classes, and keywords.
වස්තු, මොඩියුල, ශ්‍රිත, පන්ති සහ මූල පද පිළිබඳ ලේඛන ප්‍රදර්ශනය කිරීමට ශ්‍රිතය භාවිතා කරයි.

```
help(print)
```

This will display the documentation for the `print()` function, explaining how it works, its parameters, and examples of usage.
මෙමගින් `print()` ශ්‍රිතය ක්‍රියා කරන ආකාරය, එහි පරාමිති සහ භාවිතයේ උදාහරණ පැහැදිලි කරමින් එය සඳහා ලේඛනයක් පෙන්වනු ඇත.

Output:

```
Help on built-in function print in module builtins:

print(*args, sep=' ', end='\n', file=None, flush=False)
    Prints the values to a stream, or to sys.stdout by default.

    sep
      string inserted between values, default a space.
    end
      string appended after the last value, default a newline.
    file
      a file-like object (stream); defaults to the current sys.stdout.
    flush
      whether to forcibly flush the stream.
```

It provides useful information such as method descriptions, usage details, and parameter requirements, helping users understand how to use them effectively.
එය method විස්තර, භාවිත විස්තර, සහ පරාමිති අවශ්‍යතා වැනි ප්‍රයෝජනවත් තොරතුරු සපයන අතර, ඒවා ඵලදායී ලෙස භාවිතා කරන්නේ කෙසේද යන්න තේරුම් ගැනීමට පරිශීලකයින්ට උපකාර කරයි.

- **`*args`:** What to print.
  මුද්‍රණය කළ යුතු දේ.

- **`sep`:** Separator between multiple items.
  බහු අයිතම අතර බෙදුම්කරු.
  (default: space)

- **end**: What to add at the end.
  අවසානයේ එකතු කළ යුතු දේ (default: newline)

- **`file`**: Output destination.
  ප්‍රතිදාන ගමනාන්තය
  (default: console)
  .

- **`flush`**: Force immediate display
  ක්ෂණික දර්ශනය ක්‍රියාත්මක කිරීම
  (default: False).

OPERATORS
මෙහෙයවන

A symbol which is used to represent a operation which should be performed on two data is an operator
දත්ත දෙකක් මත සිදුකල යුතු කර්මයක් නිරූපණය කිරීම සඳහා භාවිතා කරන සංකේතයක් මෙහෙයවනයක් ලෙස හැඳින්වේ

Types of operators in Python
Python වල මෙහෙයවන වර්ග

Operator types represent the type of calculations which should be done on data.
දත්ත මත සිදුකල යුතු ගණනයකිරීම් වර්ගය මේවා මගින් නිරූපණය කරයි

1. Arithmetic operators
අංක ගණිතමය මෙහෙයවන

Arithmetic operators in Python perform basic mathematical operations, such as addition, subtraction, multiplication, division, and exponentiation, on numerical data types.
පයිතන් හි අංක ගණිතමය මෙහෙයවන නිඛිලමය දත්ත වර්ග මත එකතු කිරීම, අඩු කිරීම, ගුණ කිරීම, බෙදීම සහ බලය ලබා ගැනීම වැනි මූලික ගණිතමය මෙහෙයුම් සිදු කරයි.

2. Logical operators
තාර්කික මෙහෙයවන

Evaluate boolean expressions, using `and`, `or`, and `not` to combine or invert conditions.
කොන්දේසි ඒකාබද්ධ කිරීම හෝ ප්‍රතිලෝම කිරීම සඳහා සහ භාවිතයෙන් බූලියානු ප්‍රකාශන ඇගයීමට ලක් කරයි.

3. Comparison operators
සංසන්දනාත්මක මෙහෙයවන

Compare values and return `True` or `False` based on relations like equality or inequality.
සමානාත්මතාවය හෝ අසමානතාවය වැනි සම්බන්ධතා මත පදනම්ව අගයන් සංසන්දනය කර සත්‍ය හෝ අසත්‍යය ලබා ප්‍රත්‍යාගමනය කරයි.

4. Bitwise operators
බිටු අනුසාරිත මෙහෙයවන

Operate on binary representations of integers, manipulating individual bits.
තනි තනි බිටු හසුරවමින්, නිඛිලවල ද්විමය නිරූපණයන් මත ක්‍රියා කරයි.

5. Membership operators
සාමාජික මෙහෙයවන

Check if an element exists within a sequence (`in`, `not in`).
අනුක්‍රමයක් තුළ අවයව තිබේ දැයි පරීක්ෂා කරයි (`in`, `not in`).

6. Identity operators
හඳුනාගැනීමේ මෙහෙයවන

Determine if two variables refer to the same object in memory (`is`, `is not`).
විචල්‍ය දෙකක් මතකයේ ඇති එකම වස්තුවකට යොමු වන්නේද යන්න තීරණය කරයි (`is`, `is not`).

7. Assignment operators
ආදේශ කිරීමේ මෙහෙයවන

Assign values to variables, optionally performing an operation (e.g., +=, -=).
විචල්‍යයන් වෙත අගයන් පවරයි, විකල්පයක් ලෙස මෙහෙයුම්ද සිදු කළ හැක.(උදා: +=, -=).

ictfromabc.com

Arithmetic operators
අංක ගණිතමය මෙහෙයවන

Addition
එකතුකිරීම

Adds the second data with the first.
දෙවන දත්තය පළමු දත්තය සමග එකතු කිරීමට භාවිතා කරයි

```
5 + 2
7
```

Subtraction
අඩුකිරීම

Subtracts second data from the first.
දෙවන දත්තය පළමු දත්තයෙන් අඩු කිරීමට භාවිතා කරයි

```
5 - 2
3
```

Multiplication
ගුණකිරීම

Multiplies the two data.
දත්ත දෙක ගුණ කරයි

```
5 * 2
10
```

Normal Division
සාමාන්‍ය බෙදීම

Divides the first data from the second and displays the exact answer with decimals.
පළමු දත්තය දෙවන දත්තයෙන් බෙදා දශමස්ථාන සහිතව පිළිතුර ලබාදේ

```
5 / 2
2.5
```

Floor division / Integer division
නිඛිල බෙදීම

Divides the first data from the second and displays the answer without decimals.
පළමු දත්තය දෙවන දත්තයෙන් බෙදා දශමස්ථාන රහිතව පිළිතුර ලබාදේ

Used to know the exact number of times that one data is inside of another.
යම් දත්තයක් තුල තවත් දත්තයක් හරියටම කී වතාවක් පවතින්නේදැයි සොයාගැනීමට භාවිතා කල හැක

```
5 // 2
2
```

Modulus division / Remainder division
බෙදීමක ඉතිරි ලබා ගැනීම

Divides the first data exactly from the second and displays the amount remaining.
පළමු දත්තය දෙවන දත්තයෙන් හරියටම බෙදූ විට ඉතිරිවන අගය ලබාදේ

```
5 % 2
1
```

Exponentiation
බලය ලබා ගැනීම

Gives the first data raised to the power of the second data
පළමු දත්තය දෙවන දත්තයේ බලයට නංවා ලබාදේ

```
5 ** 2
25
```

NOTE 1

Result of a calculation which includes several types of data will be displayed in the data type which occupies the highest amount of memory size in a computer.
ගණනය කිරීමක් සඳහා විවිද වර්ග වල දත්ත භාවිතා කිරීමේදී ලැබෙන අවසාන පිලිතුර ඒ අතරින් පරිගණකය තුල වැඩිම ඉඩකඩ අත්පත් කරගන්නා දත්ත වර්ගයෙන් ලැබේ

```
5 + 2.0
7.0
```

```
5 * True
6
```

```
5.0 - True
4.0
```

NOTE 2

Normal division always produces a floating-point number as the result.
සාමාන්‍ය බෙදීම මගින් සෑමවිටම ඉපිලෙන ලක්ෂ්‍ය සංඛ්‍යා ආකාරයෙන් පිළිතුරු ලබාදේ

```
6 / 2
3.0
```

```
5 / True
5.0
```

```
True / True
1.0
```

Arithmetic operator precedence
ගණිතමය මෙහෙයවන ප්‍රමුඛතා අනුපිලිවෙල

The order in which operators are evaluated in an expression which contains more than one operator is the operator precedence
මෙහෙයවන එකකට වඩා වැඩි ගණනක් සහිත ප්‍රකාශණයක ඒවා ඇගයීමට ලක් කල යුතු පිලිවෙල මෙහෙයවන ප්‍රමුඛතා අනුපිලිවෙල ය

1. Parentheses ( )
2. Exponentiation **
3. Multiplication *
   Division /
   Integer division //
   Modulus division %
4. Addition +
   Subtraction -

When operators of the same type or the same precedence are available in an expression, they should be executed from left to right in the existing order.
එකම වර්ගයේ හෝ එකම ප්‍රමුඛතාවයේ මෙහෙයවන ගණිත කර්මයක් තුල පවතින විට ඒවා වමේ සිට දකුණට පවතින ආකාරයෙන් ප්‍රමුඛතාවය ලබා දී ක්‍රියාත්මක කල යුතුය

Parentheses are used to change the normal operator precedence.
සාමාන්‍ය මෙහෙයවන ප්‍රමුඛතා අනුපිලිවෙල වෙනස් කිරීමට වරහන් භාවිතා කරයි

therefore, it has the maximum priority
එමනිසා එයට වැඩිම ප්‍රමුඛතාවය පවතී

NOTE 3

Exponentiation in python is carried out from right to left of the expression.
Python භාශාවේදී බලය ලබා ගැනීමේ ප්‍රකාශණ දකුනේ සිට වමට ක්‍රියාත්මක වේ

```
2 ** 3 ** 2
512
```

In general math, the answer is 64.
සාමාන්‍ය ගණිතයේදී මෙහි පිළිතුර 64 කි

ictfromabc.com

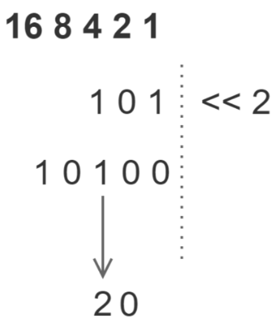
Bitwise Operators
බිටු අනුසාරිත මෙහෙයුම්

These operators are used to perform bitwise calculations on integers after converting them into binary.
නිඛිල සංඛ්‍යා ද්වීමය සංඛ්‍යා බවට පරිවර්තනය කිරීමෙන් පසු බිටු අනුසාරිත මෙහෙයුම් සිදුකිරීමට මෙම මෙහෙයවන භාවිතා කරයි

Left shift

The left shift operator shifts the bits of a number to the left by a specified number of positions.
Left shift මෙහෙයවනය විසින් සංඛ්‍යාවක බිටු නියමිත ස්ථාන ගණනකින් වමට තල්ලු කරයි.

```
5 << 2
```
```
20
```



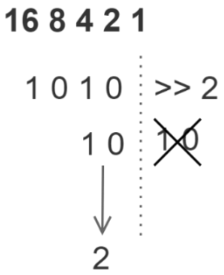
We can use the following formula to easily do the calculation.
පහසුවෙන් ගණනය කිරීම සිදු කිරීම සඳහා අපට පහත සූත්‍රය භාවිතා කළ හැකිය.

$n << x = n * 2^{x}$

Right shift

The right shift operator shifts the bits of a number to the right by a specified number of positions.
Right shift මෙහෙයවනය විසින් සංඛ්‍යාවක බිටු නිශ්චිත ස්ථාන ගණනකින් දකුණට තල්ලු කරයි.

```
10 >> 2
```
```
2
```



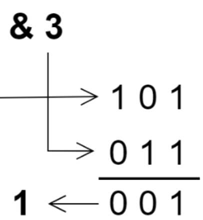
We can use the following formula to easily do the calculation.
පහසුවෙන් ගණනය කිරීම සිදු කිරීම සඳහා අපට පහත සූත්‍රය භාවිතා කළ හැකිය.

$n >> x = n // 2^{x}$

Bitwise AND

Compares each bit of two numbers and returns 1 if both bits are 1, otherwise returns 0.
සංඛ්‍යා දෙකක සෑම බිට් එකක්ම සසඳා බිටු දෙකම 1 නම් 1 ලබා දෙයි, එසේ නොමැතිනම් 0 ලබා දෙයි.

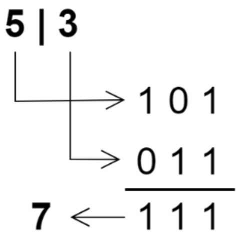
```
5&3
```
```
1
```



Bitwise OR

Compares each bit of two numbers and returns 1 if either of the bits is 1.
සංඛ්‍යා දෙකක සෑම බිට් එකක්ම සසඳා බිටු දෙකෙන් එකක් 1 නම් 1 ලබා දෙයි.

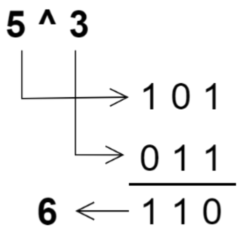
5|3
7



Bitwise XOR

Compares each bit of two numbers and returns 1 if the bits are different, otherwise returns 0.
සංඛ්‍යා දෙකක සෑම බිට් එකක්ම සසඳා බිටු වෙනස් නම් 1 ලබා දෙයි, එසේ නොමැතිනම් 0 ලබා දෙයි.

5^3
6



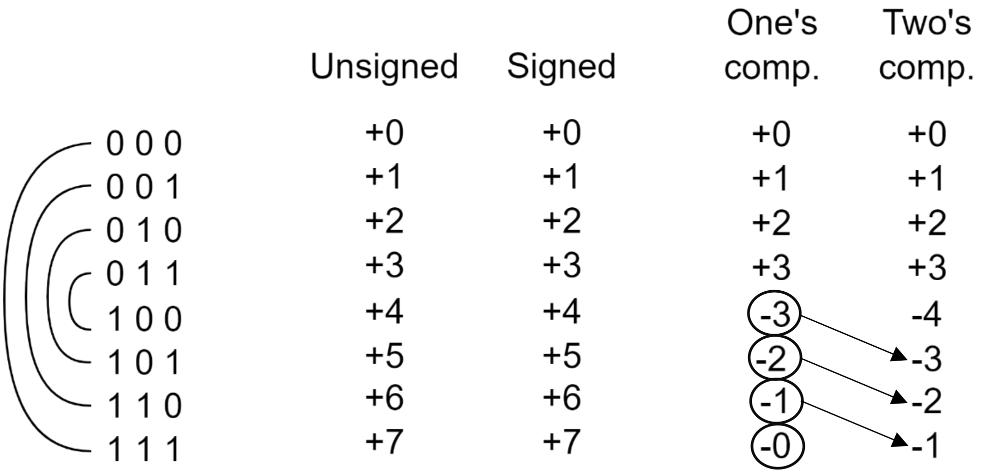
Bitwise NOT

Inverts all the bits of a number (Ex.1 becomes 0 and 0 becomes 1).
සංඛ්‍යාවක සියලුම බිටු පෙරළයි (උදා.1, 0 බවටත් 0, 1 බවටත් පත් වේ).

~2
-3

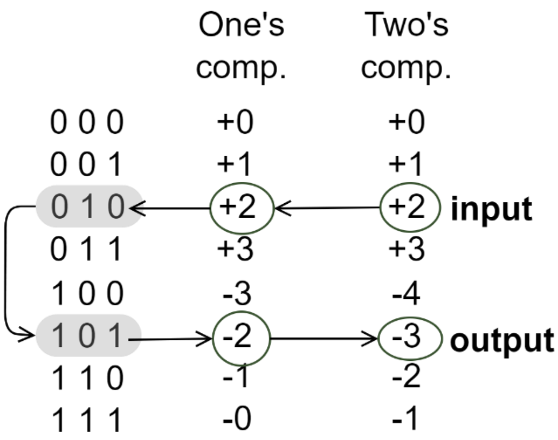
This operator also changes the sign of the number in most systems that use two's complement for representing signed integers.
මෙම මෙහෙයවනය මගින් ලකුණුවත් පූර්ණ සංඛ්‍යා නියෝජනය කිරීම සඳහා දෙකෙහි අනුපූරකය භාවිතා කරන බොහෝ පද්ධතිවල සංඛ්‍යා ලකුණ වෙනස් කරයි.

Bitwise NOT explained further
බිටු අනුසාරිත NOT පිළිබඳ වැඩිදුර පැහැදිලි කිරීම



Bitwise NOT with positive numbers
ධන සංඛ්‍යා සමඟ බිටු අනුසාරිත NOT

For example, if we ask the computer to find the bitwise NOT of 2,
උදාහරණයක් ලෙස, අප පරිගණකයෙන් 2 හි (Bitwise NOT) සොයා ගැනීමට ඉල්ලා සිටියහොත්,



One’s comp. and two’s comp. of a positive number is the same.
ධන සංඛ්‍යාවක එකෙහි සහ දෙකෙහි අනුපූරකය දෙකම එකම වේ.

1. The computer first converts 2 into binary which is 010.
පරිගණකය මුලින්ම 2 ද්විමය බවට පරිවර්තනය කරයි, එනම් 010 වේ.

2. Then, that binary number will be inverted by bitwise NOT, resulting in 101.
ඉන්පසු, එම ද්විමය සංඛ්‍යාව bitwise NOT මගින් ප්‍රතිලෝම කරන අතර එහි ප්‍රතිඵලයක් ලෙස 101 ලැබේ.

3. Since the computer outputs the result in two’s complement, the result will be the two’s complement of 101.
පරිගණකය ප්‍රතිඵලය දෙකෙහි අනුපූරකයෙන් ප්‍රතිදානය කරන බැවින්, ප්‍රතිඵලය 101හි දෙකෙහි අනුපූරකය වනු ඇත.

4. If we look at the above diagram, it shows that the two’s complement of 101 is -3.
අප ඉහත රූප සටහන දෙස බැලුවහොත්, 101හි දෙකෙහි අනුපූරකය -3 බව එමගින් පෙනේ.

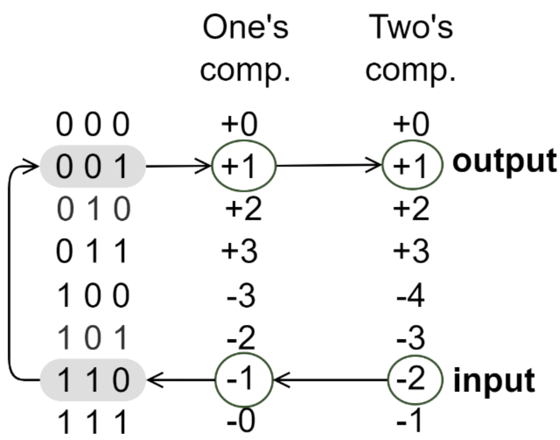
Bitwise NOT with negative numbers
සෘණ සංඛ්‍යා සමඟ බිටු අනුසාරිත NOT

```
~-2
1
```

-2 is represented as -1 in one’s complement.
-2, එකෙහි අනුපූරකයේදී -1 ලෙස නිරූපණය කෙරේ.

When bitwise NOT is applied to 110, the answer will be 001.
110 ට බිටු අනුසාරිත NOT යෙදූ විට පිළිතුර 001 වනු ඇත.

Since the one’s complement and two’s complement of 001 is the same, the output will be +1.
001 හි එකෙහි අනුපූරකය සහ දෙකෙහි අනුපූරකය සමාන බැවින්, ප්‍රතිදානය +1 වනු ඇත.



Logical operators
තාර්කික කාරක

Used to check the validity of any Boolean expression.
බූලීය ප්‍රකාශණයක වලංගුතාවය පෙන්වීම සඳහා භාවිතා කරයිග

Theses can be used to make complex logical expressions.
සංකීර්ණ තර්ක ප්‍රකාශණ ගොඩනැගීම සඳහා භාවිතා කිරීමේ හැකියාව පවතී.

1. Logical not operator
තාර්කික not කාරකය

It inverts the boolean value of something.
බූලීය අගයක් කණපිට හැරවීම මෙමගින් සිදුවේ

Should be used before the expression.
ප්‍රකාශණයකට පෙර භාවිතා කල යුතුය

```
not True
False
Not False
True
```

2. Logical or operator
තාර්කික or කාරකය

Outputs True when at least one of the two expressions combined by the operator is true.
මෙහෙයවනය මගින් සම්බන්ද කරණ ප්‍රකාශණ දෙකෙන් එකක්වත් සත්‍ය වන විට ප්‍රතිදානය සත්‍ය ලෙස ලබාදේ

Should be used between the expressions.
ප්‍රකාශණ වලට මැදින් භාවිතා කල යුතුය

```
False or False
False
False or True
True
True or False
True
True or True
True
```

3. Logical and operator
තාර්කික and කාරකය

Outputs True when both two expressions combined by the operator is true.
මෙහෙයවනය මගින් සම්බන්ද කරණ ප්‍රකාශණ දෙකම සත්‍ය වන විට ප්‍රතිදානය සත්‍ය ලෙස ලබාදේ

Should be used between the expressions.
ප්‍රකාශණ වලට මැදින් භාවිතා කල යුතුය

```
False and False
False
False and True
False
True and False
False
True and True
True
```

NOTE:
In python, the null value of each data type is False. All the other values are true.
ඕනෑම දත්ත වර්ගයක ශූන්‍ය නිරූපණය කරන අවස්ථාව අසත්‍ය ලෙසද මේ හැරුණු විට ඉතිරි සියලු අගයන් සත්‍ය ලෙස සලකයි

```
not 0
True

not []
True

not {}
True

not ()
True

not ''
True

not False
True

not 0.0
True
```


ictfromabc.com

Comparison operators
සැසඳුම් කාරක

Operators which are used to compare two variables or values and return True or False are known as comparison operators.
විචල්‍ය හෝ අගයන් දෙකක් සංසන්ධනය කරමින් සත්‍ය හෝ අසත්‍ය ලෙස ප්‍රතිදාන ලබාදීම සඳහා සැසඳුම් කාරක භාවිතා කරයි

==	equal සමාන
!=	not equal අසමාන
>	greater than විශාල
>=	greater than or equal විශාල හෝ සමාන
<	less than කුඩා
<=	less than or equal කුඩා හෝ සමාන

Let us take two initialized variables
පෙර අර්ථ දැක්වූ විචල්‍ය දෙකක් භාවිතා කරමු

```
a=1
b=2
```

To check if operands are equal
විචල්‍ය සමානදැයි විමසීම සඳහා

```
a==b
False
```

NOTE: This operator compares the hash() values of both sides
මෙම මෙහෙයවනය මගින් දෙපස hash() අගයන් සංසන්දනය කරයි

```
2==2.0
True
```

To check if operands are not equal
විචල්‍ය අසමානදැයි විමසීම සඳහා

```
a!=b
True
```

To check if left operand is greater
වම්පස විචල්‍ය විශාලදැයි විමසීම සඳහා

```
a>b
False
```

To check if left operand is greater or equal
විශාල හෝ සමානදැයි විමසීම සඳහා

```
a>=b
False
```

To check if left operand is smaller
වම්පස විචල්‍ය කුඩාදැයි විමසීම සඳහා

```
a<b
True
```

To check if left operand is smaller or equal
කුඩා හෝ සමානදැයි විමසීම සඳහා

```
a<=b
True
```

Identity operators
හඳුනාගැනීමේ මෙහෙයවන

NOTE: This operator compares the `id()` values of both sides
මෙම මෙහෙයවනය මගින් දෙපස `id()` අගයන් සංසන්දනය කරයි

is

`x=2`	`# id = 4565874`
`y=2`	`# id = 4565874`
`z=2.0`	`# id = 2154782`
`x is y`	
`True`	
`y is z`	
`False`	

is not

`x=2`	`# id = 4565874`
`y=2`	`# id = 4565874`
`z=2.0`	`# id = 2154782`
`x is not y`	
`False`	
`y is not z`	
`True`	

Membership operators
සාමාජික මෙහෙයවන

Used to check if an element is available or not available in a sequence.
Sequence එකක element එකක් පවතීද නොපවතීද සෙවීමට භාවිතා කරයි

in

`'m' in 'Man'`
`False`

not in

`'m' not in 'Man'`
`True`

Assignment operator discussed further
ආදේශ කිරීමේ මෙහෙයවනය වැඩිදුර අධ්‍යනය

(1) Normal assignment
සාමාන්‍ය ආදේශය

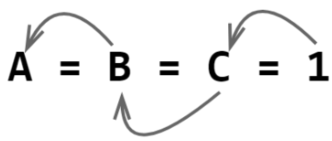
Here, the expression on the right side is assigned into the left side variable.
දකුණු පස ප්‍රකාශය වම් පස විචල්‍යට ආදේශ කිරීම මෙමගින් සිදුවේ

`a = 3`
`b = 'Ravindu'`
`c = 5*4.0`
`d = False or True`

Answer of the calculations or logics are assigned after the execution.
ගණනය කිරීම් සහ තර්ක ක්‍රියාත්මක වීමෙන් පසු ඒවායේ පිළිතුර ආදේශ වීම සිදුවේ

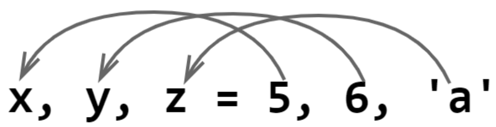
(2) Multiple assignment of same value
එකවර සමාන අගයන් කිහිපයක් ආදේශය

`A = 1`	`A = B = C = 1`
`B = 1`	
`C = 1`	



(3) Multiple assignment of different values
එකවර වෙනස් අගයන් කිහිපයක් ආදේශය

`X = 5`	`X,Y,Z = 5,6,'a'`
`Y = 6`	
`Z = 'a'`	



APPLICATION:

Swapping two variables
විචල්‍ය දෙකක අගය හුවමාරු කිරීම

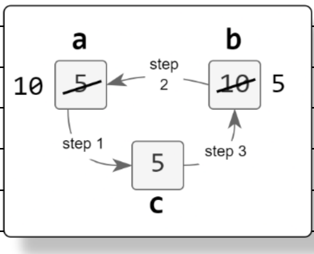
```
a=5
b=10
print(a,b)
5 10
a,b=b,a
print(a,b)
10 5
```

In other languages this must be done manually using a 3rd variable
වෙනත් භාෂා වලදී මෙය තුන්වන විචල්‍යක් භාවිතයෙන් හස්තීයව සිදුකල යුතුය

```
a=5
b=10
print(a,b)
5 10

c=a
a=b
b=c

print(a,b)
10 5
```



Mistakes

```
a,b = 1,2,3
```

Gives an error. too many values to unpack.

```
a,b,c = 1,2
```

Gives an error. not enough values to unpack.

This happens because the number of variables is not the same as the number of items in the expression list.
විචල්‍ය ප්‍රමාණය ප්‍රකාශණ ලැයිස්තුවේ ඇති අවයව ප්‍රමාණයට සමාන නොවන නිසා මෙය සිදුවේ

Augmented assignment
වැඩි දියුණු කරන ලද ආදේශය

```
x+=1     equals     x=x+1
x-=2     equals     x=x-2
x*=2     equals     x=x*2
x/=6     equals     x=x/6
x%=2     equals     x=x%2
x//=5    equals     x=x//5
x**=2    equals     x=x**2
x>>=2    equals     x=x>>2
x<<=2    equals     x=x<<2
x&=2     equals     x=x&2
x|=2     equals     x=x|2
x^=2     equals     x=x^2
```

Examples:

```
x = 5
x += 1
print(x) # 6

x = 5
x *= 2
print(x) # 10

x = 12
x /= 6
print(x) # 2.0

x = 5
x %= 2
print(x) # 1

x = 10
x //= 5
print(x) # 2

x = 8
x >>= 2
print(x) # 2

x = 3
x <<= 2
print(x) # 12
```


ictfromabc.com

Operator precedence
මෙහෙයවන ප්‍රමුඛතා අනුපිළිවෙල

1. Arithmetic Operators

Precedence Order:

**	highest precedence among arithmetic operators අංක ගණිතමය මෙහෙයවන අතර ඉහළම ප්‍රමුඛතාවය
* / // %	second highest precendence දෙවන ඉහළම ප්‍රමුඛතාවය
+ -	the lowest precedence among arithmetic operators අංක ගණිතමය මෙහෙයවන අතර අඩුම ප්‍රමුඛතාවය

Arithmetic operators have higher precedence than comparison and logical operators, so they are evaluated first in expressions.
සැසඳුම් සහ තාර්කික මෙහෙයවන වලට වඩා අංක ගණිතමය මෙහෙයවන වලට ඉහළ ප්‍රමුඛතාවයක් ඇත, එබැවින් ඒවා ප්‍රකාශනවලදී පළමුව ඇගයීමට ලක් කෙරේ.

2. Bitwise Operators

Precedence Order:

<< >>	shift operators have the **second** highest precedence මෙහෙයවනවලට **දෙවන** ඉහළම ප්‍රමුඛතාවය ඇත
&	next highest ඊළඟ ඉහළම
^	next highest ඊළඟ ඉහළම
|	Lowest අඩුම

Bitwise operators (excluding NOT) have lower precedence than arithmetic operators but higher precedence than comparison and logical operators.
බිටු අනුසාරිත මෙහෙයවන (NOT හැර) වලට අංක ගණිතමය මෙහෙයවනවලට වඩා අඩු ප්‍රමුඛතාවයක් ඇති නමුත් සැසඳුම් සහ තාර්කික මෙහෙයවනවලට වඩා ඉහළ ප්‍රමුඛතාවයක් ඇත.

3. Comparison Operators

Precedence Order:
Comparisons have equal precedence.
සැසඳීම්වලට සමාන ප්‍රමුඛතාවයක් ඇත.

Comparison operators have lower precedence than arithmetic and bitwise operators but are evaluated before logical operators.
සැසඳුම් කාරක වලට අංක ගණිතමය සහ බිටු අනුසාරිත මෙහෙයවනයවලට වඩා අඩු ප්‍රමුඛතාවයක් ඇති නමුත් තාර්කික කාරක වලට පෙර ඇගයීමට ලක් කෙරේ.

4. Membership and Identity Operators

Precedence Order:
Membership and identity operators have equal precedence and sit just below comparison operators.
Membership සහ identity මෙහෙයවනවලට සමාන ප්‍රමුඛතාවයක් ඇති අතර සැසඳීමේ මෙහයවනවලට වඩා මඳක් පහළින් සිටී.

They have lower precedence than arithmetic and bitwise operators but are evaluated before logical operators.
ඒවාට අංක ගණිතමය සහ බිටු අනුසාරිත මෙහෙයවනවලට වඩා අඩු ප්‍රමුඛතාවයක් ඇති නමුත් තාර්කික මෙහෙයවන වලට පෙර ඇගයීමට ලක් කෙරේ.

5. Logical Operators

Precedence Order:

not	highest precedence among logical operators තාර්කික මෙහෙයවන අතර ඉහළම ප්‍රමුඛතාවය
and	second highest දෙවන ඉහළම ප්‍රමුඛතාවය
or	lowest precedence අඩුම ප්‍රමුඛතාවය

Logical operators have the lowest precedence among all types, meaning they are evaluated last in an expression unless parentheses change the order.
තාර්කික මෙහෙයවන වලට සියලු වර්ග අතර අඩුම ප්‍රමුඛතාවය ඇත, එනම් වරහන් මගින් අනුපිළිවෙල වෙනස් නොකරන්නේ නම් ඒවා ප්‍රකාශනයක අවසාන වශයෙන් ඇගයීමට ලක් කෙරේ.

Parentheses
Parentheses have the highest precedence over all operators.
සියලුම මෙහෙයවන වලට වඩා වරහන් වලට ඉහලම ප්‍රමුඛතාවයක් ඇත.

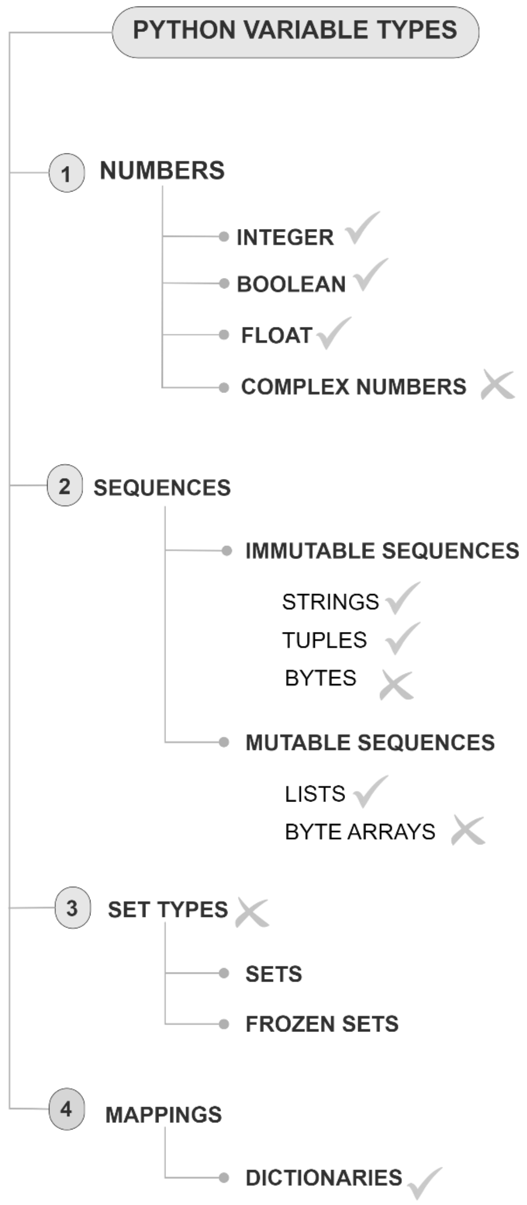
NOTE 1: Parentheses
Parentheses are not operators. They are used to change the precedence of the order in which an expression is evaluated.
වරහන් යනු මෙහෙයවන නොවේ. ප්‍රකාශනයක් ඇගයීමට ලක් කරන අනුපිළිවෙලෙහි ප්‍රමුඛතාවය වෙනස් කිරීමට ඒවා භාවිතා වේ.

NOTE 2: Bitwise NOT
In python, bitwise NOT has the next highest precedence after parentheses. Therefore, it has the highest precedence over other operators.
පයිතන් හි, වරහන් වලින් පසුව bitwise NOT හට මීළඟ ඉහළම ප්‍රමුඛතාවය හිමිවේ. එබැවින්, අනෙකුත් මෙහෙයවනවලට වඩා ඉහළම ප්‍රමුඛතාවක් මෙයට ඇත.

Total operator precedence (summary)

Parentheses	( )
Unary plus	+
Unary minus	-
Bitwise NOT	**~**
Exponentiation	**
Multiplication	*
Division	/
Integer division	//
Modulus division	%
Addition	+
Subtraction	-
Left shift	<<
Right shift	>>
Bitwise AND	&
Bitwise XOR	^
Bitwise OR	|
Comparisons	<
	<=
	>
	>=
	!=
	==
Assignments operators	=
	+=, -=, /=,
	*=, //=, %=,
	**=
Identity	is
	is not
Membership	in
	not in
Boolean NOT	not
Boolean AND	and
Boolean OR	or

Python variable types
පයිතන් භාෂාවේ විචල්‍ය වර්ග



type() function

The `type()` function in Python is used to get the data type of an object.
වස්තුවක දත්ත වර්ගය ලබා ගැනීමට පයිතන් හි ඇති `type()` ශ්‍රිතය භාවිතා කරයි.

Primitive data types of variables
විචල්‍ය වල ප්‍රාථමික දත්ත වර්ග

Integer variables store any whole number.
ඕනෑම පූර්ණ සංඛ්‍යාවක් නිඛිල විචල්‍යයක් ලෙස ගබඩා වේ

```
x=12
type(x)
<class 'int'>
```

Floating point variables store any number with a floating-point value.
ඕනෑම පූර්ණ නොවන සංඛ්‍යාවක් ඉපිලෙන ලක්ෂ්‍ය විචල්‍යයක් ලෙස ගබඩා වේ

```
y=12.2
type(y)
<class 'float'>
```

Boolean variables are used to store True or False
සත්‍ය සහ අසත්‍ය බව බූලීය විචල්‍යයක් තුල ගබඩා වේ

```
a=True
type(a)
<class 'bool'>
```

```
b=False
type(b)
<class 'bool'>
```

String variables store sequences of characters (text).
විචල්‍ය සංකේත අකුරු (පාඨය) අනුක්‍රම සටහන් කරයි

```
a="ictfromabc"
type(a)
<class 'str'>
```

Lists are ordered collections of items that can hold different data types.
ලැයිස්තු යනු විවිධ දත්ත වර්ග රැඳවිය හැකි අයිතමවල ඇණවුම් එකතුවකි.

```
a=[10, 20, 30, 40]
type(a)
<class 'list'>
```

Tuples are ordered collections like lists, but they are immutable (cannot be changed after creation).
ටියුපල්ස් ලැයිස්තු මෙන් ක්‍රමවත් එකතුවක් වන අතර, එය නිර්මාණයෙන් පසු වෙනස් කළ නොහැකි (අවිකල) වේ

```
a=(10, 20, 30, 40)
type(a)
<class 'tuple'>
```

Sets are unordered collections of unique items.
Sets යනු අද්විතීය අයිතමවල ඇණවුම් නොකළ එකතුවකි.

```
a={10, 20, 30}
type(a)
<class 'set'>
```

Dictionaries store data as key-value pairs and are mutable. Keys must be unique and are often strings.
Dictionaries දත්ත යතුරු-අග යුගල ලෙස ගබඩා කරයි හා ඒවා වෙනස් කළ හැක. යතුරු (Keys) අද්විතීය විය යුතු අතර, ඒවා බොහෝ විට strings වේ.

```
person={"name":"Ravindu","age":29}
type(person)
<class 'dict'>
```

Variables
විචල්‍ය

A temporary storage location in the main memory of a computer paired with a name (an identifier) which can contain some quantity of information is known as a variable.
තොරතුරු ප්‍රමාණයක් රඳවා තබාගත හැකි නාමයක් හා යුගලනය වූ (හඳුන්වනයක්) පරිගණකයෙහි ප්‍රධාන මතකය තුල වෙන්කරගත් තාවකාලික ගබඩා ස්ථානයක් විචල්‍යක් ලෙස හැඳින්වේ

We can create variables anywhere in the code. No pre declaration of variables before the beginning of the program is needed to reserve memory space.
කේතයේ ඕනෑම තැනක විචල්‍ය සෑදීම සිදුකල හැක. විචල්‍ය මතක තබාගැනීම සඳහා වැඩසටහන ආරම්භයට පෙර මතකය තුල ඉඩ වෙන් කිරීමක් අවශ්‍ය නොවේ.

Creation of a variable automatically happens when we assign a value.
විචල්‍යකට අගයක් ආදේශ කරන අවස්ථාවේදී විචල්‍යය සෑදීම ඉබේම සිදුවේ.

We also do not need to inform the computer about the data type of the variable that we are going create.
Python භාෂාවේදී නිර්මාණය කරන විචල්‍යයේ දත්ත වර්ගය පිලිඹඳ පරිගණකයට පෙර දැනුම්වත් කිරීමක් සිදු කිරීමක් අවශ්‍ය නොවේ.

Python dynamically allocates the space needed for a variable from the memory according to what we are going to store.
Python භාෂාව මගින් ගබඩා කරන දත්තය අනුව අවශ්‍ය ඉඩ ප්‍රමාණය ඉබේම තීරණය කිරීම සිදුකරයි.

Variable creation in different languages
විවිධ භාෂා වල විචල්‍ය නිර්මාණය

Language	Example	Typing	Declaration Requirements
Python	`x = 5`	Dynamic	No type or keyword needed; type inferred. වර්ගයක් හෝ මූල පදයක් අවශ්‍ය නොවේ; වර්ගය අනුමාන කරයි.
JavaScript	`let x = 5`	Dynamic	let for mutable, const for constants. mutable සඳහා let , නියත සඳහා const .
Java	`int x = 5;`	Static	Must declare type explicitly; type is fixed. වර්ගය පැහැදිලිව ප්‍රකාශ කළ යුතුය; වර්ගය ස්ථාවර වේ
PHP	`$x = 5;`	Dynamic	No type or keyword needed; variables use $. වර්ගයක් හෝ මූල පදයක් අවශ්‍ය නොවේ; විචල්‍ය $ භාවිතා කරයි.
C++	`int x = 5;`	Static	Must declare type explicitly; type is fixed. වර්ගය පැහැදිලිව ප්‍රකාශ කළ යුතුය; වර්ගය ස්ථාවර වේ
Swift	`var x = 5`	Static	var for mutable, let for constants. mutable සඳහා var , නියත සඳහා constants .
Kotlin	`val x = 5`	Static	var for mutable, val for constants. mutable සඳහා var , නියත සඳහා val .
Ruby	`x = 5`	Dynamic	No type or keyword needed; type inferred. වර්ගයක් හෝ මූල පදයක් අවශ්‍ය නොවේ; වර්ගය අනුමාන කරයි .
R	`x <- 5`	Dynamic	No type or keyword needed; type inferred. වර්ගයක් හෝ මූල පදයක් අවශ්‍ය නොවේ; වර්ගය අනුමාන කරයි .
Pascal	`var x: Integer; x:=5;`	Static	var keyword required; type fixed after declaration. var මූල පදය අවශ්‍යයි; ප්‍රකාශයෙන් පසුව වර්ගය ස්ථිර වේ .

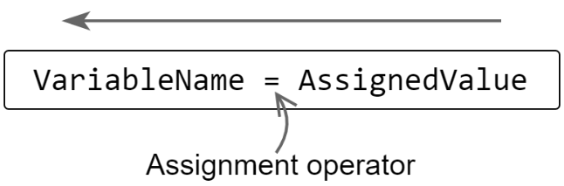
Dynamic Typing
No type declaration needed; the type is inferred from the value.
ගතික ටයිප් කිරීම: type ප්‍රකාශයක් අවශ්‍ය නොවේ; වර්ගය අගයෙන් අනුමාන කෙරේ.

