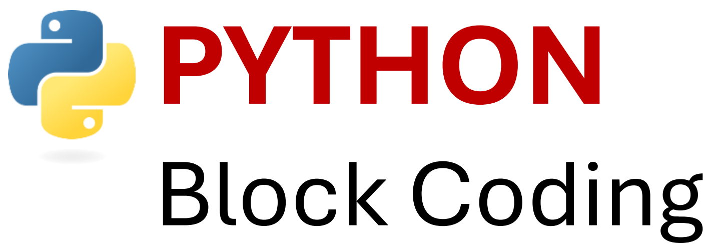


USER MANUAL

A step-by-step user guide to getting started on learning Python Programming Language through Block Coding. Learn syntax, build projects and solve problems using Physical Python Block Coding.

CREATED BY

AITinkr	Prepared by Suresh Kadari	Last Updated: *30 January 2025*

Copyright CEDURA TESTSOL

Safety Information

To ensure the safe and enjoyable use of this toy, please read and follow these safety instructions carefully:

AGE RECOMMENDATION:
This kit is suitable for children aged 8 years and older.

SUPERVISION
Not mandatory as this kit has how-to-use instructions and projects. However, in the case of young children, we recommend supervision to prevent accidental swallowing or misuse of blocks.

STORAGE
Keep the blocks out of the reach of younger siblings or pets. After playtime, store the blocks in a secure container.

CLEANING
Clean the blocks periodically with a dry cloth. Ensure the blocks are completely dry, and do not use nails or sharp objects to pick them up. The stickers on the blocks may come off due to usage or handling. In such cases, please use glue to fix them back.

CHOKING HAZARD
Not suitable for children under 5 years due to small parts that may pose a choking hazard. Small parts can block airways if swallowed. If a child places a block in their mouth or swallows one, seek medical attention immediately.

CEDURA © Copyrighted Content. All rights reserved

Table of Contents

- ❑ Kit Contents
- ❑ Essential Instructions
- ❑ Overview of the Blocks
- ❑ How to use the Output Chart
- ❑ What is Block Coding?
- ❑ How to Block Code?
- ❑ Data & Data Types
- ❑ Explanation of Individual Blocks
- ❑ Assignment Solutions

CEDURA © Copyrighted Content. All rights reserved

Kit Contents

Wooden Box

Output Chart

Black Board (Behind / Flipside)

Magnetic White Board

Value Area

Coding Area

Variables logging Area

Sketch Pen with Eraser (Magnetic)

Sprite

Duster (white board)

Storage Area

Latch for lock (with screw) - 2No.

Chalks for Black Board

Instruction / Coding Booklet

Various types of Coding Blocks*

Screwdriver for fixing latch (not included)

* Includes Blank Blocks

Getting Started:

- Check the QC seal. Do not accept if broken.
- Unpack the contents and identify them.
- Fix 2 No. latches. Remove the screws, insert the latch and re-screw them
- Place the Chart on the board
- Read the instructions properly before using the kit. If in doubt, write to us at hello@schoolforai.com
- Use blank blocks to create your own coding blocks. Please share your ideas with us and we may publish them

CEDURA © Copyrighted Content. All rights reserved

Essential Instructions

Dos and Don'ts

- Always read the instructions before using the kit.
- Keep small parts away from toddlers and small children. There is a danger of accidental swallowing.
- Blocks are designed to be reusable. You can write on the whiteboard, on the blocks, and on the blackboard.
- Use the duster or an eraser properly without leaving marks on the whiteboard. Do not use hard sketch pens or ball pens, but light sketch pens only.
- To erase your writing on the sticker properly, use a hand sanitiser or glass cleaner liquid. Use small quantities only.
- Do not write hard on the white space on the block. It may leave impressions and make them not reusable. In such a rare case, you could create your own sticker and stick it on the block.
- Do not clean the board or blocks with hard chemicals or abrasive fluids.
- Do not write hard on the magnetic whiteboard
- Always write small text and values in the white space. Please consider the names of the variables or values assigned to them or the message in the output block to be a small one. The goal is to learn the coding within the available space.
- It may be possible that the sticker may come off after long usage. If it happens, you may stick them back with any standard glue. You could even go creative, and design stickers as required.
- Use the blank blocks supplied in the kit to create your own block code and showcase your creativity. Alternatively, you could use them in place of blocks if you lose some. You could also trace the design on cardboard and prepare blocks on your own.

NOTE:
This is a kit mimicking block coding without an actual computer. Hence, sometimes we need to assume things like clicking the space bar or seeking input from a user or such. This is a gamified fun-filled learning experience and also helps you to be creative. We cannot expect the flexibility and sophistication of online computer-based block coding while using the kit

CEDURA © Copyrighted Content. All rights reserved

Overview of the Blocks - Total 129

Event Blocks

Block	Label	Count
Start	Start Block	# 1
	Connector	# 5
	Connector Extended	# 2
End	End Block (short)	# 1
End	End Block (extended)	# 1
End	End Block (long)	# 1

Conditional Blocks

Block	Label	Count
if	If Condition	# 2
else	Else (short)	# 2
else	Else (extended)	# 2
elif	Elif Condition	# 2
:	Colon End	# 4
pass	Pass	# 1
match	Match	# 1
case	Case (short)	# 1
case	Case (extended)	# 3

Control Loop Blocks

Block	Label	Count
while	While Control Loop	# 2
for	For Control Loop	# 2
break	Break	# 1
continue	Continue	# 1

Operator Blocks

Arithmetic

Block	Label	Count
+	Add	# 1
-	Subtract	# 1
X	Multiply	# 1
/	Divide	# 1
%	Mod	# 1
//	Floor Division	# 1
**	Exponentiation	# 1

Assignment

Block	Label	Count
=	Equal to	# 6

Comparison

Block	Label	Count
<	Lesser than	# 2
<=	Lesser than or equal	# 2
>	Greater than	# 2
>=	Greater than or equal	# 2
==	Equals to	# 1
!=	Not Equal	# 1

Logical

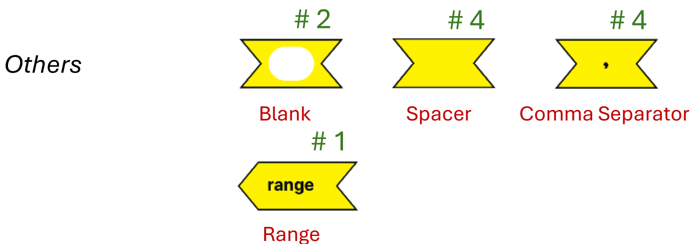
Block	Label	Count
and	AND	# 1
or	OR	# 1
not	NOT	# 1