Static Typing
Requires a type to be declared; type is fixed.
ස්ථිතික ටයිප් කිරීම: වර්ගයක් ප්‍රකාශ කිරීමට අවශ්‍යවේ; වර්ගය ස්ථාවර වේ

Declaration Needed
Indicates whether you must declare the type or use specific keywords (Ex. let, var).
ඔබ වර්ගය ප්‍රකාශ කළ යුතුද නැතහොත් විශේෂිත මූල පද භාවිත කළ යුතුද යන්න දක්වයි (උදා., let, var)

Creating variables and assigning values
විචල්‍ය සාදා අගයන් ආදේශය

For this the assignment operator is used between the variable name and the value
මේ සඳහා විචල්‍ය නාමය සහ අගය අතරට ආදේශ කිරීමේ මෙහෙයවනය භාවිත කරයි



NOTE: Statements including the assignment operator should always be read from right to left
ආදේශ කිරීමේ මෙහෙයවනය සහිත ප්‍රකාශ සෑම විටම දකුණේ සිට වමට කියවිය යුතුය

Naming variables
විචල්‍ය නාමකරණය

1. A variable name can only contain letters, digits, and underscores (_). No other special characters or spaces are allowed inside them.
විචල්‍ය නාමයක පැවතිය හැක්කේ අකුරු, ඉලක්කම් සහ යටි ඉරි පමණි(_). ඒ තුළ වෙනත් කිසිදු විශේෂ සලකුණු හෝ හිස් අවකාශ පැවතිය නොහැක

2. A variable name cannot be started with a digit. It can only be started with the remaining types, a letter, or an underscore.
විචල්‍ය නාමයක් ඉලක්කමකින් ආරම්භ විය නොහැක. ආරම්භ විය හැක්කේ ඉතිරි වර්ග දෙකෙන්, එනම් අකුරකින් හෝ යටි ඉරකින් පමණි

3. Language reserved words cannot be used as variable names.
භාශාවේ ඇවිරුනු පද විචල්‍ය නාම ලෙස භාවිතා කල නොහැක

4. Python is a case sensitive language.
පයිතන් යනු capital / simple සංවේදී භාශාවකි

5. A variable can be of any length.
විච්‍යලයක් ඕනෑම දිගකින් පැවතිය හැක.

Practices in defining variable names.
විචල්‍ය නාම සෑදීමේදී ඇති පුරුදු

Multi word variable names are defined in the following three forms.
වචන කිහිපයක විචල්‍ය නාම පහත පරිදි අර්ථදැක්වීම සිදුකරයි

Pascal Case
Each word starts with a capital letter:
සෑම වචනයක්ම කැපිටල් අකුරකින් ආරම්භ වේ

```
MyVariableName = "John"
```

Camel Case
Each word, except the first, starts with a capital letter:
පලමු වචනය හැර සෑම වචනයක්ම කැපිටල් අකුරකින් ආරම්භ වේ

```
myVariableName = "John"
```

Snake Case
Each word is separated by an underscore character:
සෑම වචනයක්ම යටි ඉරක් මගින් වෙන් කරනු ලබයි

```
my_variable_name = "John"
```


ictfromabc.com

Special Variables and Conventions
විශේෂ විචල්‍යයන් සහ සම්මුතීන්

Single Underscore (_):
A single underscore is often used as a throwaway variable when a value is needed but not used.
අගයක් අවශ්‍ය නමුත් භාවිතා නොකරන විට තනි යටි ඉරි බොහෝ විට ඉවතලන විචල්‍යයක් ලෙස භාවිතා කරයි.

Double Leading Underscore (__var)
Used to make a variable name "private" within a class.
පන්තියක් (class) තුළ "පුද්ගලික" විචල්‍ය නාමයක් සෑදීමට භාවිතා කරයි .

Trailing Underscore (var_)
Used to avoid conflicts with reserved keywords, Ex. class_ instead of class.
වෙන් කළ මූලපද සමඟ ගැටුම් වළක්වා ගැනීමට භාවිතා කරයි, උදා., class වෙනුවට class

Examples for Variable Names That Cause Errors
දෝෂ ඇති කරන විචල්‍ය නම් සඳහා උදාහරණ::

```
class
for
First name
Age and gender
1st_place
My-variable
total$cost
```

Examples for Correct Variable Names
නිවැරදි විචල්‍ය නම් සඳහා උදාහරණ

```
class_name
my_variable
list1
MyName
_hello
hello_
AgeAndGender
```

Displaying a variable
විචල්‍යක් පෙන්වීම

`x=1`
`print(x)`
`1`

Displaying several variables together
විචල්‍ය කිහිපයක් එකවර පෙන්වීම

`y=2`
`print(x,y)`
`1 2`

We can use **sep** parameter in the `print()` function to separate the values with a character that we want.
අපට අවශ්‍ය ාඅනුලක්ෂණයක් මගින් අගයන් වෙන් කිරීමට `print()` ශ්‍රිතයේ `sep` පරාමිතිය භාවිතා කළ හැක.

The default is `sep=" "`.
මෙහි පෙරනිමිය `sep=" "` වේ.

`print(x, y, sep="-")`
`1-2`

Displaying variables with strings
Strings සමග විචල්‍ය පෙන්වීම

`print('value of x is',x)`
`value of x is 1`

ictfromabc.com

Calculations with variables
විචල්‍ය සමග ගණනයකිරීම්

```
z=x+y
print(z)
3
print(x+y)
3
```

Calculations with variables and values
විචල්‍ය සහ අගයන් සමග ගණනයකිරීම්

```
print(x+1)
2
Print(x*y)
2
Print(y**2)
4
```

Reusing a variable again and again
එකම විචල්‍ය නැවත නැවත භාවිතය

```
print(x)
1
x=x+1
print(x)
2
print(y)
2
y = x + y**2
print(y)
5
```

Constants
නියත

Constants are variables whose values are intended to remain unchanged throughout the program.
නියතයන් යනු වැඩසටහන පුරාවටම නොවෙනස්ව පැවතීමට අදහස් කරන විචල්‍ය වේ.

While Python doesn't have built-in support for constants like some other languages, developers typically use a naming convention to indicate a constant by writing the variable name in ALL CAPITALS.
Python හට වෙනත් සමහර භාෂා වලදී මෙන් නියතයන් සෑදීම සඳහා සහය නොමැති අතර, සංවර්ධකයින් විසින් සාමාන්‍යයෙන් විචල්‍ය නාමය සියල්ල කැපිටල් අකුරු වලින් ලිවීමෙන් නියතයක් දැක්වීමට නාම කිරීමේ සම්මුතියක් භාවිතා කරයි.

```
PI = 3.14159
GRAVITY = 9.8
```

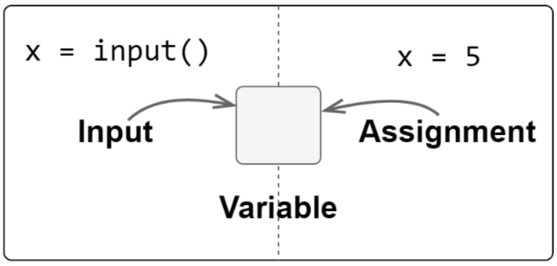
Example.

In **y = mx + c**:

- m and c are fixed constants.
  m සහ c ස්ථාවර නියතයන් වේ.
- x is the independent variable.
  x යනු ස්වායත්ත විචල්‍යයයි.
- y is the dependent variable, determined by m, c, and x.
  y යනු m, c, මෙන්ම x මගින් තීරණය කරනු ලබන පරායත්ත විචල්‍යය වේ.

Getting input from the user
පරිශීලකයාගෙන් ආදාන ලබාගැනීම

Values for a variable can be assigned from the inside of the code or taken as an input from the user when the program is running.
විචල්‍යක් සඳහා අගය කේතය ඇතුලතින් ආදේශ කල හැකි අතර කේතය ක්‍රියාත්මක වන අතරතුර පරිශීලකයාගෙන් ඉල්ලා ගැනීමද කල හැකිය



NOTE:
In python, input values are assigned in the moment which the blank variables are created.
විචල්‍ය සාදන අවස්තාවේදීම ආදාන අගයන් ආදේශ කිරීම python හිදී සිදුවේ

In other languages blank variables should be created before taking user input.
වෙනත් භාශා වලදී පරිශීලකයාගෙන් ආදාන ලබාගැනීමට පෙර විචල්‍ය සාදා තිබිය යුතුය

Input is taken later in another step
ආදාන ලබාගන්නේ වෙනත් පියවරකදීය

Inputs are taken as follows
ආදාන ලබාගන්නේ පහත පරිදිය

```
VariableName = input('prompt')
```

NOTE:
Prompt is the message shown to the user instructing him to input a data
Prompt යනු දත්තයක් ලබා දෙන ලෙස පරිශීලකයාට පෙන්වන පණිවුඩයයි

```
x=input('give length')
y=input('give width')
print(length*width)
```

NOTE:
`input()` function always takes the user input as a string value by default and assigns to the variable
`input()` ශ්‍රිතය මගින් සෑමවිටම පරිශීලකයා ආදානය කරන දත්තය string ආකාරයෙන් විචල්‍ය තුල ගබඩා කිරීම සිදුකරයි

To change the data type of the input, casting should be done
ආදානයේ දත්ත වර්ගය වෙනස් කිරීම සඳහා ජීවිසබට සිදු කල යුතුය

```
x=int(input('give length'))
y=int(input('give width'))
print('area is',length*width)
```

As a good practice, before every output which is related with a user input, we must inform the user how the output was generated
පුරුද්දක් ලෙස පරිශීලකයා විසින් ලබාදුන් ආදානයකට අදාල සෑම ප්‍රතිදානයකටම පෙර එය ලැබුණු ආකාරය පරිශීලකයා දැනුම්වත් කල යුතුය

Casting

Converting a specific data type into another data type for temporary usage, using functions is known as casting.
එක් දත්ත වර්ගයක් තාවකාලික භාවිතය සඳහා වෙනත් දත්ත වර්ගයකට පරිවර්ථනය කරගැනීම casting ලෙස හඳුන්වයි

Types of Casting
Casting වර්ග

1.Implicit Casting

Implicit casting (or automatic type conversion) in Python happens when the interpreter automatically converts one data type to another without explicit programmer instructions.
පැහැදිලි ක්‍රමලේඛක උපදෙස් නොමැතිව පරිවර්තකයා එක් දත්ත වර්ගයකට ස්වයංක්‍රීයව පරිවර්තනය කරන විට පයිතන් හි Implicit casting (හෝ ස්වයංක්‍රීය ආකාරයේ පරිවර්තනය) සිදුවේ.

This happens when there’s no risk of data loss or ambiguity in the conversion
දත්ත නැතිවීමේ අවදානමක් හෝ පරිවර්තනයේ අපැහැදිලි බවක් නොමැති විට මෙය සිදු වේ

Output automatically adopts the data type that occupies more memory
ප්‍රතිදානය ස්වයංක්‍රීයව වැඩි මතකයක් ඇති දත්ත වර්ගය භාවිතා කරයි

Integer and Float point number:

a=1
b=3.14
print(a + b)
4.14

Boolean and Integer

a=True
b=3
print(a + b)
4

Boolean and Floating point number

a=True
b=3.14
print(a + b)
4.14

2.Explicit Casting

Explicit casting (also known as type conversion) in Python involves manually converting a variable from one data type to another using built-in functions.
පයිතන් හි Explicit casting හි (වර්ග පරිවර්තනය ලෙසද හැඳින්වේ) built-in ශ්‍රිත භාවිතයෙන් එක් දත්ත වර්ගයකින් තවත් දත්ත වර්ගයකට විචල්‍යයක් අතින් පරිවර්තනය කිරීම ඇතුළත් වේ.

1. String Casting

- **String to Integer**

a = "42"
print(int(a))
42

- **String to List**

a = "hello"
print(list(a))
['h','e','l','l','o']

ictfromabc.com

- **String to float**

```
a = "3.14"
print(float(a))
3.14

b = "1"
print(float(b))
1.0
```

Note:
Below code will give a `value error`
පහත කේතය මගින් `value error` එකක් ලබා දෙනු ඇත

```
b = "ictfromabc"
print(float(b))
#gives a valueerror

print(int(b))
#gives a valueerror
```

String to Boolean

Converts a non-empty string to True, and an empty string to False.
හිස් නොවන string එකක් True බවටත්, හිස් string එකක් False බවටත් පරිවර්තනය කරයි.

```
a = "ictfromabc"
print(bool(a))
True

a=""
print(bool(a))
False
```

2. Integer Casting

- **Integer to String**

```
a = 1
a = str(a)
print(a)
1

print(type(a))
<class 'str'>
```

- **Integer to float**

```
a = 1
a = float(a)
print(a)
1.0

print(type(a))
<class 'float'>
```

- **Integer to Boolean**

Converts 0 to False, and any non-zero integer to True
0 අසත්‍ය බවටත්, ශුන්‍ය නොවන පූර්ණ සංඛ්‍යාවක් සත්‍ය බවටත් පරිවර්තනය කරයි

```
a = 1
a = bool(a)
print(a)
True

print(type(a))
<class 'bool'>
```

3. Float Casting

- **Float to Integer**

```
a = 3.99
a = int(a)
print(a)
3

print(type(a))
<class 'int'>
```

- **Float to string**

```
a = 3.14
a = str(a)
print(a)
3.14

print(type(a))
<class 'str'>
```

- **Float to Boolean**

Converts 0.0 to False, and any non-zero float to True.
0.0 False බවට පරිවර්තනය කරයි, සහ ඕනෑම ශුන්‍ය නොවන float එකක් True බවට පරිවර්තනය කරයි.

```
a = 3.14
a = bool(a)
print(a)
True

print(type(a))
<class 'bool'>
```

4. Boolean Casting

- **Boolean to Integer**

Converts True to 1 and False to 0
True 1 සහ False 0 බවට පරිවර්තනය කරයි

```
a = True
a = int(a)
print(a)
1

print(type(a))
<class 'int'>
```

- **Boolean to String**

```
a = True
a = str(a)
print(a)
True

print(type(a))
<class 'str'>
```

- **Boolean to Float**

```
a = True
a = float(a)
print(a)
1.0

print(type(a))
<class 'float'>
```

5. List Casting

- **List to String**

  Method 1:

```
a=["a","b","c"]
print(str(a))
["a","b","c"]
```

  Method 2 :

```
a=["a","b","c"]
print("".join(a))
abc
```

  `join()` method will be discussed later
  `join()` method පසුව සාකච්ඡා කෙරේ

- **List to Tuple**

```
a=["a","b","c"]
print(tuple(a))
("a","b","c")
```

- **List to Set**

```
a=["a","b","c"]
print(set(a))
("a","b","c")
```

- **List to Dictionary**

  Converts a list of tuples (key-value pairs) to a dictionary
  ටපල් ලැයිස්තුවක් (යතුරු-අගය යුගල) dictionary එකකට පරිවර්තනය කරයි

```
a=[('a',10),('b',20)]
print(dict(a))
{'a': 10, 'b': 20}
```

6. Tuple Casting

- **Tuple to List**

```
a=(1,2,3)
print(list(a))
["1","2","3"]
```

- **Tuple to String**

  Method 1:

```
a=(10,20,30)
print(str(a))
(10,20,30)
```

  Method 2:

```
a=('a','b','c')
print("".join(a))
abc
```

7.Set Casting

- **Set to list**

```
a={10,20,30}
print(list(a))
[10,20,30]
```

- **Set to tuple**

```
a={10,20,30}
print(tuple(a))
(10,20,30)
```

- **Set to string**

```
a={10,20,30}
print(str(a))
{10,20,30}
```

Control Structures
පාලන ව්‍යුහ

A block of programming that chooses a direction in for the program to flow based on given parameters.
දී ඇති පරාමිතීන් අනුව වැඩසටහනට ගලා යාමට දිශාවක් තීරණය කරන ක්‍රමලේඛණ කොටසක් පාලන ව්‍යුහයකි

Control structures control the flow of execution of a program.
පාලන ව්‍යුහයන් මගින් වැඩසටහනක් ක්‍රියාත්මක කිරීමේ ප්‍රවාහය පාලනය කරයි.

They determine how and when certain sections of code are executed based on conditions or repeated actions.
කොන්දේසි හෝ පුනරාවර්තන ක්‍රියා මත පදනම්ව කේතයේ ඇතැම් කොටස් ක්‍රියාත්මක කරන්නේ කෙසේද සහ කුමන වේලාවේද යන්න ඒවා මගින් තීරණය කරයි.

Need for control structures for better program execution
වඩා හොඳ වැඩසටහන් ක්‍රියාත්මක කිරීමක් සඳහා පාලන ව්‍යුහ වල අවශ්‍යතාවය

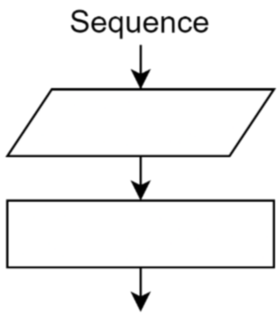
- Direct program flow and decision-making.
  සෘජු වැඩසටහන් ප්‍රවාහය සහ තීරණ ගැනීම.

- Execute specific code based on conditions.
  කොන්දේසි මත පදනම්ව නිශ්චිත කේත ක්‍රියාත්මක කිරීම.

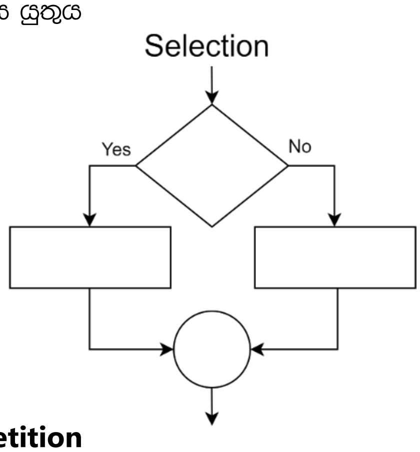
Types of Control Structures
පාලන ව්‍යුහ වර්ග

Sequence	අනුක්‍රමය
Selection	වරණය/තේරීම
Repetition	පුණර්කරණය

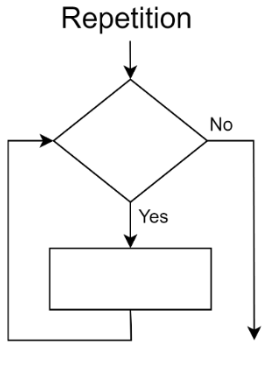
Sequence
අනුක්‍රමය
Statements are executed from the beginning to end in a strict order.
මුල සිට අගට නොවෙනස් වන පිළිවෙලකට එකක් පසුපස එකක් උපදෙස් ක්‍රියාත්මක වේ



Selection
වරණය
A decision is made depending on a condition. should always be two ways.
අවශ්‍යතාවයක් මත පදනම්ව තීරණයක් ගනු ලබයි. ප්‍රතිදානය සෑමවිටම දෙපසකට පමණක් බෙදිය යුතුය



Repetition
පුණරුක්තිය
One or several statements of an algorithm are repeated for a fixed number of times depending on a condition.
ඇල්ගොරිතමයක පියවර එකක් හෝ කිහිපයක්, යම් අවශ්‍යතාවයක් මත පදනම්ව නියත වාර ප්‍රමාණයක් නැවත නැවත ක්‍රියාත්මක වීම සිදුවේ



Data structures / Sequences
දත්ත ව්‍යුහ /අනුක්‍රමය

A sequence is a collection of items that can be indexed and iterated over.
අනුක්‍රමයක් යනු සුචිගත සහ පුනරාවර්තනය කළ හැකි අයිතම එකතුවකි.

Sequences can be of different types, and they allow you to store and manage multiple values in an organized way.
අනුක්‍රමයක් විවිධ වර්ගවල විය හැකි අතර, ඒවා සංවිධානාත්මක ආකාරයකින් බහු අගයන් ගබඩා කිරීමට සහ කළමනාකරණය කිරීමට ඔබට ඉඩ සලසයි.

Types of data structures

1. Lists

Ordered, mutable (changeable) collection of items.
අනුපිළිවෙලක් ඇති, වෙනස් කළ හැකි අයිතම එකතුවකි.

Syntax: Defined with square brackets.
කොටු වරහන් මගින් අර්ථ දක්වනු ලබයි.
Ex. `my_list = [1, 2, 3]`

Ideal for collections of related data, like a list of numbers or strings.
සංඛ්‍යා හෝ තන්තු ලැයිස්තුවක් වැනි අදාළ දත්ත එකතු කිරීම් සඳහා වඩාත් සුදුසුය.

2. Tuples

Ordered but immutable collection of items.
අනුපිළිවෙලක් ඇති නමුත් වෙනස් කළ නොහැකි අයිතම එකතුවකි.

Syntax: Defined with parentheses.
සාමාන්‍ය වරහන් මගින් අර්ථ දක්වනු ලබයි.
Ex. `my_tuple = (1, 2, 3)`

Used for fixed collections of items, where immutability is important, like coordinates `(x, y)` or other constant data.
ඛණ්ඩාංක ((ο, හ)) හෝ වෙනත් නියත දත්ත වැනි වෙනස් නොවන බව වැදගත් වන අයිතමවල ස්ථාවර එකතුවක් තබා ගැනීමට භාවිතා වේ.

3. Dictionaries

Unordered collection of key-value pairs, mutable.
යතුරු-අගය යුගලවල අනුපිළිවෙලක් නැති වෙනස් කළ හැකි එකතුවකි.

Syntax: Defined with curly braces.
සගල වරහන් මගින් අර්ථ දක්වනු ලැබේ.
Ex.
```
my_dict = {"key1": "value1",
"key2": "value2"}
```

Suitable for mappings, where you need to associate unique keys with values, such as a phone book.
දුරකථන අංක සහිත පොතක් වැනි අගයන් සමඟ අනන්‍ය යතුරු සම්බන්ධ කිරීමට අවශ්‍ය වන සිතියම්ගත කිරීම් සඳහා සුදුසු වේ.

5. Strings
Ordered, immutable sequence of characters.
අනුපිළිවෙලක් ඇති, වෙනස් කළ නොහැකි අනුලක්ෂණ අනුපිළිවෙලකි.

Syntax: Defined with quotes.
උද්ධෘත මගින් අර්ථ දක්වනු ඇත.
Ex. `my_string = "Hello"`

4. Sets

Unordered collection of unique items, mutable.
වෙනස් කළ හැකි අනන්‍ය අයිතමවල අනුපිළිවෙලක් නොමැති එකතුවකි.

Syntax: Defined with curly braces.
සගළ වරහන් මගින් අර්ථ දක්වනු ඇත.
Ex. `my_set = {1, 2, 3}`

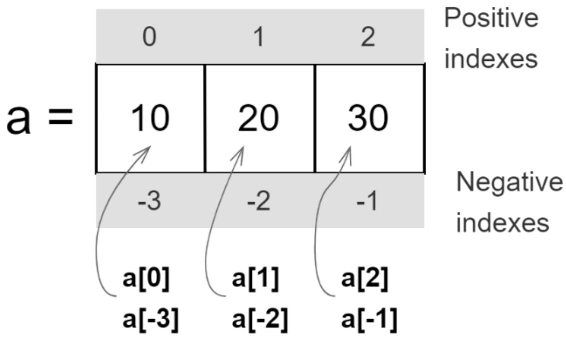
Used to store unique items and perform mathematical set operations like union, intersection, etc.
අනන්‍ය අයිතම ගබඩා කිරීමට සහ union, intersection වැනි ගණිතමය මෙහෙයුම් සිදු කිරීමට භාවිතා කරයි.

ictfromabc.com

Lists
ලැයිස්තු

Lists are used to store several data under the same identifier.
එකම හඳුන්වන නාමය යටතේ දත්ත කිහිපයක් රඳවා තබා ගැනීම සඳහා ඛසිවි භාවිතා කරයි

A list contains items separated by commas and enclosed by square brackets.
list එකක් තුල comma යොදා වෙන් කල දත්ත කොටු වරහන් තුල අන්තර්ගතව පවතී



Items in a list can be different data types.
මේවා තුල විවිද වර්ග වල දත්ත පැවතිය හැක

Elements can be added, edited, and deleted after creating a list (lists are mutable)
පවතින කසිවි තුලට අලුත් දත්ත ඇතුලත් කිරීමට, පවතින දත්ත වෙනස් කිරීමට සහ මකා දැමීමට හැකියාව පවතී

Has an order of values.
අගයන් අනුපිළිවෙලක් පවතී.

Can contain duplicate values.
සමාන අගයන් කිහිපයක් පැවතිය හැකිය.

Elements of a list can be individually accessed using a unique index.
අනන්‍ය වූ index භාවිතයෙන් එම දත්ත වෙත වෙන් වෙන්ව පිවිසීම සිදු කල හැක

Structure of a list inside main memory of a computer may look like follows

පරිගණකයේ මතකය තුල list එකක ව්‍යුහය පහත පරිදිය.

```
K=[10, False, 20, 3.14, True, 5.1]
```

Accessing list elements can be done in two methods as follows
list එකක අවයව වලට ප්‍රවේශ වීම පහත පරිදි ක්‍රම දෙකකට සිදුකල හැකිය.

From left to right using positive indexes
වමේ සිට දකුණට ධන දර්ශක යොදා ගනිමින්

```
print(K[0]) # 10
print(K[2]) # 20
```

From right to left using negative indexes
දකුණේ සිට වමට සෘණ දර්ශක යොදා ගනිමින්

```
print(K[-1]) # 5.1
print(K[-5]) # False
```

Modifying list elements can be done using the following method.
List එකක අවයව වෙනස් කිරීම පහත ක්‍රමය මගින් සිදුකල හැක.

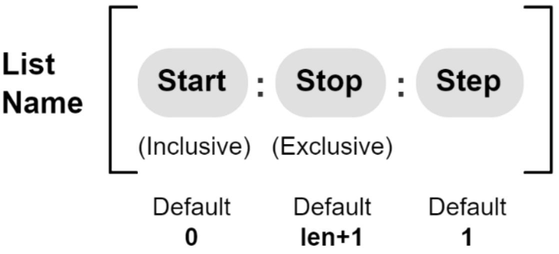
```
J=[10, 20, 30, 40, 50]
```

```
J[1]=500
print(J)
[10, 500, 30, 40, 50]
```

Slicing

Accessing several elements of a list can be done by defining start, stop and step values.
ආරම්භක, අවසාන සහ පියවර අගයන් යොදාගනිමින් එකවර list එකක අවයව කිහිපයක් වෙත ප්‍රවේශ වීම

0	1	2	3	4
10	20	30	40	50
-5	-4	-3	-2	-1



Default values are used when user defined values are not given.
පරිශීලකයා විසින් අර්ථදක්වන ලද අගයන් ලබා දී නොමැති විට සම්මත අගයන් භාවිත වේ

```
J = [10, 20, 30, 40, 50]

print(J[1:4])
[20, 30, 40]

print(J[1:4:2])
[20, 40]

print(J[::2])
[10, 30, 50]

print(J[:-1:])
[10, 20, 30, 40]

print(J[-1:0:-1])
[50, 40, 30, 20]
```

Modifying a slice of a list at once can be done using the following method.
List එකක කොටසක් එකවර වෙනස් කිරීම පහත ක්‍රමය මගින් සිදුකල හැක.

Changing the exact number of values
හරියටම අගයන් ගණනක් වෙනස් කිරීම

```
marks = [10, 20, 30, 40, 50]

marks[1:3] = [60, 70]
print(marks)
[10, 60, 70, 40, 50]
```

Changing less values with more
අඩු අගයන් ගණනක් වැඩි අගයන් ගණනකින් වෙනස් කිරීම

```
marks = [10, 20, 30, 40, 50]

marks[1:3] = [60, 70, 80, 90]
print(marks)
[10, 60, 70, 80, 90, 40, 50]
```

Changing more values with less
වැඩි අගයන් ගණනක් අඩු අගයන් ගණනකින් වෙනස් කිරීම

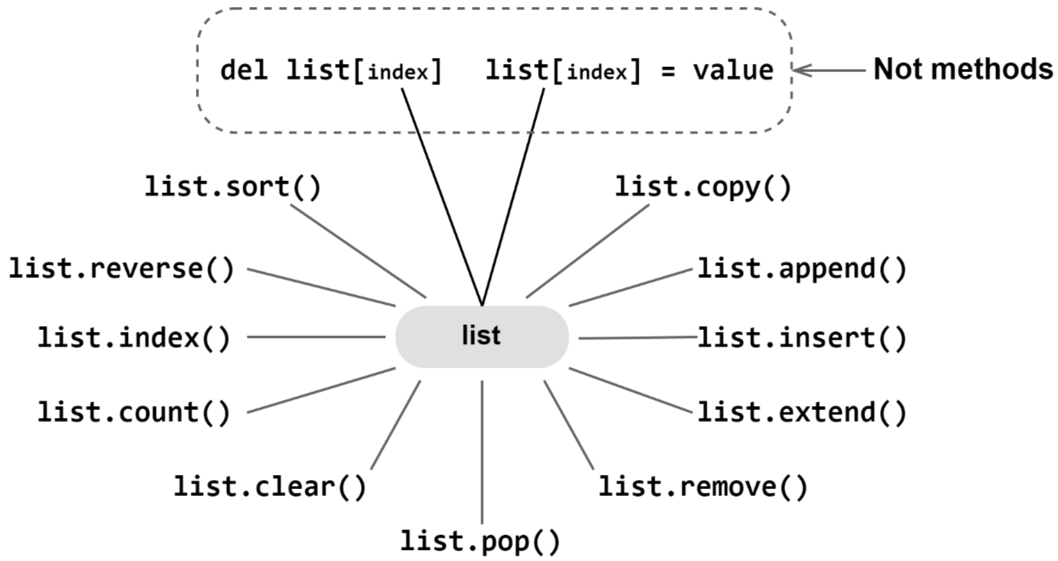
```
marks = [10, 20, 30, 40, 50]

marks[1:3] = [100]
print(marks)
[10, 100, 40, 50]
```

Adding elements to lists cannot be done by calling unavailable indexes
නොපවතින indexes කැඳවා list එකකට අලුත් අවයව එක් කිරීම සිදු කල නොහැක

```
R=[]
R[0]=34
gives an error.
```

Using methods with lists
ලැයිස්තු සමග methods භාවිතා කිරීම



append() method is used to add new elements into a list.
list එකකට අලුත් අවයව එකතු කිරීම සඳහා append () method එක භාවිතා කරයි

```
R.append(2)
print(R)
[2]
```

append() method can add only one element to the end of a list at a time.
list එකක අගට එක් වරකට එක් අවයවයක් පමණක් එක් කිරීම සඳහා append() method එක භාවිතා කරයි

```
R.append(3,4,5)
gives an error.
```

insert() method is used to add an element to a specific index of a list.
list එකක නිශ්චිත index එකකට අවයවයක් එක් කිරීම සඳහා insert() method එක භාවිතා කරයි.

```
a=[10, 20, 30]
a.insert(2,'k')
```

This inserts 'k' into index 2 of list a
List a හි index 2 ලෙස 'k' ඇතුලත් කරයි

```
print(a)
[10, 20, 'k', 30]
```

`extend()` method is used to extend a list by another list

list එකක් තවත් list එකකින් දීර්ග කිරීම සඳහා `extend()` method එක භාවිතා කරයි.

```
x=[1, 2, 3]
y=[4, 5, 6]

x.extend(y)
print(x)
[1, 2, 3, 4, 5, 6]
```

NOTE:

Lists can be extended by any iterable object (string, tuple, set, dictionary)

ඕනෑම දත්ත වර්ගයක් මගින් list extend කිරීම සිදුකල හැක.

```
z='hello'
y.extend(z)
print(y)
[4, 5, 6, 'h', 'e', 'l', 'l','o']
```

`del` command is used to Delete the whole list

මුලු list එකම මකා දැමීම සඳහා `del` විධානය භාවිතා කරයි

```
T=[10, 20, 30, 40]
del T       # deletes the list.
print(T)    # gives an error.
```

`del` command can be also used to delete specific index elements from a list

list එකකින් නිශ්චිත index එකක අවයව මකා දැමීම සඳහාද del විධානය භාවිතා කරයි.

```
S = [10, 20, 30, 40, 50]

del S[0]
print(S)
[20, 30, 40, 50]
```

`remove()` method can be used to remove the first coming specific item from a list

list එකක මුලින්ම හමුවන නිශ්චිත දත්තයක් ඉවත් කිරීම සඳහා `remove()` method එක භාවිතා කරයි

```
b=list('colombo')
print(b)
['c','o','l','o','m','b','o']
b.remove('o')
print(b)
['c','l','o','m','b','o']
b.remove('o')
print(b)
['c','l','m','b','o']
b.remove('o')
print(b)
['c','l','m','b']
b.remove('o')
gives an error
```

`pop()` method can be used to remove a specific item at a given index

දී ඇති index එකක දත්තයක් ඉවත් කිරීම සඳහා `pop()` method එක භාවිතා කල හැක

```
c=list('colombo')
print(c)
['c','o','l','o','m','b','o']

c.pop(4)
'm'
print(c)
['c', 'o', 'l', 'o', 'b', 'o']
```

Here the popped data will be displayed on the screen, or we can pass it to another variable.

මෙහිදී pop වන දත්තය තිරය මත දර්ශණය වන අතර, අවශ්‍ය නම් එය වෙනත් විචල්‍යයක් තුලට ලබාගත හැකිය

```
x = c.pop(5)
print(x)
'o'
```

NOTE:
If we do not specify which item to remove, pop() method removes the last item of the list.
ඉවත් කල යුත්තේ කුමන දත්තයද යැයි සඳහන් නොකල විට pop() method එක මගින් list එකක අවසාන දත්තය ඉවත් කරයි.

clear() method can be used to remove all the elements from a list but keep the empty list as a data structure in the memory.
list එකකින් සියලුම අවයව ඉවත් කර, හිස් දත්ත ව්‍යුහය පමණක් මතකයේ තබා ගැනීම සඳහා clear() method එක භාවිතා කරයි.

```
x = [10, 20, 30, 40, 50]
x.clear()
print(x)
[]
```

reverse() method reverses the order of a list by rearranging the list items backwards.
reverse() method එක මගින් ලැයිස්තු අයිතම අග සිට මුලට නැවත සැකසීමෙන් ලැයිස්තුවේ අනුපිළිවෙල ආපසු හරවයි.

```
x = [10, 50, 40, 30, 20]
x.reverse()
print(x)
[20, 30, 40, 50, 10]
```

count() method is used to determine the number of times a specified value occurs in a list.
ලැයිස්තුවක නිශ්චිත අගයක් ඇති වාර ගණන තීරණය කිරීමට count() ක්‍රමය භාවිතා කරයි.

```
numbers = [1, 2, 3, 2, 2, 4]
count_of_twos = numbers.count(2)
print(count_of_twos)
3
```

sort() method sorts the list in ascending order by default.
sort() method එක මගින් පෙරනිමියෙන් ලැයිස්තුව ආරෝහණ අනුපිළිවෙලට sort කරයි.

```
x = [10, 50, 40, 30, 20]
x.sort()
print(x)
[10, 20, 30, 40, 50]
```

parameters

```
list.sort(reverse, key)
```

reverse
reverse=True will sort the list in descending order. Default is reverse=False
reverse=True ලැයිස්තුව අවරෝහණ අනුපිළිවෙලට වර්ග කරයි. පෙරනිමිය reverse=False වේ

key
The key parameter allows you to specify a function that determines the criteria by which elements should be sorted.
අවයව sort කළ යුතු නිර්ණායක තීරණය කරන ශ්‍රිතයක් නියම කිරීමට key පරාමිතිය මගින් ඔබට ඉඩ සලසයි.

Ex.

Sorting a list of strings by length instead of alphabetical order

```
words = ["apple", "banana",
"kiwi", "grape"]

words.sort(key=len)

print(words)
#output
['kiwi', 'grape', 'apple',
'banana']
```

index() Method

`index()` method in Python is used to find the position of the first occurrence of a specified value in a list.
ලැයිස්තුවක නිශ්චිත අගයක පළමු සිදුවීමේ පිහිටීම සොයා ගැනීමට පයිතන් හි index() ක්‍රමය භාවිතාකරයි.

Syntax:

```
list.index(value, start, end)
```

- **value**: The element whose position is to be found.
  පිහිටීම සොයා ගත යුතු අවයවය.
- **start** (optional): The starting index from where the search begins.
  සෙවීම ආරම්භ වන ස්ථානයේ සිට ආරම්භක දර්ශකය.
- **end** (optional): The ending index where the search stops (exclusive).
  සෙවීම නතර කරන අවසන් දර්ශකය (ඇතුළත් නොවේ)

```
numbers = [10, 20, 30, 20, 40, 20]
numbers.index(20, 2, 5)
print(numbers.index(20, 2, 5))
3
print(numbers.index(20))
```

1

When the first occurrence of the specified value is found, its index is returned.
නිශ්චිත අගයේ පළමු සිදුවීම සොයාගත් විට, එහි දර්ශකය ආපසු ලබා දෙනු ලැබේ.

If the value is not found, a `ValueError` is raised.
අගය සොයාගත නොහැකි නම්, `ValueError` මතු වේ.

copy() method

The copy() method is used to create a copy of a list.
ලැයිස්තුවක පිටපතක් සෑදීමට copy() ක්‍රමය භාවිතා කරයි.

This means it creates a new list object that has the same elements as the original list, but is a separate instance in memory.
මෙයින් අදහස් කරන්නේ එය මුල් ලැයිස්තුවට සමාන අවයව ඇති නමුත් මතකයේ වෙනම අවස්ථාවක් වන නව ලැයිස්තු වස්තුවක් නිර්මාණය කරන බවයි.

Changes in the original list do not affect the copied list.
මුල් ලැයිස්තුවේ වෙනස්වීම් පිටපත් කළ ලැයිස්තුවට බලපාන්නේ නැත.

```
# Original list
flat_list = [1, 2, 3, 4]

# Using copy()
copied_flat_list=flat_list.copy()

# Modifying the original list
flat_list[0] = 100

print("Original List:", flat_list)
[100, 2, 3, 4]

print("Copied List:",
copied_flat_list)
[1, 2, 3, 4]
```

Additionally, **list()** function can also be used to create a copy.
ඊට අමතරව, පිටපතක් සෑදීමට list() ශ්‍රිතය ද භාවිතා කළ හැක.

```
a = [10, 20, 30, 40, 50]
b = list(a)
a[0] = 100
print(a) # [100, 20, 30, 40, 50]
print(b) # [10, 20, 30, 40, 50]
```

NOTE

Assigning a List vs Copying a List
ලැයිස්තුවක් පැවරීම එදිරිව ලැයිස්තුවක් පිටපත් කිරීම

Assigning a List
ලැයිස්තුවක් පැවරීම (y = x)

- Both variables point to the same memory reference.
  විචල්‍ය දෙකම එකම මතක යොමු වෙත යොමු වෙයි.

- Changes made to one list reflect in the other.
  එක් ලැයිස්තුවකට කරන ලද වෙනස්කම් අනෙක් ලැයිස්තුවටද බලපායි.

- No new list is created, so memory usage is minimal.
  නව ලැයිස්තුවක් සාදනු නොලැබේ, එබැවින් මතක භාවිතය අවම වේ.

Copying a List
ලැයිස්තුවක් පිටපත් කිරීම (y = x.copy()):

- A new list is created with the same elements as the original.
  නව ලැයිස්තුවක් මුල් පිටපතට සමාන මූලද්‍රව්‍යය සමඟ නිර්මාණය වේ.

- Changes made to one list do not affect the other.
  එක් ලැයිස්තුවකට කරන ලද වෙනස්කම් අනෙකට බලපාන්නේ නැත.

- Uses more memory since a new list object is created.
  නව ලැයිස්තුක් සෑදූ බැවින් වැඩි මතකයක් භාවිතා කරයි.

copy() with Nested Lists

when the list contains nested lists (lists within lists), the behavior of copy() differs.
ලැයිස්තුවේ කැදලි ලැයිස්තු (ලැයිස්තු තුළ ලැයිස්තු) ඇති විට, පිටපතේ copy() හැසිරීම වෙනස් වේ.

A new outer list is created, but the inner lists are not independently copied.
නව බාහිර ලැයිස්තුවක් සාදනු ලැබේ, නමුත් අභ්‍යන්තර ලැයිස්තු ස්වාධීනව පිටපත් නොකෙරේ.

Instead, references to the inner lists are shared between the original and copied
ඒ වෙනුවට, අභ්‍යන්තර ලැයිස්තු වෙත යොමු කිරීම් මුල් සහ පිටපත් කළ ලැයිස්තු අතර බෙදා ගනී.

This means that changes made to the inner lists of the original list will also affect the inner lists of the copied list.
මෙයින් අදහස් කරන්නේ මුල් ලැයිස්තුවේ අභ්‍යන්තර ලැයිස්තු වලට කරන ලද වෙනස්කම් පිටපත් කළ ලැයිස්තුවේ අභ්‍යන්තර ලැයිස්තු වලටද බලපාන බවයි.

```
# Original nested list
nestedL = [[1, 2, 3], [4, 5, 6]]

# Using copy()
copied_nestedL = nestedL.copy()

# Modifying an inner list in the
# original nested list
nestedL[0][0] = 100

# Results
print("Original Nested List:",
nestedL)
[[100, 2, 3], [4, 5, 6]]

print("Copied Nested List:",
copied_nestedL)
[[100, 2, 3], [4, 5, 6]]
```

Using functions with lists
lists සමග ශ්‍රිත භාවිතය

Example List

```
Q = [10, 50, 30, 25, 65]
```

1. Displaying the List
ලැයිස්තුව පෙන්වීම

This prints the entire list to the screen.
මෙය සම්පූර්ණ ලැයිස්තුවම තිරයට මුද්‍රණය කරයි.

```
print(Q)
[10, 50, 30, 25, 65]
```

2. Counting Elements
මූලද්‍රව්‍යය ගණන් කිරීම

This returns the number of elements in the list.
මෙය ලැයිස්තුවේ ඇති මූලද්‍රව්‍ය ගණන ආපසු ලබා දෙයි.

```
len(Q)
5
```

3. Finding the Minimum Value
අවම අගය සොයා ගැනීම

This gives the smallest value in the list.
මෙය ලැයිස්තුවේ කුඩාම අගය ලබා දෙයි.

```
min(Q)
10
```

4. Finding the Maximum Value
උපරිම අගය සොයා ගැනීම

This gives the largest value in the list.
මෙය ලැයිස්තුවේ විශාලතම අගය ලබා දෙයි.

```
max(Q)
65
```

5. Checking Data Type
දත්ත වර්ගය පරීක්ෂා කිරීම

This shows the data type of the object.
මෙය වස්තුවක දත්ත වර්ගය පෙන්වයි.

```
type(Q)
<class 'list'>
```

6. Sorting in Ascending Order
ආරෝහණ අනුපිළිවෙලින් වර්ග කිරීම

This returns a new list with elements sorted in ascending order, without changing the original list.
මෙය මුල් ලැයිස්තුව වෙනස් නොකර, ආරෝහණ අනුපිළිවෙලට සකසන ලද මූලද්‍රව්‍ය සහිත නව ලැයිස්තුවක් ලබා දෙයි.

```
sorted(Q)
[10, 25, 30, 50, 65]
```

7. Reversing the List
ලැයිස්තුව ආපසු හැරවීම

This returns a new list with the elements in reverse order, leaving the original list unchanged.
මෙය මුල් ලැයිස්තුව නොවෙනස්ව තබමින් ප්‍රතිලෝම අනුපිළිවෙලෙහි මූලද්‍රව්‍ය සහිත නව ලැයිස්තුවක් ලබා දෙයි.

```
list(reversed(Q))
[65, 25, 30, 50, 10]
```

8. Summing Elements
ලැයිස්තුවක අගයන් එකතු කිරීම

This calculates and returns the sum of all elements in the list.
මෙය ලැයිස්තුවේ ඇති සියලුම මූලද්‍රව්‍යවල එකතුව ගණනය කර ආපසු ලබා දෙයි.

```
sum(Q)
180
```

Tuples

Tuples also can be used to store several data under the same identifier.
එකම හඳුන්වන නාමය යටතේ දත්ත කිහිපයක් රඳවා තබා ගැනීම සඳහා Tuples ද භාවිතා කල හැක.

A tuple contains items separated by commas and enclosed by round brackets.
Tuple එකක් තුල comma යොදා වෙන් කල දත්ත රවුම් වරහන් තුල අන්තර්ගතව පවතී

Tuples can also be created without using brackets.
වරහන් භාවිතා නොකරද tuples සෑදීම සිදුකල හැක.

Items in a tuple can be of different data types.
මේවා තුල විවිද වර්ග වල දත්ත පැවතිය හැක

Elements cannot be added, edited, and deleted after creating a tuple (tuples are immutable)
පවතින tuple තුලට අලුත් දත්ත ඇතුලත් කිරීමට, පවතින දත්ත වෙනස් කිරීමට සහ මකා දැමීමට හැකියාව ක් නැත.

Has an order of values.
අගයන් අනුපිළිවෙලක් පවතී.

Elements of a tuple can be individually accessed using a unique index.
අනන්‍ය වූ සබාුං එකක් භාවිතයෙන් එම දත්ත වෙත වෙන් වෙන්ව පිවිසීම කල හැක

Can contain duplicate values.
සමාන අගයන් කිහිපයක් පැවතිය හැකිය.

The structure of a tuple in the main memory of a computer looks like follows.
පරිගණකයේ මතකය තුල tuple එකක ව්‍යුහය පහත පරිදිය.

```
K = 1, False, 2, 3.14, True, 5.1
```

Accessing tuple elements can be done in two methods as follows.
Tuple එකක අවයව වලට ප්‍රවේශ වීම පහත පරිදි ක්‍රම දෙකකට සිදුකල හැකිය.

From left to right using positive indexes
වමේ සිට දකුණට ධන දර්ශක යොදා ගනිමින්

From right to left using negative indexes
දකුණේ සිට වමට සෘණ දර්ශක යොදා ගනිමින්

Because tuples are immutable,
tuples සෑදූ පසු වෙනස් කල නොහැකි බැවින්

Modifying tuple elements cannot be done.
Tuple එකක අවයව වෙනස් කිරීම සිදු කල නොහැක.

```
K = 1, False, 2, 3.14, True, 5.1
K[0] = 10 # GIVES AN ERROR
```

Adding elements cannot be done
අවයව එක් කල නොහැක.

```
R = ()
R.append()  gives an error.
R.extend()  gives an error.
R.insert()  gives an error.
```

Deleting elements from a tuple cannot be done
tuple එකකින් අවයව මකා දැමීම සිදුකල නොහැක.

```
S = 1, 2, 3, 4, 5
del S[0]    gives an error.
S.remove()  gives an error.
S.pop()     gives an error.
```

Deleting the whole tuple can be done
මුළු tuple එකම මකා දැමීම සිදු කල හැක.

```
S = 1, 2, 3, 4, 5
del S        # deletes the tuple.
print(S)     # gives an error.
```

Slicing

Slicing in tuples is done in the same way as in lists.
Tuples වල slicing ද ලැයිස්තු වල මෙන් ම සිදු කෙරේ.

```
T = 10, 20, 30, 40, 50

print(T[0:4])
(10, 20, 30, 40)

print(T[-1::-1])
(50, 40, 30, 20, 10)

print(T[::2]
(10, 30, 50)
```

Default values are used when user defined values are not given.
පරිශීලකයා විසින් අර්ථදක්වන ලද අගයන් ලබා දී නොමැති විට සම්මත අගයන් භාවිත වේ

Using methods with tuples
tuples සමග methods භාවිතය

copy()

```
Q = 10 , 50 , 30 , 25 , 65
R = Q.copy()
```

Gives an error because tuples are immutable.
Tuples වෙනස් කළ නොහැකි බැවින් දෝශ පණිවුඩයක් දිස්වේ

The reasoning behind this is that you don't need to "copy" a tuple in the same sense you do with a mutable object
මෙය පිටුපස ඇති තර්කය වන්නේ, වෙනස් කළ හැකි වස්තුවක් සමග සිදු කරන ආකාරයටම එකක් "පිටපත්" කිරීමට අවශ්‍ය නොවන බවයි.

you can't change a tuple after it's created anyway, so the concept of copying doesn't apply in the same way.
එය නිර්මාණය කළ පසු කෙසේවත් එය වෙනස් කළ නොහැක, එබැවින් පිටපත් කිරීමේ සංකල්පය එකම ආකාරයකින් අදාල නොවේ.

count()
Returns the number of occurrences of an element in the tuple.
Tuple එකෙහි ඇති අවයවයක සිදුවීම් ගණන ආපසු ලබා දෙයි.

```
tup = (10, 20, 20, 30, 40, 20)
print(tup.count(20))
3
```

index()
Returns the index of the first occurrence of an element in the tuple.
එකෙහි ඇති අවයවයක පළමු සිදුවීමට අදාල දර්ශකය ආපසු ලබා දෙයි.

```
tup = (10, 20, 30, 40)
print(tup.index(30))
2
```

Using functions with tuples
Tuples සමග ශ්‍රිත භාවිතය

`Q = 10 , 50 , 30 , 25 , 65`

`print(Q)`
`(10, 50, 30, 25, 65)`
This displays the tuple on the screen.
තිරය මත tuple එක පෙන්වයි

`type(Q)`
`<class 'tuple'>`
Displays the data type of the tuple.
Tuple එකෙහි දත්ත වර්ගය පෙන්වයි

`len()`
Returns the number of elements in the tuple.
Tuple එකෙහි ඇති අවයව ගණන ආපසු ලබා දෙයි.

`tup = (12, 34, 56)`
`print(len(tup))`
`3`

`min()`
Returns the smallest element in the tuple.
Tuple එකෙහි ඇති කුඩාම අවයවය ආපසු ලබා දෙයි.

`tup = (56, 23, 89)`
`print(min(tup))`
`23`

max()
Returns the largest element in the tuple.
Tuple එකෙහි ඇති විශාලතම අවයවය ආපසු ලබා දෙයි.

`tup = (56, 23, 89)`
`print(max(tup))`
`89`

`sum()`
Returns the sum of all the elements in the tuple (if they are numeric).
Tuple එකෙහි ඇති සියලුම අවයව වල එකතුව (ඒවා සංඛ්‍යා නම්) ලබා දෙයි.

`tup = (12, 34, 56)`
`print(sum(tup))`
`102`

`sorted()`
Rearranges the tuple in ascending order.
Tuple එක ආරෝහණ පිලිවෙලට සකසයි

`tup = (34, 12, 56, 89)`
`print(tuple(sorted(tup)))`
`(12, 34, 56, 89)`

`reversed()`
Rearranges the tuple in reverse order.
Tuple එක එකට සකසයි

`tup = (34, 12, 56, 89)`
`print(tuple(reversed(tup)))`
`(89, 56, 12, 34)`

`tuple()`
Converts an iterable (Ex. list, string) into a tuple.
Tuple එකක් iterable එකක් බවට පරිවර්තනය කරයි.

`lst = [12, 34, 56]`
`print(tuple(lst))`
`(12, 34, 56)`

Strings

Strings are arrays of bytes representing Unicode characters.
String යනු යුනිකේත සංකේත නිරූපණය කරන බයිට් ආරාවකි.

A string is made up of zero or more alphabetical, numerical, or special characters.
string එකක් සෑදී ඇත්තේ අක්ශර, අංක හෝ විශේෂ සංකේත කිහිපයකිනි

They are enclosed in either single quotation marks, double quotation marks, or triple quotation marks.
ඒවා තනි උද්‍ධෘත පාඨ, ද්වී උද්‍ධෘත පාඨ හෝ උද්‍ධෘත පාඨ ත්‍රිත්ව තුල ඇතුලත් කර පවතී

Characters of a string can be individually accessed using a unique index.
අනන්‍ය වූ index එකක් භාවිතයෙන් එම අනුලක්ශණ වෙත වෙන් වෙන්ව පිවිසීම සිදු කල හැක

```
x = "Hello"
print(x[0])
H
```

Characters cannot be added, edited or deleted after creating a string (strings are immutable)
පවතින strings තුලට අලුත් අනුලක්ශණ ඇතුලත් කිරීමට, පවතින අනුලක්ශණ වෙනස් කිරීමට සහ මකා දැමීමට හැකියාව නොපවතී

Can contain duplicate characters.
සමාන අගයන් කිහිපයක් පැවතිය හැකිය.

The structure of a string in the main memory of a computer looks as follows.
පරිගණකයේ මතකය තුල string එකක ව්‍යුහය පහත පරිදිය.

```
K = 'Helloworld'
```

Accessing string characters can be done in two methods as follows.
String එකක අවයව වලට ප්‍රවේශ වීම පහත පරිදි ක්‍රම දෙකකට සිදුකල හැකිය.

From left to right using positive indexes
වමේ සිට දකුණට ධන දර්ශක යොදා ගනිමින්

From right to left using negative indexes
දකුණේ සිට වමට ඍණ දර්ශක යොදා ගනිමින්

Because strings are immutable,
Strings සෑදූ පසු වෙනස් කල නොහැකි බැවින්

Modifying string characters cannot be done.
string එකක අවයව වෙනස් කිරීම සිදු කල නොහැක.

```
K[5] = "m" # Gives an error
```

Adding elements to a string cannot be done.
string වලට අවයව එක් කල නොහැක.

```
R='hello'
R.append( ) # gives an error.
R.extend( ) # gives an error.
R.insert( ) # gives an error.
```

Deleting elements from a string cannot be donne.
String එකකින් අවයව මකා දැමීම සිදුකල නොහැක.

```
S='hello'
del S[0]    gives an error.
S.remove( ) gives an error.
S.pop( )    gives an error.
```

Deleting the whole string can be done.
මුළු string එකම මකා දැමීම සිදු කල හැක

S='hello'	
del S	deletes the string.
print(S)	gives an error.

Slicing

Slicing in strings is done in the same way as in lists and tuples.
Strings වල slicing ද ලැයිස්තු සහ tuples වල මෙන් ම සිදු කෙරේ.

text = "Hello, World!"

print(text[0:5])
Hello

print(text[7:12])
World

print(text[:5])
Hello

print(text[7:])
Hello

print(text[-6:])
World

print(text[::2])
Hlo ol!

print(text[-1::-1])
!dlroW ,olleH

Default values are used when user defined values are not given.
පරිශීලකයා විසින් අර්ථදක්වන ලද අගයන් ලබා දී නොමැති විට සම්මත අගයන් භාවිත වේ

Using functions with strings

Q='hello'

`print(Q)`
This displays the string on the screen
තිරය මත string එක පෙන්වයි
`'hello'`

`len(Q)`
Displays the element count in the string
String එක තුල ඇති අවයව ගණන පෙන්වයි
5

`min(Q)`
Displays the minimum element of the string.
string එකෙහි අවම අවයවය පෙන්වයි
`'e'`

`max(Q)`
Displays the maximum element of the string
String එකෙහි උපරිම අවයවය පෙන්වයි
`'o'`

NOTE:
Here, the ASCII values are considered when choosing the `min` and `max` values.
මෙහිදී, `min` සහ `max` අගයන් තෝරාගැනීමේදී ASCII අගයන් සලකනු ලැබේ.

`type(Q)`
Displays the data type of the string
String එකෙහි දත්ත වර්ගය පෙන්වයි
`<class 'str'>`

`sorted(Q)`
Arranges the string in the ascending order
string එක ආරෝහණ පිළිවෙලට සකසයි
`['e', 'h', 'l', 'l', 'o']`

`sorted(Q,reverse=True)`
Arranges the string in descending order.
string එක අවරෝහණ පිළිවෙලට සකසයි
`['o', 'l', 'l', 'h', 'e']`

`list(reversed(Q))`
arranges the string in the reverse order.
string එක අවරෝහණ පිළිවෙලට සකසයි
`['o', 'l', 'l', 'e', 'h']`

ictfromabc.com

Using methods with strings
strings සමග methods භාවිතය

`x = y.copy()`
Gives an error
දෝශ පණිවුඩයක් දිස්වේ

`index()` method
Displays the index of the first coming specific element in a string
string එකක මුලින්ම හමුවන නිශ්චිත දත්තයට අදාල index එක පෙන්වයි

```
k="Hello, World!"
index=k.index("World")
print(index)
7
```

`replace()` method
Used to replace a substring of a string with another substring
String එකක කොටසක් වෙනත් කොටසක් මගින් ප්‍රතිස්ථාපනයට භාවිතා කරයි

```
k='GoodMorning!'
k.replace('Morning','Evening')
'GoodEvening!'
```

`join()` method
This is used to join a sequence of data into a single string using a special string as intermediate.
විශේෂ string එකක් අතරමැදියා ලෙස භාවිතා කරමින් දත්ත sequence එකක් එක් string එකක් බවට පත් කිරීමට මෙය භාවිතා කරයි

```
a='kurunegala'
'-'.join(a)
'k-u-r-u-n-e-g-a-l-a'
```

`capitalize()` method
Returns a string where the first letter is upper case, and the rest is lower case.
පළමු අක්ශරය කැපිටල් සහ ඉතිරි අක්ශර සිම්පල් අකුරු සහිත string එකක් ලබාදේ.

```
x = "hello WORLD"
print(x.capitalize())
Hello world
```

`title()` method
Make the first letter in each word upper Case සෑම වචනයකම පළමු අකුර capital බවට හරවයි.

```
x = "hello WORLD"
print(x.capitalize())
Hello World
```

`lower()` method
`casefold()` method
Returns a string where all the characters are lower case.
සෑම අක්ශරයක්ම සිම්පල් අකුරු සහිත string එකක් ලබාදේ.

```
x='RAVINDU'
print(x.lower())
ravindu
```

`upper()` method
Converts a string into upper case.
string එකක් capital අකුරු බවට හරවයි.

```
x='ravindu'
print(x.upper())
RAVINDU
```

`center()` method

Returns the aligned string, using a specified character (default is space)
නිශ්චිත සංකේතයක් භාවිතා කරමින් මැදට එකෙල්ල කරන ලද string එකක් ලැබේ

Syntax:

```
string.center(length, character)
```

`length` - Length of the returned string
පිළිතුර ලෙස ලැබෙන string එකේ දිග

`character` - What to fill the missing space
හිස්තැන් පිරවිය යුතු සංකේතය

```
x = 'Ravindu'
x.center(25)
'         Ravindu         '
x. center(25,'_')
'_________Ravindu_________'
```

`count()` method

Returns the number of substrings inside a specific part of a string
string එකක යම් කොටසක් තුල පවතින නිශ්චිත substrings ප්‍රමාණය ලබාදේ

Syntax:

```
string.count(value, start, end)
```

`value` – required substring
අවශ්‍ය කරන substring එක

`start/end` – index range of the string (default values are 0 and infinity)
ප්‍රධාන string එකේ ආරම්භය සහ අවසානය.

```
x='hello'
x.count('l')
2
x.count('ll')
1
x.count('l',0,2)
0
```

`strip()` method

Used to remove the whitespaces at the beginning and end of a string.
string එකක මුලට සහ අගට ඇති whitespaces ඉවත් කිරීම සඳහා භාවිතා කරයි.

```
x=' Ravindu Bandaranayake\n'
print(x.strip())
Ravindu Bandaranayake
```

We can also specify the characters to be removed if other than whitespaces.
whitespaces වලට අමතරව වෙනත් සංකේත ඉවත් කිරීමට අවශ්‍ය නම් ඒවාද ලබා දිය හැකිය.

```
x='aaaaRavindu Bandaranayakeaaaa'
print(x.strip('a'))
Ravindu Bandaranayake
```

This method will not remove characters in the middle of a string.
මෙම method එක මගින් string එකක මධ්‍යයේ ඇති characters ඉවත් කිරීම සිදු නොකරයි.

```
x=' Ravindu      Bandaranayake\n '
print(x.strip())
Ravindu      Bandaranayake
```

`lstrip()` and **`rstrip()`** methods can be separately used to only remove the characters before the beginning of the string or after the end of the string.
Sring එකක මුලට සහ string එකක අගට ඇති characters වෙන වෙනම ඉවත් කිරීම සඳහා `lstrip()` සහ `rstrip()` යන methods දෙක භාවිතා කල හැකිය.

```
x=' Ravindu Bandaranayake\n'
x.lstrip()
'Ravindu Bandaranayake\n'
x.rstrip()
' Ravindu Bandaranayake'
```

split() method

Each word of a string is separated into an item in a list.

String එකක සෑම වචනයක්ම කොටස් වලට කඩා list එකක අගයක් බවට පරිවර්තනය කරයි.

We can specify a character from which the string should be split, or by default python will split the string from any whitespace in it.

string එක කොටස් කල යුතු අගය අපට ලබා දිය හැකි අතර සම්මත වශයෙන් python භාෂාව මගින් string එකෙහි ඇති ඕනෑම whitespace එකකින් එය කොටස් වලට කඩයි.

```
x = 'code never lies'
y = x.split()
print(y)
['code', 'never', 'lies']
```

```
z = x.split('e')
print(z)
['cod', ' n', 'v', 'r li', 's']
```

The number of maximum splits which should be done can also be specified.

කොටස් වලට කඩා වෙන්කල යුතු අවස්ථා ගණනද අවශ්‍ය නම් ලබාදිය හැකිය.

```
x = 'code never lies'
y = x.split(' ',1)
print(y)
['code', 'never lies']
```

isalnum()

Returns True if all characters in the string are alphanumeric

තන්තුවෙහි සියලුම අනුලක්ෂණ අක්ෂර සහ සංඛ්‍යා නම් සත්‍ය ලෙස ලබා දෙයි

```
string = "John123"
print(string.isalnum())
True
```

```
string = "John@123"
print(string.isalnum())
False
```

isalpha()

Returns True if all characters in the string are in the alphabet

තන්තුවෙහි ඇති සියලුම අනුලක්ෂණ හෝඩියේ අකුරුනම් සත්‍ය ලබා දෙයි

```
string = "John"
print(string.isalpha())
True
```

```
string = "John123"
print(string.isalpha())
False
```

isascii()

Returns True if all characters in the string are ascii characters

තන්තුවෙහි ඇති සියලුම අනුලක්ෂණ ascii අක්ෂර නම් සත්‍ය ලෙස ලබා දෙයි

```
string = "John123"
print(string.isascii())
True
```

isdigit()

Returns True if all characters in the string are digits

තන්තුවෙහි ඇති සියලුම අනුලක්ෂණ සංඛ්‍යා නම් සත්‍ය ලබා දෙයි

```
string = "8000"
print(string.isdigit())
True
```

isdecimal()

Returns True if all characters in the string are decimals

තන්තුවේ සියලුම අනුලක්ෂණ දහයේ පාදයේ නම් සත්‍ය ලෙස ලබා දෙයි

```
string = "123"
print(string.isdecimal())
True
```

```
string = "John123"
print(string.isdecimal())
False
```

isidentifier()

Returns True if the string is an identifier

තන්තුව identifier එකක් නම් සත්‍ය ලෙස ලබා දෙයි

A string is considered a valid identifier if it only contains alphanumeric letters (a-z) and (0-9), or underscores (_).

තන්තුවක අනුලක්ෂණ අකුරු (a-z) සහ (0-9) හෝ යටි ඉරි (_) පමණක් අඩංගු වන්නේ නම් එය වලංගු එකක් ලෙස සැලකේ.

A valid identifier cannot start with a number, or contain any spaces.

වලංගු identifier එකක් සංඛ්‍යාවකින් ආරම්භ විය නොහැක, හෝ කිසිදු එක්ක් අඩංගු විය නොහැක.

```
x = "LinkToWebsite"
y = "myCategory2"
w = "2ndFolder"
z = "Gas Report"
```

```
print(x.isidentifier())
True
print(y.isidentifier())
True
print(w.isidentifier())
False
print(z.isidentifier())
False
```

islower()

Returns True if all characters in the string are lower case

තන්තුවෙහි සියලුම අනුලක්ෂණ කුඩා අකුරු නම් සත්‍ය ලෙස ලබා දෙයි

```
string = "andrew"
print(string.islower())
True
```

```
string = "Andrew"
print(string.islower())
False
```

isupper()

Returns True if all characters in the string are upper case

තන්තුවේ සියලුම අනුලක්ෂණ විශාල අකුරු නම් සත්‍ය ලෙස ලබා දෙයි

```
string = "JOHN"
print(string.isupper())
True
```

```
string = "John"
print(string.isupper())
False
```

isnumeric()

Returns True if all characters in the string are numeric

තන්තුවෙහි ඇති සියලුම අනුලක්ෂණ සංඛ්‍යාත්මක නම් සත්‍ය ලබා දෙයි

```
string = "11738"
print(string.isnumeric())
True
```

```
string = "kld11738"
print(string.isnumeric())
False
```

isprintable()

Returns True if all characters in the string are printable

තන්තුවෙහි සියලුම අනුලක්ෂණ මුද්‍රණය කළ හැකි නම් සත්‍ය ලෙස ලබා දෙයි

```
string = "Hello World"
print(string.isprintable())
True
```

```
string = "Hello\nWorld"
print(string.isprintable())
False
```

isspace()

Returns True if all characters in the string are whitespaces

තන්තුවෙහි සියලුම අක්ෂර whitespaces නම් සත්‍ය ලෙස ලබා දෙයි

```
string = "        "
print(string.isspace())
True
```

```
string = "  .     "
print(string.isspace())
False
```

istitle()

Returns True if the string follows the rules of a title

තන්තුව මාතෘකාවක රීති අනුගමනය කරන්නේ නම් සත්‍ය ලබා දෙයි

```
string = "Heading 1"
print(string.istitle())
True
```

```
string = "We were taught lies"
print(string.istitle())
False
```

Dictionaries

Dictionaries are also used to store several data under the same identifier.
එකම හඳුන්වන නාමය යටතේ දත්ත කිහිපයක් රඳවා තබා ගැනීම සඳහා Dictionarie භාවිතා කරයි

Dictionaries are a type of mapping which provide a lookup facility based on keys we define.
අප විසින් අර්ථ දක්වන යතුරු මගින් දත්ත සෙවීමට Dictionaries තුලදී අවස්තාව සැලසේ

Contains key:value pairs inside of curly brackets.
කොමා මගින් වෙන් කල යතුරුඃඅගයන් යුගල සඟල වරහන් තුල පවතී

Items can be accessed using keys.
අගයන් යතුරු යොදාගනිමින් ප්‍රවේශ කරගත හැකිය.

Keys can only be integers or strings.
යතුරු ලෙස භාවිත කල හැක්කේ නිඛිල හෝ strings පමණි.

Items in a list can be different data types.
මේවා තුල විවිද වර්ග වල දත්ත පැවතිය හැක

Dictionaries do not allow duplicate values.
මේවා අනුපිටපත් අගයන්ට සහය නොදක්වයි

After Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries were unordered.
Python 3.7 සංස්කරණයෙන් පසු dictionaries පිළිවෙලක් සහිත දත්ත වර්ගයක් වන අතර Python 3.7 සහ ඉන් පෙර ඒවායේ පිළිවෙලක් අර්ථදක්වා නැත.

Keys can be integers or strings.
යතුරු integers හො strings විය හැක

Dictionaries are mutable.
(Can be changed)
නිර්මාණය කිරීමෙන් පසු මේවා වෙනස් කල හැකිය.

Structure of a dictionary
Dictionary එකක ව්‍යුහය

```
person1 = {'name':'Rav','age':26}
```

Accessing dictionary elements
dictionary අවයව වලට ප්‍රවේශ වීම

```
person1['name']
'Rav'
person1.get("name")
'Rav'
person1.keys()
dict_keys(['name', 'age'])
person1.values()
dict_values(['Rav', 26])
```

Adding data to a dictionary
Dictionary එකකට දත්ත ඇතුලත් කිරීම

```
person1['phone'] = '0714555513'
person1.update({"gender":"male"})
print(person1)
{'name':'Rav','age':26,
'phone':'0714555513',
'gender':' male'}
```

Element keys which are originally not present in a dictionary can be added.
dictionary එකක මුලදී නොපවතින keys වලට අවයව එක් කිරීමක් සිදු කල හැක

Updating a dictionary
Dictionary එකක අවයව වෙනස් කිරීම

```
person1['phone'] = 'unknown'
person1.update({"age":27})
print(person1)
{'name':'Rav','age':27,
'phone':'unknown',
'gender':' male'}
```

Deleting values from a dictionary
Dictionary එකකින් අවයවයක් මකා දැමීම

pop()

Removes the item with the specified key and returns its value; if the key is not found, it returns the default value if provided, otherwise raises a `KeyError`.
නිශ්චිතව සඳහන් කර ඇති යතුරෙහි අයිතමය ඉවත් කර එහි අගය ආපසු ලබා දෙයි; යතුර සොයාගත නොහැකි නම්, පෙරනිමි අගයක් සපයා ඇත්නම් එය ලබා දෙයි, එසේ නොමැතිනම් `KeyError` දෝෂයක් මතු කරයි.

syntax
`pop(key, default)`

```
d = {'a': 78, 'b': 90, 'c': 12}
value = d.pop('b')
print(value)
90
print(d)
{'a': 78, 'c': 12}
```

```
# Using pop() with default value
value = d.pop('d', 0)
print(value)
0
```

popitem()
Removes and returns the last inserted (key, value) pair from the dictionary.
Dictionary එකට අවසන් වරට ඇතුළත් කළ (යතුර, අගය) යුගලය ඉවත් කර ආපසු ලබා දෙයි.

```
d = {'m': 34, 'n': 56, 'o': 78}
pair = d.popitem()
print(pair)
('o', 78)
print(d)
{'m': 34, 'n': 56}
```

In Python 3.7 and later versions last item is removed by `popitem()` method. Earlier it was a random item because dictionaries were unordered.
Python 3.7 සංස්කරණයෙන් පසු popitem() method එක භාවිතයෙන් අවසාන යතුරු අගය යුවල ඉවත් කල හැකි අතර Python 3.6 සහ ඉන් පෙර ඒවායේ පිළිවෙලක් පැවතියේ නැති නිසා අහඹු අගයක් ඉවත්වී යයි.

Deleting all values from a dictionary
Dictionary එකකින් අවයව සියල්ල මකා දැමීම

```
d = {'x': 12, 'y': 34, 'z': 56}
d.clear()
print(d)
{}
```

Deleting a dictionary
Dictionary එකක් මකා දැමීම

```
del person1
print(person1)
# generates an error.
```

Commonly used methods

`dictionary.get()` method

Is used with a dictionary to retrieve the value for a specified key.
නිශ්චිත යතුරක් සඳහා අගයක් ලබා ගැනීමට dictionary එකක් සමඟ භාවිතා වේ

```
k = {'name': 'John', 'age': 30}
y = k.get('name')
print(y)
John
```

`dictionary.items()` method

Used with dictionaries to return a list of dictionary's key-value tuple pairs.
Dictionary එකක යතුරු-අගය tuple යුගල list එකක් ආපසු ලබාගැනීම සඳහා භාවිතා වේ

```
k = {'name':'John','age': 30}
z = k.items()

print(z)
dict_items([('name', 'John'),
('age', 30)])
```

`dictionary.keys()`

This method returns a view object that displays all the keys in the dictionary.
මෙම එක එකෙහි සියලුම යතුරු පෙන්වන දර්ශන වස්තුවක් ලබා දෙයි.

```
print(my_dict.keys())
dict_keys(['a', 'b', 'c'])
```

`dictionary.values()`

This method returns a view object that displays all the values in the dictionary.
මෙම එක එකෙහි ඇති සියලුම අගයන් පෙන්වන දර්ශන වස්තුවක් ලබා දෙයි.

```
my_dict = {'a': 1, 'b': 2, 'c': 3}

print(my_dict.values())
dict_values([1, 2, 3])
```

`fromkeys()` methods

fromkeys() method is used to create a new dictionary from a given sequence of keys, with all values initialized to the same specified value.
fromkeys() ක්‍රමය භාවිතා කරනුයේ දී ඇති යතුරු අනුපිළිවෙලකින් නව dictionary එකක් නිර්මාණය කිරීම සඳහා වන අතර, සියලු අගයන් එකම නිශ්චිත අගයට ආරම්භ කර ඇත.

Syntax

```
dict.fromkeys(keys, value=None)
```

- **keys**: A sequence (Ex. list, tuple, or string) of keys that will become the keys in the dictionary.
  dictionary එකේ යතුරු බවට පත්වන යතුරු අනුපිළිවෙලක් (උදා., list, tuple, string).

- **value**: The value assigned to each key. If not provided, the default is None.
  එක් එක් යතුරට පවරා ඇති අගය. සපයා නොමැති නම්, පෙරනිමිය None වේ

Example 1: Basic Usage

```
keys = ['a', 'b', 'c']
value = 0

newd = dict.fromkeys(keys, value)
print(newd)

{'a': 0, 'b': 0, 'c': 0}
```

Example 2: Default Value (None)

```
keys = ['x', 'y', 'z']

newd = dict.fromkeys(keys)
print(newd)

{'x': None, 'y': None, 'z': None}
```

NOTE:
Copying a dictionary cannot be simply done using assignment operator. It will only generate a reference to the original dictionary, not a separate copy as we discussed in lists.
Dictionary එකක පිටපතක් ලබාගැනීම ආදේශ කිරීමේ මෙහෙයවනය භාවිතයෙන් පහසුවෙන් සිදුකරගත නොහැක. ලැයිස්තු වලදී සාකාච්චා කළ ආකාරයට එමගින් නිර්මාණය වන්නේ මුල් dictionary එකට යොමුවක් පමණි.

```
person1={'name':'Rav','age':28}
person2=person1

print(person2)
{'name':'Rav','age':28}

person2['name']='Chamath'

print(person2)
{'name':'Chamath','age':28}

print(person1)
{'name':'Chamath','age':28}
```

This can be done by `dict()` function.
මෙය `dict()` ශ්‍රිතය භාවිතයෙන් සිදුකල හැකිය.

```
person1={'name':'Rav','age':28}
person2=dict(person1)
print(person2)
{'name':'Rav','age':28}
person2['name']='Chamath'
print(person2)
{'name':'Chamath','age':28}
print(person1)
{'name':'Rav','age':28}
```

`y = x.copy()`
Also generates a separate copy as discussed in lists
ලැයිස්තු වලදී සාකච්චා කළ ආකරයටම වෙනම පිටපතක් ලබාගැනීම සඳහා `copy()` method එකද භාවිතා කල හැකිය

Using functions with dictionaries.
Dictionaries සමග ශ්‍රිත භාවිතය

```
Q={1:'one',2:'two',3:'three'}
```

`print (Q)`
Displays the dictionary on the screen.
තිරය මත dictionary එක පෙන්වයි
`{1:'one',2:'two',3:'three'}`

`len (Q)`
Displays the number of values.
අවයව ගණන පෙන්වයි
3

`min (Q)`
displays the minimum key of the list.
list එකෙහි අවම key එක පෙන්වයි
1

`max (Q)`
Displays the maximum key of the list
list එකෙහි උපරිම key එක පෙන්වයි
3

`type (Q)`
displays the data type of the dictionary
Dictionary එකෙහි දත්ත වර්ගය පෙන්වයි
`<class 'dict'>`

`sorted (Q)`
Arranges in the ascending order of keys.
keys වැඩිවන පිළිවෙලට සකසයි
`[ 1, 2, 3 ]`

The following methods and functions cannot be used with dictionaries.
මෙම ශ්‍රිත සහ ක්‍රම dictionaries සමග භාවිතා කල නොහැක

```
append ()
insert ()
remove ()
reversed ()
index ()
count ()
```

Sets

A set is a data structure that stores unique, unordered items.
set එකක් යනු අනන්‍ය, අනුපිලිවෙලක් නොමැති අයිතම ගබඩා කරන දත්ත ව්‍යුහයකි.

Sets are primarily used to eliminate duplicates and for performing set operations like union, intersection, and difference.
sets මූලික වශයෙන් භාවිතා කරනුයේ අනුපිටපත් ඉවත් කිරීමට සහ union, intersection, difference සහ වැනි කුලක ආශ්‍රිත සිදු කිරීම සඳහා ය.

Sets automatically remove duplicate values, so each element in a set is unique.
Sets ස්වයංක්‍රීයව අනුපිටපත් අගයන් ඉවත් කරයි, එබැවින් set එකක එක් එක් අයිතම අනන්‍ය වේ.

You can add or remove elements from a set after it's created.
Set එකක් නිර්මාණය කළ පසු ඔබට එයට අයිතම එක් කිරීමට හෝ ඉවත් කිරීමට හැකිය.

The elements inside a set must be immutable types (like numbers, strings, or tuples).
Set එකක් තුළ ඇති අයිතම වෙනස් කළ නොහැකි දත්ත වර්ග (numbers, strings, tuples වැනි) විය යුතුය.

Creating a set
You can create a set using curly braces {} or the `set()` constructor:
ඔබට සගල වරහන් හෝ `set()`constructor එක භාවිතයෙන් එකක් සෑදිය හැක:

```
# Using curly braces
my_set = {1, 2, 3, 4}
# Using set() constructor
another_set = set([1, 2, 3, 4])
type(my_set); type(another_set)
<class 'set'>
<class 'set'>
```

Sets don't maintain a specific order of elements. So, you cannot be sure in which order the items will appear.
Sets නිශ්චිත අයිතම අනුපිළිවෙලක් පවත්වා නොගනී. එබැවින් අයිතම දිස්වන්නේ කුමන අනුපිළිවෙලටදැයි ඔබට සහතික කල නොහැක.

```
my_set = {1, 2, 3, 4}
print(my_set)
# random combination
# {2, 1, 3, 4} or
# {1, 3, 4, 2} or
# {4, 2, 3, 1} etc.
```

Cannot be referred to by index or key.
දර්ශකයක් හෝ යතුරක් මගින් අයිතම වෙත යොමු විය නොහැක.

```
print(my_set[0])
# gives an error
```

NO duplicates

```
my_set = {1, 2, 3, 4, 1, 2}
print(my_set)
```

The output will contain only the unique values (1, 2, 3, 4) in a random order.
ප්‍රතිදානය තුළ අහඹු අනුපිළිවෙලකට අනන්‍ය අගයන් (1, 2, 3, 4) පමණක් අඩංගු වේ.

NOTE:
True and 1 are considered as the same value in sets.
True සහ 1 යන අගයන් වලදී එකම අගය ලෙස සැලකේ.

False and 0 are considered as the same value in sets
False සහ 0 යන අගයන් වලදී එකම අගය ලෙස සැලකේ

```
my_set ={0, 1, 2, 3, 4, True, False}
print(my_set)
{1, 4, 3, 0, 2} # random output
```

`len()` function can be used to get the length of a set.
Set එකක දිග ලබා ගැනීමට `len()` ශ්‍රිතය භාවිතා කළ හැක.

```
my_set ={1, 2, 3, 4}
print(len(my_set))
4
```

```
my_set ={0, 1, 2, 3, 4, True, False}
print(len(my_set))
5
```

Accessing items in a set cannot be done by using indexes or keys.
Set එකක අයිතම වෙත ප්‍රවේශ වීම දර්ශක හෝ යතුරු භාවිතයෙන් කළ නොහැක.

But items can be accessed by looping through the set, or asking if a specific item is in the set.
නමුත් set එක හරහා ලූප් කිරීමෙන් හෝ නිශ්චිත අයිතමයක් set එකෙහි තිබේදැයි විමසීමෙන් අයිතම වෙත ප්‍රවේශ විය හැකිය.

Looping

```
my_set ={1, 2, 3, 4}

for item in my_set:
      print(item)
# any random combination
1
2
4
3
```

Checking

```
print(2 in my_set)
True
```

Adding items to a set can be done by using `add()` function.
Set එකකට අයිතම එකතු කිරීම `add()` ශ්‍රිතය භාවිතයෙන් කළ හැක.

```
my_set ={1, 2, 3, 4}
my_set.add("Hello")

print(my_set)
{1, "Hello", 2, 4, 3}
```

Items from another set can be added to the current set by using `update()` function.
`update()` ශ්‍රිතය භාවිතයෙන් වෙනත් set එකක අයිතම වත්මන් set එකට එක් කළ හැක.

```
my_set ={1, 2, 3, 4}
another_set = {"Hello", 10}

my_set.update(another_set)

print(my_set)
{1, 2, 3, 4, 'Hello', 10}
```

Adding sequences can also be done
අනුක්‍රම එකතු කිරීම ද කළ හැකිය

```
my_set ={1, 2, 3, 4}
new_list = ["Hello", 10]

my_set.update(new_list)

print(my_set)
{1, 2, 3, 4, 'Hello', 10}
```

Removing items in a set
Set එකක ඇති අයිතම ඉවත් කිරීම

remove()

Removes a specified item. If the item is not in the set, `remove()` will raise an error.

නිශ්චිත අයිතමයක් ඉවත් කරයි. අයිතමය set එකෙහි නොමැති නම්, `remove()` මගින් දෝෂයක් ඇති කරයි.

discard()

Also removes a specified item, but does not raise an error if the item to remove does not exist.

මෙමගින්ද නිශ්චිත අයිතමයක් ඉවත් කරයි, නමුත් ඉවත් කළ යුතු අයිතමය නොමැති නම් දෝෂයක් මතු නොකරයි.

```
my_set ={1, 2, 3, 4}
my_set.remove(3)
my_set.discard(1)

print(my_set)
{2, 4}
```

pop()

Removes a random item. Cannot be sure about what item gets removed.

අහඹු අයිතමයක් ඉවත් කරයි. ඉවත් කරන්නේ කුමන අයිතමයද යන්න ගැන සහතික විය නොහැක.

The removed value will be the return value of pop().

ඉවත් කළ අගය pop()හි ප්‍රත්‍යාගමන අගය වනු ඇත.

```
my_set ={1, 2, 3, 4}

x = my_set.pop()
print(x)
2 # can be any value in the set

print(my_set)
{1, 3, 4}
```

clear()

Removes all the items in the set. Doesn't delete the set.

Set එකෙහි ඇති සියලුම අයිතම ඉවත් කරයි. Set එක මකන්නේ නැත.

```
my_set ={1, 2, 3, 4}
my_set.clear()
print(my_set)
set()
```

del

Deletes the entire set.

සම්පූර්ණ set එකම මකා දමයි.

```
my_set ={1, 2, 3, 4}
del my_set
print(my_set)
# Gives an error
```

copy() method

The copy() method is used to create a shallow copy of a set.

set එකක නොගැඹුරු පිටපතක් සෑදීම සඳහා copy() ක්‍රමය භාවිතා කරයි.

A new set is created with the same elements as the original set.

මුල් set එකට සමාන මූලද්‍රව්‍ය සමඟ නව set එකක් නිර්මාණය වේ.

Modifications made to the copied set are not reflected in the original set, and changes to the original set do not affect the copied one.

පිටපත් කරන ලද set එකට කරන ලද වෙනස් කිරීම් මුල් set එකේ පිළිබිඹු නොවන අතර මුල් set එකේ වෙනස්කම් පිටපත් කළ එකට බලපාන්නේ නැත.

```
original_set = {1, 2, 3, 4}
copied_set = original_set.copy()

print("Original Set:",
original_set)
{1, 2, 3, 4}

print("Copied Set:", copied_set)
{1, 2, 3, 4}

# An element is added to the
# copied set
copied_set.add(5)

print("Original Set:",
original_set)
{1, 2, 3, 4}

print("Copied Set:", copied_set)
{1, 2, 3, 4, 5}
```

Joining sets

There are several ways to join two or more sets.
Sets දෙකක් හෝ වැඩි ගණනක් සම්බන්ධ කිරීමට ක්‍රම කිහිපයක් තිබේ.

```
set_a = {1, 2, 3}
set_b = {3, 4, 5}
```

Union (| or `.union()`):
Combines all elements from both sets, removing duplicates.
අනුපිටපත් ඉවත් කරමින් sets දෙකෙන්ම සියලුම අවයව ඒකාබද්ධ කරයි.

```
result1 = set_a | set_b
result2 = set_a.union(set_b)

print(result1)
{1, 2, 3, 4, 5}

print(result2)
{1, 2, 3, 4, 5}
```

Intersection (& or `.intersection()`):
Returns elements that are common to both sets.
Sets දෙකටම පොදු අවයව ලබා දෙයි.

```
result1 =set_a & set_b
result2 =set_a.intersection(set_b)

print(result1)
{3}

print(result2)
{3}
```

Difference (- or `.difference()`):
Returns elements that are in the first set but not in the second.
පළමු set එකෙහි ඇති නමුත් දෙවැන්නෙහි නොමැති අයිතම ලබා දෙයි.

```
result1 =set_a - set_b
result2 =set_a.difference(set_b)

print(result1)
{1, 2}

print(result2)
{1, 2}
```

Symmetric Difference
(^ or `.symmetric_difference()`):
Returns elements that are in either of the sets, but not in both (no overlap).
Sets දෙකෙන් ඕනෑම එකක ඇති, නමුත් sets දෙකෙහිම නැති අයිතම ලබා දෙයි.

```
result1=set_a ^ set_b
result2=set_a.symmetric_difference
(set_b)
print(result1, result2)
{1, 2, 4, 5} {1, 2, 4, 5}
```

`difference_update()` method

The difference_update() method is used to modify a set by removing all elements found in another specified set .
වෙනත් නිශ්චිත set එකක ඇති සියලුම මූලද්‍රව්‍ය ඉවත් කිරීමෙන් set එකක් වෙනස් කිරීමට difference_update() ක්‍රමය භාවිතා කරයි.