Membership

Block	Label	Count
in	IN	# 1
not in	NOT IN	# 1

CEDURA © Copyrighted Content. All rights reserved

Others



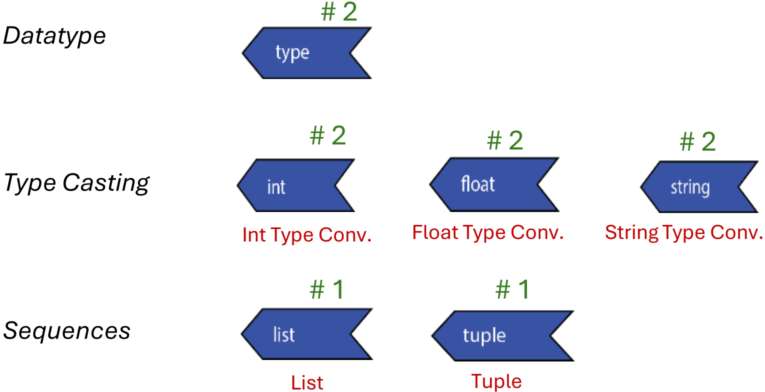
** Use these when you find a shortfall of operator blocks

Datatype Blocks

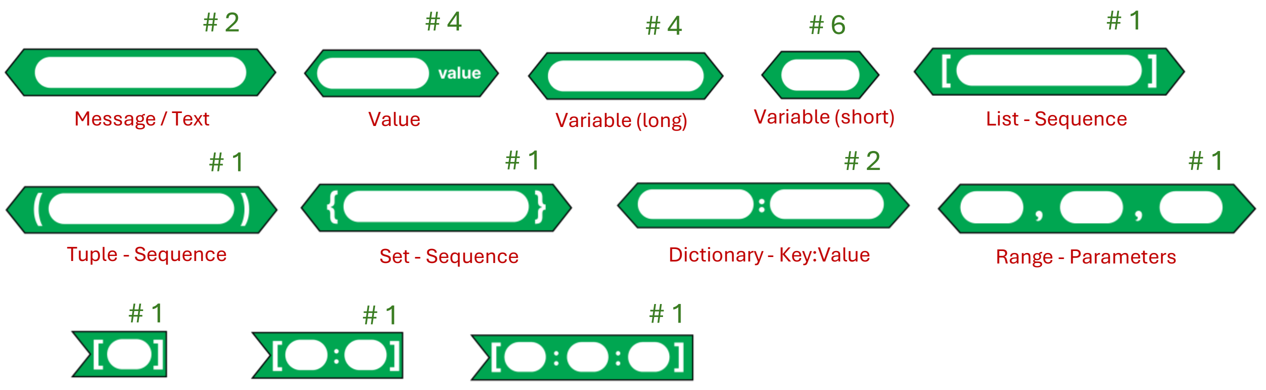
Datatype

Type Casting

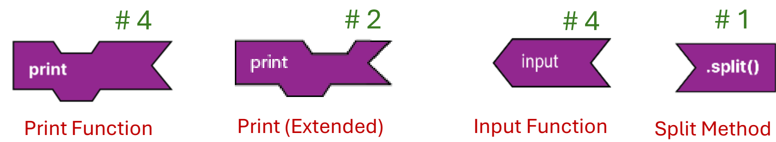
Sequences



Value / Variable Blocks



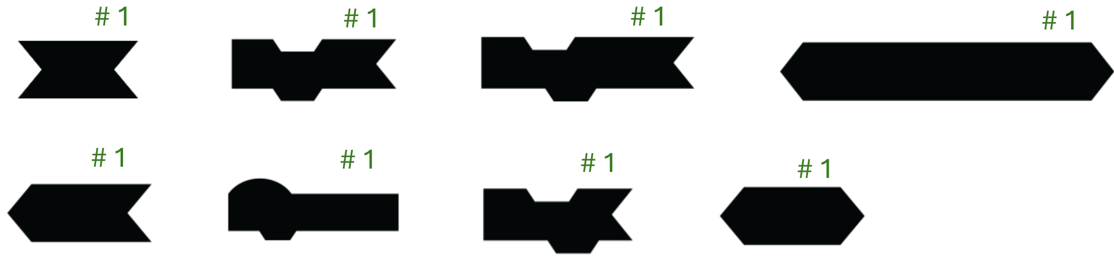
Input & Output Blocks



CEDURA © Copyrighted Content. All rights reserved

Sprite Block

Blank Blocks



Note: Use these blank blocks to create your logic or even a new block-coding programming language. Why not? All you need is an idea, creativity and a need that should be fulfilled.

CEDURA © Copyrighted Content. All rights reserved

How to use the Output Chart

While working on Block Coding you are expected to showcase the output to prove the correct execution of your code. However, it is hard to do so on this physical Block Coding kit. Hence we have introduced the Output Chart that facilitates Sprite movement and a few other actions to help you learn the coding concepts. You could move the Sprite on this Output Chart while explaining your code.

Sprite Docking Area
Space Click
Output Value Space
PYTHON
Flag Click
SPACE
VALUE
Variable Block
Value Block
White Space for calculations and values
Variable-value pair to write intermediatory and final output values

Output Chart

Move the Sprite on the chart and imitate the click actions. You could even write the output value in the value space on the top right-hand side. Read the X and Y axis coordinates to move the Sprite on the movement area. You could re-print the chart if required by downloading it from our resources. If you have any queries or need help in this regard please write to us at hello@schoolforai.com.

CEDURA © Copyrighted Content. All rights reserved

What is Block Coding?

Block coding is a visual programming method that allows users to create programs by stacking blocks of code, rather than writing text-based code. Each block represents a specific instruction, making it an accessible way to learn programming concepts without dealing with syntax complexities.

Block coding serves as a simplified, visual approach to programming, designed to make coding more accessible, especially for beginners, children, and non-technical users. Its primary purpose is to introduce fundamental programming concepts in an engaging and error-free manner.

KEY FEATURES:

Visual Interface: Uses drag-and-drop blocks to build code. Blocks are often color-coded and shaped to indicate compatibility.

Syntax-Free: No need to worry about typos, brackets, or punctuation errors.

Event-Driven: Focuses on actions triggered by events (e.g., clicking a button, moving a sprite).

Immediate Feedback: Offers real-time execution of code, helping users see the results of their logic instantly.

POPULAR TOOLS:

Scratch: Developed by MIT, Scratch is widely used for teaching kids programming through animations, games, and stories. One of the most famous visual programming interfaces.

Blockly: A Google-developed library for creating visual programming interfaces. Widely used to create programs, learn coding and solve problems.

Code.org: Offers block-based coding activities for different levels along with problem-solving.

Block coding is an excellent way for kids to develop essential skills and prepare for a technology-driven future. Its visual, beginner-friendly approach makes it accessible and fun, enabling children to grasp complex concepts in a simplified manner. Block coding helps in developing computational thinking, problem-solving skills, logical reasoning, algorithm design, creativity and innovation, and boosts confidence.

CEDURA © Copyrighted Content. All rights reserved

Why Learn PYTHON

Python is an excellent programming language for kids to learn because of its simplicity and versatility. Its easy-to-read syntax resembles everyday language, making it an ideal choice for beginners. Unlike other languages that require complex setups, Python allows kids to focus on understanding programming concepts without getting bogged down by syntax errors. This makes the learning process smoother and more enjoyable. Python also encourages creativity, as kids can quickly build games, animations, and even simple applications, turning their ideas into reality.

Beyond its simplicity, Python prepares kids for the future by introducing them to one of the most widely used programming languages in the world. It is a gateway to exciting fields such as artificial intelligence, data science, robotics, and web development. Learning Python at an early age not only boosts logical thinking and problem-solving skills but also builds confidence to tackle more complex programming languages later on. With its abundant learning resources and supportive community, Python opens doors to endless possibilities, empowering kids to be creators and innovators in an increasingly tech-driven world.

ADVANTAGES OF LEARNING PYTHON:

Python is one of the best programming languages for children to start their coding journey. Its simplicity and versatility offer numerous benefits that go beyond just learning to program. Here are the key advantages:

Easy to Understand:
Python's simple and readable syntax makes it ideal for kids. They can quickly grasp programming concepts without being overwhelmed by complex code structures.

Builds Problem-Solving Skills:
Coding in Python teaches kids how to break down problems into smaller steps, fostering logical thinking and structured reasoning.

Encourages Creativity:
Python enables kids to create games, animations, and applications, allowing them to bring their ideas to life and express their creativity.

Widely Used in Real-World Applications:
Python's widespread use in fields like artificial intelligence, data science, and web development makes it a valuable skill for the future.

Future-Ready Skill:
Learning Python early positions kids for success in a technology-driven world, opening up career opportunities in various industries as they grow.

Prepares for Advanced Topics:
Python acts as a stepping stone for exploring more advanced topics such as machine learning, robotics, and IoT.

CEDURA © Copyrighted Content. All rights reserved

How to Block Code?

Block coding is simple and can be used by all children. We have provided blocks and instructions to guide you through your learning process. This is especially suitable for young kids from 8 to 12 years age group. However, younger kids can also try Block Coding and build simple skills.

You will find different types of Blocks in this kit, which serve as the fundamental building units for creating Python programs. Programming using these physical blocks involves dragging and placing these blocks on the magnetic board, connecting them to form a sequence to solve a problem, In this manual, we will explore how Python programming works in Block Coding, dive into the various categories of blocks available, and outline the basic steps to get started. Additionally, we are offering exciting beginner projects to help you kick-start your coding journey and unleash your creativity.

Please note that even though this Blocking Coding kit, was designed with inspiration from MIT Scratch Block coding, there are limitations in using these physical Blocks. Hence, a few Blocks are not available and also few Blocks are modified to build this physical Block Coding kit. The goal is to create an environment where kids to learn and practice coding without any digital device. There is no limitation for challenges, creativity and coding.

SAMPLE PROJECT 1:

Create a Python Block code to add two number and print the output.

Algorithm:

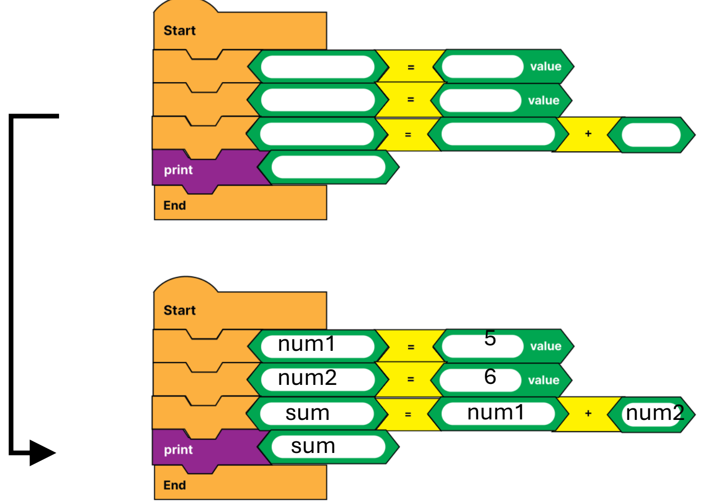
1. Start the Program
2. Consider a variable and assign a number (num1)
3. Consider another variable and assign a number (num2)
4. Now, add both num1 and num2 and store the result in another variable (sum)
5. Print the result (sum)
6. End the Program

Coding Steps:

1. Select the required code block to solve the above problem. In this case, we need different blocks
    a. An EVENT block to start the sequence
    b. A VARIABLE block to hold a number.
    c. An OPERATOR & VALUE block to assign value to the variable.
    d. A VARIABLE block to hold a second number.
    e. An OPERATOR & VALUE block to assign value to the second variable.
    f. A VARIABLE block to hold a result.
    g. VARIABLE & OPERATOR blocks to add the above variables
    h. An OUTPUT block to display the result
    i. An END Block to indicate end of the program

CEDURA © Copyrighted Content. All rights reserved

2. Select the right Code Blocks from the kit. You could use your algorithm thinking to decide on the Blocks to solve the given problem.
3. Use the output Chart as required to store the intermediary and/or final output values
4. Join / Snap the selected Code Blocks as per the logic following the sequence of activities. It is recommended to write down your algorithm so that you can simply follow it.



5. Code Blocks are supplied blank, with few having white spaces. Here, you can write the value or text as per the problem statement. Please be careful not to leave permanent marks.
6. On the OUTPUT side, you could write whether the output i.e., the sum of two numbers
7. The END block indicates the end of the code block.

SAMPLE PROJECT 2:

Create a Python Block code to identify whether a person is eligible to vote or not.