Unlike the difference() method, which returns a new set, difference_update() modifies the original set in place and does not return a new set.
නව set එකක් ආපසු ලබා දෙන difference() ක්‍රමය මෙන් නොව, difference_update() මුල් කුලකය වෙනස් කරන අතර නව set එකක් ආපසු ලබා නොදේ.

```
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 6}

set1.difference_update(set2)

print("Modified Set1:", set1)
{1, 2, 5}

print("Set2 remains unchanged:", set2)
{3, 4, 6}
```

`intersection_update()` method

intersection_update() method is used to modify a set by retaining only the elements that are common with one or more specified sets.
intersection_update() භාවිතා කරනුයේ නිශ්චිත set එකක් හෝ කිහිපයක් සමඟ පොදු වන මූලද්‍රව්‍ය පමණක් රඳවා තබා ගනිමින් set එකක් වෙනස් කිරීමට ය.

The original set is updated in place, and no new set is created.
මුල් set එක තැනින් තැන යාවත්කාලීන කර ඇති අතර නව set එකක් නිර්මාණය නොවේ.

```
set1 = {1, 2, 3, 4, 5}
set2 = {3, 4, 6}
set3 = {4, 5, 6}

set1.intersection_update(set2, set3)

print("Modified Set1:", set1)
{4}

print("Set2 remains unchanged:", set2)
{3, 4, 6}

print("Set3 remains unchanged:", set3)
{4, 5, 6}
```

Unpacking

Unpacking refers to the process of assigning elements from an iterable to a number of variables in a single, concise statement.
Unpacking යනු තනි, සංක්ෂිප්ත ප්‍රකාශයක් තුළ iterable එකක සිට විචල්‍ය ගණනාවකට අවයව ආදේශ කිරීමේ ක්‍රියාවලියයි.

To do this the number of variables on the left of the assignment operator must match the number of values in the list.
මෙය සිදුකිරීම සඳහා ආදේශ කිරීමේ මෙහෙයවනයෙන් වම්පස පවතින විචල්‍ය ප්‍රමාණය list එක තුල පවතින අගයන් ගණනට සමාන විය යුතුය.

If the values in the list may exceed the number of variables, we can use an asterisk mark to a variable to put the exceeding values to a list inside it.
List එක තුල පවතින අවයව ගණන විචල්‍ය ගණන ඉක්මවා යන්නේ නම් ඉතිරි අවයව යම් විචල්‍යකට list එකක් සේ ඇතුලත් කිරීම සඳහා ඒ සමග asterisk ලකුණක් භාවිතා කල හැකිය.

lists

```
marks = [10, 20, 30, 40, 50, 60]

student1, student2, *other = marks

print(student1) # 10
print(student2) # 20
print(other) # [30, 40, 50, 60]

student1, *student2, other = marks

print(student1) # 10
print(student2) # [20, 30, 40, 50]
print(other) # 60
```

Tuples

```
marks = 10, 20, 30, 40, 50, 60

student1, student2, *other = marks
print(student1) # 10
print(student2) # 20
print(other) # [30, 40, 50, 60]

student1, *student2, other = marks
print(student1) # 10
print(student2)  # [20, 30, 40, 50]
print(other) # 60
```

Strings

```
s = "abc"
a, b, c = s
print(a, b, c) # a b c

s = "abcdefg"
a, b, c, *other = s
print(a) # a
print(b) # b
print(c) # c
print(other) # ['d','e','f','g']
```

Dictionary's Keys

```
d ={'key1': 1,'key2': 2,'key3': 3}
k1, k2, k3 = d
print(k1, k2, k3) # key1 key2 key3

key1, *other = d
print(key1) # key1
print(other) # ['key2', 'key3']
```

Sets

```
s = {70, 80, 90}
a, *b = s
print(a) # 70
print(b) # [80,90] # order may vary
```


ictfromabc.com

Summary of common functions and methods

Functions

Functions	Lists	Tuples	Strings	Dictionaries	Sets
`len()`	✓	✓	✓	✓	✓
`min()`	✓	✓	✗	✗	✓
`max()`	✓	✓	✗	✗	✓
`sum()`	✓	✓	✗	✗	✓
`sorted()`	✓	✓	✗	✗	✓
`reversed()`	✓	✓	✗	✗	✓

Methods

Methods	Lists	Tuples	Strings	Dictionaries	Sets
`append()`	✓	✗	✗	✗	✓
`extend()`	✓	✗	✗	✗	✓
`insert()`	✓	✗	✗	✗	✗
`remove()`	✓	✗	✗	✗	✓
`pop()`	✓	✗	✗	✓	✓
`popitem()`	✗	✗	✗	✓	✗
`clear()`	✓	✗	✗	✓	✓
`index()`	✓	✓	✓	✗	✗
`count()`	✓	✓	✓	✗	✗
`sort()`	✓	✗	✗	✗	✗
`reverse()`	✓	✗	✗	✗	✗
`keys()`	✗	✗	✗	✓	✗
`values()`	✗	✗	✗	✓	✗
`items()`	✗	✗	✗	✓	✗
`add()`	✗	✗	✗	✗	✓
`update()`	✗	✗	✗	✓	✓
`strip()`	✗	✗	✓	✗	✗
`split()`	✗	✗	✓	✗	✗
`join()`	✗	✗	✓	✗	✗
`replace()`	✗	✗	✓	✗	✗

Nesting data structures in one another
දත්ත ව්‍යුහයන් එකිනෙක තුළ කැදලි කිරීම

Lists

- **List of Lists**

```
matrix = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
```

- **List of Tuples**

```
points = [(1, 2), (3, 4), (5, 6)]
```

Here, the list `points` is mutable, but the items (tuples) within it are immutable.
මෙහිදී, `points` යන ලැයිස්තුව mutable වේ නමුත් එහි ඇති අයිතම (tuples) immutable වේ.

```
points[0] = (8,9)
print(points)
[(8, 9), (3, 4), (5, 6)]
```

```
points[0][0] = 10
# This will generate an ERROR
```

Tuples

- **Tuple of Tuples**

```
coordinates = (
      (0,1),
      (2,4),
      (4,5)
)
```

- **Tuple of Lists**

```
points = ([1, 2], [3, 4], [5, 6])
```

Here, the `points` tuple is immutable, but the items (lists) within it are mutable.
මෙහි `points` යන tuple එක immutable වේ නමුත් එහි ඇති අයිතම (ලැයිස්තු) mutable වේ.

```
points[0] = [8,9]
# This will generate an ERROR
points[0][0] = 10
print(points)
([10, 2], [3, 4], [5, 6])
```

- **Strings**

NOTE:
Strings cannot be embedded inside each other. So, it is always a one level data structure.
String එකක් තුල string එකක් ඇතුලත් කල නොහැක. එමනිසා සෑමවිටම strings එකක් එක් මට්ටමක පමණක් පවතී.

Adding other data structure into strings is worthless as it becomes a single element.
එක අගයක් බවට පත්වන බැවින් strings තුලට අනෙකුත් දත්ත ව්‍යුහ දැමීමේ පදනමක් නැත

Accessing items in nested data structures
කැදලි දත්ත ව්‍යුහයන් තුළ අයිතම වෙත ප්‍රවේශ වීම

```
R = [10, 20, [40, 50, [70, 80]]]
```

```
print(R[2])
[40, 50, [70, 80]]
```

```
print(R[2][1])
50
print(R[2][-2])
50
print(R[-1][1])
50
print(R[-1][-2])
50
```

Calculations with data structures
දත්ත ව්‍යුහයන් සමග ගණනය කිරීම්

Out of all calculations, there are only two calculations that can be done data structures.
සියලු ගණනය කිරීම් අතරින් දත්ත ව්‍යුහයන් සමග සිදුකල හැකි ගණනයකිරීම් වර්ග ඇත්තේ දෙකක් පමණි.

Addition
එකතු කිරීම

Data structures of the same type can be added
එකම වර්ගයේ දත්ත ව්‍යුහ එකට එකතු කල හැක

- **Lists**

```
x = [1, 2, 3]
y = [7, 8, 9]
print(x+y)
[1, 2, 3, 7, 8, 9]
print(y+x)
[7, 8, 9, 1, 2, 3]
print(x,y)
[1, 2, 3] [7, 8, 9]
```

- **Tuples**

```
a = 10,20,30
b = 40,50,60
print(a+b)
(10, 20, 30, 40, 50, 60)
print(b+a)
(40, 50, 60, 10, 20, 30)
```

- **Strings**

```
p='hello'
q='world'
print(p+q)
helloworld
```

This is called string concatenation.
මෙය string concatenation ලෙස හැඳින්වේ

Multiplication
ගුණ කිරීම

Data structures can only be multiplied by numbers (Integers and Boolean)
නිඛිල සහ බූලීය අගයන් මගින් පමණක් දත්ත ව්‍යුහයන් ගුණ කිරීම සිදුකල හැක.

- **Lists**

```
p = [1, 2, 3]
print(p*2)
[1, 2, 3, 1, 2, 3]
print(p*False)
[]
print(p*True)
[1, 2, 3]
```

- **Tuples**

```
k = 1,2,3
print(k*2)
(1,2,3,1,2,3)
print(k*True)
(1,2,3)
print(k*False)
()
```

- **Strings**

```
x='ha'
print(x*5)
hahahahaha
print(x*True)
ha
print(x*False)
```

NOTE:
Dictionaries and sets cannot be added together.
Dictionaries and sets එකිනෙක එකතු කල නොහැක

Dictionaries and sets cannot be multiplied.
Dictionaries and sets ගුණ කිරීමද සිදුකල නොහැක

Dictionaries and sets cannot be extended.
Dictionaries and sets දිගු කල නොහැක

Special instances of data structures
විශේෂ දත්ත ව්‍යුහ ආකාර

Singleton data structures
එක් අවයවයක් සහිත දත්ත ව්‍යුහ

- **Lists**

```
y=[1]
print(y)
[1]
type(y)
<class 'list'>
```

- **Tuples**

```
a = (1)
print(a)
1
type(a)
<class 'int'>
```

Enclosing with brackets does not create a tuple.
වරහන් භාවිතය මගින් සංවෘත කිරීමෙන් tuple එකක් නොසෑදේ

```
b = 1,
print(b)
(1,)
type(b)
<class 'tuple'>
```

Key factor is availability of more than one data even if brackets are not available
ප්‍රධාන සාධකය වන්නේ වරහන් පැවතීමකින් තොරව හෝ දත්ත එකකට වඩා පැවතීමයි

- **Strings**

```
a='b'

print(a)
b

type(a)
<class 'str'>
```

Python does not have a character data type. A single character is simply a string with a length of 1.
Python භාෂාවේ character දත්ත වර්ගයක් නොපවතින අතර මෙහිදී character එකක් යනු සරලව එක් සංකේතයක් සහිත string එකකි

- **Dictionaries**

```
b={1:'one'}

print(b)
{1:'one'}

type(b)
<class 'dict'>
```

- **Sets**

```
a={1}

print(a)
{1}

type(a)
<class 'set'>
```


ictfromabc.com

Empty data structures
හිස් දත්ත ව්‍යුහ

- **Lists**

```
x=[] # or x = list()

print(x)
[]

type(x)
<class 'list'>
```

- **Tuples**

```
c=() # or c = tuple()

print(c)
()

type(c)
<class 'tuple'>
```

There is no point of creating an empty tuple because it is not editable further. Empty tuple is the only point where we should use brackets to create a tuple.
තවදුරටත් වෙනස් කිරීමට ඇති නොහැකියාව නිසා empty tuple සෑදීමෙන් ඇති ඵලක් නැත Tuple සෑදීමේදී වරහන් භාවිත කලයුතු එකම ස්ථානය හිස් tuple වලදීය

- **Strings**

```
x='' # or x = str()
print(x)
type(x)
<class 'str'>
```

There is no point in creating an empty string because it is not editable further.
තවදුරටත් වෙනස් කිරීමට ඇති නොහැකියාව නිසා empty string සෑදීමෙන් ඇති ඵලක් නැත

- **Dictionaries**

```
a={} # or a = dict()

print(a)
{}

type(a)
<class 'dict'>
```

- **Sets**

```
q = set()
print(q)
set()

type(q)
<class 'set'>
```

Empty sets cannot be created with only curly brackets since it creates an empty dictionary.
සගල වරහන් මගින් හිස් dictionary එකක් නිර්මාණය කරන බැවින් සගල වරහන් මගින් පමණක් Empty sets සෑදිය නොහැක.

ictfromabc.com

Formatted output
සැකසූ ප්‍රතිදාන

Formatted output allows you to control the way data is displayed.
සැකසූ ප්‍රතිදානය ඔබට දත්ත දර්ශනය වන ආකාරය පාලනය කිරීමට ඉඩ සලසයි.

It's particularly useful for displaying numbers, strings, and other values in a specific format. Python offers several ways to format output.
එය විශේෂිත ආකෘතියකින් numbers, strings සහ වෙනත් අගයන් දර්ශනය කිරීමට විශේෂයෙන් ප්‍රයෝජනවත් වේ. Python ප්‍රතිදාන සකස් කිරීමට ක්‍රම කිහිපයක් ලබා දෙයි.

1. `f-strings`
An f-string is a string prefixed with 'f' or 'F'. Here, expressions inside curly braces {} can be evaluated at runtime.
F-string යනු 'f' හෝ 'F' මුලට යොදා ඇති string එකකි. මෙහිදී, සඟල වරහන් {} තුළ ඇති ප්‍රකාශන ක්‍රියාත්මක වීමේදී ඇගයීමට ලක්වේ.

```
name='John'
age=30

print(f"{name} is {age} years
old.")
John is 30 years old.

value=12.3456

print(f"Formatted Value:
{value:.2f}")
Formatted Value: 12.35
```

2. `format()` method
The `format()` method allows you to insert values into a string using placeholders, denoted by curly braces {}.
`format()` method එක මඟින් ඔබට සඟල වරහන් {} මගින් දක්වනු ලබන placeholders භාවිතා කරමින් string එකකට අගයන් ඇතුළු කිරීමට ඉඩ සලසයි.

```
name='John'
age=30
template="{0} is {1} years old."
print(template.format(name, age))
John is 30 years old.

print("{name} is {age} years
old.".format(name='John', age=30))
John is 30 years old.

value=12.3456
print("Formatted Value:
{:.2f}".format(value))
Formatted Value: 12.35
```

3. `% Operator`
This is an older method which is still used but less readable than the others. It uses % symbols as placeholders.
මෙය තවමත් භාවිතා කරන පැරණි ක්‍රමයක් වන නමුත් අනෙක් ඒවාට වඩා කියවීමට අපහසුය. එය placeholders ලෙස % සංකේත භාවිතා කරයි.

```
name='John'
age=30

print("%s is %d years old." %
(name, age))
John is 30 years old.

value = 12.3456
print("Formatted Value: %.2f" %
value)
Formatted Value: 12.35
```

%s is String format

It inserts the argument as a string. If it is not a string, it's converted to a string using the `str()` function.
එය තර්කය තන්තුවක් ලෙස ඇතුළත් කරයි. එය තන්තුවක් නොවේ නම්, `str()` ශ්‍රිතය භාවිතයෙන් තන්තුවක් බවට පරිවර්තනය වේ.

```
name = "Ravindu"
print("My name is %s" % name)
My name is Ravindu
```

%r is Repr format

It inserts the argument as a string but uses the `repr()` function to create the representation.
එය තර්කය තන්තුවක් ලෙස ඇතුළු කරන නමුත් නිරූපණය නිර්මාණය කිරීමට `repr()` ශ්‍රිතය භාවිතා කරයි.

```
value = "Ravindu\n"
print("Value is %r" % value)
Value is 'Ravindu\n'
```

%d and %i are Integer formats

It inserts the argument as a decimal integer.
එය තර්කය දශම පූර්ණ සංඛ්‍යාවක් ලෙස ඇතුළත් කරයි.

```
age=27
print("I am %d years old" % age)
I am 27 years old
```

%f is Floating-point format

It inserts the argument as a floating-point number and allows you to control the number of decimal places.
එය තර්කය ඉපිලෙන ලක්ෂ්‍ය සංඛ්‍යාවක් ලෙස ඇතුළු කරන අතර දශම ස්ථාන ගණන පාලනය කිරීමට ඔබට ඉඩ සලසයි.

```
value=12.3456
```

```
print("Value is %f" % value)
Value is 12.345600
```

```
print("Value is %.2f" % value)
Value is 12.35
```

%g is General format

It formats the number as either a floating-point or in scientific notation, depending on the value and precision.
එය අගය සහ නිරවද්‍යතාවය මත පදනම්ව, ඉපිලෙන ලක්ෂ්‍යයක් ලෙස හෝ විද්‍යාත්මක අංකනයකින් සංඛ්‍යාව සංයුති කරයි.

```
value=12.3456
print("Value is %g" % value)
Value is 12.3456
value = 0.000123456
print("Value is %g" % value)
Value is 0.000123456
```

%e is Exponential format

It inserts the argument in scientific notation.
එය විද්‍යාත්මක අංකනයෙන් තර්කය ඇතුල් කරයි.

```
value=12.3456
print("Value is %e" % value)
Value is 1.234560e+01
```

%s or %r - for string
%d or %i - for integers
%f or %g or %e - for floating point

```
print(100)
print(1200)
print(200)
print(2500)
100
1200
200
2500
```

```
print('%5d'%100)
print('%5d'%1200)
print('%5d'%200)
print('%5d'%2500)
  100
 1200
  200
 2500
```

Line joining
පේළි එක් කිරීම

1. **Explicit line joining**
   ව්‍යංග පේළි එක් කිරීම

We can break lines by using backslash.
Backslash එකකින් පේළි වෙන් කල හැක

```
print('hello\
world')
```

These two physical lines logically represent the following expression.
මෙම භෞතික පේළි දෙක මගින් පහත ප්‍රකාශණය නිරූපණය කරයි

```
print('helloworld')
```

2. **Implicit line joining**
   සෘජු පේළි එක් කිරීම

Expressions inside parentheses `{},[],()` and triple quotes """ """ , ''' ''' can be split over to more than one physical line without using backslashes.
වරහන් `{},[],()` සහ """ """ , ''' ''' උද්ධෘත පාඨ තුන තුල පවතින ප්‍රකාශණ \ backslashes භාවිතයකින් තොරව භෞතික පේළි එකකට වඩා වැඩි ගණනකට දීර්ඝ කල හැක

```
x=[1,2,
3,4,5]
```

```
print("""hello
World""")
```

3. **Using semicolons**
   තිත්කොමා භාවිතය

```
a=2
b=3
```

This can be represented in the same line as follows using semicolons to separate instructions.
මෙය, semicolon භාවිතය මගින් එකම පේළියේ පහත පරිදි නිරූපණය කල හැක

```
a=2; b=3
```

4. **Blank lines**
   හිස් පේළි

A logical line that contains only white spaces (space, tab, new-line-character) is ignored.
white spaces පමණක් පවතින තාර්කික පේළි අර්ථවින්‍යාසකය මගින් නොසලකාහරී

In the standard interactive interpreter, a blank line terminates a multi-line statement.
තවද සාමාන්‍ය අර්ථවින්‍යාසකයේදී පේළි කිහිපයක ප්‍රකාශණයක් හිස් පේළියක් මගින් අවසන් කරන්නේ යැයි සැලකේ

ictfromabc.com

Conditional Statements
කොන්දේසිමය ප්‍රකාශන

if

This statement can be used to execute a set of instructions only if a condition is true.
යම් අවශ්‍යතාවයක් තෘප්ත වන්නේ නම් පමණක් උපදෙස් සමූහයක් ක්‍රියාත්මක කිරීම සඳහා මෙම ව්‍යුහය භාවිතා කරයි

```
if(condition):
    instructions to be executed
    when the condition is true.
    කොන්දේසිය සත්‍ය විට ක්‍රියාත්මක කල
    යුතු උපදෙස්
```

Instructions will be neglected when the condition is not satisfied.
අවශ්‍යතාවය තෘප්ත නොවන විට උපදෙස් නොසලකාහැරීම මෙහිදී සිදුවේ

```
x=int(input('give marks : '))

if x>=50:
      print('passed')
      print('good')
print('goodbye')
```

Instructions which should be executed depending on the condition should be indented in the same level if there is more than one instruction.
අවශ්‍යතාවය තෘප්ත වීම මත පදනම්ව ක්‍රියාත්මක විය යුතු උපදෙස් එකකට වඩා පවතී නම් ඒවා එකම මට්ටමේ සබාැඛව කල යුතුය

The first instruction without indentation will mark the end of the structure.
Indentation නොමැතිව පවතින පළමු කේත පේලිය මගින් මෙම ව්‍යුහයේ අවසානය ලබාදේ

if else

With this statement we can provide the computer with two sets of instructions to be executed when the condition is true & false
යම් අවශ්‍යතාවයක් තෘප්ත වීම හෝ නොවීම මත පදනම්ව ක්‍රියාත්මක විය යුතු එකිනෙකට වෙනස් උපදෙස් මාලාවන් දෙකක් පරිගණකයට ලබාදීම සඳහා මෙම ව්‍යුහය භාවිතා කළ හැක

```
if (condition):
    instructions to be
    executed when the
    condition is true
    කොන්දේසිය සත්‍ය විට ක්‍රියාත්මක
    කල යුතු උපදෙස්
else:
    instructions to be
    executed when the
    condition is false
    කොන්දේසිය අසත්‍ය විට ක්‍රියාත්මක
    කල යුතු උපදෙස්
```

The computer will always choose one side of them and neglect the other.
එවිට පරිගණකය විසින් ඒවායින් එකක් තෝරා ගන්නා අතර අනෙක නොසලකාහැරීම සිදුකරයි

```
x=int(input('give marks : '))

if x>=50:
      print('passed')
      print('good')
else:
    print('failed')
    print('bad')
print('goodbye!')
```

The statements of each sector can be indented in different levels of indent.
එක් එක් කොටසට අදාල උපදෙස් වෙන වෙනම මට්ටම් දෙකක indent කිරීම සිදුකල හැක

Nested if else

Using if else structure inside another if else structure is known as nesting.
if else ව්‍යුහයක් තුල if else ව්‍යුහයක් භාවිතා කිරීම nesting ලෙස හඳුන්වයි

This can be used when there are more than two choices in the conditional statement.
අවශ්‍යතා කුලකය තුල අවශ්‍යතා දෙකකට වඩා පවතින විට මෙය භාවිතා කරයි

The checking condition should be broken into several Boolean conditions to implement this.
මෙය භාවිතය සඳහා සලකන අවශ්‍යතාවය සරල බූලීය අවශ්‍යතා කිහිපයකට වෙන් කරගැනීම සිදුකල යුතුය

There are two types of nesting.
Nesting වර්ග දෙකකි

1. if in if
2. if in else

Ex: Let's take a student assessment program.

```
x=int(input('give marks : '))

if x>=75:
    print('A')
else:
    if x>=65:
        print('B')
    else:
        if x>=55:
            print('C')
        else:
            if x>=35:
                print('S')
            else:
                print('F')
print('goodbye!')
```

```
if(condition 1):
    statement 1
else:
    if(condition 2):
        statement 2
    else:
        if(condition 3):
            statement 3
        else:
            statement 4
```

Statement 1 is executed when condition 1 is true.
1 වන කොන්දේසිය සත්‍ය වන විට 1 වන ප්‍රකාශණය ක්‍රියාත්මක වේ

Statement 2 is executed when condition 1 is false and condition 2 is true.
1 වන කොන්දේසිය අසත්‍ය වන විට සහ 2 වන කොන්දේසිය සත්‍ය වන විට 2 වන ප්‍රකාශණය ක්‍රියාත්මක වේ

Statement 3 is executed when condition 1 and condition 2 are false and condition 3 is true.
1 වන කොන්දේසිය සහ 2 වන කොන්දේසිය අසත්‍ය වන විටත් 3 වන කොන්දේසිය සත්‍ය වන විටත් 3 වන ප්‍රකාශණය ක්‍රියාත්මක වේ

Statement 4 is executed when condition 1 condition 2 and condition 3 are false.
1 වන කොන්දේසිය 2 වන කොන්දේසිය සහ 3 වන කොන්දේසිය අසත්‍ය වන විට 4 වන ප්‍රකාශණය ක්‍රියාත්මක වේ

Ex: Let's take a system login:

```
un=input('input username : ')
pw=input('input password : ')

if un=='abc':
      if pw=='xyz':
            print('logged in!')
      else:
            print('incorrect pw')
else:
   print('incorrect un or pw')
```

```
if(condition 1):
    if(condition 2):
          if(condition 3):
                statement 1
          else:
                statement 2
    else:
          statement 3
else:
    statement 4
```

Statement 1 is executed when condition1 condition 2 and condition 3 are true.
1 වන කොන්දේසිය 2 වන කොන්දේසිය සහ 3 වන කොන්දේසිය සත්‍ය වන විට 1 වන ප්‍රකාශණය ක්‍රියාත්මක වේ

Statement 2 is executed when condition 1 and condition 2 are true and condition 3 is false.
1 වන කොන්දේසිය සහ 2 වන කොන්දේසිය සත්‍ය වන විටත් 3 වන කොන්දේසිය අසත්‍ය වන විටත් 2 වන ප්‍රකාශණය ක්‍රියාත්මක වේ

Statement 3 is executed when condition 1 is true and condition 2 is false.
1 වන කොන්දේසිය සත්‍ය වන විට සහ 2 වන කොන්දේසිය අසත්‍ය වන විට 3 වන ප්‍රකාශණය ක්‍රියාත්මක වේ

Statement 4 is executed when condition 1 is false. 1 වන කොන්දේසිය අසත්‍ය වන විට 4 වන ප්‍රකාශණය ක්‍රියාත්මක වේ

if elif else

in python, the traditional nesting of if else statement is made simple as follows, such that you can check each condition one after the other.
Python භාෂාවේදී ඉහත සඳහන් සාම්ප්‍රදායික **if else** ක්‍රමය පහත පරිදි එක් අවශ්‍යතාවකට පසු අනෙක් අවශ්‍යතාව පිළිවෙලින් සලකා බැලිය හැකි පරිදි පහසු කර තිබේ.

```
x=int(input('give marks :'))

if x>=75:
      print('A pass')
elif x>=65:
      print('B pass')
elif x>=55:
      print('C pass')
elif x>=35:
      print('S pass')
else:
      print('Failed!')

print('goodbye!')
```

elif conditions are executed if the immediate previous condition was not true.
පෙර පැවති ආසන්නතම කොන්දේසිය සත්‍ය නොවූයේ නම් elif කොන්දේසිය ක්‍රියාත්මක වේ

Statements inside else are executed if all the previous conditions were not true.
පෙර පැවති සියලුම කොන්දේසි සත්‍ය නොවූයේ නම් else තුල ඇති ප්‍රකාශණ ක්‍රියාත්මක වේ

pass statement

pass statement is used to fill an if statement when there are no instructions to be given to avoid errors.
if ප්‍රකාශණයක් තුල යෙදීමට උපදෙස් නොමැති විට දෝශයක් ලැබීම වලක්වාගැනීම සඳහා pass භාවිතා කල හැකිය.

```
x=int(input('give marks :'))

if x>=75:
	pass
```

Shorthand if

If you have only one statement to execute depending on the conditional statement, you can put it on the same line as the if statement.
කොන්දේසිය මත පදනම්ව ක්‍රියාත්මක කිරීම සඳහා එක් ප්‍රකාශණයක් පමණක් පවතී නම් if ප්‍රකාශණය සමග එය එකම පේලියේ ලිවිය හැකිය

```
a=10
b=200
if a>b: print("a is bigger")
```

Shorthand if else

If you have only two statements to execute depending on the condition, one for if, and one for else, you can put it all on the same line as follows.
කොන්දේසිය මත පදනම්ව ක්‍රියාත්මක කිරීම සඳහා if ප්‍රකාශණය සඳහා එකක් සහ else ප්‍රකාශණය සඳහා එකක් බැගින් ප්‍රකාශණ දෙකක් පමණක් පවතී ඒවා පහත පරිදි එකම පේලියේ ලිවිය හැකිය

```
a=10
b=200
print("A") if a>b else print("B")
```

Repetition
පුනරුක්තිය

while loop

This structure can be used to repeat executing the same instruction sequence while a given condition is true.
දී ඇති කොන්දේසියක් සත්‍ය වනතෙක් එයට අදාල උපදෙස් අනුක්‍රමයක් ක්‍රියාත්මක කිරීම සඳහා මෙම ව්‍යුහය භාවිතා කල හැක

```
x=0
while x<5:
	print('hello')
	x=x+1
print('end')
```

```
control variable
while(condition):
    statements
    increment/decrement
```

Condition is examined before looping starts.
ලූපය ක්‍රියාත්මක වීම ආරම්භ වීමට පෙර කොන්දේසිය සත්‍ය දැයි සලකා බලයි

Content is executed only if the condition is true.
සලකන අවශ්‍යතාවය සත්‍ය වන්නේ නම් පමණක් අන්තර්ගතය ක්‍රියාත්මක වේ

If the condition is false at the beginning itself, the content won't be executed.
ආරම්භයේදීම සලකන අවශ්‍යතාවය අසත්‍ය වන්නේ නම් අන්තර්ගතය ක්‍රියාත්මක නොවේ

Repetition ends when the condition becomes false.
සලකන අවශ්‍යතාවය තවදුරටත් සත්‍ය නොවන බවට පත්වූ පසු පුණර්කරණය අවසාන වේ

If the condition doesn't become false anyway, it will be an infinite loop.සලකන කොන්දේසිය කිසිවිටෙකත් අසත්‍ය නොවන්නේ නම් එය අනන්ත වාරයක් ක්‍රියාත්මක වනු ඇත
NOTE:

If the content of a loop is a constant, number of times that a loop is going to execute can be only changed by
ලූපයක අන්තර්ගතය නියත වන්නේ නම්, ලූපයක් ක්‍රියාත්මක වන වාර ගණන වෙනස් කල හැක්කේ,

1. changing the value of the control variable
   පාලන විචල්‍යයේ ආරම්භක අගය මගින්

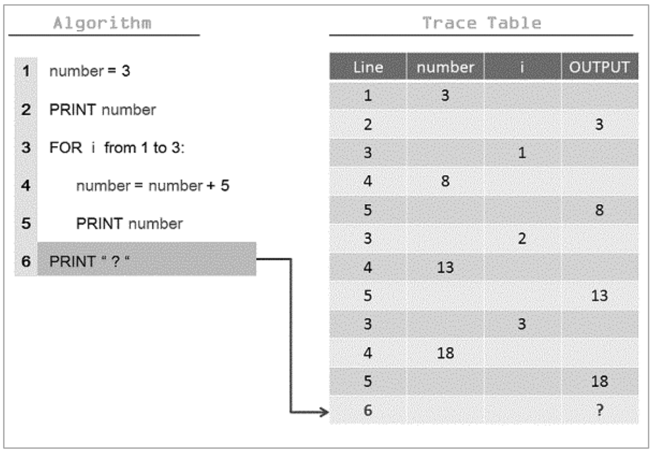
2. changing the condition
   කොන්දේසිය වෙනස් කිරීම මගින්

3. changing the increment / decrement
   ඉහල යාම හෝ පහල යාම වෙනස් කිරීම මගින්

Hand Tracing a loop

Manually simulating each iteration of the loop to understand its behavior and track variable changes step by step is known as hand tracing.
ලූපයේ සෑම පුනරාවර්තනයකදීම එහි හැසිරීම තේරුම් ගැනීමට සහ පියවරෙන් පියවර විචල්‍ය වෙනස්කම් නිරීක්ෂණය කිරීමට එම ලූපය අතින් අනුකරණය කිරීම hand tracing ලෙස හැඳින්වේ.

A trace table can be used for this.
මේ සඳහා trace table එකක් භාවිතා කළ හැකිය.


Algorithm

```
1 number = 3
2 PRINT number
3 FOR i from 1 to 3:
4     number = number + 5
5     PRINT number
6 PRINT " ? "
```

Trace Table

Line	number	i	OUTPUT
1	3		
2			3
3		1	
4	8		
5			8
3		2	
4	13		
5			13
3		3	
4	18		
5			18
6			?

Examples

Difficulty: level 1

A basic countdown loop that decreases a number from 5 to 1.
සංඛ්‍යාවක් 5 සිට 1 දක්වා අඩු කරන මූලික ගණනය කිරීමේ ලූපයක්.

```
count = 5
while count > 0:
    print(count)
    count = count - 1
print("Blast off!")
```

This loop prints numbers from 5 down to 1, then exits when `count` reaches 0.
මෙම ලූපය 5 සිට 1 දක්වා සංඛ්‍යා මුද්‍රණය කරයි, පසුව `count` එක 0ට ළඟා වූ විට ලූපයෙන් ඉවත්වෙයි.

Trace table

Initial values → count=5

rep	print(count)	count-=1
1	5	4
2	4	3
3	3	2
4	2	1
5	1	0

Output will be,

```
5
4
3
2
1
Blast off!
```

Difficulty: level 2

A loop that adds user input to a running total until it reaches or exceeds a target sum.
ඉලක්ක ගත එකතුවකට ළඟා වන හෝ ඉක්මවන තෙක් ධාවන වන එකතුවකට පරිශීලක ආදානය එක් කරන ලූපයක්.

```
ts = 20 # target_sum
cs = 0  # current_sum

while cs < ts:
    promt = "Enter a number"
    num = int(input(prompt))
    cs = cs + num

    print(f"Current sum: {cs}")

print("Target reached!")
```

If the inputs are `5, 7, 10`
ආදාන 5, 7, 10 නම්,

Trace table

Initial values → ts = 20
→ cs = 0

rep	num	cs+=num	print()
1	5	5	Current_sum: 5
2	7	12	Current_sum: 12
3	10	22	Current_sum: 22

The output will be
ප්‍රතිදානය පහත පරිදි වනු ඇත.

```
Current_sum: 5
Current_sum: 12
Current_sum: 22
Target reached!
```

Difficulty: level 3

A loop that generates Fibonacci numbers until the numbers exceed a given limit.
සංඛ්‍යා දී ඇති සීමාව ඉක්මවා යන තෙක් Fibonacci සංඛ්‍යා ජනනය කරන ලූපයක්.

```
limit = 50
a, b = 0, 1

while a <= limit:
    print(a, end=" ")
    a, b = b, a + b
```

The Fibonacci sequence is a sequence in which each number is the sum of the two preceding ones. The first two numbers are predefined as 0 and 1.
Fibonacci අනුක්‍රමය යනු එක් එක් සංඛ්‍යාව පෙර ඇති සංඛ්‍යා දෙකේ එකතුව වන අනුක්‍රමයකි. පළමු සංඛ්‍යා දෙක 0 සහ 1 ලෙස පූර්ව නිර්වචනය කර ඇත

Trace table
Initial values → limit = 50
→ a = 0
→ b = 1

rep	print(a)	a=b	b=a+b
1	0	1	1
2	1	1	2
3	1	2	3
4	2	3	5
5	3	5	8
6	5	8	13
7	8	13	21
8	13	21	34
9	21	34	55
10	34	55	89

The output will be

```
0 1 1 2 3 5 8 13 21 34
```

Difficulty: level 4

A program to check if a word is a palindrome. A palindrome reads the same forwards and backwards (like "racecar" or "madam"). The code will compare characters from the start and end of the word, moving towards the center.
වචනයක් palindrome එකක් ද යන්න පරීක්ෂා කිරීමට වැඩසටහනක්. palindrome එකක් ඉදිරියට සහ පසුපසට කියවෙන්නේ එකම ආකාරයටය ("racecar" හෝ "madam" වැනි). කේතය වචනයේ ආරම්භයේ සහ අවසානයේ සිට කේන්ද්‍රය දෙසට ගමන් කරමින් අක්ෂර සංසන්දනය කරයි.

```
word = "racecar"  # Input word
is_palindrome = True  # Flag to indicate if the word is a palindrome
start = 0
end = len(word) - 1

while start < end:
    if word[start] != word[end]:  # Compare characters from start and end
        is_palindrome = False
        break
    start += 1  # Move start index forward
    end -= 1    # Move end index backward

if is_palindrome:
    print(f"'{word}' is a palindrome.")
else:
    print(f"'{word}' is not a palindrome.")
```

Trace table

Initial values → is_palindrome = True
→ start = 0
→ end = len(word) – 1 = 6

rep	if word[start] != word[end]	start+=1	end-=1
1	-	1	5
2	-	2	4
3	-	3	3

```
racecar is a palindrome
```

For a non-palindrome word – "London"

rep	if word[start]!=word[end]	start+=1	end-=1
1	is_palindrome=False; break		

```
London is not a palindrome
```


ictfromabc.com

Difficulty: level 5

A loop that finds and prints all prime numbers up to a specified limit.
නිශ්චිත සීමාවක් දක්වා සියලුම ප්‍රථමක සංඛ්‍යා සොයාගෙන මුද්‍රණය කරන ලූපයක්.

```
limit = 10
num = 2  # Starting from the first prime number

while num <= limit:
    is_prime = True
    divisor = 2
    while divisor <= int(num ** 0.5):  # Only check up to square root of num
        if num % divisor == 0:
            is_prime = False
            break
        divisor += 1
    if is_prime:
        print(num, end=" ")
    num += 1
```

Trace table

Initial values → limit = 10
→ num = 2

Outer loop			**Inner loop**			**Outer loop continued**	
rep	is_prime	divisor	rep	if num%divisor==0	divisor+=1	if is_prime print(num)	num+=1
1	True	2	-	-	-	2	3
2	True	2	-	-	-	3	4
3	True	2	1	is_prime = False	-	-	5
4	True	2	1	-	3	5	6
5	True	2	1	is_prime = False	-	-	7
6	True	2	1	-	3	7	8
7	True	2	1	is_prime = False	-	-	9
8	True	2	1	-	3		
			2	is_prime = False	-	-	10
9	True	2	1	is_prime = False	-	-	11

The output will be 2 3 5 7
ප්‍රතිදානය 2 3 5 7

range() function

Used to generate a list of integers in a specific range instantly.
නිශ්චිත පරාසයකට අයත් වන නිඛිල ලැයිස්තුවක් ලබාගැනීම සඳහා භාවිතා කරයි

Syntax

```
range(start, stop, step)
```

start (optional): The starting integer of the sequence. The default is 0.
අනුක්‍රමයේ ආරම්භක පූර්ණ සංඛ්‍යාව. පෙරනිමිය 0 වේ.

stop (required): The stopping integer of the sequence. The range goes up to (but does not include) this value.
අනුක්‍රමයේ නැවතුම් පූර්ණ සංඛ්‍යාව. පරාසය මෙම අගය දක්වා ඉහළ යයි (නමුත් ඇතුළත් නොවේ).

step (optional): Increment or decrement between each integer in the sequence. The default is 1.
අනුපිළිවෙලෙහි එක් එක් නිඛිල අතර වැඩි කිරීම හෝ අඩු කිරීම. පෙරනිමිය +1 වේ.

can be used in 3 ways.
ආකාර තුනකට භාවිතා කල හැක

range(stop)
range(start, stop)
range(start, stop ,[step])

All arguments should be integers.
සෑම අවයවයක්ම නිඛිලයක් විය යුතුය

```
list(range(8))
[0, 1, 2, 3, 4, 5, 6, 7]
```

```
list(range(-2,5))
[-2, -1, 0, 1 , 2, 3, 4]
```

```
list(range(2,10,2))
[2, 4, 6, 8]
```

```
list(range(8,3,-1))
[8,7,6,5,4]
```

```
list(range(8,3,-2))
[8, 6, 4]
```

```
list(range(-8))
[ ]
```

Setting step to 0 gives an error.
ඉහලයාම 0 වූ විට දෝශයක් ලැබේ

```
list(range(1, 5, 0))
# Raises ValueError: range() arg 3
# must not be zero
```

The value of stop should be greater than value of start if no step is given or step is positive.
පියවර අගයක් නැත්නම් හෝ පියවර අගය ධන නම් ආරම්භක අගයට වඩා අවසාන අගය විශාල විය යුතුය

The value of stop should be less than value of start if step value is negative.
පියවර අගය සෘණ නම් ආරම්භක අගයට වඩා අවසාන අගය කුඩා විය යුතුය

NOTE:
Converting `range()` to Other Sequence Types in Python
පයිතන් හි `range()` වෙනත් අනුක්‍රමික වර්ග වලට පරිවර්තනය කිරීම

Conversion to Tuple

```
tuple(range(8))
(0, 1, 2, 3, 4, 5, 6, 7)
```

Conversion to Set

```
set(range(8))
{0, 1, 2, 3, 4, 5, 6, 7}
```

for loop

here, a condition depending on a data sequence is introduced to the control variable.
මෙහිදී පාලන විචල්‍යට දත්ත අනුක්‍රමයක් මත පදනම්වූ කොන්දේසියක් ලබාදේ

included instructions are executed exactly the number of times equivalent to the number of elements in the conditional data sequence.
අන්තර්ගත උපදෙස් සමූහය ක්‍රියාත්මක වන්නේ හරියටම කොන්දේසියෙහි ඇති දත්ත අනුක්‍රමයේ අවයව ගණනට සමාන වාර ප්‍රමාණයකි

this loop is better to be used when we know the amount it should repeat in the first place.
පුණර්කරණය විය යුතු වාර ගණන දන්නා විට මෙම ව්‍යුහය භාවිතා කිරීම වඩාත් සුදුසුය

```
for variable in (sequence):
      statements
```

The default increment in for loop is 1
ව්‍යුහයේ සම්මත ඉහලයාමේ අගය 1 ය

Conditional data sequence can be obtained by
කොන්දේසියේ දත්ත අනුක්‍රමය ලබාගත හැකි ක්‍රම

1. using `range()` function
2. using a string
3. using a list
4. using a tuple
5. using a dictionary
6. using a file

range function

```
for a in range(0,5,1):
      print(a)
```

```
0
1
2
3
4
```

```
for b in range(5,0,-1):
      print(b)
```

```
5
4
3
2
1
```

with strings

```
for c in 'kurunegala':
      print(c)
```

```
k
u
r
u
n
e
g
a
l
a
```

with lists

```
for x in [1, 2, 3, 4, 5]:
      print(x-1, end',')
```

```
0,1,2,3,4
```


ictfromabc.com

with tuples

```
for y in (1, 2, 3, 4, 5):
      print ('hello')

hello
hello
hello
hello
hello
```

with dictionaries

```
a={'name':'Ravindu','age':24}
for z in a:
      print(z)

name
age

for z in a.values():
      print(z)

Ravindu
24

for z in a.keys():
      print(z)

name
age
```

with files

```
file = open("example.txt", "r")

for line in file:
        print(line.strip())
```

Using else statement in for loop
for පුණර්කරණ ව්‍යුහය තුල else භාවිතය

If we assign an else statement to a for loop, because the condition in the for loop is always true, the else statement will execute after the loop is finished.
for loop එකක් සෑමවිටම සත්‍ය වන බැවින් එය සෑමවිටම ක්‍රියාත්මක වන අතර, else උපදෙසක් යෙදූ විට for loop එක ක්‍රියාත්මකවී අවසන් වූ පසු else කොටස ක්‍රියාත්මක වේ.

```
for x in range(6):
  print(x)
else:
  print("Goodbye!")

0
1
2
3
4
5
Goodbye!
```

Using else Statement with while loop
while පුණර්කරණ ව්‍යුහය තුල else භාවිතය

We can run a block of code once when the condition of a while loop no longer is true:
while ව්‍යුහයක කොන්දේසිය තවදුරටත් සත්‍ය නොවන විට ක්‍රියාත්මක වීම සඳහා else තුල ප්‍රකාශණ ලබාදිය හැකිය

```
i=1
while i<4:
      print(i)
      i=i+1
else:
    print("no longer less than 4")
1
2
3
no longer less than 4
```

break statement

This statement is used to stop a repetition control structure in the middle of its functioning, depending on whether a certain condition is fulfilled.
ක්‍රියාත්මක වීම අතරතුරදී යම්කිසි කොන්දේසියක් මත පදනම්ව පුණර්කරන පාලන ව්‍යුහයකින් නැවැත්වීම සඳහා මෙම විධානය යොදාගනී

Without the break statement:

```
i = 0
while i < 5:
    if i == 3:
        print("Encountered 3")
    print(f"Processing {i}")
    i = i + 1
print("Loop ended without break")
```

Trace table

Initial values → i = 0

rep	i	i == 3	print()
1	0	False	Processing 0
2	1	False	Processing 1
3	2	False	Processing 2
4	3	True	Encountered 3 + Processing 3
5	4	False	Processing 4

```
Processing 0
Processing 1
Processing 2
Encountered 3
Processing 3
Processing 4
Loop ended without break
```

With the break statement:

```
i = 0
while i < 5:
    if i == 3:
        print("Encountered 3")
        print("Terminated...")
        break
    print(f"Processing {i}")
    i = i + 1
print("Loop ended with break")
```

Trace table

Initial values → i = 0

rep	i	i == 3	print()
1	0	False	Processing 0
2	1	False	Processing 1
3	2	False	Processing 2
4	3	True	Encountered 3 Terminated...

```
Processing 0
Processing 1
Processing 2
Encountered 3
Terminated...
Loop ended with break
```

continue statement.