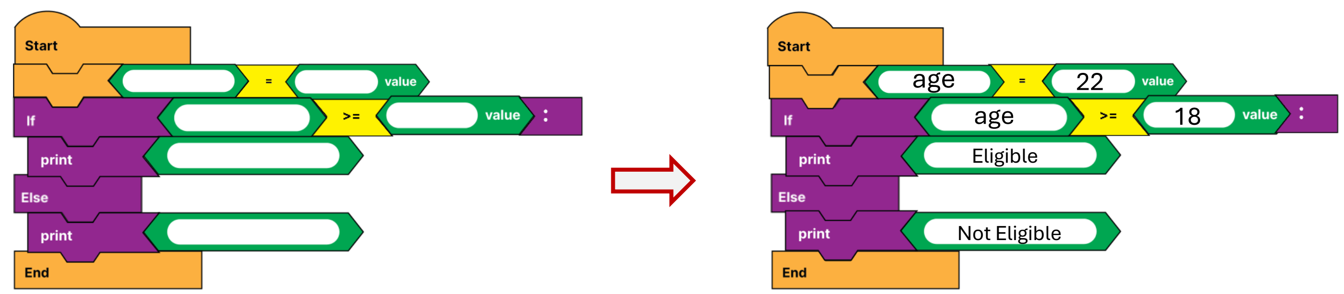
Algorithm:

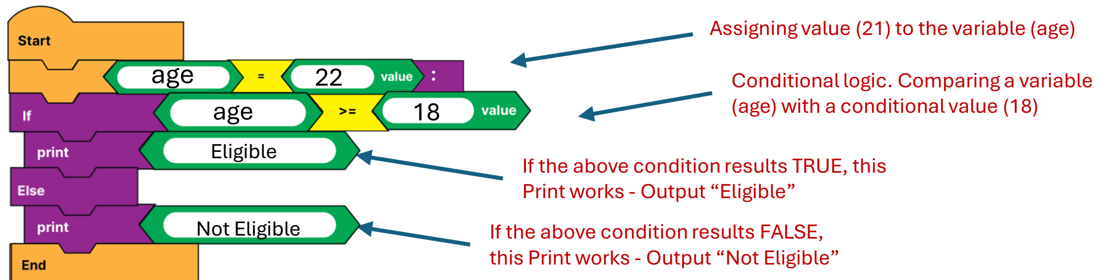
1. Start the Program
2. Consider a variable (age) and assign the age of the person (22)
3. Check the condition i.e., whether the age of the person is greater than or equal to 18
4. The result could be TRUE or FALSE.
5. If TRUE, print 'Eligible'
6. If FALSE, print 'Not Eligible'
7. End the Program

CEDURA © Copyrighted Content. All rights reserved

Steps:

1. Select the required code block to solve the above problem. In this case, we need different blocks
    a. An EVENT block to start the sequence
    b. A VARIABLE block (used along with connector block) to hold age.
    c. A VALUE block to assign value to the age variable.
    d. A CONDITIONAL block to make a decision
    e. A VARIABLE block for comparing
    f. A VALUE block to use in comparison
    g. An OUTPUT block to print the desired message when the condition is TRUE
    h. A VALUE block to declare a message.
    i. A CONDITIONAL block to act when the condition is FALSE.
    j. An OUTPUT block to print the desired message when the condition is FALSE.
    k. A VALUE block to declare a message.
    l. An END Block to indicate end of the program
2. Select the right Code Blocks from the kit. You could use your algorithm thinking to decide on the Blocks to solve the given problem.
3. Use the output Chart as required to store the intermediary and/or final output values
4. Join / Snap the selected Code Blocks as per the logic following the sequence of activities. It is recommended to write down your algorithm so that you can simply follow it.





CEDURA © Copyrighted Content. All rights reserved

5. Code Blocks are supplied blank, with few having white spaces. Here, you can write the value or text as per the problem statement. Please be careful not to leave permanent marks.
6. On the OUTPUT side, you could write whether the output would be ‘Eligible’ or ‘Not Eligible’
7. The END block indicates the end of the code block.

What is a Statement?

A statement in Python is a single line of code or instruction that performs a specific action. It can include variable assignments, function calls, control flow structures, or other programming commands. For example, x = 10 is an assignment statement that sets the value of x to 10, while print(x) is a statement that outputs the value of x. Python executes statements sequentially unless control flow statements like if, while, or for alter the order. Statements do not always produce a value; their primary purpose is to instruct the interpreter to perform an action. Understanding Python statements is essential for organizing and structuring code effectively.

Example Statements:

```
var = 10

sum = num1 + num2

if a > 5 and b <10 :
```

What is an Expression?

An expression in Python is a combination of variables, operators, and values that evaluates to a result or value. For example, 2 + 3 is an arithmetic expression that evaluates to 5, while x > 5 is a Boolean expression that evaluates to True or False depending on the value of x. Expressions can be used within statements to compute values or control program flow.. Unlike statements, expressions always return a value, making them fundamental to calculations and decision-making in Python programs.

Example:

CEDURA © Copyrighted Content. All rights reserved

Understanding Data & Data Types

Data is a measurement or an observation. In generic terms, Data refers to the raw information that can be collected, processed, and used for various purposes. In programming, data serves as the foundation for building software and performing computational tasks. Data can represent facts, observations, or instructions that are stored and manipulated by a program. Examples of data include numbers, text, images, and measurements. How data is organized and categorized plays a critical role in ensuring its effective use, which is why programming languages define **data types** to classify and manage different kinds of data.

Data types specify the **kind of value** a variable can hold and dictate the operations that can be performed on it. Common data types include integers, floating-point numbers, strings, and Boolean values. For example, integers represent whole numbers, while floating-point numbers handle decimals. Strings are sequences of characters used to represent text, and Boolean values (True or False) are used in decision-making logic. Advanced data types, such as lists, tuples, sets, and dictionaries, allow programmers to work with collections of data efficiently. Understanding data and its types is fundamental in programming as it ensures proper utilization of memory, avoids errors, and enables the creation of robust applications.

Integer (int)

The integer data type in programming represents whole numbers, both positive and negative, without any decimal or fractional components. In Python, integers are defined using the int class, and they can handle a wide range of values, as Python supports arbitrarily large integers. This makes the integer data type highly versatile for mathematical operations such as addition, subtraction, multiplication, and division. Integers are commonly used in programming for tasks like counting iterations in loops, indexing elements in data structures, and performing calculations. Declaring an integer in Python is straightforward, as the interpreter automatically assigns the int type to whole number values.