This statement is used to stop the current repeating cycle of a loop from somewhere and restart from the next repeating cycle depending on whether a condition is fulfilled.
පුණර්කරන පාලන ව්‍යුහයක පුණර්කරන අවස්තාවක් යම් කොන්දේසියක් මත පදනම්ව අතරමැද ස්ථානයකින් මගහැරයමින් ඊලඟ වාරයේ සිට ක්‍රියාත්මක කරවීම සඳහා මෙම විධානය යොදාගනී

Without the continue statement:

```
i = 0
while i < 5:
    if i == 3:
        print("Encountered 3,
        but processing anyway")
    print(f"Processing {i}")
    i += 1
print("Loop ended without continue")
```

Trace table

Initial values → i = 0

rep	i	i == 3	print()	i += 1
1	0	False	Processing 0	1
2	1	False	Processing 1	2
3	2	False	Processing 2	3
4	3	True	Encountered 3, but processing anyway + Processing 3	4
5	4	False	Processing 4	5

```
Processing 0
Processing 1
Processing 2
Encountered 3, but processing anyway
Processing 3
Processing 4
Loop ended without continue
```

With the continue statement:

With continue

```
i = 0
while i < 5:
    if i == 3:
        print("Skipping 3")
        i += 1
        continue
    print(f"Processing {i}")
    i += 1
print("Loop ended with continue")
```

Trace table

Initial values → i = 0

rep	i	i == 3	print()	i += 1
1	0	False	Processing 0	1
2	1	False	Processing 1	2
3	2	False	Processing 2	3
4	3	True	Skipping 3	4
5	4	False	Processing 4	5

```
Processing 0
Processing 1
Processing 2
Skipping 3
Processing 4
Loop ended with continue
```

NOTE:

In a while loop, the control variable (i) must be updated before using continue, or the loop will run indefinitely.

while ලූපයක, continue භාවිතා කිරීමට පෙර පාලන විචල්‍යය (i) අනිවාර්යයෙන් යාවත්කාලීන කළ යුතුය, නැතහොත් ලූපය නොනැවති ක්‍රියාත්මක වේ.

This happens because the continue statement skips the rest of the current iteration, including the code where the variable would typically be incremented or decremented.

මෙය සිදු වන්නේ විචල්‍යයේ අගය සාමාන්‍යයෙන් වැඩි කරන හෝ අඩු කරන කේතය ඇතුළුව, වත්මන් පුනරාවර්තනයේ ඉතිරි කේත කොටස් සියල්ල continue ප්‍රකාශය මගින් මග හරින බැවිනි.

Incorrect Placement of Increment

Increment එකෙහි වැරදි ස්ථානගත කිරීම

```
i = 0
while i < 5:
    if i == 3:
        continue
    print(f"Processing {i}")
    i += 1  # This line is skipped
```

Correct Placement to Avoid Infinite Loop

නොනවතින ලූපය වළක්වා ගැනීම සඳහා නිවැරදි ස්ථානගත කිරීම

```
i = 0
while i < 5:
    if i == 3:
        i += 1
      # Increment before continue
        continue
    print(f"Processing {i}")
    i += 1
```

Nested Loops

එකක් තුල එකක් පවතින පුණර්කරණ ව්‍යුහ

The inner loop will be executed one time for each iteration of the outer loop.

බාහිර පුණර්කරණ ව්‍යුහය ක්‍රියාත්මක වන සෑම අවස්ථාවකටම එක බැගින් අභ්‍යන්තර පුණර්කරණ ව්‍යුහය ක්‍රියාත්මක වීම මෙහිදී සිදුවේ

```
x=5
for i in range(0, x):
      for j in range(0, i+1):
            print("*",end="")
      print("\r")
```

```
*
**
***
****
*****
```

```
rows = 5

for i in range(1, rows + 1):
    for j in range(rows - i):
        print(" ", end="")

    for k in range(2 * i - 1):
        print("*", end="")
    print()

for i in range(rows - 1, 0, -1):
    for j in range(rows - i):
        print(" ", end="")

    for k in range(2 * i - 1):
        print("*", end="")
    print()
```

```
    *
   ***
  *****
 *******
*********
 *******
  *****
   ***
    *
```

Examples

Difficulty: level 1

A loop that calculates the sum of all digits in a given integer.
දී ඇති පූර්ණ සංඛ්‍යාවක සියලුම ඉලක්කම්වල එකතුව ගණනය කරන ලූපයක්.

```
number = 12345
sum_of_digits = 0

for digit in str(number):
    sum_of_digits += int(digit)

print(f"The sum of digits in
{number} is {sum_of_digits}")
```

Trace table

Initial values → number = 12345
→ sum_of_digits = 0

str(number) = '12345'

rep	digit	sum_of_digits+=int(digit)
1	'1'	1
2	'2'	3
3	'3'	6
4	'4'	10
5	'5'	15

```
The sum of digits in 12345 is 15
```

Difficulty: level 2

A loop that finds and prints all divisors of a given number.
දී ඇති සංඛ්‍යාවක් බෙදෙන සියලුම අගයන් සොයාගෙන මුද්‍රණය කරන ලූපයක්.

```
num = 10
divisors = []

for i in range(1, num + 1):
    if num % i == 0:
        divisors.append(i)

print(f"Divisors of {num}:",
divisors)
```

Trace table

Initial values → num = 10
→ divisors = []

range(1,num+1) = [1,2,3,4,5,6,7,8,9,10]

rep	i	if num%i==0	divisors
1	1	True	[1]
2	2	True	[1,2]
3	3	False	[1,2]
4	4	False	[1,2]
5	5	True	[1,2,5]
6	6	False	[1,2,5]
7	7	False	[1,2,5]
8	8	False	[1,2,5]
9	9	False	[1,2,5]
10	10	True	[1,2,5,10]

```
[1, 2, 5, 10]
```

Difficulty: level 3

A loop that counts the number of vowels in a given string
දී ඇති තන්තුවක ස්වර අක්ෂර ගණන ගණනය කරන ලූපයක්

```
text = "Hi! How are you?"

vowel_count = 0

vowels = {'a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', 'O', 'U'}

for char in text:
    # Check if the character is a vowel
    if char in vowels:
        vowel_count += 1

print("Number of vowels:", vowel_count)
```

Trace table

Initial values → vowel_count = 0

rep	char	if char in vowels	vowel_count+=1
1	H	False	-
2	i	True	1
3	!	False	-
4	" "	False	-
5	H	False	-
6	o	True	2
7	w	False	-
8	" "	False	-
9	a	True	3
10	r	False	-
11	e	True	4
12	" "	False	-
13	y	False	-
14	o	True	5
15	u	True	6
16	?	False	-

```
Number of vowels: 6
```

Difficulty: level 4

A loop that counts how often each character appears in a string, ignoring spaces.
හිස්තැන් නොසලකා හරිමින් එක් එක් අනුලක්ෂණය තන්තුවක කොපමණ වාරයක් දිස් වේ දැයි ගණනය කරන ලූපයක්.

```
text = "hello world"
count = {}

for char in text:
    if char != " ":
        if char in count:
            count[char] += 1
        else:
            count[char] = 1

print("Character frequencies:", count)
```

Trace table

rep	char	if char !=" "	if char in count	count[char]+=1	count[char]=1
1	'h'	True	False	-	{'h':1}
2	'e'	True	False	-	{'h':1,'e':1}
3	'l'	True	False	-	{'h':1,'e':1,'l':1}
4	'l'	True	True	{'h':1,'e':1,'l':2}	-
5	'o'	True	False	-	{'h':1,'e':1,'l':2,'o':1}
6	' '	False			
7	'w'	True	False	-	{'h':1,'e':1,'l':2,'o':1','w':1}
8	'o'	True	True	{'h':1,'e':1,'l':2,'o':2, 'w':1}	
9	'r'	True	False	-	{'h':1,'e':1,'l':2, 'o':2,'w':1, 'r':1}
10	'l'	True	True	{'h':1,'e':1,'l':3, 'o':2,'w':1,'r':1}	-
11	'd'	True	False	-	{'h':1,'e':1,'l':3,'o':2,'w':1,r ':1,'d':1}

```
Character frequencies: {'h': 1, 'e': 1, 'l': 3, 'o': 2, 'w': 1, 'r': 1,
'd': 1}
```

Difficulty: level 5

Nested loops for element-wise addition of two matrices.

න්‍යාස දෙකක අවයව අනුව එකතු කිරීම සඳහා වූ කැදලි ලූප.

```
matrix1 = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
matrix2 = [
    [9, 8, 7],
    [6, 5, 4],
    [3, 2, 1]
]

result = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]

for i in range(len(matrix1)):
    for j in range(len(matrix1[0])):
        result[i][j] = matrix1[i][j] + matrix2[i][j]
for row in result:
    print(row)
```

Trace table

range(len(matrix1)) = [0,1,2]

range(len(matrix1[0])) = [0,1,2]

Outer loop		**Inner loop**		
rep	i	rep	j	result[i][j] = matrix1[i][j] + matrix2[i][j]
1	0	1	0	[[10, 0, 0], [0, 0, 0], [0, 0, 0]]
		2	1	[[10, 10, 0], [0, 0, 0], [0, 0, 0]]
		3	2	[[10, 10,10], [0, 0, 0], [0, 0, 0]]
2	1	1	0	[[10, 10,10], [10, 0, 0], [0, 0, 0]]
		2	1	[[10, 10,10], [10, 10, 0], [0, 0, 0]]
		3	2	[[10, 10,10], [10, 10, 10], [0, 0, 0]]
3	2	1	0	[[10, 10,10], [10, 10, 10], [10, 0, 0]]
		2	1	[[10, 10,10], [10, 10, 10], [10, 10, 0]]
		3	2	[[10, 10,10], [10, 10, 10], [10, 10, 10]]

```
[10, 10,10]
[10, 10,10]
[10, 10,10]
```

List comprehension

List comprehension is a concise way to create lists based on existing lists or other iterable objects.
List comprehension යනු පවතින ලැයිස්තු හෝ වෙනත් පුනරාවර්තනය කළ හැකි වස්තු මත පදනම්ව ලැයිස්තු සෑදීමේ සංක්ෂිප්ත ක්‍රමයකි.

Syntax

```
[exp. for item in iterable if cond.]
```

Examples.

Square each number in a list.
ලැයිස්තුවක සෑම සංඛ්‍යාවක්ම වර්ග කිරීම.

```
nums = [1, 2, 3, 4, 5]
squares = [i ** 2 for i in nums]
print(squares)
```

```
[1, 4, 9, 16, 25]
```

This iterates over nums, calculates `i ** 2` for each `i`, and appends the result to `squares`
මෙය `nums` මත පුනරාවර්තනය වේ, එක් එක් `i` සඳහා `i ** 2` ගණනය කර ප්‍රතිඵලය `squares` හි අගට එකතු කරයි.

Square each even number in a given range.
දී ඇති පරාසයක එක් එක් ඉරට්ටේ සංඛ්‍යාව වර්ග කරයි.

```
S = [i**2 for i in range(1,11) if
i % 2 == 0]
print(S)
```

```
[4, 16, 36, 64, 100]
```

Here, `if num % 2 == 0` adds a condition to only square even numbers.
මෙහි, `if num % 2 == 0` මගින් ඉරට්ටේ සංඛ්‍යා පමණක් වර්ග කිරීමට කොන්දේසියක් එක් කරයි.

Creating a list of products for each pair of elements from two lists.
ලැයිස්තු දෙකක එක් එක් අවයව යුගලය සඳහා වන ගුණිත ලැයිස්තුවක් නිර්මාණය කිරීම.

```
list1 = [1, 2, 3]
list2 = [4, 5]

products = [x * y for x in list1
for y in list2]

print(products)
```

```
[4, 5, 8, 10, 12, 15]
```

Each `x` from `list1` is paired with each y from `list2`, resulting in the following pairs:
`list1` හි ඇති සෑම `x` එකක්ම `list2` හී ඇති සෑම `y` එකක් සමඟම යුගල වන අතර, එහි ප්‍රතිඵලයක් ලෙස පහත යුගල ලැබේ:

1 with 4, 1 with 5
2 with 4, 2 with 5
3 with 4, 3 with 5

In the list comprehension, `x * y` is the expression being evaluated.
List comprehension තුළ, `x * y` යනු ඇගයීමට ලක් කෙරෙන ප්‍රකාශනයයි.

For each pair `(x, y)`, `x * y` calculates their product and adds it to the `products` list.
`(x, y)` එක් එක් යුගල සඳහා, `x * y` මගින් ගුණිතය ගණනය කර ලැයිස්තුවට එකතු කරයි.

```
1 * 4 = 4
1 * 5 = 5
2 * 4 = 8
2 * 5 = 10
3 * 4 = 12
3 * 5 = 15
```

So, `products = [4, 5, 8, 10, 12, 15]`

Functions
ශ්‍රිත

A block of organized and reusable code that performs a single, related action is known as a function.
එක් නිශ්චිත කාර්යයක් සිදුකිරීම සඳහා භාවිතා කරන සංවිධානාත්මක සහ නැවත භාවිත කල හැකි කේත කොටසක් ශ්‍රිතයක් ලෙස හැඳින්වේ.

Functions can be used when we need to do the same task repeatedly in a program.
ක්‍රමලේඛණයක් තුල එකම කාර්‍ය්‍ය නැවත නැවත සිදු කිරීමට ඇති විටදී ශ්‍රිත භාවිතා කල හැක .

It avoids the difficulty of writing the same segment of code again and again.
එකම කේතය නැවත නැවත ලිවීමේ අසීරුතාවයෙන් වැලකීම සඳහා උපකාරී වේ

It saves time and prevents errors.
එය කාලය ඉතිරි කරන අතර දෝශ අවම කරයි

There are pre-defined functions and functions defined by us.
පෙර අර්ථ දක්වන ලද ශ්‍රිත පවතින අතර අප විසින්ද අවශ්‍ය පරිදි ශ්‍රිත අර්ථ දක්වා ගැනීමට පහසුකම් පවතී

A function only runs when it is called.
ශ්‍රිතයක් ක්‍රියාත්මක වන්නේ එය කැඳවූ විටදී පමණි.

Even a which is function defined by us in a program only runs when it is called.
අපි විසින් වැඩසටහනක් තුළ නිර්වචනය කරන ලද ශ්‍රිතයක් පවා ක්‍රියාත්මක වන්නේ එය කැදවූ විට පමණි.

NOTE:

parameters vs arguments

The terms "parameters" and "arguments" are often used interchangeably but they do have different meanings in a function.
parameters සහ arguments යන පද දෙක බොහෝ අවස්ථා වලදී හුවමාරු කරමින් භාවිත වුවත් ඒවාට අදාල නිශ්චිත තේරුම් පවතී.

Parameters are the variables that are defined when creating the original function.
Parameters යනු ශ්‍රිත අර්ථදැක්වීමේදී ශ්‍රිත තුලට අගයන් රැගෙන යාම සඳහා නිර්මාණය කරන විචල්‍යයන්ය.

They act as placeholders waiting for the "input" to the function.
ඒවා ආදානයක් ලැබෙන තෙක් placeholders එකක් ලෙස හැසිරේ.

Arguments are the actual values or that are passed into a function when it is called.
Arguments යනු ශ්‍රිතයක් කැඳවන විට ඒ තුලට යවන්නාවූ සත්‍ය අගයන්ය.

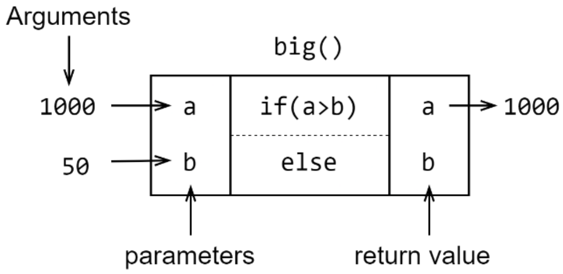
They act as assigned values for the parameters of a function at an instance.
නැතහොත් එක් එක් අවස්ථාවේදී parameters සඳහා ලබාදී ඇති අගයන්ය.

ictfromabc.com

User defined functions
පරිශීලකයා විසින් අර්ථදක්වන ලද ශ්‍රිත

keyword def is used to define functions.
ශ්‍රිත අර්ථදැක්වීමට def යන මූලපදය භාවිතා කරයි

```
def FunctionName(parameters):
      statements
      return value
FuntionName(Arguments)
```



Variables defined related with a function are known as parameters.
ශ්‍රිතයකට අදාලව අර්ථදක්වන විචල්‍යයන් පරාමිතීන් ලෙස හැඳින්වේ

We can add any number of parameters for a function. Parameters are used to pass values as input to a function.
ශ්‍රිතයකට පරාමිතීන් ඕනෑතරම් ප්‍රමාණයක් ලබාදිය හැක ශ්‍රිතයකට ආදාන ලෙස අගයන් ලබාදීම සඳහා පරාමිතීන් භාවිතා කරයි

Functions are called by using the function name followed by parenthesis.
ශ්‍රිතයක නම සහ එයට පසුව යෙදෙන වරහන් යුගලයකින් ශ්‍රිතයක් කැඳවනු ලබයි

A function can return data as a result.
ශ්‍රිතයක් මගින් ප්‍රතිඵල ලෙස දත්ත return කෙරේ.

User defined functions can be of 4 types.
පරිශීලකයා විසින් අර්ථදැක්විය හැකි ශ්‍රිත 4කි

1. With parameters with return
2. Without parameters without return
3. Without parameters with return
4. With parameters without retur

1. Functions with parameters and return
පරාමිතීන් සහ ප්‍රත්‍යාගමන අගයන් ඇති ශ්‍රිත

```
def big(x,y):
      if x>y:
            return x
      else:
            return y

print(big(5,10))
10
```

2. Functions with no parameters and no return
පරාමිතීන් සහ ප්‍රත්‍යාගමන අගයන් නැති ශ්‍රිත

```
def logo():
      print('******************')
      print('****ictfromabc****')
      print('******************')

logo ()
******************
****ictfromabc****
******************
```

3. Functions with no parameters but with return
පරාමිතීන් නැති නමුත් ප්‍රත්‍යාගමන අගයන් ඇති ශ්‍රිත

```
def greet():
      greeting = "Hello World"
      return greeting
print(greet())
```

4. Functions with parameters but without return
පරාමිතීන් ඇති නමුත් ප්‍රත්‍යාගමන අගයන් නැති ශ්‍රිත

```
def greet(name):
      print("Hello ", name)
greet("John")
```

Arbitrary Arguments, *args

When we don't know how many arguments will be passed into the function, we add an * before the parameter name in the function definition.
ශ්‍රිතයක් තුලට arguments කොපමණ ගණනක් ආදානය කරන්නේදැයි නොදන්නා විටදී parameters ඉදිරියෙන් * ලකුණ භාවිතා කරයි.

```
def addone(*parameter):
    for i in parameter:
        print(i+1)

addone(1, False, 3.14, True)
2
1
4.140000000000001
2
```

Default parameter values
සම්මත පරාමිති අගයන්

Default values for one or several parameters of the function can be assigned.
ශ්‍රිතයක පරාමිතීන් එකකට හෝ කිහිපයකට සම්මත අගයන් ලබා දිය හැකිය

Those values will be automatically used if any other values for them are not manually given.
පරාමිතීන් සඳහා හස්තීයව අගයන් ලබා නොදුන් විට එම අගයන් ස්වයංක්‍රීයව භාවිතා වීම සිදුවේ

```
def town(x, y='Kurunegala'):
      print(x, 'is from', y)

town('Ravindu')
Ravindu is from Kurunegala

town('Tharindu', 'Colombo')
Tharindu is from Colombo
```

Keyword arguments
යතුරුපද පරාමිතීන්

These can be used to pass values specifically to a parameter.
යම් නිශ්චිත පරාමිතියකට අගයක් ලබාදීම සඳහා මෙම ක්‍රමය භාවිතා කරයි

For this, the user must have a clear understanding about the function
මේ සඳහා පරිශීලකයාට ශ්‍රිතයේ අර්ථදැක්වීම සම්බන්ධව පැහැදිලි අදහසක් තිබිය යුතුය

```
def total(a, b=100, c=200):
      print(a+b+c)

total(100)
400

total(100, 200)
500

total(100, 200, 300)
600
```

In the above example, arguments are in the exact order of the parameters.
ඉහත උදාහරණයේ දී, පරාමිතිවල අනුපිළිවෙලටම arguments ද පවතී.

If we want to skip a particular parameter when passing arguments, it can be done as follows.
Arguments ලබා දීමේදී යම් පරාමිතියක් මඟ හැරීමට අවශ්‍ය නම්, එය පහත පරිදි කළ හැකිය.

```
total(a=200, c=400)
700

total(b=300, a=100)
600

total(500,400)
1100
```


ictfromabc.com

Parameter passing methods

1. Passing by Value
When a function receives a copy of the argument's value, changes made to the parameter inside the function do not affect the original argument outside the function.
ශ්‍රිතයකට argument එකෙහි අගයේ පිටපතක් ලැබුණු විට, ශ්‍රිතය තුළ ඇති පරාමිතියට සිදු කරන ලද වෙනස්කම් ශ්‍රිතයෙන් පිටත මුල් argument එකට බලපාන්නේ නැත.

2. Passing by Reference
When a function receives a reference (or address) to the original object, changes made to the parameter inside the function affect the original argument outside the function.
ශ්‍රිතයකට මුල් වස්තුව වෙත යොමුවක් (හෝ ලිපිනයක්) ලැබෙන විට, ශ්‍රිතය තුළ ඇති පරාමිතියට සිදු කරන ලද වෙනස්කම් ශ්‍රිතයෙන් පිටත මුල් argument එකට බලපායි.

Whether changes to the object inside the function affect the original object depends on whether the object is mutable or immutable.
ශ්‍රිතය තුළ වස්තුවේ වෙනස්කම් මුල් වස්තුවට බලපාන්නේද යන්න රඳා පවතින්නේ වස්තුව mutable ද immutable ද යන්න මතය.

Mutable Objects (Ex. lists, dictionaries):

Here, the memory address of the object is passed to the function.
මෙහිදී වස්තුවේ මතක ලිපිනය ශ්‍රිතයට යවනු ලැබේ.

This means changes made to the object inside the function will affect the original object outside the function.
මෙයින් අදහස් කරන්නේ ශ්‍රිතය තුළ ඇති වස්තුවට කරන ලද වෙනස්කම් ශ්‍රිතයෙන් පිටත මුල් වස්තුවට බලපාන බවයි.

Immutable Objects (Ex. integers, strings, tuples):

A copy of the original object is passed to the function.
මුල් වස්තුවේ පිටපතක් ශ්‍රිතයට යවනු ලැබේ.

Therefore, any changes made to the object inside the function will not affect the original object.
එමනිසා, ශ්‍රිතය තුළදී වස්තුවට කරන ලද කිසිදු වෙනසක් මුල් වස්තුවට බලපාන්නේ නැත.

Passing Mutable Object (List)

```
def modify_list(lst):
    lst.append(4)

my_list = [1, 2, 3]
modify_list(my_list)

print("Modified list:", my_list)
```

```
Modified list: [1, 2, 3, 4]
```

Passing Immutable Object (Integer)

```
def modify_number(num):
    num += 1

my_num = 5
modify_number(my_num)

print("Original number:", my_num)
```

```
Original number: 5
```


ictfromabc.com

Recursive functions
ආවර්ථීක ශ්‍රිත

In programming, a recursive function is a function that calls itself inside itself to solve a problem.
ශ්‍රිතයක් තුල එම ශ්‍රිතයම කැඳවා ගැටළු විසඳන වර්ගයේ ශ්‍රිත ආවර්ථීක ශ්‍රිත ලෙස හැඳින්වේ.

Here, a conditional statement is used to break recurrence.
මෙහිදී පුණර්කරණය පාලනය කිරීම සඳහා තාර්කික ප්‍රකාශයක් භාවිතා කරයි

It's called the base condition.
එය මූලික කොන්දේසිය නමින් හැඳින්වේ.

```
def factorial(n):
    if n==1:
        return 1
    else:
        return n*factorial(n-1)
print(factorial(5))
120
```

Function Call	Explanation	Result
factorial(5)	5×factorial(4)	?
factorial(4)	4×factorial(3)	?
factorial(3)	3×factorial(2)	?
factorial(2)	2×factorial(1)	?
factorial(1)	return 1	1
factorial(2)	2×factorial(1)	2
factorial(3)	3×factorial(2)	6
factorial(4)	4×factorial(3)	24
factorial(5)	5×factorial(4)	120

To calculate factorial(5), we need to calculate 5×factorial(4), so we look at the next row to see what factorial(4) is.
factorial(5) සාදාගැනීම සඳහා factorial(4) සාදාගත යුතුය. එමනිසා factorial(4) කුමක්දැයි සොයාගත යුතුය.

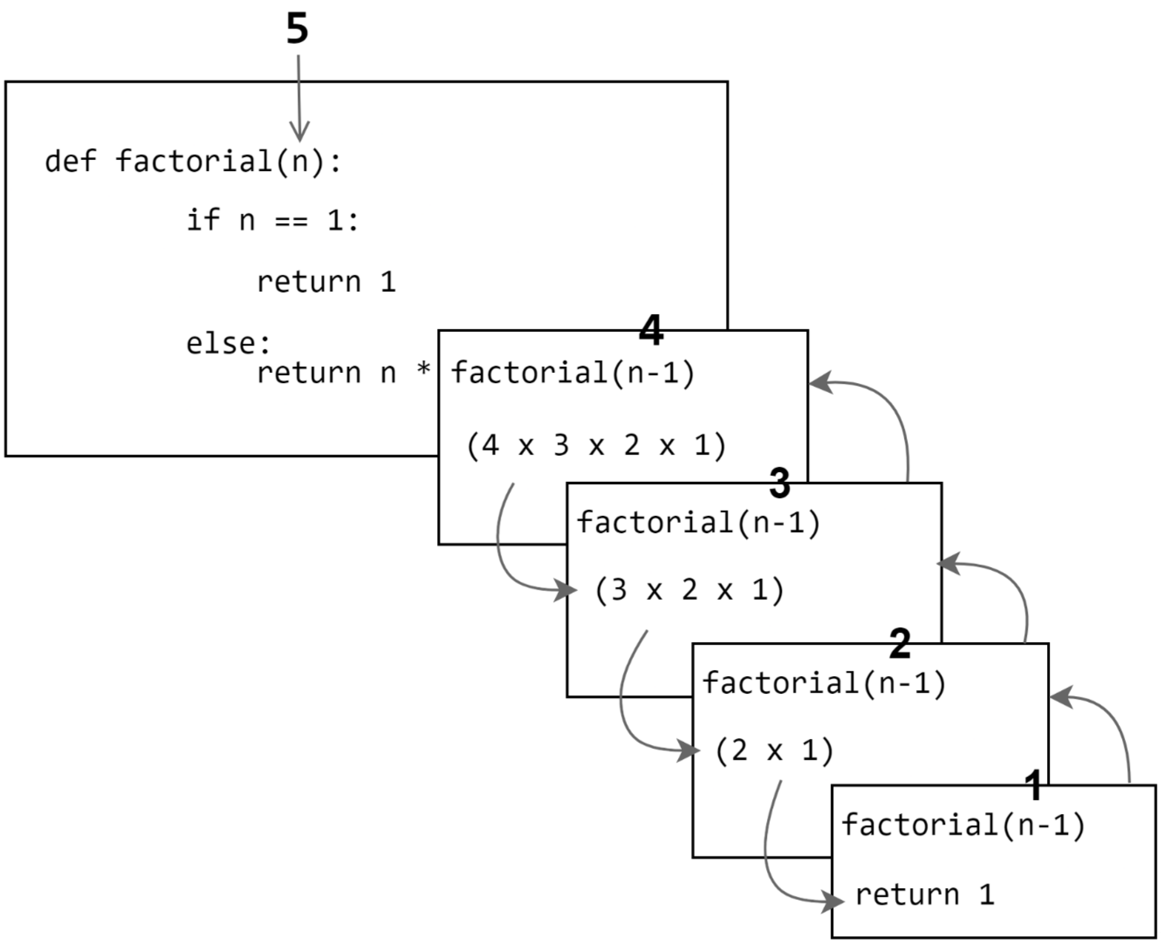
factorial(4) is 4×factorial(3), so we look at the next row to see what factorial(3) is.
factorial(4) සාදාගැනීම සඳහා factorial(3) සාදාගත යුතුය. එමනිසා factorial(3) කුමක්දැයි සොයාගත යුතුය.

We continue this pattern until we reach factorial(1), which is our base case and simply returns 1.
මෙම රටාව factorial(1) දක්වා ගමන් කල යුතුය. එවිට එමගින් පිළිතුර ලෙස 1 ලැබේ.

Then we can work our way back up, multiplying the numbers together to get the result of 120.
එවිට නැවත ආපස්සට ක්‍රියාත්මක වෙමින් සියලු සංඛ්‍යා ගුණකිරීම සිදුකර120 ලෙස පිළිතුර ලබාගත හැක.

NOTE:
A recursive function is like a magical mirror that can reflect itself. Imagine you have a series of mirrors, each reflecting the image of the previous one.
ආවර්ථීක ශ්‍රිතයක් යනු එයම පරාවර්තනය කළ හැකි ඉන්ද්‍රජාලික කැඩපතක් වැනිය. ඔබ සතුව දර්පණ මාලාවක් ඇතැයි සිතන්න, ඒ සෑම එකක්ම පෙර රූපයේ රූපය පිළිබිඹු කරයි.

Demontration
රූප සටහනක් ආශ්‍රයෙන් පැහැදිලි කිරීම



Another recursive function example that reverses a string
තන්තුවක් ආපසු හරවන ආවර්තීක් ශ්‍රිතයකට උදාහරණයක්

```
def reverse_string(s):
    # Base case: If the string is empty or has one character, return it
    if len(s) <= 1:
        return s
    # Reverse the rest of the string and append the first character
    return reverse_string(s[1:]) + s[0]

result = reverse_string("hello")
print(f"Reversed string: {result}")
```

```
Reversed string: olleh
```

NOTE:

Global variables vs Local variables

ගෝලීය විචල්‍ය සහ ස්ථානීය විචල්‍ය

Variables which are created inside a function are known as local variables.
They can only be called inside the function.
ශ්‍රිතයක් තුල නිර්මාණය කරගන්නා විචල්‍ය ස්ථානීය විචල්‍ය ලෙස හැඳින්වේ. ඒවා කැඳවිය හැක්කේ එම ශ්‍රිතය තුල සිට පමණි.

They get destroyed when the function finishes executing.
ශ්‍රිතය ක්‍රියාත්මකවී අවසානයේ ඒවා මතකයෙන් මැකී ඉවත්වී යයි.

```
def function1():
    z=50
    print(z)

function1()
42
print(z)
error
```

Variables which are created normally as we always did in the main program are known as global variables.
ප්‍රධාන වැඩසටහන තුල සාමාන්‍ය ආකාරයෙන් සාදාගන්නා විචල්‍ය ගෝලීය විචල්‍ය ලෙස හඳුන්වයි.

They can be called anywhere in the program, inside or out of a function.
ඒවා ශ්‍රිතයක් තුල වේවා ශ්‍රිතයක් තුල නොවේවා වැඩසටහනේ ඕනෑම ස්ථානයක සිට කැඳවිය හැකිය.

They retain their value until the program is running.
වැඩසටහන ක්‍රියාත්මක වනතෙක් මේවා ඒවායේ අගය රඳවා පවත්වාගනී.

```
x=100
def function2(y):
      x=10
      x=x+y
      print(x)

function2(50)
60
print(x)
150
```

We can even use the same identifier names for local variables and global variables. Python will not be confused.
ස්ථානීය විචල්‍ය සහ ගෝලීය විචල්‍ය සඳහා එකම නම් පවා භාවිතා කල හැකි අතර එමගින් Python අපහසුතාවයට පත් නොවේ

```
x=100
def function2(y):
      global x
      x = x+y
      print(x)

function2(50)
150
```

To address global scope variables from inside a function, the **global** keyword is used before a variable name.
ශ්‍රිතයක් තුල සිට ගෝලීය විචල්‍ය වෙත ප්‍රවේශ වීම සඳහා විචල්‍ය නාමයට ඉදිරියෙන් global යතුරුපදය භාවිතා කරයි.

2021- 42

What will be the result when the following Python code is executed?
පහත පයිතන් කේතය ක්‍රියාත්මක කළ විට ලැබෙන ප්‍රතිඵලය කුමක්ද?

```
x = 10
def myfun(a):
    global x
    a = x + a
    x = 30
    return a

print(myfun(x))
```

Explanation

Here, the global variable x has been declared as global within the function.
මෙහිදී x ගෝලීය විචල්‍යය ශ්‍රිතය තුළ ගෝලීය ලෙස ප්‍රකාශ කර ඇත.

This means, x can be modified inside the function, affecting the original value of x.
මෙයින් අදහස් වන්නේ, x හි මුල් අගය වෙනස් කරමින් ශ්‍රිතය තුළදී x වෙනස් කළ හැකි බවයි.

When `myfun` is called, the same x variable has been passed as the parameter, which sets the value of a to 10.
`myfun` ඇමතූ විට, එම x විචල්‍යයම පරාමිතිය ලෙස ශ්‍රිතය තුළට ලබා දෙන අතර එය a හි අගය 10 ලෙස සකසයි.

Inside the function, a is updated by adding the current value of a and x, and then assigning the result back to a.
ශ්‍රිතය තුළදී, සහ x හි වත්මන් අගයන් එකතු කිරීමෙන් පසුව ප්‍රතිඵලය නැවත a වෙත ආදේශ කිරීමෙන a යාවත්කාලීන කරනු ලබයි.

x is also set to 30 which changes the original value from 10 to 30.
x ද 30 ලෙස සකසනු ලබන අතර එමඟින් එහි මුල් අගය 10 සිට 30 දක්වා වෙනස් වේ.

Finally, a is returned and printed.
අවසාන වශයෙන්, a ප්‍රත්‍යාගමනය කර මුද්‍රණය කරනු ලැබේ.

2020 – 38

What would be the output when the following Python code is executed?
පහත පයිතන් කේතය ක්‍රියාත්මක කළ විට ලැබෙන ප්‍රතිදානය කුමක්ද?

```
x = 1
def myfunc(p, q):
    global x
    p, q = q, p
    x = x + p

myfunc(x, 3)
print(x)
```

Explanation

This question also has the same approach as the previous one.
මෙම ප්‍රශ්නයටද පෙර ප්‍රවේශයට සමාන ප්‍රවේශයක් ඇත.

When `myfunc` is called, parameter p is assigned the value of the global variable x, and parameter q is assigned 3.
`myfunc` ඇමතූ විට, p පරාමිතියට ගෝලීය විචල්‍යය වන x හි අගය ද, q පරාමිතියට 3 ද ආදේශ වනු ලැබේ.

Within the function, the values of p and q are swapped, making p = 3 and q = 1.
ශ්‍රිතය තුළදී, p = 3 සහ q = 1 බවට පත් කරමින් p සහ q හි අගයන් හුවමාරු කරනු ලබයි.

After that, the value of global variable x is updated by adding the current values of x and p, and then assigning the result which is 4 back to x.
ඉන් පසු, x සහ p හි වත්මන් අගයන් එකතු කිර, ඉන්පසු ප්‍රතිඵලය වන 4 x වෙත නැවත ආදේශ කිරීමෙන් ගෝලීය විචල්‍ය වන x හි අගය යාවත්කාලීන කරනු ලැබේ.

Since, x is declared as global at the start of the function, the original value of x is changed from 1 to 4.
ශ්‍රිතය ආරම්භයේදී x ගෝලීය යනුවෙන් ප්‍රකාශ කර ඇති බැවින්, x හි මුල් අගය 1 සිට 4 දක්වා වෙනස් වේ.

Therefore, `print(x)` prints 4.
එබැවින් `print(x)` මගින් 4 මුද්‍රණය කෙරේ.

Number conversion functions
සංඛ්‍යා පරිවර්තන ශ්‍රිත

can be used to convert a given base 10 number into the specific base.
10 පාදයේ සංඛ්‍යාවක් වෙනත් නිශ්චිත පාදයකට පරිවර්තනය කිරීම සඳහා භාවිතා කරයි

These functions return a string with a prefix of the specific base as the output.
මෙම ශ්‍රිත වල ප්‍රතිදානය ලෙස අදාල පාදයේ උපසර්ගයක් සහිත string එකක් ලබාදේ

If used with prefixes, the same functions can be used to convert a number represented by any specific base into its own kind.
උපසර්ගත් සමග භාවිතා කරන්නේ නම් මෙම ශ්‍රිත වෙනත් ඕනෑම පාදයක සංඛ්‍යාවක් තම පාදය වෙත හරවා ගැනීම සඳහා භාවිතා කල හැක

`bin()` function
converts into binary.
ද්වීමය සංඛ්‍යා පාදයට පරිවර්තනය කරයි

bin(50)
'0b110010'

bin(0o62)
'0b110010'

bin(0x32)
'0b110010'

`oct()` function
converts into octal.
අෂ්ඨමය සංඛ්‍යා පාදයට පරිවර්තනය කරයි

oct(50)
'0o62'

oct(0b110010)
'0o62'

oct(0x32)
'0o62'

`hex()` function
converts into hexadecimal.
ශඩ් දශමය සංඛ්‍යා පාදයට පරිවර්තනය කරයි

hex(50)
'0x32'

hex(0b110010)
'0x32'

hex(0o62)
'0x32'

`int()` function
can be used to convert a number represented by any other base into a base 10 number.
වෙනත් ඕනෑම පාදයක සංඛ්‍යාවක් 10 පාදයට පරිවර්තනය කිරීම සඳහා භාවිතා කරයි

int(0b110010)
50

int(0o62)
50

int(0x32)
50

prefixes
උපසර්ග

```
0b   - binary
0o   - octal
0x   - hexadecimal
```

Commonly used built-in functions
බහුලව භාවිතා වන built-in ශ්‍රිත

`type()` function

This function is used to determine the data type of a variable or a data structure.
විචල්‍යයක හෝ දත්ත ව්‍යුහයක දත්ත වර්ගය හඳුනාගැනීම සඳහා භාවිතා කරයි

```
a = 1
b = 3.14
c = True
d = 'hello'
e = [10, 20, 30]
f = 40, 50, 60
g = {'age':25, 'name':'rav'}
```

```
type(a)
<class 'int'>
```

```
type(b)
<class 'float'>
```

```
type(c)
<class 'bool'>
```

```
type(d)
<class 'str'>
```

```
type(e)
<class 'list'>
```

```
type(f)
<class 'tuple'>
```

```
type(g)
<class 'dict'>
```

`sum()` function

Used to add numeric data in lists tuples or dictionaries.
lists, tuples සහ dictionaries වල සංඛ්‍යා එකතු කිරීම සඳහා භාවිතා කරයි

Do not work with non-numeric data.
අංකමය නොවන දත්ත සමග ක්‍රියා නොකරයි

```
x=[10,20,30,40,50]
sum(x)
150
```

```
y=(10,20,30,40,50)
sum(y)
150
```

```
z={1:10,2:20,3:30,4:40,5:50}
sum(z)
15
```

```
sum(z.values())
150
```

```
sum(z.keys())
15
```

`sum()` function with start value

Defines an initial value to sum up the total.
එකතු කිරීමට ආරම්භක වටිනාකමක් ලබාදේ

```
x=[10,20,30,40,50]
sum(x,start=100)
250
```

`divmod()` function

`divmod(a,b)` returns `a//b` and `a%b`
`divmod(a,b)` මගින් `a//b` සහ a%b ලබාදේ

```
divmod(7,3)
(2, 1)
```

abs() function

gives the absolute value of a number
සංඛ්‍යාවක මාපාංකය ලබාදේ

abs(5)
5
abs(-5)
5

min() and max()

Returns the smallest and the largest item in a data structure or a given set of data.
දත්ත ව්‍යුහයක හෝ දී ඇති දත්ත සමූහයක ඇති උපරිම සහ අවම අගය ප්‍රත්‍යාගමනය කරයි

Do not work with different data types.
වෙනස් දත්ත වර්ග සමග ක්‍රියා නොකරයි

type 1:
function(sequence)

x=[10,20,30,40,50]
min(x)
10

y=(10,20,30,40,50)
max(y)
50

k='kurunegala'
max(k)
'u'

z={1:10,2:20,3:30,4:40,5:50}
min(z)
1

type 2:
function(arg1, arg2,... argN)

max(10,50,30,25,65)
65

pow() function - two arguments

pow(x,y) returns x**y
pow(x,y) මගින් x**y ලබාදේ

pow(10,-2)
0.01
pow(25,1/2)
5.0

pow() function - three arguments

pow(x,y,z) returns x**y%z
pow (x, y, z) මගින් x**y%z ලබාදේ

pow(2,3,3)
2

help() function

provides help about any keyword in python
Python හි ඕනෑම මූලපදයක් සම්බන්ධව උදව් ලබා දේ

id() function

displays the memory identity of an object.
විචල්‍යක් අඩංගු මතක අවකාශයේ හඳුනාගැනීමේ ලිපිනය පෙන්වයි

a=5
b=3
c=2
d=b+c
e=5.0
f=-5

id(a); id(b); id(c)
id(d); id(e); id(f)

id() of a, b, c, e, f are different
a, b, c, e, f ගේ id() අසමාන වේ

id() of a and d is the same
a and d ගේ id() සමාන වේ

`math` Module in python

Use `help()` function to learn more about the `math` module
`math` මොඩියුලය ගැන වැඩිදුර ඉගෙන ගැනීමට `help()` ශ්‍රිතය භාවිතා කරන්න

NOTE:
Modules
Usually, a module is a collection of functions which belong to the same category
Module එකක් යනු එකම වර්ගයට අයත් ශ්‍රිත කිහිපයක එකතුවකි

```
import math
```

When using external modules, we must import them into our program at the beginning of the code.
බාහිර මොඩියුල භාවිතා කරන විට, කේතයේ ආරම්භයේදීම අපගේ වැඩසටහනට ඒවා import කළ යුතුය.

`math.sqrt()` function
Gives the square root of any number
ඕනෑම සංඛ්‍යාවක වර්ගමූලය ලබාදේ

```
import math
math.sqrt(5)
2.23606797749979
```

`math.floor()` function

```
math.floor(1.5)
1
math.floor(-1.5)
-2
```

NOTE
An easy method to call functions in a module is as follows.
Module එකක් තුල පවතින ශ්‍රිතයක් කැඳවීම සඳහා පහසු ක්‍රමයක් පහත පරිදි වේ

```
f = math.floor
f(3.14)
3
```

`hash()` function
Returns the integer value of an object.
යම් වස්තුවක නිඛිල අගය පෙන්වයි

Hash value may be the same even if the data types are different.
දත්ත වර්ග වෙනස් වුවත් ඒවායේ අගයන් සමාන විය හැක

This function is used to compare whether two values are the same.
අගයන් දෙකක් සමාන දැයි පරීක්ෂා කිරීම සඳහා මෙය භාවිතා කරයි

```
a,b,c,d=1,1.0,True,-1
hash(a); hash(b); hash(c); hash(d)
1 1 1 -1
```

`len()`function
Returns the number of items of a container
දත්ත ව්‍යුහයක පවතින දත්ත ගණන පෙන්වයි

```
a='Kurunegala'
b=[10,20,30,40]
c=(40,50,60)
d={'a':24,'b':22}

len(a)
10
len(b)
4
len(c)
3
len(d)
2
```

`chr()` function

Returns the UNICODE character of a given integer

දෙන ලද නිඛිලයක UNICODE අක්ෂරය ලබාදේ

```
chr(65)
'A'
chr(97)
'a'
```

Methods

Methods are also functions which are always called on an object.

සෑමවිටම වස්තුවක් මත ක්‍රියාත්මක කරවන ශ්‍රිත මේවාය.

```
object.method(arguments)
```

Methods can alter the object on which it is called, but functions normally get input, do something and return an output only.

මේවා මගින් එම වස්තුවට බොහෝවිට වෙනස්කම් සිදුකරන අතර සාමාන්‍ය ශ්‍රිත සිදුකරනුයේ ආදාන ලබාගෙන ඒවාට අදාල යම් ක්‍රියාවලියක් මගින් ප්‍රතිදාන ලබාදීම පමණි

`list.append()` method

used to add new elements to the end of a list.

list එකක අගට අලුත් අවයව එකතු කිරීම සඳහා භාවිතා කරයි

`list.insert()` method

used to insert data before a specific index of an existing list

පවතින එකක නිශ්චිත සබාැං එකකට පෙර දත්ත ඇතුලත් කිරීම සඳහා භාවිතා කරයි

`list.pop()` method
`dictionary.pop()` method

used to remove data in a specific index of a list and return its value.

list එකක පවතින නිශ්චිත එකක දත්තයක් ඉවත් කර ගෙනැවිත් පෙන්වීමට භාවිතා කරයි

`list.extend()` method

Used to extend an existing list with the elements from any other type of sequence (list,tuple,string).

පවතින කසිඑ එකක් තවත් sequence එකකින් අවයව ගෙන දීර්ඝ කිරීම සඳහා භාවිතා කරයි

`list.copy()` method
`dictionary.copy()` method
`set.copy()` method

Copies one data structure into another variable.

වෙනත් විචල්‍යක් තුලට දත්ත ව්‍යුහයක දත්ත පිටපත් කරගනී

`list.remove()` method
`set.remove()` method

Used to remove the first coming specific element from a list or a set.

list එකක හෝ set එකක පවතින මුලින්ම හමුවන නිශ්චිත දත්තය ඉවත් කිරීම සඳහා භාවිතා කරයි

`list.count()` method
`string.count()` method
`tuple.count()` method

Number of times that an element is in a sequence (list or a string)

යම් අවයවයක් sequence එකක් තුල පවතින වාර ගණන මෙමගින් පෙන්වයි

`list.clear()` method
`dictionary.clear()` method
`set.clear()` method

clears all the elements form a list/dict

list/dict එකකින් සියලු අවයව ඉවත් කරයි

`list.index()` method
`string.index()` method
`tuple.index()` method

Displays the index of the first coming required element in a sequence.

sequence එකක අවශ්‍යකරන දත්තයේ පළමුවෙන්ම හමුවන index එක පෙන්වයි.

`list.sort()` method

Used with lists to sort the elements in ascending order by default.

අවයව ආරෝහණ පිළිවෙලට වර්ග කිරීමට lists සමඟ භාවිතා කරයි

`list.reverse()` method

Used with lists to reverse the order of the elements in the list.

List එකක ඇති අවයව වල අනුපිළිවෙල ආපසු හැරවීමට භාවිතා වේ

`dictionary.keys()` method

Used with dictionaries to return a list of all the keys in the dictionary.

Dictionary එකක ඇති සියලුම යතුරු ලැයිස්තුවක් ආපසු ලබා දීමට භාවිතා වේ

`dictionary.values()` method

Used with dictionaries to return a list of all the values in the dictionary.

Dictionary එකක ඇති සියලුම අගයන් ලැයිස්තුවක් ආපසු ලබා දීමට භාවිතා වේ

`dictionary.get()` method

Is used with a dictionary to retrieve the value for a specified key.

නිශ්චිත යතුරක් සඳහා අගයක් ලබා ගැනීමට . එකක් සමඟ භාවිතා වේ

`dictionary.objects()` method

Used with dictionaries to return a list of dictionary's key-value tuple pairs.

Dictionary එකක යතුරු-අගය tuple යුගල list එකක් ආපසු ලබාගැනීම සඳහා භාවිතා වේ

`dictionary.popitem()` method

Used to remove and return the last key-value pair from the dictionary.

dictionary එකකින් අවසන් යතුරු-අගය යුගලය ඉවත් කර ආපසු ලබා දීමට භාවිතා කරයි.

other:
`dictionary.fromkeys()` method
`dictionary.update()` method
`dictionary.setdefault()` method

`string.find()` method

displays the lowest available starting index of a sub string in a main string.

String එකක් තුල සොයන්නාවූ sub string එකක ඇති අවම ආරම්භක සබාැං එක ලබාදේ

ictfromabc.com

Error handling with `try: except:`
`try: except:` සමගින් දෝෂ පාලනය

The `try` and `except` statements are used to handle exceptions, which are errors that occur during the execution of a program.
ව්‍යතිරේක හෙවත් වැඩසටහන ක්‍රියාත්මක වීමේදී සිදු වන දෝෂ හැසිරවීමට `try` සහ `except` ප්‍රකාශ භාවිතා කරනු ලැබේ.

Syntax

```
try:
    risky_code

except ExceptionType:
    handle_exception
```

try block:
Contains the code that might raise an exception. If an error occurs within this block, Python immediately stops execution here and jumps to the corresponding `except` block.
ව්‍යතිරේකයක් මතු කළ හැකි කේතය අඩංගු වේ. මෙම බ්ලොක් එක තුළ දෝෂයක් ඇති වුවහොත්, පයිතන් වහාම මෙහි ක්‍රියාත්මක කිරීම නවත්වා අනුරූප `except` බ්ලොක් එක වෙත පනිනු ලැබේ.

except block:
Catches and handles the specified exception. If the type of exception matches `ExceptionType`, the code in this block executes.
නිශ්චිත ව්‍යතිරේකය අල්ලාගෙන හසුරුවයි. ව්‍යතිරේක වර්ගය `ExceptionType` වලට ගැලපේ නම්, මෙම කොටසේ ඇති කේතය ක්‍රියාත්මක වේ.

Example

```
try:
    num = int(input("Enter a
    number: "))
    result = 10 / num
    print("Result:", result)

except ValueError:
    print("Error: You must enter
    an integer.")

except ZeroDivisionError:
    print("Error: Cannot divide by
    zero.")
```

If the user inputs a non-integer value, a `ValueError` occurs, and the first except block handles it.
පරිශීලකයා නිඛිල නොවන අගයක් ඇතුළත් කළහොත්, `ValueError` හටගනී, පලමු වැනි except block එක එය හසුරුවයි.

If the user enters 0, a `ZeroDivisionError` occurs, and the second except block handles it.
පරිශීලකයා 0 ඇතුලත් කළහොත්, `ZeroDivisionError` හටගනී, දෙවැනි except block එක මගින් එය හසුරුවයි.

If no exception occurs, the program runs normally and prints the result.
ව්‍යතිරේකයක් සිදු නොවේ නම්, වැඩසටහන සමාන්‍ය ආකාරයෙන් ක්‍රියාත්මක වන අතර ප්‍රතිඵලය මුද්‍රණය කරයි.

Common types of exceptions

```
TypeError
ValueError
IndexError
KeyError
ZeroDivisionError
AttributeError
NameError
```


ictfromabc.com

File operation
ගොනු හැසිරවීම

File operations are fundamental actions that allow you to interact with files on our system. Under file operation we discuss about the following aspects
ගොනු හැසිරවීම යනු අපගේ පරිගණකයේ පවතින ගොනු සමග අන්තර්ක්‍රියා කිරීමට භාවිත කරන සිද්ධාන්තයකි. මේ යටතේ පහත කරුණු ගැන සාකච්චා කරයි.

Creating a new file
Opening a file
Writing in a file
Closing a file
Appending to a file
Overwriting a file
Reading from a file
Seeking within a file
Deleting a file

The following functions and methods will be used to discuss the lesson.
මේ සඳහා පහත functions සහ methods භාවිතා කරයි.

`open()` function
`close()` method
`write()` method
`writelines()` method
`read()` method
`readlines()` method
`tell()` method
`seek()` method

What is a file object?
file object එකක් යනු කුමක්ද?

A file object in Python is a memory location (like a variable) that's created in the main memory of a computer when you open a file from the secondary storage using the open function.
එකක් යනු open function එක භාවිතා කරමින් පරිගණකයේ ද්විතියික මතකය තුල පවතින ගොණුවක් ගන්වාගෙන විවෘත කිරීමේදී ප්‍රාථමික මතකය තුල නිර්මාණය වන මතක අවකාශයකි.

`open()` function

This function is used to create a file in the secondary storage of a computer and open it inside a file object in the main memory of a computer for further modifications.
පරිගණකයේ ද්විතියික ආචයනය තුල ගොණුවක් නිර්මාණය කර එය තවදුරටත් වෙනස්කිරීම් සඳහා ප්‍රධාන මතකය තුල නිර්මාණය කරගත් file object එකක් තුල විවෘත කර දීම මෙමගින් සිදුවේ.

It has two parameters.
මෙයට පරාමිතීන් දෙකක් පවතී.

It requires a filename and a mode
එයට ගොනු නාමයක් සහ mode එකක් අවශ්‍ය වේ

```
open(filename,mode)
```

File modes

File modes determine the type of operation you can perform on a file (Ex. reading, writing, or appending).
ගොනු මාදිලි මගින් ඔබට ගොනුවක් මත සිදු කළ හැකි මෙහෙයුම් වර්ගය කුමක්ද යන්න තීරණය කරයි (උදා: කියවීම, ලිවීම හෝ එකතු කිරීම).

When opening a file using `open()` function, you specify the file mode as the second argument, which controls how the file is accessed.
`open()` ශ්‍රිතය භාවිතයෙන් ගොනුවක් විවෘත කරන විට, ගොනුව වෙත ප්‍රවේශ වන ආකාරය පාලනය කරන දෙවන තර්කය ලෙස ගොනු මාදිලිය සඳහන් කරයි.

There are main 4 different modes.
ගොණුවක් විවෘත කරගත හැකි ප්‍රධාන ආකාර හතරකි.

"r" or the read only mode. (default)

A file will open in this mode even if we don't define a mode for the file to open.
This will open a file only for reading, and this will generate an error if the file is not available in secondary storage.
mode එකක් යොදා නොමැති අවස්ථාවකදී පවා ගොනුවක් විවෘත වන්නේ මෙම ආකාරයෙනි. මෙමගින් ගොණුවක් කියවීම සඳහා පමණක් විවෘත වන අතර, එවැනි ගොණුවක් ද්වීතියික ආචයනයේ නොපවතී නම් දෝශයක් ලබාදේ.

```
file = open("example.txt", "r")
content = file.read()
print(content)
file.close()
```

"w" or the write mode(overwrite)

Used to open an already available file in the secondary storage for writing data.
දත්ත ලිවීම සඳහා දැනටමත් ද්වීතියික මතකයේ ගබඩා කරගෙන ඇති ගොණුවක් විවෘත කරදීම මෙමගින් සිදුවේ.

Opening a file in this mode will erase all the data in the file if any data is available and if the file is not present in the secondary storage this mode will also create the file for you.
මෙහිදී ගොනුවේ දත්ත පවතී නම් ඒවා සියල්ල මැකී ගොස් නැවත මුල සිට දත්ත ලිවීම ආරම්භ වන අතර එවන් ගොනුවක් ද්වීතියික ආචයනයේ නොපවතී නම් එය නිර්මාණය වී විවෘත වීම සිදුවේ.

```
file = open("example.txt", "w")
file.write("This will overwrite
any existing content.")
file.close()
```

"x" or the create mode.

It is used to create a new file in secondary storage and open it in the main memory to write data.
ද්වීතියික ආචයනය තුල නව ගොණුවක් නිර්මාණය කරමින් ප්‍රධාන මතකය තුල එය විවෘත කරගෙන ඒ තුල දත්ත ලිවීම සඳහා මෙම ආකාරය භාවිතා කරයි.

This generates an error if the file is already present.
දැනටමත් ගොණුවක් පවතී නම් දෝශයක් ලබාදේ.

```
try:
   file = open("newfile.txt", "x")
   file.write("This file was
   created using 'x' mode.")
   file.close()

except FileExistsError:
   print("File already exists!
   Choose a different file name.")
```

In this example,
මෙම උදාහරණයේ,

If "`newfile.txt`" does not exist, it will be created, and the text will be written to it.
"`newfile.txt`" නොපවතියි නම්, එය සාදනු ලබන අතර, පෙළ එයට ලියනු ඇත.

If "`newfile.txt`" already exists, a `FileExistsError` will be raised, and the program will print a warning message.
"`newfile.txt`" දැනටමත් පවතී නම්, `FileExistsError` දෝෂය මතු වනු ඇත, සහ වැඩසටහන අනතුරු ඇඟවීමේ පණිවිඩයක් මුද්‍රණය කරනු ඇත.

"a" or the append mode.

Also used to open an already available file in the secondary storage for writing data.
මෙමගින් සිදුවන්නේද දත්ත ලිවීම සඳහා දැනටමත් ද්විතියික මතකයේ ගබඩා කරගෙන ඇති ගොණුවක් විවෘත කර දීමයි.

But opening a file in this mode will not erase the data in the file if any data is available and allows you to write to the end of the file. If the file is not present in the secondary storage this mode will also create the file for you.
මෙහිදී ගොනුවේ දත්ත පවතී නම් ඒවා කිසිවක් වෙනස් නොවී ගොනුවේ අගට නව දත්ත ලිවීම ආරම්භ වන අතර එවන් ගොනුවක් ද්විතියික ආචයනයේ නොපවතී නම් එය නිර්මාණය වී විවෘත වීම සිදුවේ.

```
file = open("example.txt", "a")
file.write("\nAdding this line to
the end of the file.")
file.close()
```

In addition, you can specify if the file should be handled as binary or text mode.
ඊට අමතරව, ගොනුව ද්විමය හෝ පෙළ ආකාරයෙන් හැසිරවිය යුතුද යන්න ඔබට සඳහන් කළ හැක.

"t" - Text - Default value. Text mode

Text mode is useful for reading and writing standard text files.
එක සම්මත පෙළ ගොනු කියවීමට සහ ලිවීමට ප්‍රයෝජනවත් වේ.

"b" - Binary - Binary mode

This is essential for handling non-text files, like images, videos, and executables.
පින්තූර, වීඩියෝ සහ ක්‍රියාත්මක කළ හැකි යෙදුම් වැනි පෙළ-නොවන ගොනු හැසිරවීමට මෙය අත්‍යවශ්‍ය වේ.

```
file = open("img.png", "rb")
```

File methods

`close()` method
This method is used to close an open file inside a file object in the main memory. changes made to a file will not show in the storage until you close the file from the memory.
ප්‍රධාන මතකයේ file object එකක් තුල විවෘතව පවතින ගොණුවක් වසා දැමීම සඳහා මෙය භාවිතා කරයි. ගොණුවක් ප්‍රධාන මතකයෙන් වසා දැමීම සිදුවනතෙක් එයට සිදුකරන ලද වෙනස්කම් ද්විතියික ආචයනයේ දැකීමට නොහැක.

Writing to a file
ගොනුවක් තුළට ලිවීම

`write()` method.
This method can be used to write a single data string on the file.
ගොනුවක් තුල එක් දත්ත string එකක් ලිවීම සඳහා මෙම method එක භාවිතා කරයි.

```
f1=open('sample.txt','x')
f1.write('Helloworld')
f1.close()
f2=open('sample.txt')
f2.read()
'Helloworld'
f2.close()
```

`writelines()` method
This method can be used to write more than one string at a time into a file using the strings as objects placed in a list.
මෙම method එක මගින් list එකක ඇතුලත් කරන ලද objects කිහිපයක් ලෙස ලබාදුන් විට එකම ගොනුව තුල එකවර string කිහිපයක් ලිවීම සිදුකල හැක.

```
f3=open('sample.txt','w')
f3.writelines(['line1\n','line2\n'])
f3.close()
f4=open('sample.txt')
f4.read()
'line1\nline2\n'
f4.close()
```

Reading a file
ගොනුවක් කියවීම

`read()` method.
Can be used to read an entire file one time from the beginning to the end and get its entire content as a single string.
සම්පූර්ණ ගොණුවක අන්තර්ගතය මුල සිට අගට එකවරක් කියවා තනි string එකකට ලබාගැනීමට භාවිතා කරයි.

```
f1=open('fileoperation.txt','x')
f1.write('RavinduBandaranayake\n')
f1.write('0714555513')
f1.close()
```

```
f2=open('fileoperation.txt')
f2.read()
'RavinduBandaranayake\n0714555513'
f2.close()
```

```
f3=open('fileoperation.txt','a')
f3.write('\nictfromabc.com')
F3.close()
```

read(n) can specify the number of characters to be read at a time without reading the total file at once.
සම්පූර්ණ ගොනුවම එකවර කියවීමෙන් තොරව read(n) මගින් එක් වරකට කියවිය යුතු characters ගණන ලබාදිය හැකිය.

```
f4=open('fileoperation.txt')
f4.read(20)
'Ravindu Bandaranayak'
f4.read(10)
'e\n07145555'
f4.read(20)
'13\nictfromabc.com'
```

`readline()` method.

This method can be used to read a file line by line from the beginning once.
පමම method එක ගොනුවක් මුල සිට අගට එක් වරක් පේලියෙන් පේලිය කියවීම සඳහා භාවිතා කල හැක.

```
f5=open('fileoperation.txt')
f5.readline()
'Ravindu Bandaranayake\n'
f5.readline()
'0714555513\n'
f5.readline()
'ictfromabc.com\n'
f5.readline()
''
f5.close()
```

`readlines()` method

This method returns all the lines in a file one by one as elements of a list.
මෙම ශ්‍රිතය මගින් ගොනුවක පේලි එකින් එක කියවා ඒවා සියල්ල list එකක elements ලෙස පෙලගස්වා ලබාදේ.

```
f6=open('fileoperation.txt')
f6.readlines()
['Ravindu Bandaranayake\n',
'0714555513\n','ictfromabc.com\n']
f6.close()
```

Navigating in a File
ගොනුවක් තුල සැරිසැරීම

`seek(position)` method.
Can be used to navigate the cursor to a specific position inside a file.
ගොනුවක් තුල නිශ්චිත ස්ථානයකට cursor එක රැගෙන යාම සඳහා seek(position) method එක භාවිතා කරයි.

You can move the file pointer to any part of the file by specifying the byte offset.
බයිට් ඕෆ්සෙට් නියම කිරීමෙන් ඔබට ගොනු දර්ශකය ගොනුවේ ඕනෑම කොටසකට ගෙන යා හැක.

seek(**offset, from_what**):

offset
The number of bytes to move.
චලනය කිරීමට ඇති බයිට ගණන.

from_what
Determines the reference point for the offset:
ඕෆ්සෙට් සඳහා යොමු ලක්ෂ්‍යය තීරණය කරයි:

0: Start of the file (default).
1: Current position.
2: End of the file.

```
file = open("fileoperation.txt")
file.read()
'RavinduBandaranayake\n'
file.seek(10)
file.read()
'ndaranayake\n'
file.close()
```

`tell()` method.

Can be used to ask the computer to return the current position of the cursor in the file.
ගොනුවක් තුල cursor එක පවතින ස්ථානය පරිගණකයෙන් අසා දැනගැනීම සඳහා tell( ) method එක භාවිතා කරයි.
It tells you where the file pointer is located after reading or writing.
කියවීමෙන් හෝ ලිවීමෙන් පසු ගොනු දර්ශකය පිහිටා ඇති ස්ථානය කුමක්ද යන්න පවසයි.

```
f1=open('fileoperation.txt')
f1.readline()
'Ravindu Bandaranayake\n'
f1.tell()
23
f1.close()
```

Deleting a file
ගොණුවක් මකාදැමීම

To delete a file, we must import the OS module and run the `os.remove()` method.
ගොනුවක් මකා දැමීමට, OS මොඩියුලය import කර `os.remove()` method එක ක්‍රියාත්මක කළ යුතුය.

```
f = open('ravindu.txt','x')
f.close()

import os

os.remove('ravindu.txt')
f = open('ravindu.txt')
error
```

Creating and deleting new directories (folders) using python can also be done.
python භාවිතයෙන් නව directories (folders) සෑදීම සහ මකාදැමීමද සිදුකලහැක.

```
import os
os.mkdir('helloworld')
os.rmdir('helloworld')
```

A Python code that reads student marks from a file, calculates the total of their marks, and writes the results with their names to another file.

ගොනුවකින් ශිෂ්‍ය ලකුණු කියවා, ඔවුන්ගේ ලකුණුවල එකතුව ගණනය කර වෙනත් ගොනුවකට සිසුන්ගේ නම් සමඟ ප්‍රතිඵල ලියනු ලබන පයිතන් කේතයක්.

```python
# Open the input file (marks.txt) in read mode
marks_file = open("marks.txt", "r")

# Read all lines from the file
lines = marks_file.readlines()

marks_file.close()

marks_totals = []

# Process each line
for line in lines:
    parts = line.split(", ")

    name = parts[0]  # The first part is the name
    marks = []  # Create an empty list for marks


    for i in range(1, len(parts)):
        marks.append(int(parts[i]))

    # Calculate the total of marks
    total = 0
    for mark in marks:
        total += mark

    marks_totals.append(f"{name}, {total}")

total_file = open("total.txt", "w")

for total_line in marks_totals:
    total_file.write(total_line + "\n")

total_file.close()
```

marks.txt

```
John, 85, 90, 78
Alice, 88, 76, 92
Bob, 70, 80, 65
Azeem, 60, 43, 67
Bella, 34, 64, 90
```

total.txt

```
John, 253
Alice, 256
Bob, 215
Azeem, 170
Bella, 188
```


ictfromabc.com

Connecting to a database in python
පයිතන් භාවිතයෙන් දත්ත සමුදායකට සම්බන්ධ වීම

Connecting Python to a database allows you to retrieve, manipulate, and manage data within your application.
පයිතන් දත්ත සමුදායකට සම්බන්ධ කිරීමෙන් ඔබගේ යෙදුම තුළදී දත්ත ලබා ගැනීමට, හැසිරවීමට සහ කළමනාකරණය කිරීමට ඉඩ සලසයි.

To interact with databases in Python, Database Drivers should be used.
පයිතන් හි දත්ත සමුදායන් සමඟ අන්තර් ක්‍රියා කිරීමට, දත්ත සමුදා ධාවක භාවිතා කළ යුතුය.

They are libraries that let us communicate with specific types of databases.
ඒවා අපට නිශ්චිත දත්ත සමුදායන් සමඟ සන්නිවේදනය කිරීමට ඉඩ සලසන libraries වේ.

In Python, MySQL or SQLite can be used to connect to databases.
පයිතන් හිදී, MySQL හෝ SQLite දත්ත සමුදායන් වෙත සම්බන්ධ වීමට භාවිතා කළ හැක.
(According to the syllabus)

MySQL
To connect to a MySQL database, a driver (like `mysql-connector-python`) that allows Python to communicate with MySQL needs to be installed.
එහිඹ දත්ත ගබඩාවකට සම්බන්ධ වීමට, එහිඹ සමඟ සන්නිවේදනය කිරීමට Python හට ඉඩ දෙන ධාවකයක් (`mysql-connector-python` වැනි) ස්ථාපනය කිරීමට අවශ්‍ය වේ.

SQLite
An additional driver is not needed because SQLite is built into Python.
SQLite පයිතන් තුළම ගොඩනගා ඇති නිසා අමතර ධාවකයක් අවශ්‍ය නොවේ.

The sqlite3 module can directly be used.
sqlite3 මොඩියුලය සෘජුවම භාවිතා කළ හැක.

(1) MySQL database connection

Here, a library called `mysql-connector-python` should be installed.
මෙහිදී `mysql-connector-python` නම් library එකක් ස්ථාපනය කළ යුතුය.

The `mysql-connector-python` library acts as a connector between Python and a MySQL database.
`mysql-connector-python` library එක පයිතන් සහ MySQL දත්ත සමුදාය අතර සම්බන්ධකයක් ලෙස ක්‍රියා කරයි.

Python cannot directly communicate with a MySQL database without this library.
මෙම library එක නොමැතිව පයිතන් හට MySQL දත්ත ගබඩාවක් සමඟ සෘජුව සන්නිවේදනය කළ නොහැක.

This library provides the necessary functions to:
මෙම library එක මගින් පහත අවශ්‍යතාවන් සපයයි.

- Open a connection to the MySQL database.
  MySQL දත්ත සමුදායකට සම්බන්ධතාවක් ස්ථාපිත කිරීම.
- Execute SQL commands (such as queries and updates).
  SQL විධාන (විමසුම් සහ යාවත්කාලීන වැනි) ක්‍රියාත්මක කිරීම.
- Manage the database connection and handle any errors.
  දත්ත සමුදා සම්බන්ධතාවය කළමනාකරණය කර ඕනෑම දෝෂයක් හැසිරවීම.

To install `mysql-connector-python`, **pip** is used.
`mysql-connector-python` ස්ථාපනය කිරීමට, pip භාවිතා වේ.

ictfromabc.com

What is pip?
pip යනු කුමක්ද

pip is Python's package installer.
pip යනු පයිතන් හි පැකේජ ස්ථාපකයයි.

It enables easy installation, updating, and management of additional libraries (or packages).
එය අමතර libraries (හෝ පැකේජ) පහසුවෙන් ස්ථාපනය කිරීම, යාවත්කාලීන කිරීම සහ කළමනාකරණය කිරීමට ඉඩ සලසයි.

By running the `pip install` command, pip retrieves the specified library from the Python Package Index (PyPI) and installs it on the system.
`pip install` විධානය ක්‍රියාත්මක කිරීමෙන්, pip විසින් නියම කරන ලද library එක Python Package Index (PyPI) වෙතින් ලබාගෙන එය පද්ධතිය මත ස්ථාපනය කරයි.

What is PyPI?
PyPI යනු කුමක්ද

The Python Package Index (PyPI) is an online repository of Python packages.
Python Package Index (PyPI) යනු පයිතන් පැකේජ වල මාර්ගගත ගබඩාවකි.

It hosts thousands of packages created by developers, allowing users to easily find, download, and install libraries that extend Python's functionality.
පරිශීලකයින්ට Python හි ක්‍රියාකාරීත්වය පුළුල් කරන libraries පහසුවෙන් සොයා ගැනීමට, බාගත කිරීමට සහ ස්ථාපනය කිරීමට ඉඩ සලසමින් එමගින් සංවර්ධකයින් විසින් නිර්මාණය කරන ලද පැකේජ දහස් ගණනකට සත්කාරකත්වය සපයයි.

1. Installing the MySQL Connector
MySQL සම්බන්ධකය ස්ථාපනය කිරීම

How to Use pip

To install the package, the command `pip install [package name]` is run on the command line.
පැකේජය ස්ථාපනය කිරීම සඳහා, `pip install [package name]` යන විධානය විධාන පේළිය මත ධාවනය කෙරේ.

```
pip install mysql-connector-python
```

This command tells pip to locate the `mysql-connector-python` package on PyPI, download it, and install it.
මෙම විධානය මගින් PyPI මත `mysql-connector-python` පැකේජය සොයා සොයා ගෙන එය බාගත කර ස්ථාපනය කිරීමට pip වෙත උපදෙස් ලබා දෙයි.

2. Verifying the Installation
ස්ථාපනය සහතික කරගැනීම

After installation, verify that the package was installed correctly.
ස්ථාපනය කිරීමෙන් පසු, පැකේජය නිවැරදිව ස්ථාපනය වී ඇත්දැයි තහවුරු කර ගන්න.

This can be done by opening a Python interpreter and importing the library:
පයිතන් අර්ථවින්‍යාසකයක් විවෘත කර library එක ආයාත කිරීමෙන් මෙය කළ හැකිය:

```
import mysql.connector
```

If no error appears, the installation was successful
දෝෂයක් හට නොගනී නම්, ස්ථාපනය සාර්ථක වී ඇත

3. Establishing a Connection

A connection to the MySQL database is created by specifying the host, username, password, and database name.
MySQL දත්ත සමුදාය වෙත සම්බන්ධතාවයක් නිර්මාණය කරනුයේ සත්කාරකය, පරිශීලක නාමය, මුරපදය සහ දත්ත සමුදායේ නම සඳහන් කිරීමෙනි.

host: Specifies the server location of the MySQL database. For a local database, this is often localhost.
MySQL දත්ත සමුදායේ සේවාදායක ස්ථානය සඳහන් කරයි. ස්ථානීය දත්ත සමුදායක් සඳහා, මෙය බොහෝ විට localhost වේ.

username: The MySQL account name used to access the database.
දත්ත සමුදායට ප්‍රවේශ වීමට භාවිතා කරන MySQL ගිණුමේ නම වේ.

password: The password for the specified MySQL username.
නිශ්චිත MySQL පරිශීලක නාමයක් සඳහා වන මුරපදය.

database name: The name of the specific database to connect to within the MySQL server.
MySQL සේවාදායකය තුළ සම්බන්ධ වීමට අවශ්‍ය දත්ත සමුදායේ නම.

```
import mysql.connector

conn = mysql.connector.connect(
        host="localhost",
        user="your_username",
        password="your_password",
        database="your_database"
)
```

4. Creating a Cursor

A cursor is an object that enables a Python program to send SQL commands (like SELECT, INSERT, UPDATE, DELETE) to the database.
Cursor යනු දත්ත සමුදාය වෙත SQL විධාන (SELECT, INSERT, UPDATE, DELETE වැනි) යැවීමට Python වැඩසටහනකට ඉඩ සලසන object එකකි.

It establishes a channel between Python and the database through which SQL queries are sent and responses are received.
එමගින් පයිතන් සහ දත්ත සමුදාය අතර SQL විමසුම් යවන සහ ප්‍රතිචාර ලැබෙන නාලිකාවක් ස්ථාපිත කරයි.

How to create a cursor

```
cursor = conn.cursor()
```

`conn.cursor():`
This method is called on an existing database connection object, creating a cursor specifically for that connection.
මෙම method එක දැනට පවතින දත්ත සමුදා සම්බන්ධතා වස්තුවක් මත අමතනු ලැබේ, එමගින් එම සම්බන්ධතාවය සඳහාම වූ විශේෂ කර්සරයක් නිර්මාණය කරයි.

Once a cursor is created, SQL queries can be executed.
කර්සරයක් නිර්මාණය කළ පසු, SQL විමසුම් ක්‍රියාත්මක කළ හැක.

For example, `cursor.execute("SELECT * FROM employees")` sends a query to retrieve all rows from the employees table.
උදාහරණයක් ලෙස, `cursor.execute("SELECT * FROM employees")` මගින් වගුවෙන් සියලුම පේළි ලබා ගැනීමට විමසුමක් යවයි.

5. Creating a Table

```
cursor.execute('''
    CREATE TABLE employees (
        id INT AUTO_INCREMENT
        PRIMARY KEY,
        name VARCHAR(20),
        age INTEGER
    )
''')
```

`cursor.execute()` method

The `cursor.execute()` method is used to run SQL commands in the database through the cursor.
Cursor එක හරහා දත්ත සමුදායේ SQL විධානයන් ක්‍රියාත්මක කිරීමට `cursor.execute()` method එක භාවිතා කරයි.

It sends an SQL statement from Python to the database.
එය පයිතන් සිට දත්ත ගබඩාවට SQL ප්‍රකාශයක් යවයි

Syntax

```
cursor.execute(sql_query, parameters)
```

`sql_query`:
The SQL command to be executed, provided as a string.
ක්‍රියාත්මක කළ යුතු SQL විධානය, එය තන්තුවක් ලෙස ලබා දෙනු ලැබේ.
(Ex. "SELECT * FROM employees")

`parameters` (optional):
Values to be safely inserted into the query to prevent SQL injection attacks.
SQL injection attacks වැළැක්වීම සඳහා විමසුමට ආරක්ෂිතව ඇතුළත් කළ යුතු අගයන්.

6. Inserting data
දත්ත ඇතුළත් කිරීම

Method 1

```
cursor.execute('''
INSERT INTO employees (name, age)
VALUES (?, ?)''', ("Alice", 30))
```

Method 2

```
query = """
   INSERT INTO employees(name,age)
   VALUES (%s, %s)"""
values = ("Alice", 30)

cursor.execute(query, values)
```

A new row is added to the employees table, targeting the `name` and `age` columns.
`name` සහ `age` තීරු ඉලක්ක කර ගනිමින් සේවක වගුවට නව පේළියක් එක් කෙරේ.

`("Alice", 30)`
This tuple provides the values for the placeholders. `"Alice"` is inserted as the name, and `30` as the age.
මෙම tuple එක මඟින් placeholders සඳහා අගයන් සපයයි. `"Alice"` යන අගය `name` ලෙසත්, `30` යන අගය `age` ලෙසත් ඇතුළත් කරනු ලැබේ.

Purpose of `?` and `%s`
These symbols act as placeholders, making the query parameterized.
මෙම සංකේත මගින් විමසුම පරාමිතිකරණය කරමින් placeholders එක ලෙස ක්‍රියා කරයි.

They prevent SQL injection by separating the query structure from the data.
ඒවා මගින් දත්ත වලින් විමසුම් ව්‍යුහය වෙන් කිරීමෙන් SQL injection වළක්වයි.

Inserting multiple data rows
දත්ත පේළි කිහිපයක් ඇතුළත් කිරීම

```
query = """
   INSERT INTO employees(name,age)
   VALUES (%s, %s)"""
values = [
      ("Alice", 30),
      ("Bob", 24),
      ("Charlie", 28),
      ("Diana", 32)
]

cursor.executemany(query, values)
```

executemany()
This method executes the INSERT statement for each tuple in values, inserting multiple rows with a single call.
මෙම method එක මගින් එක් ඇමතුමකින් පේළි කිහිපයක් ඇතුළත් කරමින් values හි ඇති එක් එක් tuple අගයන් සඳහා INSERT ප්‍රකාශය ක්‍රියාත්මක කරයි.

7. Committing the Changes

```
conn.commit()
```

In many databases (including SQLite and MySQL), changes are initially made in a temporary state.
බොහෝ දත්ත සමුදායන්හි (SQLite සහ MySQL ඇතුළුව), වෙනස්කම් මුලින් සිදු කරනු ලබන්නේ තාවකාලික තත්වයකයදීය.

Calling commit() finalizes these changes and saves them permanently to the database.
commit() මගින් මෙම වෙනස්කම් අවසන් කර ඒවා ස්ථිරවම දත්ත සමුදායට සුරකියි.

employees table

id	name	age
1	Alice	30
2	Bob	24
3	Charlie	28
4	Diana	32

NOTE:
Committing is only needed when there's a change in the database, such as with INSERT, UPDATE, or DELETE statements.
Commit කිරීම් අවශ්‍ය වන්නේ INSERT, UPDATE, හෝ DELETE ප්‍රකාශයන් වැනි දත්ත සමුදායේ වෙනසක් ඇති විට පමණි.

When selecting data (using SELECT queries), committing is not required, as no changes are being made to the database.
දත්ත තෝරන විට (SELECT විමසුම් භාවිතා කරමින්), දත්ත සමුදායට කිසිදු වෙනසක් සිදු නොවන බැවින් commit කිරීම අවශ්‍ය නොවේ.

8. Selecting data from the database
දත්ත සමුදායෙන් දත්ත තෝරා ගැනීම

```
cursor.execute("SELECT * FROM
employees")

results = cursor.fetchall()
print(results)
for row in results:
      print(row)
```

```
[
  (1, "Alice", 30),
  (2, "Bob", 24),
  (3, "Charlie", 28)
]

(1, "Alice", 30)
(2, "Bob", 24)
(3, "Charlie", 28)
```

`fetchall()`

`fetchall()` is used to obtain every row returned by the executed query.
`fetchall()` භාවිතා කරනුයේ ක්‍රියාත්මක කළ විමසුම මගින් ආපසු ලැබෙන සෑම පේළියක්ම ලබා ගැනීමටය.

`fetchall()` returns a list of tuples.
`fetchall()` tuples වලින් සමන්විත ලැයිස්තුවක් ආපසු ලබා දෙයි.

Each tuple represents a single row in the result set, with each element in the tuple corresponding to a column in the table.
සෑම tuple එකක්ම result set එකෙහි තනි පේළියක් නියෝජනය කරයි, tuple එකෙහි එක් එක් අවයවය වගුවේ තීරුවකට අනුරූප වේ.

Common Methods

Method	Description
`fetchone()`	Retrieves one row at a time වරකට එක් පේළියක් ලබා ගනී
`fetchmany(size)`	Retrieves a specific number of rows නිශ්චිත පේළි ගණනක් ලබා ගනී
Cursor Iteration	Loops directly over the cursor කර්සරය මත සෘජුවම පුණරාවර්තනය වේ

9. Updating Data

```
update_query = """
      UPDATE employees
      SET age = %s WHERE name = %s
"""
data = (35, "Alice")
cursor.execute(update_query, data)
conn.commit()

print(cursor.rowcount, "row(s)
were updated.")
```

1

10. Deleting Data

```
delete_query = "DELETE FROM
employees WHERE name = %s"

data = ("Bob",)
cursor.execute(delete_query, data)
conn.commit()

print(cursor.rowcount, "row(s)
were deleted.")
```

1

`rowcount`

The `rowcount` attribute of a cursor returns the number of rows affected by the last executed SQL statement.
කර්සරයක `rowcount` උපලක්ෂණය මගින් අවසන් වරට ක්‍රියාත්මක කළ SQL ප්‍රකාශයෙන් බලපෑමට ලක් වූ පේළි ගණන කියක්ද යන්න ලබා දෙයි.

11. Closing `cursor()` and `conn()`

```
cursor.close()
conn.close()
```

The `close()` method on a cursor releases the resources tied to that cursor object.
Cursor එකක `close()` method එක මගින් එම cursor object එකට බැඳුනු සම්පත් මුදාහරියි.

The `close()` method on the database connection is used to terminate the session between the Python program and the database.
පයිතන් ක්‍රමලේඛය සහ දත්ත සමුදාය අතර සැසිය අවසන් කිරීමට දත්ත සමුදා සම්බන්ධතාවය මත `close()` method එක භාවිතා කරයි.