Example: var = 10

In Python, there is no need to explicitly declare variables while assigning value.

Float (float)

The **float data type** in programming is used to represent real numbers that contain decimal points or fractions. In Python, floats are defined using the float class and can handle a wide range of values, including very small or very large numbers. Floats are commonly used in applications requiring precision. For example, assigning x = 3.14 creates a float variable x representing a decimal number.

Example:

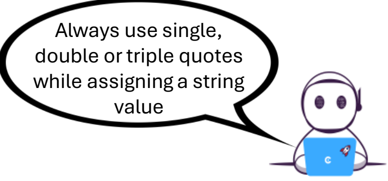
```
temp = 10.534
height = 5.2
```


CEDURA © Copyrighted Content. All rights reserved

String (str)

The **string data type** in programming represents a sequence of characters, such as letters, numbers, symbols, or spaces, enclosed within quotes. In Python, strings are defined using the str class and can be created using single quotes ('), double quotes ("), or triple quotes (''' or """) for multi-line strings. Strings are widely used for storing and manipulating text in various applications, such as displaying messages, processing user input, and generating dynamic content. Strings are immutable, meaning their values cannot be changed once created, although operations can produce new strings. This immutability ensures reliability when handling text data in Python programs.

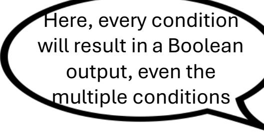
Example:
```
name = ‘python’
loc = ”Hyderabad”
```



Boolean (True / False)

The **Boolean data type** in programming represents one of two possible values: **True or False**. In Python, Booleans are defined using the bool class and are commonly used in decision-making and logical operations. Boolean values arise as the result of comparisons or conditions, such as 5 > 3 (which evaluates to True) or 4 == 7 (which evaluates to False). They play a critical role in controlling the flow of a program through conditional statements (if, else, elif) and loops (while, for). Boolean operators like **and, or,** and **not** are used to combine or negate logical expressions, allowing for more complex decision-making. For example, the expression **x > 0 and y < 10** evaluates to True only if both conditions are true. Booleans are foundational to programming, enabling the development of responsive, dynamic, and logical code structures.

Example:
```
abc = True
print(type(abc))
if  age >= 18:
```



Complex (a + bj)

The **complex data type** in Python represents complex numbers, consisting of a real part and an imaginary part. Complex numbers are written as a + bj, where a is the real part, b is the imaginary part, and j represents the square root of -1 (the imaginary unit). Python’s built-in complex type is particularly useful in mathematical computations, especially in fields like signal processing, physics, and engineering, where operations with complex numbers are common. Python supports arithmetic operations with complex numbers.

Example:
```
var  = 3 + 5j
```

Please Note: *There are other data types however, at this learning level it is fine to learn these basic data types.*

CEDURA © Copyrighted Content. All rights reserved

TOPIC ASSIGNMENT

1. Identify the below data types.

    i. 23 ______________

    ii. -26 ______________

    iii. 0 ______________

    iv. -12.5 ______________

    v. true ______________

    vi. false ______________

    vii. 3+4j ______________

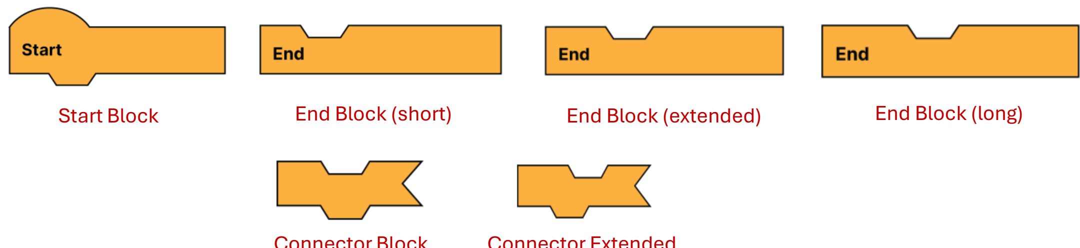
    viii. “Hii” ______________

CEDURA © Copyrighted Content. All rights reserved

UNDERSTANDING BLOCKS

EVENT Blocks

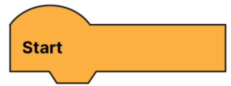
Event blocks are the orange blocks in the kit that are with a rounded top. These are the fundamental categories in block coding that act as triggers for executing actions in a program. These blocks respond to specific events or user interactions. In our kit, you will find a couple of Event Blocks that could be used while working on coding.



The event blocks are the ones that begin each sequence of code and even at the end. Except for the 'Connector Block', you cannot add the blocks in between the code. The 'Start Block' job is to wait for a specific event to happen or send a message to other blocks so they can tell the code blocks below them to go! The 'End Block' will always be located at the end of the Program.

In this manual Python Block Coding kit, you will find the Event Blocks. As we do not have any digital devices these blocks represent actions which would be mimicked by the user. These blocks will make the user understand how an event will trigger a set of actions (lines of code).

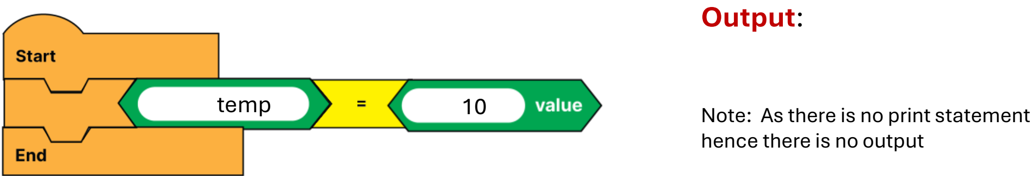
Start Block



The "Start" event block is used to start executing the chain of code only. If you have worked on other Block Coding applications you must have used this type of block earlier. However, we do not have any 'event' based triggering here. Using this block indicates the start of the code.

PRACTICE 1

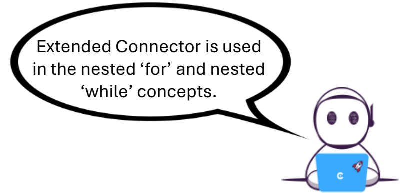
Problem Statement: *Build a Python Block Code to load a variable with value.*

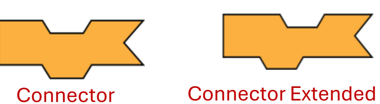


Code Explanation: *As per the problem statement we are expected to load a value into the variable. For that, we have started with the 'Start' event block which indicates the start of the program. This is followed by other coding blocks that meet the requirement.*

CEDURA © Copyrighted Content. All rights reserved

Connector Block

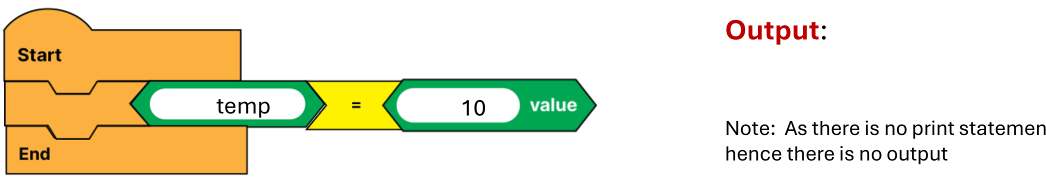




The "Connector" event block is used to connect variables and make them part of the coding. These are usually used under the 'Start' event block and wherever you need to assign value to the variables or manipulate the existing value. They help in connecting variables or conducting some operations on the variables.

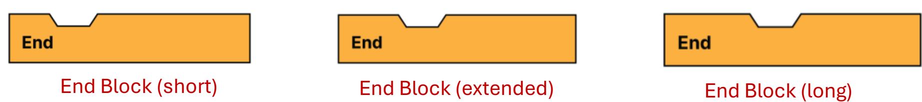
PRACTICE 1

Problem Statement: *Build a Python Block Code to load a variable with a value.*



> ***Code Explanation:*** *In the earlier example we have already used the 'Connector' block to work on the variables. This is the way we use the 'Connector' block to assign values to the variables or to manipulate them.*

End Block

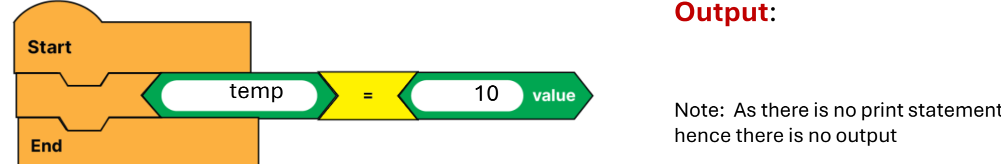


End Block (short) End Block (extended) End Block (long)

The "End" event block is always used at the end of the program. This indicates the completion of the code. You would find two different 'End' event blocks in the kit, one with a shorter 'top notch' and the other one with a longer 'top notch'. Please note that Python minds spaces (indentation). Meaning you cannot give or leave spaces in the code and all the code blocks should appear in a line. However, when working on advanced programs you will need extended 'End' block to maintain the alignment of the coding blocks.

PRACTICE 1

Problem Statement: *Build a Python Block Code to load a variable with a value.*

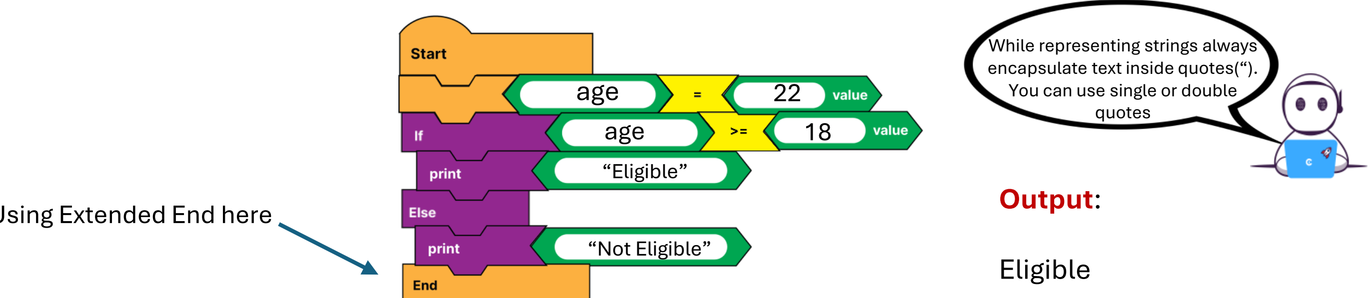


CEDURA © Copyrighted Content. All rights reserved

> ***Code Explanation:*** *Again, consider the earlier example where we have used an 'End' event block to indicate the end of the program. There cannot be any other programming blocks beyond the 'End' block. It should be the last block of the coding.*

PRACTICE 2

Problem Statement: *Build a Python Block Code with an if conditional statement to demonstrate the usage of the 'extended' End Block.*



Output:

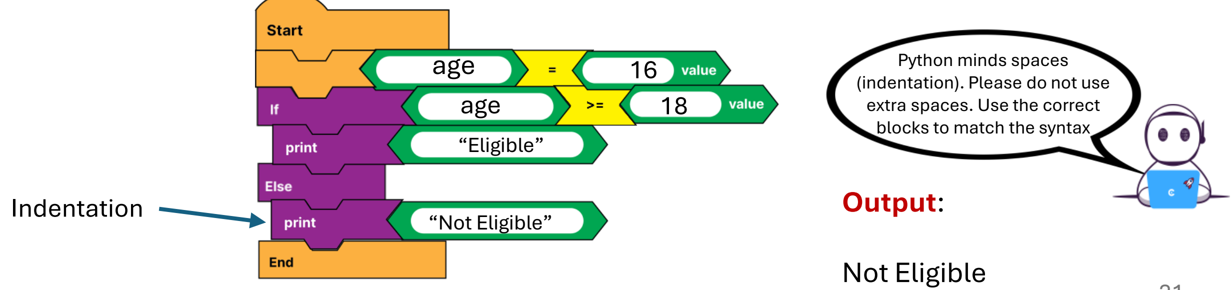
Eligible

> ***Code Explanation:*** *This code is a higher-level one however, used here to showcase how we could use the 'extended' End block to maintain the code alignment. Maintaining the alignment spacing or indentation is an important aspect of Python Programming. Remember that Python minds extra spaces, hence, do not leave any extra spacing unnecessarily.*

Understanding Indentation

Indentation in programming refers to the spaces or tabs used at the beginning of a line of code to define its structure and hierarchy. In **Python**, indentation is not just a matter of style; it is a fundamental part of the syntax. It indicates that an indented block of code is under the above statement. In the below example print(Do not use extra spaces in Python as it will violate its syntax.

Unlike many other programming languages where braces {} or keywords are used to define blocks of code, Python relies on **indentation** to determine the grouping of statements.