Not closing `cursor()` and `conn()` can lead to consumption of resources.
`cursor()` සහ `conn()` වසා නොදැමීම සම්පත් පරිභෝජනයට හේතු විය හැක.

(2) SQLite connection

SQLite is a lightweight database that is built into Python, so no additional driver installation is required.
SQLite යනු පයිතන් තුළ ගොඩනගා ඇති සැහැල්ලු දත්ත සමුදායකි, එබැවින් අමතර ධාවක ස්ථාපනයක් අවශ්‍ය නොවේ.

In SQLite, there is no need to specify host, username, password, or database name because SQLite is a file-based database.
SQLite හි, ධාරක, පරිශීලක නාමය, මුරපදය, හෝ දත්ත සමුදායේ නම සඳහන් කිරීමට අවශ්‍ය නැත, මන්ද SQLite යනු ගොනු-පාදක දත්ත සමුදායකි.

The only required information is the path to the SQLite database file.
අවශ්‍ය එකම තොරතුරු වන්නේ SQLite දත්ත සමුදා ගොනුව වෙත යන මාර්ගයයි.

SQLite doesn't run on a server, so specifying a host or port is unnecessary.
SQLite සේවාදායකයක් මත ධාවනය නොවේ, එබැවින් ධාරකයක් හෝ කෙවෙනියක් නියම කිරීම අවශ්‍ය නොවේ.

1. Importing the SQLite Library

```
import sqlite3
```

2. Creating a Connection

For SQLite, the database is often a single file. If the specified file does not exist, it will be created automatically:
SQLite සඳහා, දත්ත සමුදාය බොහෝ විට තනි ගොනුවකි. එම නිශ්චිත ගොනුව නොපවතියි නම්, එය ස්වයංක්‍රීයව සාදනු ලැබේ:

```
conn =
sqlite3.connect("example.db")
```

3. Creating a cursor

```
cursor = conn.cursor()
```

4. Creating a Table and Inserting Data

```
cursor.execute('''
      CREATE TABLE employees (
         id INTEGER PRIMARY KEY,
         name TEXT,
         age INTEGER
      )
''')

query = '''
INSERT INTO employees (name, age)
VALUES (?, ?)
'''
values = ("Alice", 30)
cursor.execute(query, values)
```

5. Committing changes

```
conn.commit()
```

6. Fetching Data

```
cursor.execute("SELECT * FROM
employees")

results = cursor.fetchall()

for row in results:
    print(row)
```

7. Closing the Connection

```
conn.close()
```

NOTE:
All the other operations done with the MySQL database can also be done with an SQLite database.
MySQL දත්ත සමුදාය සමඟ සිදු කරන අනෙකුත් සියලුම මෙහෙයුම් SQLite දත්ත සමුදායකින්ද සිදු කළ හැක.

ictfromabc.com

Sorting Algorithms in Python
පයිතන් හි වර්ග කිරීමේ ඇල්ගොරිතම

Sorting algorithms are fundamental in computer science for arranging elements in a specific order.
නිශ්චිත අනුපිළිවෙලකට මූලාංග සැකසීම සඳහා වර්ග කිරීමේ ඇල්ගොරිතම පරිගණක විද්‍යාවේ මූලික වේ.

They play a crucial role in various applications, from organizing data for efficient searching to optimizing performance in algorithms that rely on sorted data.
ඒවා කාර්යක්ෂම සෙවීම සඳහා දත්ත සංවිධානය කිරීමේ සිට වර්ග කළ දත්ත මත රඳා පවතින ඇල්ගොරිතමවල කාර්ය සාධනය ප්‍රශස්ත කිරීම දක්වා විවිධ යෙදුම්වල තීරණාත්මක කාර්යභාරයක් ඉටු කරයි.

Example:
Suppose we have an unsorted list of integers:
අප සතුව නිඛිලවල වර්ග නොකළ ලැයිස්තුවක් ඇතැයි සිතමු:

```
unsorted_list = [5, 2, 8, 1, 7]
```

Using a sorting algorithm, we can rearrange this list into ascending order:
වර්ග කිරීමේ ඇල්ගොරිතමයක් භාවිතයෙන්, අපට මෙම ලැයිස්තුව ආරෝහණ අනුපිළිවෙලට නැවත සකස් කළ හැක:

Suppose sorted () is the function for sorting a specific algorithm
Sorted() යනු නිශ්චිත ඇල්ගොරිතමයක් වර්ග කිරීම සඳහා වූ ශ්‍රිතය යැයි සිතමු

```
sorted_list=sorted(unsorted_list)
print("Sorted list:", sorted_list)
```

```
[1,2,5,7,8]
```

Understanding Time Complexity
කාල සංකීර්ණත්වය අවබෝධ කර ගැනීම

Time complexity is a measure of how an algorithm performs as the size of its input grows.
කාල සංකීර්ණත්වය යනු ඇල්ගොරිතමයක් එහි ආදානයේ ප්‍රමාණය වර්ධනය වන ආකාරය පිළිබඳ මිනුමක් වේ.

It helps evaluate the efficiency of sorting algorithms.
එය ඇල්ගොරිතම වර්ග කිරීමේ කාර්යක්ෂමතාවය ඇගයීමට උපකාරී වේ.

Common Time Complexities:
පොදු කාල සංකීර්ණ:

O(1)	Constant time. ස්ථාවර කාලය .
O(n)	Linear time. රේඛීය කාලය.
O(n^2)	Quadratic time චතුරස්රාකාර කාලය.
O(log n)	Logarithmic time ලඝුගණක කාලය.
O(n log n)	Linearithmic time රේඛීය වේලාව

Bubble Sort

Bubble sort is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order.
Bubble sort යනු ලැයිස්තුව හරහා නැවත නැවතත් පියවර ගන්නා, යාබද මූලද්‍රව්‍ය සංසන්දනය කර, ඒවා වැරදි අනුපිළිවෙලක තිබේ නම් ඒවා මාරු කරන සරල වර්ග කිරීමේ ඇල්ගොරිතමයකි.

1. **Starting at the Beginning**

   Begin at the first element of the list.
   ලැයිස්තුවේ පළමු අංගයෙන් ආරම්භ කරන්න.

2. **Comparing Adjacent Elements**
   යාබද මූලාංග සංසන්දනය කිරීම

   Compare the current element with the next element in the list.
   වත්මන් මූලාංග ලැයිස්තුවේ ඊළඟ මූලාංගය සමඟ සසඳන්න.

3. **Swapping Elements**
   මූලාංග මාරු කිරීම

   If the current element is greater than the next element (for ascending order), swap them. This way, the larger element "bubbles up" to its correct position.
   වත්මන් මූලාංගය ඊළඟ මූලාංගයට වඩා වැඩි නම් (ආරෝහණ අනුපිළිවෙල සඳහා), ඒවා මාරු කරන්න. මේ ආකාරයෙන්, විශාල මූලාංග එහි නිවැරදි ස්ථානයට "බුබුලු" කරයි.

4. **Move to the Next Pair**
   ඊළඟ යුගලයට යන්න

   Move to the next pair of adjacent elements and repeat the comparison and swapping process.
   ඊළඟට යාබද මූලාංග යුගලයට ගෙන ගොස් සංසන්දනය සහ හුවමාරු කිරීමේ ක්‍රියාවලිය නැවත කරන්න.

5. **Repeat the Process**
   ක්‍රියාවලිය නැවත කරන්න

   Continue this process for the entire list. After each pass through the list, the largest unsorted element will be placed in its correct position at the end of the list.
   සම්පූර්ණ ලැයිස්තුව සඳහා මෙම ක්‍රියාවලිය දිගටම කරගෙන යන්න. ලැයිස්තුව හරහා එක් එක් මූලාංග සමත් වූ පසු, විශාලතම වර්ග නොකල මූලාංග ලැයිස්තුවේ අවසානයේ එහි නිවැරදි ස්ථානයේ තබනු ඇත.

6. **Reduce the Range**
   පරාසය අඩු කරන්න

   After each pass, the range of elements to be sorted decreases because the last elements are already sorted. Adjust the range for the next pass.
   එක් එක් සමත් වූ පසු, අවසාන මූලාංග දැනටමත් අනුපිළිවෙලට සකසා ඇති නිසා වර්ග කළ යුතු මූලාංග පරාසය අඩු වේ. ඊළඟ අවස්ථාව සඳහා පරාසය සීරුමාරු කරන්න.

7. **Finish When Sorted**

   Repeat the process until no more swaps are needed. When a pass is made without any swaps, the list is sorted.
   තවත් හුවමාරු කිරීම් අවශ්‍ය නොවන තෙක් ක්‍රියාවලිය නැවත කරන්න. කිසිදු හුවමාරුවකින් තොරව අවසථාවක් අවසන් වූ පසු, ලැයිස්තුව අනුපිළිවෙලට සකසනු ලැබේ.

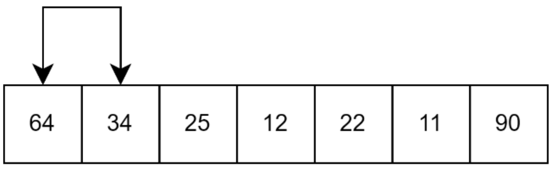
Example:

Suppose we have the following list to sort in ascending order:.
ආරෝහණ අනුපිළිවෙලින් වර්ග කිරීමට පහත ලැයිස්තුව අප සතුව ඇතැයි සිතමු

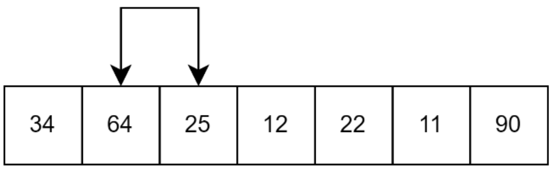
`[64, 34, 25, 12, 22, 11, 90]`

64	34	25	12	22	11	90

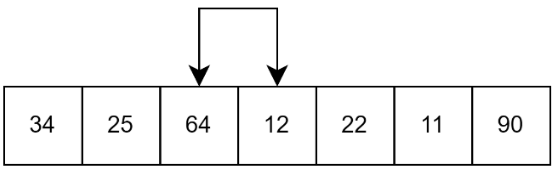
First Pass:
Compare 64 and 34. Swap because 64 > 34. List becomes [34, 64, 25, 12, 22, 11, 90].
64 සහ 34 සසඳන්න. 64 > 34 නිසා හුවමාරු කරන්න. ලැයිස්තුව [34, 64, 25, 12, 22, 11, 90] බවට පත් වේ.



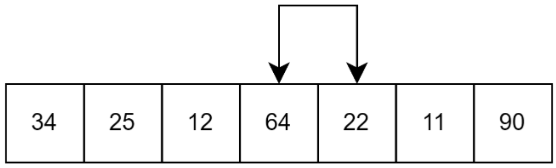
Compare 64 and 25. Swap because 64 > 25. List becomes [34, 25, 64, 12, 22, 11, 90].
64 සහ 25 සසඳන්න. 64 > 25 නිසා හුවමාරු කරන්න. ලැයිස්තුව [34, 25, 64, 12, 22, 11, 90] බවට පත් වේ.



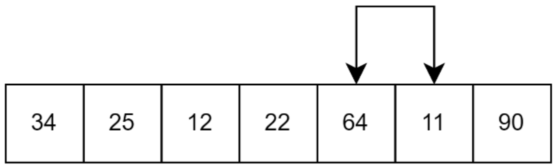
Compare 64 and 12. Swap because 64 > 12. List becomes [34, 25, 12, 64, 22, 11, 90].
64 සහ 12 සසඳන්න. 64 > 12 නිසා හුවමාරු කරන්න. ලැයිස්තුව [34, 25, 12, 64, 22, 11, 90] බවට පත් වේ.



Compare 64 and 22. Swap because 64 > 22. List becomes [34, 25, 12, 22, 64, 11, 90].
64 සහ 22 සසඳන්න. 64 > 22 නිසා හුවමාරු කරන්න. ලැයිස්තුව [34, 25, 12, 22, 64, 11, 90] බවට පත් වේ.



Compare 64 and 11. Swap because 64 > 11. List becomes [34, 25, 12, 22, 11, 64, 90].
64 සහ 11 සසඳන්න. 64 > 11 නිසා හුවමාරු කරන්න. ලැයිස්තුව [34, 25, 12, 22, 11, 64, 90] බවට පත් වේ.

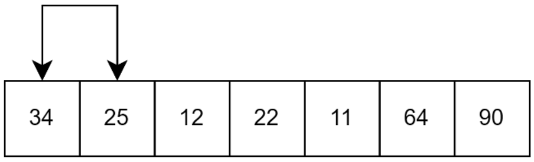


Compare 64 and 90. No swap needed.
64 සහ 90 සසඳන්න. හුවමාරුව අවශ්‍ය නොවේ.

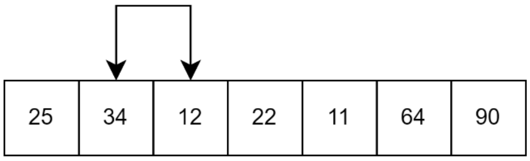
34	25	12	22	11	64	90

Second Pass:

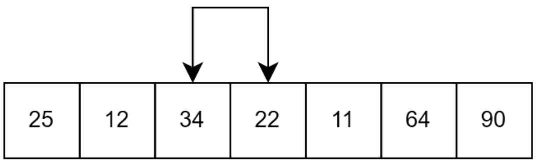
Compare 34 and 25. Swap because 34 > 25. List becomes [25, 34, 12, 22, 11, 64, 90].
34 සහ 25 සසඳන්න. 34 > 25 නිසා හුවමාරු කරන්න. ලැයිස්තුව [25, 34, 12, 22, 11, 64, 90] බවට පත් වේ.



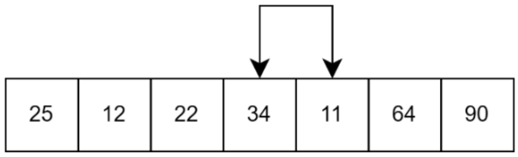
Compare 34 and 12. Swap because 34 > 12. List becomes [25, 12, 34, 22, 11, 64, 90].
34 සහ 12 සසඳන්න. 34 > 12 නිසා හුවමාරු කරන්න. ලැයිස්තුව [25, 12, 34, 22, 11, 64, 90] බවට පත් වේ.



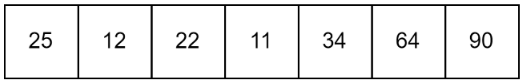
Compare 34 and 22. Swap because 34 > 22. List becomes [25, 12, 22, 34, 11, 64, 90].
34 සහ 22 සසඳන්න. 34 > 22 නිසා හුවමාරු කරන්න. ලැයිස්තුව [25, 12, 22, 34, 11, 64, 90] බවට පත් වේ.



Compare 34 and 11. Swap because 34 > 11. List becomes [25, 12, 22, 11, 34, 64, 90].
34 සහ 11 සසඳන්න. 34 > 11 නිසා හුවමාරු කරන්න. ලැයිස්තුව [25, 12, 22, 11, 34, 64, 90] බවට පත් වේ.

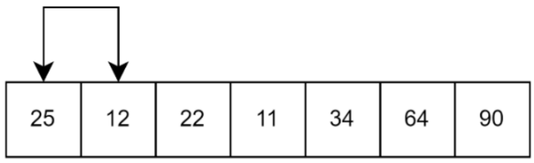


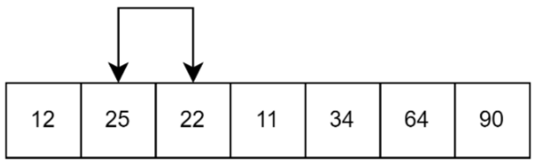
Compare 34 and 64. No swap needed.
34 සහ 64 සසඳන්න. හුවමාරුව අවශ්‍ය නොවේ.

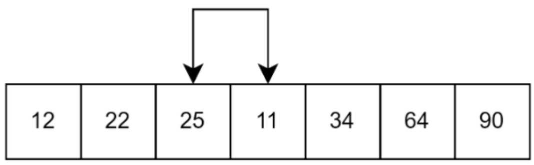


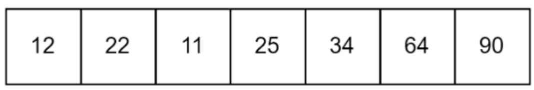
Subsequent Passes:
Continue this process until no swaps are made in a pass.
අවස්ථාවක හුවමාරුවක් සිදු නොවන තෙක් මෙම ක්‍රියාවලිය දිගටම කරගෙන යන්න.

Third Pass:

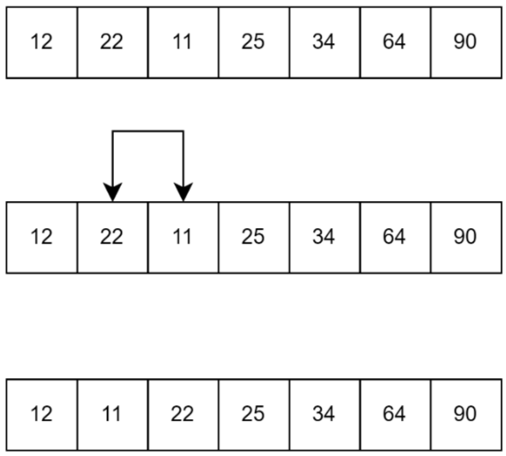




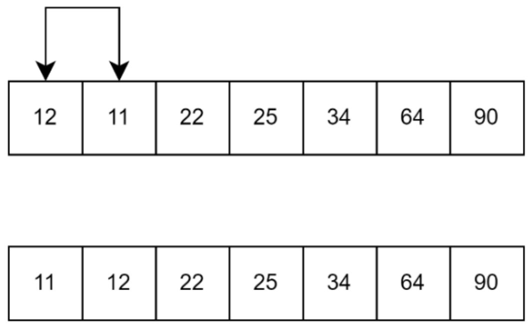




Fourth Pass:



Fifth Pass:



Python Implementation:

1. Using a for loop (Basic)

```
def bubble_sort_for(arr):
    n = len(arr)
    for i in range(n):
        for j in range(n - 1 - i):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
```

2. Using a while loop

```
def bubble_sort_while(arr):
    n = len(arr)
    i = 0
    while i < n:
        j = 0
        while j < n - 1 - i:
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
            j += 1
        i += 1
    return arr
```

3.Using a for loop with dynamic range

```
def bubble_sort_dynamic_range(arr):
    n = len(arr)
    for I in range(n):
        for j in range(n – 1 – i):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
```

4. Using a for loop with a flag (early exit)

```
def bubble_sort_for_flag(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(n - 1 - i):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        if not swapped:
            break
    return arr
```

Time Complexity

Worst Case (O(n^2)): Occurs when the array is sorted in reverse order, requiring maximum swaps.
අරාව ප්‍රතිලෝම අනුපිළිවෙලට වර්ග කළ විට සිදු වේ, උපරිම හුවමාරු කිරීම් අවශ්‍ය වේ.

Best Case (O(n)): Occurs when the array is already sorted, requiring minimum passes through the array.
අරාව දැනටමත් අනුපිළිවෙලට සකසා ඇති විට සිදු වේ, අරාව හරහා අවම ප්‍රවේශයන් අවශ්‍ය වේ.

Average Case (O(n^2)): Typically, the algorithm performs a number of comparisons proportional to the square of the number of elements.
සාමාන්‍යයෙන්, ඇල්ගොරිතම මූලාංග සංඛ්‍යාවේ වර්ගයට සමානුපාතිකව සැසඳීම් ගණනාවක් සිදු කරයි.

Possible Errors:

Index Out of Range: Accessing an element beyond the array bounds.
අරාව සීමාවෙන් ඔබ්බට මූලාංගයකට ප්‍රවේශ වීම.

Type Error: Comparing elements that are not of comparable types.සංසන්දනාත්මක වර්ග නොවන මූලාංග සංසන්දනය කිරීම

Selection Sort

Selection sort is another simple sorting algorithm, which works by repeatedly selecting the minimum (or maximum) element from the unsorted portion of the list and moving it to the beginning (or end). Selection sort යනු තවත් සරල වර්ග කිරීමේ ඇල්ගොරිතමයකි, එය ලැයිස්තුවේ වර්ග නොකළ කොටසෙන් අවම (හෝ උපරිම) මූලද්‍රව්‍යය නැවත නැවතත් තෝරා එය ආරම්භයට (හෝ අවසානයට) ගෙන යාමෙන් ක්‍රියා කරයි.

Starting at the Beginning: Start with the first element of the list.
ආරම්භයේ සිට: ලැයිස්තුවේ පළමු අංගයෙන් ආරම්භ කරන්න.

1. **Finding the Minimum Element**
   අවම මූලාංග සොයා ගැනීම

   Find the minimum element from the unsorted portion of the list.
   ලැයිස්තුවේ වර්ග නොකළ කොටසෙන් අවම මූලාංගය සොයන්න.

2. **Swapping Elements**
   මූලාංග මාරු කිරීම

   Swap the found minimum element with the first element of the unsorted portion.
   සොයාගත් අවම මූලාංගය වර්ග නොකළ කොටසේ පළමු මූලාංගය සමඟ මාරු කරන්න.

3. **Move to the Next Position**
   මීළඟ ස්ථානයට යන්න

   Move the boundary of the sorted and unsorted portions of the list by one position to the right.
   ලැයිස්තුවේ වර්ග කළ සහ නොසැලකූ කොටස්වල මායිම එක් ස්ථානයකින් දකුණට ගෙන යන්න.

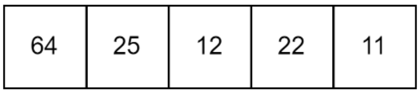
4. **Repeat the Process**
   ක්‍රියාවලිය නැවත සිදු කරන්න

   Repeat the process until the entire list is sorted.
   සම්පූර්ණ ලැයිස්තුවම වර්ග කරන තෙක් ක්‍රියාවලිය නැවත කරන්න.

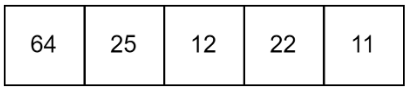
Example

Suppose we have the following list to sort in ascending order:
ආරෝහණ අනුපිළිවෙලින් වර්ග කිරීමට පහත ලැයිස්තුව අප සතුව ඇතැයි සිතමු:

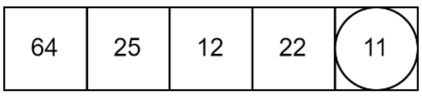
`[64, 25, 12, 22, 11]`



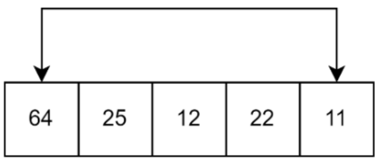
First Pass:
The unsorted list is [64, 25, 12, 22, 11].
වර්ග නොකළ ලැයිස්තුව [64, 25, 12, 22, 11] වේ.



Find the minimum element in the unsorted list. The minimum is 11.
වර්ග නොකළ ලැයිස්තුවේ අවම මූලාංගය සොයන්න. අවම අගය 11 කි.



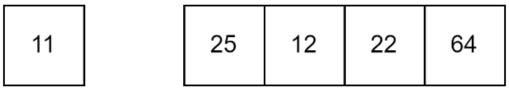
Swap 11 with the first element (64). List becomes [11, 25, 12, 22, 64].
පළමු මූලාංග (64) සමඟ 11 මාරු කරන්න. ලැයිස්තුව [11, 25, 12, 22, 64] බවට පත් වේ.



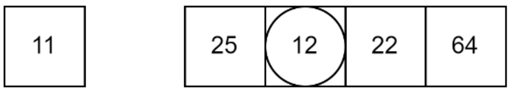
ictfromabc.com

Second Pass:

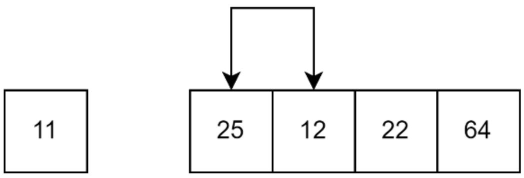
The unsorted list is [25, 12, 22, 64] (excluding the first element which is already sorted).
වර්ග නොකළ ලැයිස්තුව [25, 12, 22, 64] (දැනටමත් වර්ග කර ඇති පළමු මූලාංගය හැර) වේ.



Find the minimum element in the unsorted list. The minimum is 12.
වර්ග නොකළ ලැයිස්තුවේ අවම මූලාංගය සොයන්න. අවම අගය 12 කි.

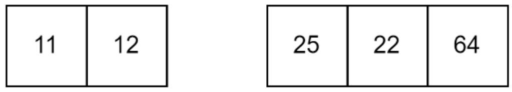


Swap 12 with the first element of the unsorted list (25). List becomes [11, 12, 25, 22, 64].
වර්ග නොකළ ලැයිස්තුවේ (25) පළමු අංගය සමඟ 12 මාරු කරන්න. ලැයිස්තුව [11, 12, 25, 22, 64] බවට පත් වේ.

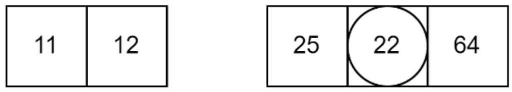


Third Pass:

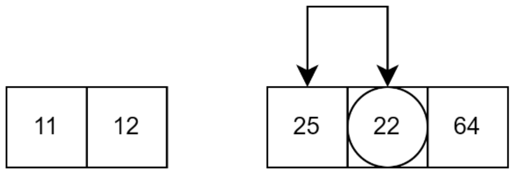
The unsorted list is [25, 22, 64].
වර්ග නොකළ ලැයිස්තුව [25, 22, 64] වේ.



Find the minimum element in the unsorted list. The minimum is 22.
වර්ග නොකළ ලැයිස්තුවේ අවම මූලාංගය සොයන්න. අවම අගය 22 කි.

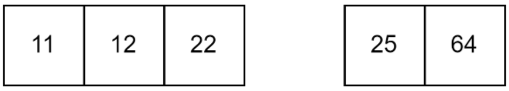


Swap 22 with the first element of the unsorted list (25). List becomes [11, 12, 22, 25, 64].
වර්ග නොකළ ලැයිස්තුවේ (25) පළමු අංගය සමඟ 22 මාරු කරන්න. ලැයිස්තුව [11, 12, 22, 25, 64] බවට පත් වේ.

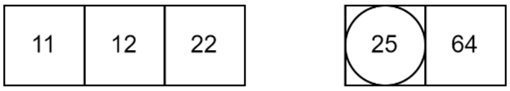


Fourth Pass:

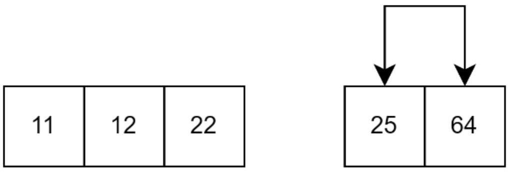
The unsorted list is [25, 64].
වර්ග නොකළ ලැයිස්තුව [25, 64] වේ.



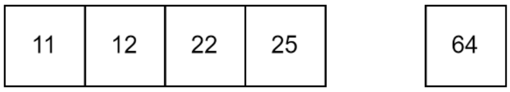
Find the minimum element in the unsorted list. The minimum is 25.
වර්ග නොකළ ලැයිස්තුවේ අවම මූලාංගය සොයන්න. අවම අගය 25 කි.



Swap 25 with the first element of the unsorted list (25). List remains [11, 12, 22, 25, 64].
වර්ග නොකළ ලැයිස්තුවේ (25) පළමු අංගය සමඟ 25 මාරු කරන්න. ලැයිස්තුව වෙනස් නොවේ [11, 12, 22, 25, 64].



Fifth Pass:
The unsorted list is [64].
වර්ග නොකළ ලැයිස්තුව [64] වේ.



Since there's only one element left, it is already sorted.
එක් අංගයක් පමණක් ඉතිරිව ඇති බැවින්, එය දැනටමත් වර්ග කර ඇත.

Python Implementation:

1. Using a for loop (Basic)

```python
def selection_sort_for(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr
```

2.Using a while loop

```python
def selection_sort_while(arr):
    n = len(arr)
    i = 0
    while i < n:
        min_idx = i
        j = i + 1
        while j < n:
            if arr[j] < arr[min_idx]:
                min_idx = j
            j += 1
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
        i += 1
    return arr
```

3. Using a for loop with a flag (early exit)

```
def selection_sort_for_flag(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        swapped = False
        for j in range(i + 1, n):
            if arr[j] < arr[min_idx]:
                min_idx = j
                swapped = True
        if swapped:
            arr[i], arr[min_idx] = arr[min_idx], arr[i]
        else:
            break
    return arr
```

4.Using a for loop with dynamic range

```
def selection_sort_dynamic_range(arr):
    n = len(arr)
    for i in range(n):
        min_idx = i
        for j in range(i + 1, n - i):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr
```

Time Complexity:

Worst Case (O(n^2)): Occurs when the array is sorted in reverse order or nearly sorted, requiring maximum comparisons.
අරාව ප්‍රතිලෝම අනුපිළිවෙලට හෝ ආසන්න වශයෙන් අනුපිළිවෙලට වර්ග කළ විට, උපරිම සැසඳීම් අවශ්‍ය වේ

Best Case (O(n^2)): Occurs when the array is already sorted, still requiring the same number of comparisons.
අරාව දැනටමත් වර්ග කර ඇති විට සිදු වේ, තවමත් සමාන සැසඳීම් සංඛ්‍යාවක් අවශ්‍ය වේ.

Average Case (O(n^2)): Generally, the algorithm performs a quadratic number of comparisons and swaps.
සාමාන්‍යයෙන්, ඇල්ගොරිතම සංසන්දනය සහ හුවමාරු කිරීම් චතුරස්‍ර සංඛ්‍යාවක් සිදු කරයි.

Possible Errors
Index Out of Range
Type Error

Insertion Sort

Insertion sort is a straightforward and efficient algorithm for small or nearly sorted datasets.
Insertion sort යනු කුඩා හෝ ආසන්න වශයෙන් වර්ග කළ දත්ත කට්ටල සඳහා සරල සහ කාර්යක්ෂම ඇල්ගොරිතමයකි.

It builds the sorted array one element at a time by comparing each new element to the already sorted elements and inserting it in its correct position.
එය එක් එක් නව මූලද්‍රව්‍ය දැනටමත් වර්ග කර ඇති මූලාංග සමඟ සංසන්දනය කර එහි නිවැරදි ස්ථානයට ඇතුළත් කිරීමෙන් වරකට එක් මූලාංගයක් ගොඩනඟයි.

Steps of Insertion Sort:
Insertion Sort හි පියවර

1. **Start with the First Element**
   පළමු මූලාංගය සමඟින් ආරම්භ කරන්න

   Consider the first element to be a sorted sublist of one element.
   පළමු මූලාංගය එක් මූලාංගයක අනුලේඛනය කළ උප ලැයිස්තුවක් ලෙස සලකන්න.

2. **Pick the Next Element**
   ඊළඟ මූලාංගය තෝරන්න

   Take the next element from the unsorted portion of the list.
   ලැයිස්තුවේ වර්ග නොකළ කොටසින් ඊළඟ මූලාංගය ගන්න.

3. **Find the Correct Position**
   නිවැරදි ස්ථානය සොයන්න

   Compare this element with the elements in the sorted sublist and find the correct position where it should be inserted.
   මෙම මූලාංගය අනුලේඛනගත කළ උප ලැයිස්තුවේ ඇති මූලාංග සමඟ සසඳා එය ඇතුළත් කළ යුතු ස්ථානය සොයා ගන්න.

4. **Shift Elements**
   මූලාංග මාරු කරන්න

   Shift all elements in the sorted sublist that are larger than the element to be inserted one position to the right to make space.
   අනුලේඛනගත කර ඇති උප ලැයිස්තුවේ ඇති සියලුම මූලාංග ස්ථානගත කිරීම සඳහා එක් ස්ථානයකට ඇතුළු කිරීමට අවශ්‍ය මූලාංගයට වඩා විශාල ඒවා ඉඩ ඇති කිරීමට දකුණට මාරු කරන්න.

5. **Insert the Element**
   මූලාංග ඇතුල් කරන්න

   Insert the element at its correct position.
   මූලාංගය එහි නිවැරදි ස්ථානයේ ඇතුල් කරන්න.

6. **Repeat the Process**
   ක්‍රියාවලිය නැවත සිදු කරන්න

   Move to the next element in the unsorted portion and repeat the process until the entire list is sorted.
   වර්ග නොකළ කොටසේ ඊළඟ මූලාංගය වෙත ගොස් සම්පූර්ණ ලැයිස්තුවම අනුපිළිවෙලට සකස් කරන තෙක් ක්‍රියාවලිය නැවත කරන්න.

Example

Suppose we have the following list to sort in ascending order:.
ආරෝහණ අනුපිළිවෙලින් වර්ග කිරීමට පහත ලැයිස්තුව අප සතුව ඇතැයි සිතමු:

`[29, 10, 14, 37, 14]`

29	10	14	37	14

First Pass

Sorted sublist: [29]
වර්ග කළ උප ලැයිස්තුව: [29]

29

Unsorted portion: [10, 14, 37, 14]
වර්ග නොකළ කොටස: [10, 14, 37, 14]

10	14	37	14

Pick the next element (10).
මීළඟ මූලාංගය තෝරන්න (10).

10

Find the correct position (before 29).
නිවැරදි ස්ථානය සොයා ගන්න (29 ට පෙර).

Shift 29 to the right.
29 දකුණට මාරු කරන්න.

10	29

Insert 10. List becomes [10, 29, 14, 37, 14].
10 ඇතුළු කරන්න 10. ලැයිස්තුව [10, 29, 14, 37, 14] බවට පත් වේ.

10	29	14	37	14

Second Pass:

Sorted sublist: [10, 29]
වර්ග කළ උප ලැයිස්තුව:[10, 29]

10	29

Unsorted portion: [14, 37, 14]
වර්ග නොකළ කොටස : [14, 37, 14]

14	37	14

Pick the next element (14).
මීළඟ මූලාංගය තෝරන්න (14).

14

Find the correct position (between 10 and 29).
නිවැරදි ස්ථානය සොයා ගන්න (10 සහ 29 අතර).
Shift 29 to the right.
29 දකුණට මාරු කරන්න.

10	14	29

Insert 14. List becomes [10, 14, 29, 37, 14].
14 ඇතුලත් කරන්න . ලැයිස්තුව [10, 14, 29, 37, 14] බවට පත් වේ.

10	14	29	37	14

ictfromabc.com

Third Pass:

Sorted sublist: [10, 14, 29]
වර්ග කළ උප ලැයිස්තුව: [10, 14, 29]

10	14	29

Unsorted portion: [37, 14]
වර්ග නොකළ කොටස: [37, 14]

37	14

Pick the next element (37).
මීළඟ මූලද්‍රව්‍යය තෝරන්න (37).

37

Find the correct position (after 29).
නිවැරදි ස්ථානය සොයා ගන්න (29 න් පසු).

No shifting needed.
මාරු කිරීම අවශ්‍ය නොවේ.

10	14	29	37

Insert 37. List becomes [10, 14, 29, 37, 14].
37 ඇතුළු කරන්න. ලැයිස්තුව [10, 14, 29, 37, 14] බවට පත් වේ

10	14	29	37	14

Fourth Pass:

Sorted sublist: [10, 14, 29, 37]
වර්ග කළ උප ලැයිස්තුව: [10, 14, 29, 37]

10	14	29	37

Unsorted portion: [14]
වර්ග නොකළ කොටස: [14]

14

Pick the next element (14).
මීළඟ මූලාංගය තෝරන්න (14).

14

Find the correct position (between 14 and 29).
නිවැරදි ස්ථානය සොයා ගන්න (14 සහ 29 අතර).

Shift 29 and 37 to the right.
29 සහ 37 දකුණට මාරු කරන්න.

10	14	14	29	37

Insert 14. List becomes [10, 14, 14, 29, 37].
ඇතුළු කරන්න 14. ලැයිස්තුව [10, 14, 14, 29, 37] බවට පත් වේ.

10	14	14	29	37

Python Implementation:

1.Using a for loop (Basic)

```
def insertion_sort_for(arr):
    n = len(arr)
    for i in range(1, n):
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j]>key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr
```

2. Using a while loop

```
def insertion_sort_while(arr):
    n = len(arr)
    i = 1
    while i < n:
        key = arr[i]
        j = i - 1
        while j >= 0 and arr[j]>key:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
        i += 1
    return arr
```

3.Using a for loop with dynamic range

```
def sort_dynamic_range(arr):
    n = len(arr)
    for i in range(1, n):
        key = arr[i]
        j = i - 1
        for _ in range(i):
            if j >= 0 and arr[j]>key:
                arr[j + 1] = arr[j]
                j -= 1
        arr[j + 1] = key
    return arr
```

4. Using Nested for loops

```
def insertion_sort_nested_for(arr):
    n = len(arr)
    for i in range(1, n):
        key = arr[i]
        for j in range(i - 1,-1,-1):
            if arr[j] > key:
                arr[j + 1] = arr[j]
            else:
                arr[j + 1] = key
                break
        else:
            arr[0] = key
    return arr
```

Time Complexity:

Worst Case (O(n^2)): Occurs when the array is sorted in reverse order, requiring maximum shifts. අරාව ප්‍රතිලෝම අනුපිළිවෙලට වර්ග කළ විට සිදු වේ, උපරිම මාරුවීම් අවශ්‍ය වේ.

Best Case (O(n)): Occurs when the array is already sorted, requiring minimum comparisons.
අවම සැසඳීම් අවශ්‍ය වන අරාව දැනටමත් වර්ග කර ඇති විට සිදු වේ.

Average Case (O(n^2)): Typically, the algorithm performs a quadratic number of comparisons and shifts. සාමාන්‍යයෙන්, ඇල්ගොරිතම සංසන්දනය සහ මාරුවීම් චතුරස්‍ර සංඛ්‍යාවක් සිදු කරයි.

Possible Errors:

Index Out of Range
Type Error

ictfromabc.com

Flowcharts
ගැලීම් සටහන්

A **flowchart** is a graphical representation of an algorithm or a process, showing the sequence of steps and how they are connected. It uses various symbols to represent different actions and decisions, making it easier to visualize the flow of logic.
ගැලීම් සටහනක් යනු ඇල්ගොරිතමයක හෝ ක්‍රියාවලියක චිත්‍රක නිරූපණයකි, පියවර අනුපිළිවෙල සහ ඒවා සම්බන්ධ වන ආකාරය පෙන්වයි. එය විවිධ ක්‍රියාවන් සහ තීරණ නිරූපණය කිරීමට විවිධ සංකේත භාවිතා කරයි, තර්කයේ ගැලීම් දෘශ්‍යමාන කිරීම පහසු කරයි.

Pseudocodes

Pseudocode is an informal way of writing an algorithm using plain language (usually a mix of English and basic programming concepts) to describe the steps involved in a process. It doesn't adhere to the syntax of any particular programming language but follows a structured format.
ව්‍යාජ කේත යනු ක්‍රියාවලියකට සම්බන්ධ පියවර විස්තර කිරීම සඳහා සරල භාෂාව (සාමාන්‍යයෙන් ඉංග්‍රීසි සහ මූලික ක්‍රමලේඛන සංකල්ප මිශ්‍රණයක්) භාවිතයෙන් ඇල්ගොරිතමයක් ලිවීමේ අවිධිමත් ක්‍රමයකි. එය කිසියම් ක්‍රමලේඛන භාෂාවක වාක්‍ය ඛණ්ඩයට අනුගත නොවන නමුත් ව්‍යුහගත ආකෘතියක් අනුගමනය කරයි.

Uses of Flowcharts and Pseudocode
ගැලීම් සටහන් සහ ව්‍යාජ කේත භාවිතය

01. Planning and Designing Programs
වැඩසටහන් සැලසුම් කිරීම

Before Coding
කේතනය කිරීමට පෙර

Both flowcharts and pseudocode help programmers outline the steps of a program or algorithm before writing any actual code. This simplifies the programming process by giving a clear picture of what the program will do, making it easier to identify any issues or improvements early on.
ගැලීම් සටහන් සහ ව්‍යාජ කේත යන දෙකම ක්‍රමලේඛකයින්ට කිසියම් සත්‍ය කේතයක් ලිවීමට පෙර වැඩසටහනක හෝ ඇල්ගොරිතමයක පියවර ගෙනහැර දැක්වීමට උපකාරී වේ. මෙමගින් ක්‍රමලේඛන ක්‍රියාවලිය සරල කරයි, වැඩසටහන මඟින් සිදු කරනු ලබන දේ පිළිබඳ පැහැදිලි අදහසක් ලබා දීමෙන්, ඕනෑම ගැටළුවක් හෝ වැඩිදියුණු කිරීම් කල් ඇතිව හඳුනා ගැනීම පහසු කරයි.

02. Communication Across Different Programming Languages
විවිධ ක්‍රමලේඛන භාෂා හරහා සන්නිවේදනය

Between Programmers
ක්‍රමලේඛකයින් අතර

When two programmers are familiar with different languages (e.g., Python and Java), flowcharts and pseudocode act as a common ground. A flowchart visually illustrates the process, while pseudocode provides a language-neutral explanation of the program’s logic. This allows both programmers to understand the core idea without needing to know the exact syntax of the other's language.
ක්‍රමලේඛකයින් දෙදෙනෙකු විවිධ භාෂා (උදා: පයිතන් සහ ජාවා) හුරුපුරුදු වූ විට, ගැලීම් සටහන සහ ව්‍යාජ කේත පොදු පදනමක් ලෙස ක්‍රියා කරයි. ගැලීම් සටහනක් ක්‍රියාවලිය දෘශ්‍යව විදහා දක්වන අතර ව්‍යාජ කේතය මඟින් වැඩසටහනේ තර්කනය පිළිබඳ භාෂා මධ්‍යස්ථ පැහැදිලි කිරීමක් සපයයි. මෙය ක්‍රමලේඛකයින් දෙදෙනාටම අනෙකාගේ භාෂාවේ නියම වාක්‍ය ඛණ්ඩය දැන ගැනීමට අවශ්‍ය නොවී මූලික අදහස තේරුම් ගැනීමට ඉඩ සලසයි.

04. Explaining to Non-Programmers
ක්‍රමලේඛකරුවන් නොවන අයට පැහැදිලි කිරීම

Simplifying Complex Logic
සංකීර්ණ තර්කනය සරල කිරීම

Flowcharts, with their visual symbols, make it easier to explain the sequence of steps in a program to someone who doesn't understand programming.
ප්‍රවාහ සටහන්, ඒවායේ දෘශ්‍ය සංකේත සමඟින්, ක්‍රමලේඛනය නොතේරෙන කෙනෙකුට වැඩසටහනක පියවර අනුපිළිවෙල පැහැදිලි කිරීම පහසු කරයි.

Pseudocode, written in plain language, can also describe the logic step by step, making the program more accessible to non-technical stakeholders like managers or clients.
සරල භාෂාවෙන් ලියා ඇති ව්‍යාජ කේත, තාර්කිකත්වය පියවරෙන් පියවර විස්තර කළ හැකි අතර, කළමනාකරුවන් හෝ සේවාදායකයින් වැනි තාක්ෂණික නොවන පාර්ශවකරුවන්ට වැඩසටහන වඩාත් ප්‍රවේශ විය හැකිය.

Example: If you need to explain a login process, a flowchart can show the flow of actions (input username, input password, check credentials, etc.), while pseudocode can describe the steps in plain English ("If password is correct, allow access; otherwise, display error").
ඔබට පිවිසුම් ක්‍රියාවලියක් පැහැදිලි කිරීමට අවශ්‍ය නම්, ගැලීම් සටහනකට ක්‍රියා ප්‍රවාහය පෙන්විය හැක (පරිශීලක නාමය ආදානය , මුරපදය ආදානය , අක්තපත්‍ර පරීක්ෂා කිරීම, ආදිය), ව්‍යාජ කේතයට සරල ඉංග්‍රීසියෙන් පියවර විස්තර කළ හැකිය ("මුරපදය නිවැරදි නම්, ප්‍රවේශයට ඉඩ දෙන්න; එසේ නොමැතිනම්, දෝෂ දර්ශනය කරන්න").

05. Debugging and Problem-Solving
නිදොස් කිරීම සහ ගැටළු විසඳීම

Visual and Textual Representation
දෘෂ්‍ය සහ පාඨමය නිරූපණය

Flowcharts can help identify errors in the flow of a program by showing the logical progression.
තාර්කික ප්‍රගතිය පෙන්වීමෙන් වැඩසටහනක ප්‍රවාහයේ දෝෂ හඳුනා ගැනීමට ගැලීම් සටහන් උපකාරී වේ.

Pseudocode, on the other hand, allows programmers to rethink the logic without getting stuck in syntax, making it easier to spot and fix logical errors or inefficiencies.
අනෙක් අතට, ව්‍යාජ කේත, ක්‍රමලේඛකයින්ට වාක්‍ය ඛණ්ඩයේ සිර නොවී තර්කනය ගැන නැවත සිතා බැලීමට ඉඩ සලසයි, එමඟින් තාර්කික දෝෂ හෝ අකාර්යක්ෂමතාව හඳුනා ගැනීම සහ නිවැරදි කිරීම පහසු කරයි.

06. Documentation

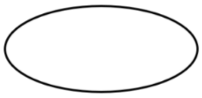
Both flowcharts and pseudocode serve as a form of documentation for future developers or team members.
අනාගත සංවර්ධකයින් හෝ කණ්ඩායම් සාමාජිකයින් සඳහා ප්‍රවාහ ප්‍රස්ථාර සහ ව්‍යාජ කේත යන දෙකම ලේඛන ආකෘතියක් ලෙස සේවය කරයි.

They provide a clear explanation of how the program works, which is especially helpful when the code is complex or written in a language others might not know.
ඔවුන් වැඩසටහන ක්‍රියා කරන ආකාරය පිළිබඳ පැහැදිලි පැහැදිලි කිරීමක් සපයයි, කේතය සංකීර්ණ වූ විට හෝ අන් අය නොදන්නා භාෂාවකින් ලියා ඇති විට එය විශේෂයෙන් උපකාරී වේ.

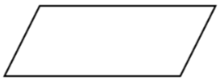
ictfromabc.com

Shapes Used in Flow Charts
ගැලීම් සටහන් වල භාවිතා කරන හැඩතල

Start / End
ආරම්භය / අවසානය

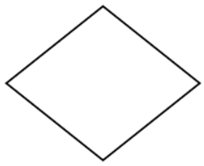


Input / Output
ආදාන / ප්‍රතිදාන

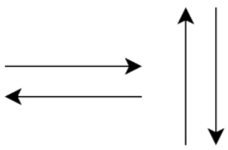


Process
ක්‍රියාවලිය

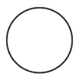
Conditions
කොන්දේසි



Flow Direction
ගැලීම් දිශාව



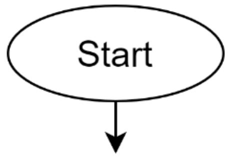
Connectors
සම්බන්ධක



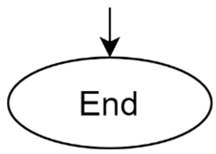
Subroutine

Uses of shapes with flow arrows
ගැලීම් සඳහා ඊතල සහිත හැඩතල භාවිතය

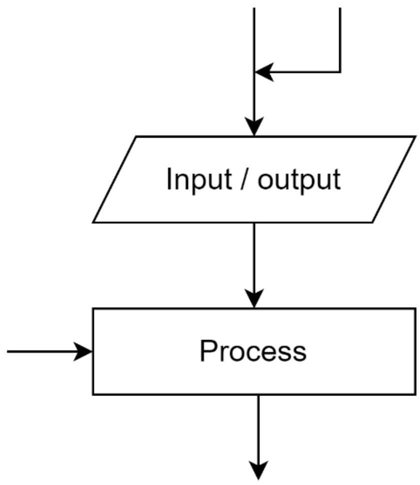
- Start/Begin can only have one arrow out.
  Start/Begin තුළ තිබිය හැක්කේ එක් ඊතලයක් පිටතට පමණි.



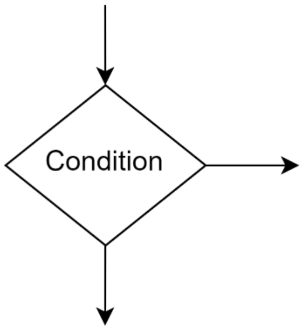
- End / Stop can only have one arrow in.
  End / Stop තුළ තිබිය හැක්කේ එක් ඊතලයක් ඇතුළතට පමණි.



- Inputs, outputs, and processes can have one or more arrows in and only one arrow out.
  ආදාන ප්‍රතිදාන සහ සැකසුම් වලට ඊතල එකක් හෝ කිහිපයක් ඇතුළතට තිබිය හැකි අතර පිටතට ඊතලයක් පමණක් තිබිය හැක.

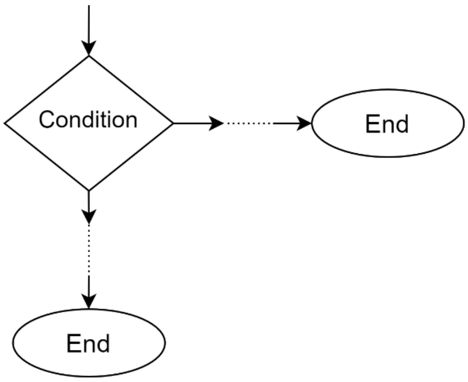


- There must be a condition to split a flow into two.
ප්‍රවාහයක් දෙකට බෙදීමට කොන්දේසියක් තිබිය යුතුය.

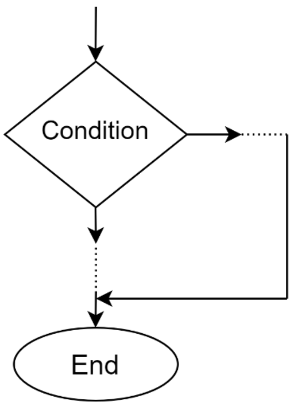


- You can end a two splitted flows with one end or two end s for each flow.
ඔබට එක් එක් ප්‍රවාහය සඳහා එක් කෙළවරක් හෝ අන්ත දෙකකින් බෙදුණු ප්‍රවාහ දෙකක් අවසන් කළ හැක.

Method 01



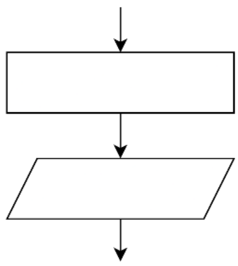
Method 02



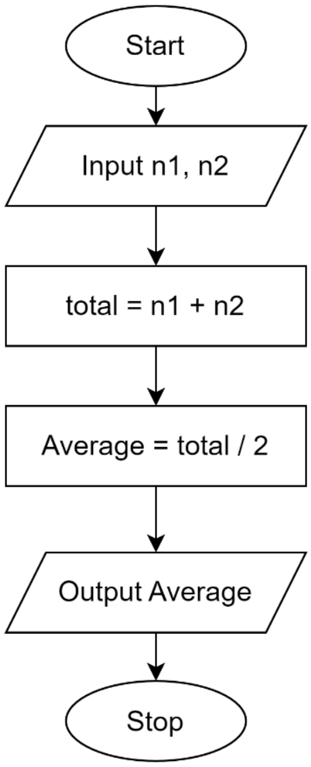
Control Structure
පාලන ව්‍යුහය

1. Sequence
අනුක්‍රමය

Sequence refers to the order in which instructions are executed in a program. In a sequence, the program follows a strict linear path, executing statements one after the other, without any branching or repetition.
අනුක්‍රමය යනු වැඩසටහනක උපදෙස් ක්‍රියාත්මක කරන අනුපිළිවෙලයි. අනුපිළිවෙලක් තුළ, වැඩසටහන දැඩි රේඛීය මාර්ගයක් අනුගමනය කරයි, කිසිදු ශාඛාවක් හෝ පුනරාවර්තනයක් නොමැතිව ප්‍රකාශ එකින් එක ක්‍රියාත්මක කරයි.

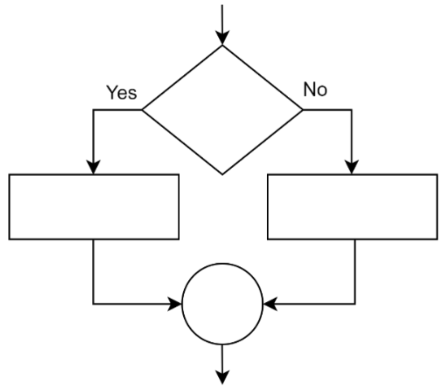


Example.

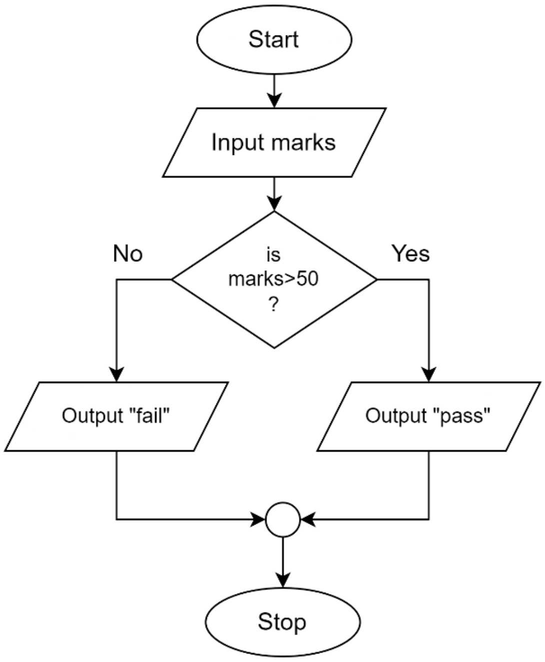


2. Selection
වරණය

Selection allows the program to choose between different paths based on conditions. It involves decision-making, typically through if-else or switch-case statements. Only one path is followed depending on the condition.
වරණය කොන්දේසි මත පදනම්ව විවිධ මාර්ග අතර තෝරා ගැනීමට වැඩසටහනට ඉඩ සලසයි. සාමාන්‍යයෙන් if-else හෝ switch-case ප්‍රකාශ හරහා තීරණ ගැනීම එයට ඇතුළත් වේ. කොන්දේසිය අනුව එක් මාර්ගයක් පමණක් අනුගමනය කරයි.

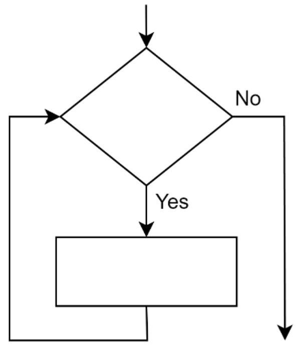


Example

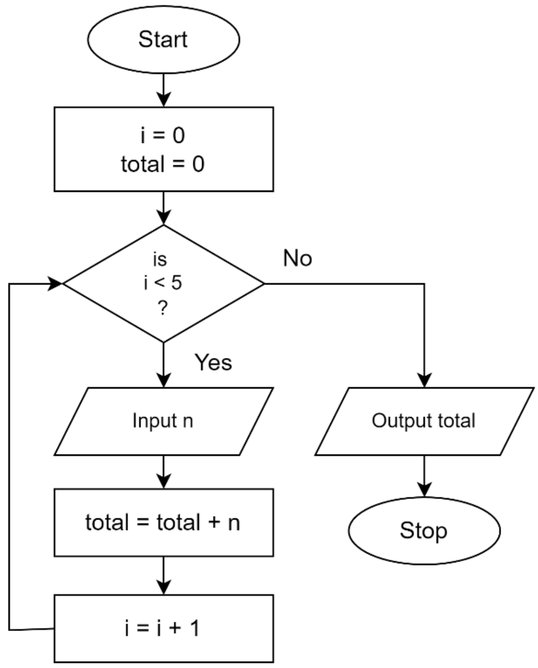


3. Repetition
පුණර්කරණය

Repetition allows a set of instructions to be executed multiple times. This control structure is used to repeat actions until a condition is satisfied or for a set number of times. Loops such as for, while, or do-while are used for this purpose.
පුණර්කරණය මඟින් උපදෙස් මාලාවක් කිහිප වතාවක් ක්‍රියාත්මක කිරීමට ඉඩ සලසයි. මෙම පාලන ව්‍යුහය කොන්දේසියක් තෘප්තිමත් වන තෙක් හෝ නියමිත වාර ගණනක් සඳහා ක්‍රියාවන් නැවත කිරීමට භාවිතා කරයි. for, while, do-while වැනි ලූප මේ සඳහා භාවිතා වේ.

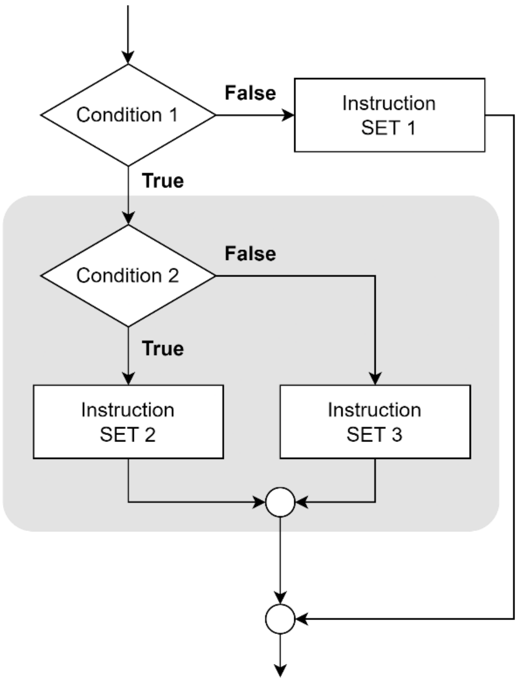


Example



Selection inside Selection
වරණය තුළ වරණය

Selection inside Selection (also known as nested selection) refers to a situation where one selection structure (such as an if-else or switch-case statement) is placed inside another selection structure.
වරණය තුළ වරණය (nested වරණය ලෙසද හැඳින්වේ) යනු එක් තේරීම් ව්‍යුහයක් (if-else හෝ switch-case ප්‍රකාශයක් වැනි) වෙනත් තේරීම් ව්‍යුහයක් තුළ තබා ඇති තත්වයකි.

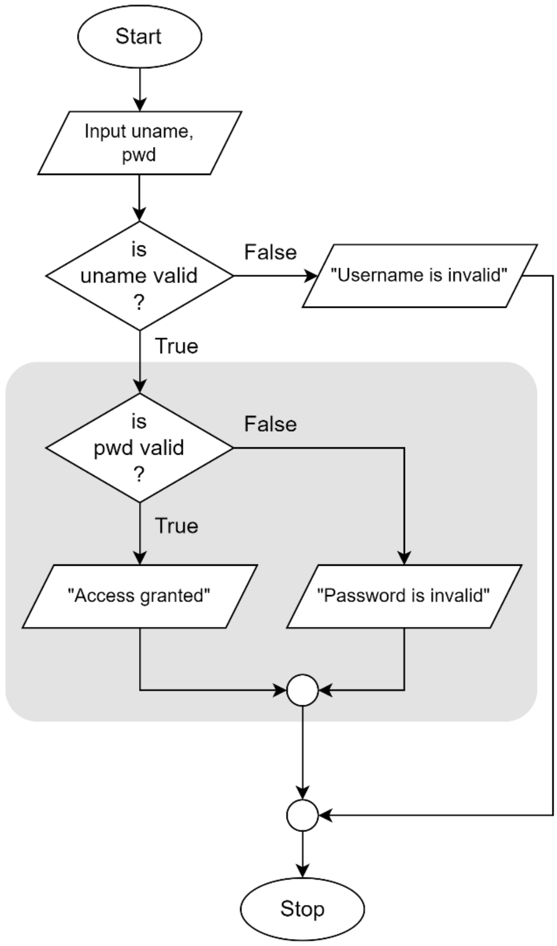


Instruction SET 1 is executed only when condition 1 is FALSE
Instruction SET 1 ක්‍රියාත්මක වන්නේ condition 1 අසත්‍ය වූ විට පමණි

Instruction SET 2 is executed only when both condition 1 and condition 2 are TRUE
Instruction SET 2 ක්‍රියාත්මක වන්නේ condition 1 සහ condition 2 දෙකම සත්‍ය වූ විට පමණි.

Instruction SET 3 is executed only when condition 1 is TRUE and condition 2 is FALSE
Instruction SET 3 ක්‍රියාත්මක වන්නේ condition 1 සත්‍ය සහ condition 2 අසත්‍ය වූ විට පමණි.

Example.



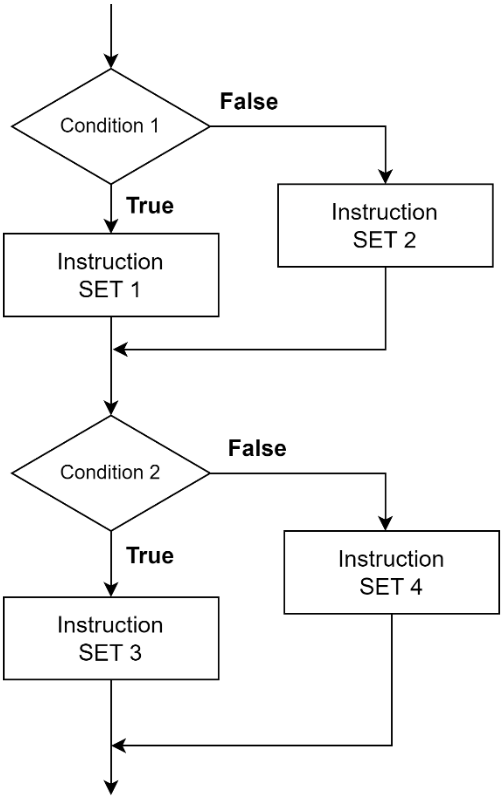
In the above example, the access will be granted only if both username and password are valid.
ඉහත උදාහරණයේදී, ප්‍රවේශය ලබා දෙන්නේ පරිශීලක නාමය සහ මුරපදය යන දෙකම වලංගු නම් පමණි.

If any of them is invalid, access will not be granted.
ඒවායින් එකක් හෝ වලංගු නොවේ නම්, ප්‍රවේශය ලබා නොදේ.

Selection after Selection
වරණයෙන් පසු වරණය

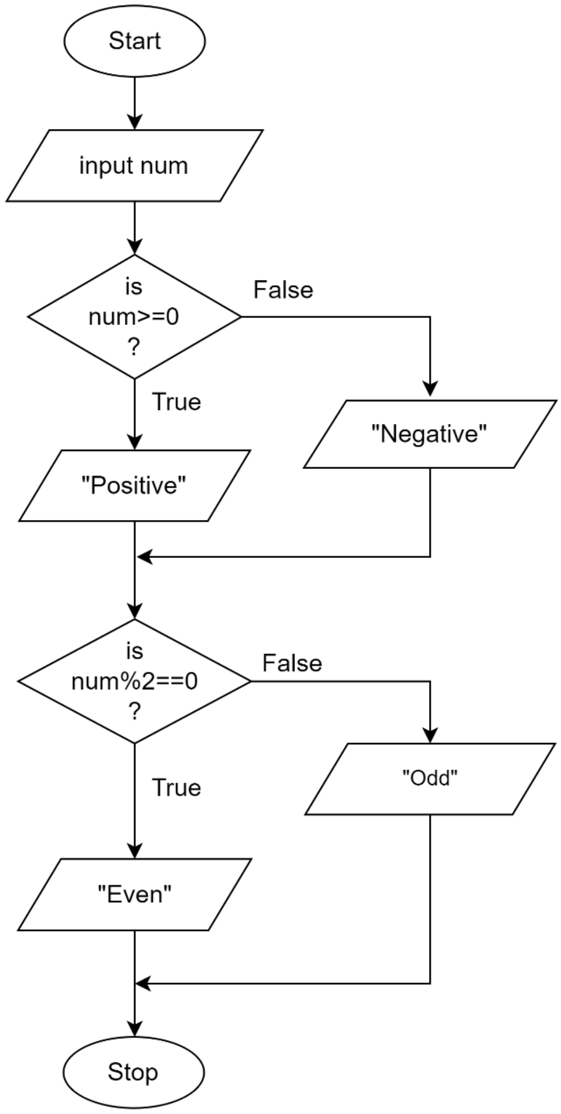
Selection after Selection refers to a scenario where multiple independent selection structures are evaluated one after the other, without any nesting.
වරණයෙන් පසු වරණය යන්නෙන් අදහස් කරන්නේ කිසිදු nesting එකකින් තොරව බහු ස්වාධීන තේරීම් ව්‍යුහයන් එකින් එක ඇගයීමට ලක් කරන අවස්ථාවකි.

Unlike selection inside selection, where one condition is evaluated inside another, in selection after selection, each condition stands alone and is checked sequentially.
වරණය තුළ වරණය මෙන් නොව, වරණයෙන් පසු වරණයේදී, සෑම කොන්දේසියක්ම තනිව පවතින අතර අනුපිළිවෙලින් පරීක්ෂා කරනු ලැබේ.



Here, Condition 1 does not have any effect on Instruction Set 3 and Instruction Set 4
මෙහිදී, Condition 1, Instruction Set 3 සහ Instruction Set 4 කෙරෙහි කිසිදු බලපෑමක් ඇති නොකරයි.

Example.

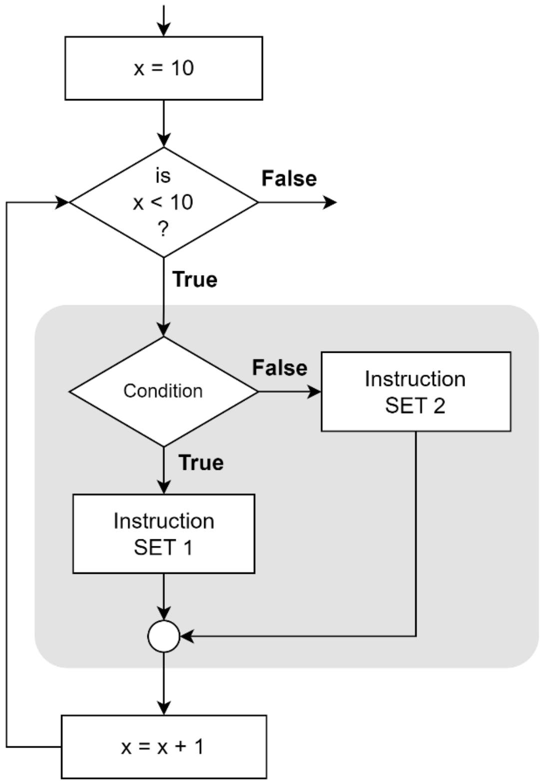


If num is -10, the outputs would be
-10 වුවහොත්, ප්‍රතිදානයන් පහත පරිදි වනු ඇත

```
Negative
Even
```

Selection inside Repetition
පුණර්කරණය තුළ වරණය

Selection inside Repetition refers to a situation where a decision-making process (selection) occurs repeatedly within a loop (repetition).
පුණර්කරණය තුළ වරණය යන්නෙන් අදහස් කරන්නේ තීරණ ගැනීමේ ක්‍රියාවලියක් (වරණය) ලූපයක් (පුණර්කරණය) තුළ නැවත නැවතත් සිදුවන තත්වයකි.

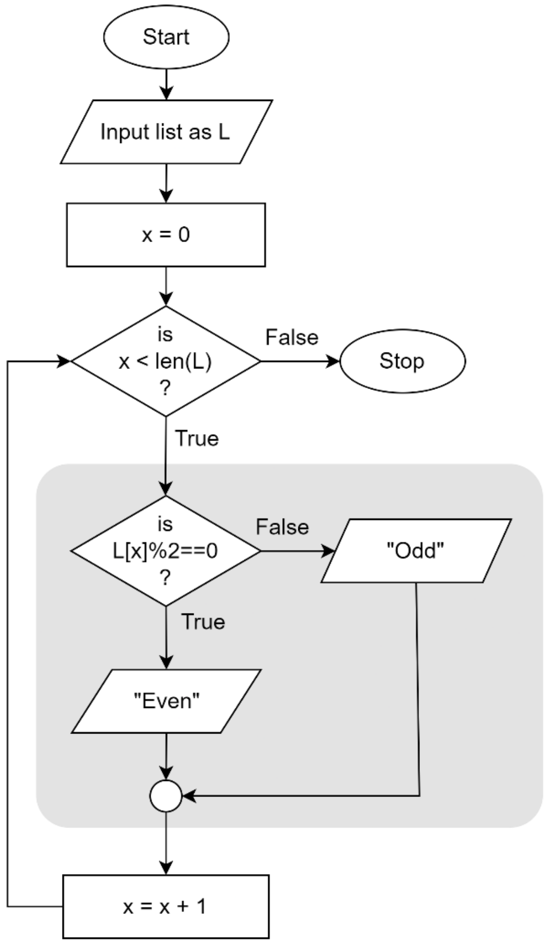


Instruction SET 1 is executed only when x < 10 and condition 1 is TRUE
Instruction SET 1 ක්‍රියාත්මක වන්නේ x < 10 සහ condition 1 සත්‍ය වූ විට පමණි.

Instruction SET 2 is executed only when x < 10 and condition 1 is FALSE
Instruction SET 2 ක්‍රියාත්මක වන්නේ x < 10 සහ condition 1 අසත්‍ය වූ විට පමණි.

Here, This loop runs 10 times while repeatedly checking the condition 1
මෙහි, condition 1 නැවත නැවත පරීක්ෂා කරන අතරතුර මෙම ලූපය 10 වතාවක් ක්‍රියාත්මක වේ

Example.

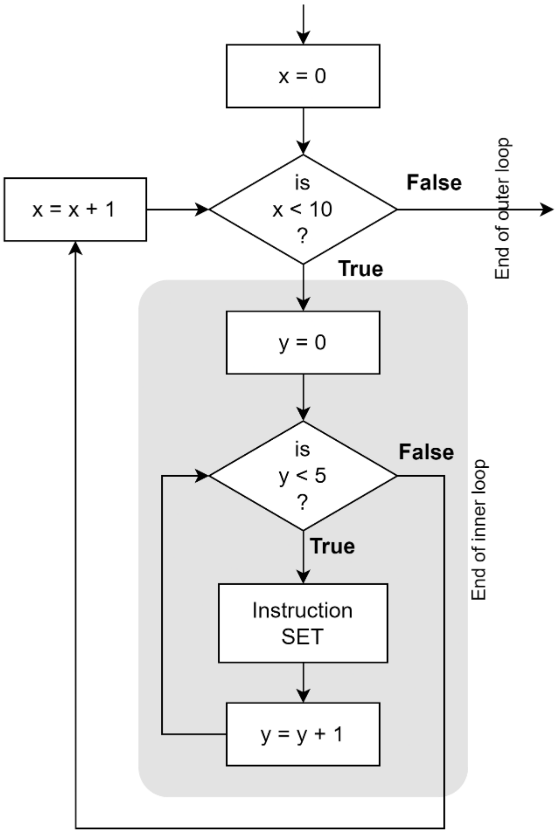


If L is `[10,12,13,51,45]`, the outputs will be
L `[10,12,13,51,45]` වුවහොත්, ප්‍රතිදාන පහත පරිදි වනු ඇත

```
Even
Even
Odd
Odd
Odd
```

Repetition inside Repetition
පුණර්කරණය තුළ පුණර්කරණය

Repetition inside Repetition refers to a control structure where one loop is placed inside another loop.
පුණර්කරණය තුළ පුණර්කරණය යනු එක් ලූපයක් තවත් ලූපයක් තුළ තබා ඇති පාලන ව්‍යුහයකි.



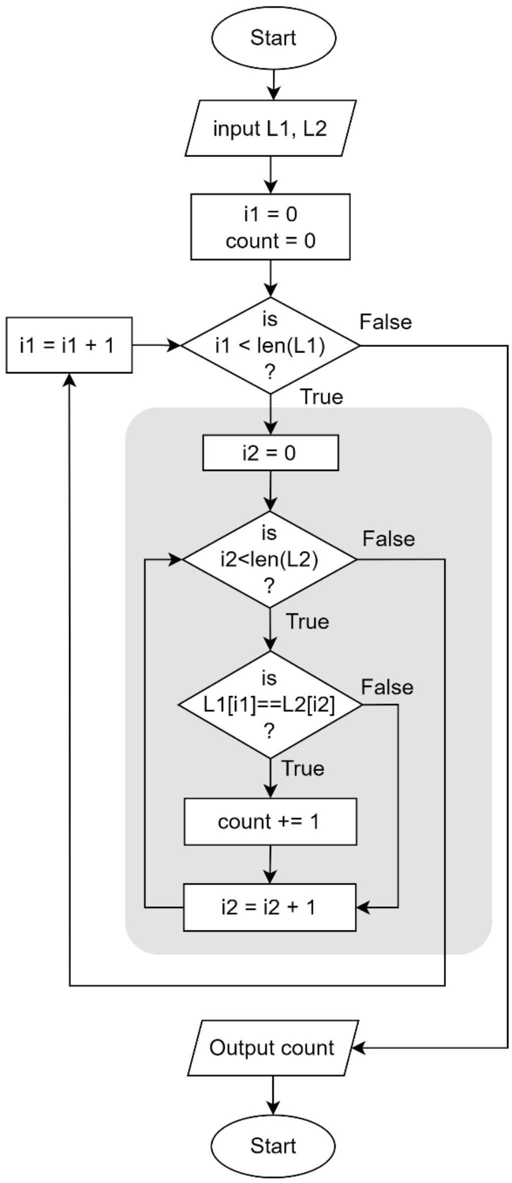
The outer loop starts by initializing x = 0, and it checks the condition x < 10. If x < 10 is true, the flow enters the outer loop and moves to the inner loop.
පිටත ලූපය x = 0 ලෙස ආරම්භ වන අතර එය x < 10 කොන්දේසිය පරීක්ෂා කරයි. x < 10 සත්‍ය නම්, ප්‍රවාහය පිටත ලූපයට ඇතුළු වී අභ්‍යන්තර ලූපය වෙත ගමන් කරයි.

Inside the outer loop, the inner loop initializes y = 0, and it checks the condition y < 5. If y < 5 is true, the inner loop executes Instruction SET.
පිටත ලූපය ඇතුළත, අභ්‍යන්තර ලූපය y = 0 ලෙස ආරම්භ කරයි, එය y < 5 තත්ත්වය පරීක්ෂා කරයි. y < 5 සත්‍ය නම්, අභ්‍යන්තර ලූපය Instruction SET ක්‍රියාත්මක කරයි.

After executing Instruction SET, y is increased by 1. Once the inner loop finishes (when y = 5), it returns to the outer loop, where x is incremented by 1.
Instruction SET ක්‍රියාත්මක කිරීමෙන් පසු, y 1 කින් වැඩි වේ. අභ්‍යන්තර ලූපය අවසන් වූ පසු (y = 5 වන විට), එය පිටත ලූපය වෙත ආපසු පැමිණේ, එහිදී x 1 කින් වැඩි වේ.

The outer loop continues to run, with the inner loop restarting for each value of x, repeating the process until x = 10.
x හි එක් එක් අගය සඳහා අභ්‍යන්තර ලූපය නැවත ආරම්භ කිරීමත් සමඟ පිටත ලූපය දිගටම ක්‍රියාත්මක වේ, x = 10 දක්වා ක්‍රියාවලිය නැවත සිදු කරයි.

Example.



Pseudocodes
ව්‍යාජ කේත

As previously explained, pseudocodes do not have a specific syntax or predefined keywords, whereas other programming languages have their unique syntax and specific keywords.
පෙර පැහැදිලි කළ පරිදි, ව්‍යාජ කේතවලට නිශ්චිත syntax එකක් හෝ පූර්ව නිර්වචනය කළ මූල පද නොමැත, නමුත් අනෙකුත් ක්‍රමලේඛන භාෂාවලටා ඒවාට අනන්‍ය වූ syntax සහ විශේෂිත මූල පද පවතී.

Comparing pseudocodes with other several programming langauages
වෙනත් ක්‍රමලේඛන භාෂා කිහිපයක් සමඟ ව්‍යාජ කේත සංසන්දනය කිරීම

Language	Print Command	Input Command
Python	`print("Hello, World!")`	`input("Enter something: ")`
Java	`System.out.println("Hello");`	`Scanner sc=new Scanner(System.in);` `String input=sc.nextLine();`
C	`printf("Hello, World!\n");`	`scanf("%s", input);`
C++	`cout << "Hello, World!";`	`cin >> input;`
JavaScript	`console.log("Hello, World!");`	`prompt("Enter something:")`
PHP	`echo "Hello, World!";`	`$input = fgets(STDIN);`
Pseudocodes	PRINT DISPLAY SHOW LOG OUTPUT EXHIBIT PRESENT ECHO PUT REVEAL REPORT WRITE	READ ACCEPT TAKE GET RECEIVE CAPTURE REQUEST SCAN ENTER FETCH PROMPT INPUT

NOTE:
Pseudocodes are not executable by computers.
ව්‍යාජ කේත පරිගණක තුළ ක්‍රියාත්මක කළ නොහැක.

They serve as a bridge between human thinking and computer programming.
ඒවා මිනිස් චින්තනය සහ පරිගණක වැඩසටහන් අතර පාලමක් ලෙස ක්‍රියා කරයි.

Benefits of Pseudocode

Helps visualize the algorithm’s flow.
ඇල්ගොරිතමයේ ගලායාම දෘශ්‍යමාන කිරීමට උපකාරී වේ.

Reduces development time by allowing pre-planning.
පූර්ව සැලසුම් කිරීමට ඉඩ දීමෙන් සංවර්ධන කාලය අඩු කරයි.

Facilitates collaborative problem-solving.
කණ්ඩායම් ලෙස ගැටළු විසඳීමට පහසුකම් සපයයි.

1. Start/Begin
ආරම්භය

Every pseudocode usually begins with a keyword like Start or Begin to indicate the initiation of the algorithm.
සෑම ව්‍යාජ කේතයක්ම සාමාන්‍යයෙන් ආරම්භ වන්නේ ඇල්ගොරිතමයේ ආරම්භය දැක්වීමට Start හෝ Begin වැනි මූල පදයකින් ය.

It signals where the algorithm starts.
එය ඇල්ගොරිතම ආරම්භ වන ස්ථානය සංඥා කරයි.

2. Defining Steps
පියවර නිර්වචනය කිරීම

Write each action or process in a clear, sequential manner. Use plain language for easy understanding
සෑම ක්‍රියාවක්ම හෝ ක්‍රියාවලියක් පැහැදිලි, අනුක්‍රමික ආකාරයෙන් ලියන්න. පහසුවෙන් තේරුම් ගැනීමට සරල භාෂාවක් භාවිතා කරන්න.

Ensure each step is self-explanatory and logically progresses.
සෑම පියවරක්ම ස්වයං පැහැදිලි කිරීම සහ තර්කානුකූලව ඉදිරියට යන බව සහතික කර ගන්න.

Example:

```
Start
   Read input value X
   Print X
```

3. Conditions
කොන්දේසි

Conditions in pseudocode allow decision-making processes. This is usually represented with keywords like IF, ELSE IF, ELSE, or ENDIF.
ව්‍යාජ කේත වල කොන්දේසි තීරණ ගැනීමේ ක්‍රියාවලීන්ට ඉඩ දෙයි. මෙය සාමාන්‍යයෙන් IF, ELSE IF, ELSE, හෝ ENDIF වැනි මූල පද වලින් නිරූපණය කෙරේ.

Conditions help the algorithm decide which path to take depending on certain criteria.
යම් නිර්ණායක මත පදනම්ව කුමන මාර්ගයක් ගත යුතුද යන්න තීරණය කිරීමට කොන්දේසි ඇල්ගොරිතමයට උපකාර කරයි.

Syntax:

```
IF <condition> THEN
    <do something>
ELSE IF <another condition> THEN
    <do something else>
ELSE
    <do something else if all
conditions fail>
ENDIF
```

Example:

```
Start
   Read input value X
   IF X > 0 THEN
       Print "X is positive"
   ELSE IF X < 0 THEN
       Print "X is negative"
   ELSE
       Print "X is zero"
   ENDIF
```

4. Loops

Loops are used when you want to repeat a set of instructions multiple times.
ඔබට උපදෙස් මාලාවක් කිහිප වතාවක් නැවත කිරීමට අවශ්‍ය විට ලූප භාවිතා වේ.

The two most common loops in pseudocode are WHILE and FOR.
ව්‍යාජ කේත වල බහුලවම ඇති ලූප දෙක වන්නේ WHILE සහ FOR වේ.

- **While Loop**

A WHILE loop runs as long as a given condition is true.
දී ඇති කොන්දේසියක් සත්‍ය වන තාක් WHILE ලූපයක් ක්‍රියාත්මක වේ.

The loop continues to execute until the condition becomes false.
කොන්දේසිය අසත්‍ය වන තෙක් ලූපය ක්‍රියාත්මක වේ.

Syntax:

```
WHILE <condition> DO
    <do something>
ENDWHILE
```

Example:

```
Start
   Set counter to 0
   WHILE counter < 5 DO
       Print "Counter is", counter
       Increment counter by 1
   ENDWHILE
```

- **For Loop**

A FOR loop runs a set number of times, iterating over a range or a sequence.
FOR ලූපයක් පරාසයක් හෝ අනුපිළිවෙලක් හරහා පුනරාවර්තනය වන වාර ගණනක් ධාවනය කරයි.

Syntax:

```
FOR <variable> =<start> TO <end> DO
    <do something>
ENDFOR
```

Example:

```
Start
   FOR i = 1 TO 5 DO
       Print "Value of i is", i
   ENDFOR
```

5. Ending the Algorithm
ඇල්ගොරිතම අවසන් කිරීම

End the pseudocode with a keyword like End or Stop to mark the conclusion of the algorithm.
ඇල්ගොරිතමයේ නිගමනය සලකුණු කිරීම සඳහා End හෝ Stop වැනි මූල පදයකින් ව්‍යාජ කේතය අවසන් කරන්න.

Example:

```
Start
   Read input value X
   Print "You entered", X
End
```

Comparing python programs and their related pseudocodes
පයිතන් වැඩසටහන් සහ ඒවාට අදාළ ව්‍යාජ කේත සංසන්දනය කිරීම

Sequence

Python Code

```
print("Enter your name:")
name = input()
print("Hello, " + name)
```

Pseudocode

```
DISPLAY "Enter your name:"
INPUT name
DISPLAY "Hello, " + name
```

Selection

Python Code

```
age = int(input("Enter age: "))
if age >= 18:
    print("You are eligible.")
else:
    print("You are not eligible.")
```

Pseudocode

```
DISPLAY "Enter your age:"
IF age >= 18 THEN
     DISPLAY "You are eligible."
ELSE
     DISPLAY "You are not eligible."
ENDIF
```

Selection inside Selection

Python Code

```
x = 8
if x > 5:
    if x < 10:
        print(1)
    else:
        print(0)
else:
    print(-1)
```

Pseudocode

```
SET x = 8
IF x > 5 THEN
    IF x < 10 THEN
        DISPLAY 1
    ELSE
        DISPLAY 0
    ENDIF
ELSE
    DISPLAY -1
ENDIF
```

Selection inside Repetition

Python Code

```
i = 1
while i <= 5:
    if i % 2 == 0:
        print(f"{i} is even")
    else:
        print(f"{i} is odd")
    i += 1
```

Pseudocode

```
SET i = 1
WHILE i <= 5 DO
    IF i MOD 2 = 0 THEN
        DISPLAY i + " is even"
    ELSE
        DISPLAY i + " is odd"
    ENDIF
    ADD 1 TO i
ENDWHILE
```

Repetition

```
i = 1
while i <= 5:
    print(i)
    i += 1
```

```
SET i = 1
WHILE i <= 5 DO
    DISPLAY i
    ADD 1 TO i
ENDWHILE
```

Repetition inside Repetition

```
i = 1
while i <= 5:
    j = 1
    while j <= 3:
        if i % j == 0:
            print("yes")
        j += 1
    i += 1
```

```
SET i = 1
WHILE i <= 5 DO
    SET j = 1
    WHILE j <= 3 DO
        IF i MODULUS j = 0 THEN
            DISPLAY "yes"
        ENDIF
        ADD 1 TO j
    ENDWHILE
    ADD 1 TO i
ENDWHILE
```

Repetition inside Selection

```
x = int(input("Enter a number: "))
if x > 0:
    count = 1
    while count <= x:
        print(count)
        count += 1
else:
    print(0)
```

```
DISPLAY "Enter a number:"
INPUT x
IF x > 0 THEN
    SET count = 1
    WHILE count <= x DO
        DISPLAY count
        ADD 1 TO count
    ENDWHILE
ELSE
    DISPLAY 0
ENDIF
```