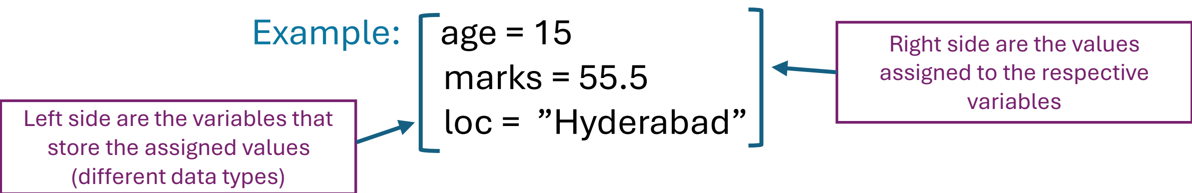
Output:

Not Eligible

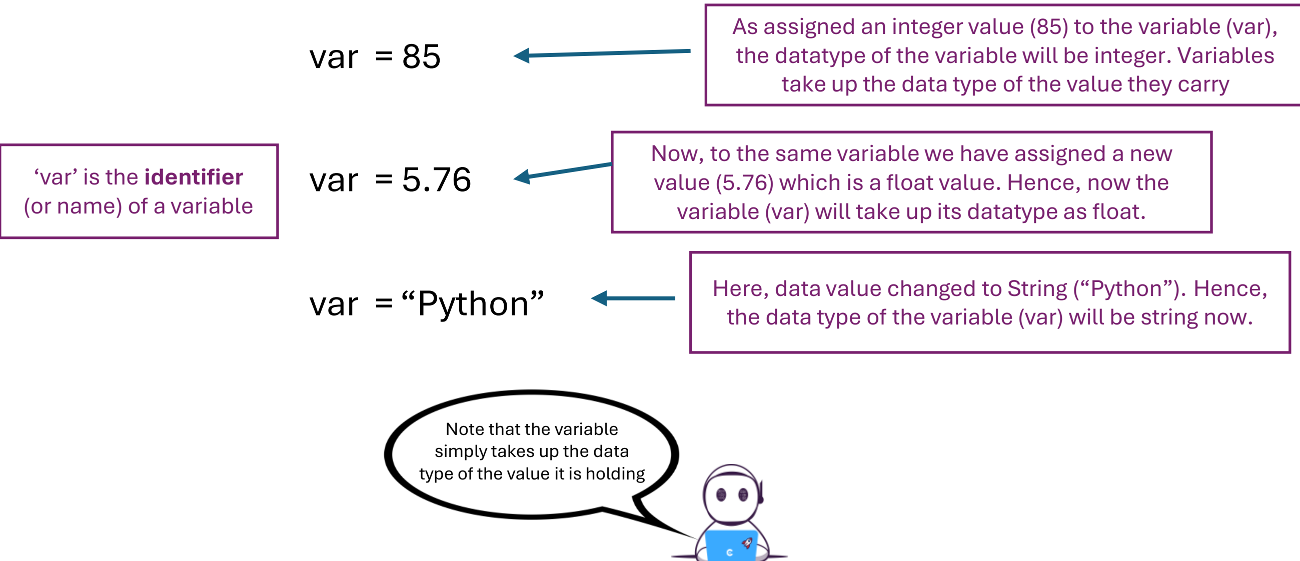
CEDURA © Copyrighted Content. All rights reserved

Exploring Variables

Variables in Python are the containers that store data values inside them. We assign values (integer, float, string or Boolean) to a variable. After assigning, these variables can be referenced and manipulated throughout a program. Every variable should have a meaning name which we call as an 'identifier'.



Python is dynamically typed, meaning you don't need to declare the type of a variable explicitly—Python determines the type based on the value assigned. For instance, writing x = 10 creates an integer variable x with the value 10, and later assigning x = "Hello" reassigns it to a string. This flexibility makes Python easy to use and beginner-friendly.



Python provides a simple syntax for creating variables, requiring only an assignment operator (=) and a valid variable name. Variables are fundamental in programming because they enable you to perform calculations, store user input, and process data dynamically. Variable names must start with a letter or an underscore and can contain letters, numbers, and underscores but cannot have spaces or special characters.

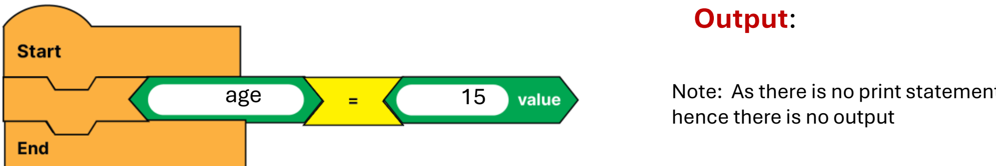
Python is case-sensitive, so 'Name' and 'name' are treated as different variables. This rule applies even for the data or value.

Valid Variable Identifiers :	Invalid Variable Identifiers :
stu_Name	1stu_Name
subject1	subject@new
empSal	

CEDURA © Copyrighted Content. All rights reserved

PRACTICE 1

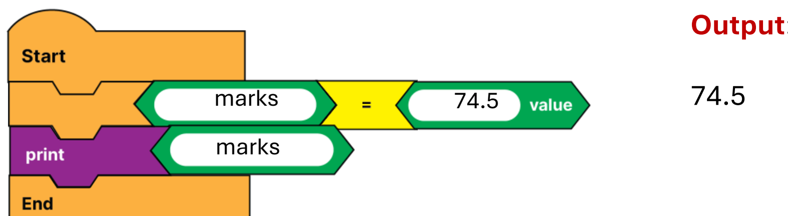
Problem Statement: *Build a Python Block Code to assign a variable with a value.*



Code Explanation: *In this program, we have considered a variable 'age' (identifier of a variable) and assigned it with an integer value (15). Now, the variable 'age' is storing a value i.e. 15. This variable could be used in the program as required. Note that the data type of the variable 'age' will be ' integer' as it stores an integer value (15) in it.*

PRACTICE 2

Problem Statement: *Build a Python Block Code to assign a variable with a value and print it.*



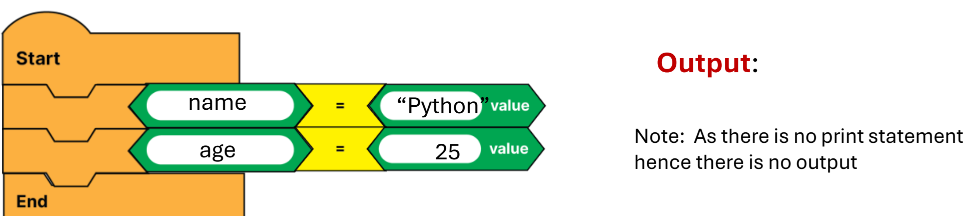
Output:

74.5

Code Explanation: *In this program, we have two steps. First, we are assigning a float value (74.5) to the variable 'marks' (identifier of a variable). Second, we are using the 'marks' variable and printing it. Once the 'print' function is executed the output will display as 74.5, as this is the value the variable 'marks' is storing inside it. Note that the data type of the variable 'marks' will be ' float' as it stores a float value (74.5) in it.*

PRACTICE 3

Problem Statement: *Build a Python Block Code to assign multiple values to multiple variables.*



Code Explanation: *In this program, we have two rows of blocks where we are considering two different variables ('name' and 'age') and assigning two different values respectively ("Python" and 25). Now the first variable data type will be 'string' as it stores a string value and the second variable data type will be 'integer' as it has an integer in it.*

CEDURA © Copyrighted Content. All rights reserved

Identifying Variable Data Type

We have learned that every variable has its own data type, and it depends on the type of value it is storing in it. For instance, if you assign an integer to a variable, the data type of that variable will be an integer as it has an integer value in it. However, while working with variables sometimes you would like to find out the data type of the variable. This situation may occur when multiple developers are working on a big program and everyone is creating new variables. Before working on an unknown variable, created by someone else, you would like to find its data type. This is where we will use the 'type()' function.

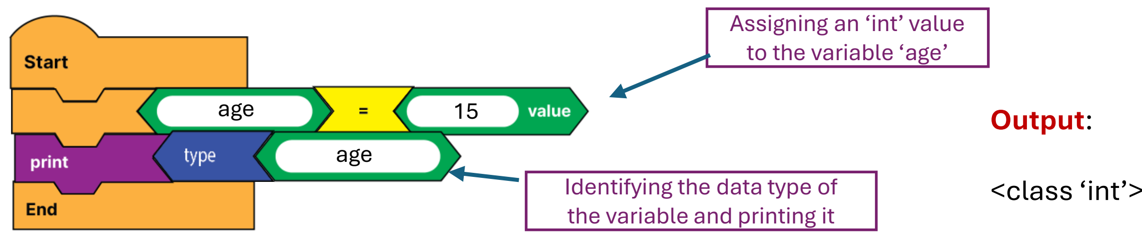
Syntax: type(object)

Example:
```
age = 15
marks = 55.5
loc =  "Hyderabad"
```

In Python, type() function is used to determine the data type of a given variable or object. The type() function is a powerful tool for understanding the nature of a variable or object, especially when working in dynamically typed languages like Python where variables do not have explicit type declarations.

PRACTICE 1

Problem Statement: *Build a Python Block Code to assign a variable with a value and find its data type.*

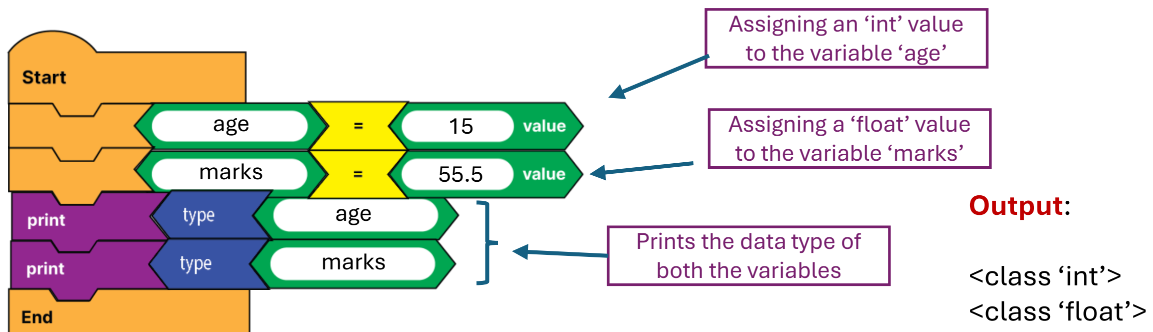


Code Explanation: *In this program, we have considered a variable 'age' (identifier of a variable) and assigned it with an integer value (15). Now, the variable 'age' is storing a value i.e. 15 hence, its data type will be 'integer'. In the next line, we are using the type() function to identify the data type of the variable 'age'. In this case, the output will be <class 'int'> as the 'age' variable stores an integer in it.*

PRACTICE 2

Problem Statement: *Build a Python Block Code to assign multiple variables with data and find their respective data types*

CEDURA © Copyrighted Content. All rights reserved



Output:

<class 'int'>
<class 'float'>

> ***Code Explanation:*** *In this program, we have two variables 'age' and 'marks' assigned with two different values 15 (integer) and 55.5 (float), respectively. In the subsequent lines of code, we are trying to identify the data type of these variables individually by using the type() function. The output would display <class 'int'> for the 'age' variable and <class 'float'> for the 'marks' variable.*

TOPIC ASSIGNMENT

1. Build a Python Block Code to assign a value to a variable and print it.
2. Build a Python Block Code to assign two variables with different values.
3. Build a Python Block Code to assign a value to a variable and then change/update the value
4. Build a Python Block Code to find out the data type of any two variables with given values: price = 56.45, loc = "Hyd", count = 56.
5. Build a Python Block Code to assign a name and a subject mark of a student and print them. Also, print its data type.
6. Build a Python Block Code to assign a Float value to a variable and find its data type.
7. Build a Python Block Code to assign a Boolean value to a variable and find its data type.
8. Build a Python Block Code to assign a String value to a variable and find its data type.

CEDURA © Copyrighted Content. All rights reserved

Type Casting in Variables

Type casting in Python refers to converting one data type into another. This process is essential when working with data from different sources or formats, ensuring compatibility and enabling operations that would otherwise result in errors. Python provides built-in functions such as int(), float(), str(), list(), and tuple() for explicit type casting. For example, converting a string "123" to an integer can be done using int("123"), allowing mathematical operations on the converted value. Similarly, typecasting can be used to handle user inputs, which are typically strings, by converting them into the required types for calculations or logical operations.

To typecast or change the data type of variables we use below blocs:

int — To convert the variable into an integer data type

float — To convert the variable into a float data type

string — To convert the variable into a string data type

Example 1:

age = 15 → Data type of ‘age’ variable is ‘**integer**’

type(age)

Output: <class ‘int’>

To convert ‘age’ variable into ‘**float**’

float(age) ← Converts variable into float (15.0)

type(age)

Output: <class ‘float’>

To convert ‘age’ variable into ‘**string**’

str(age) ← Converts variable into string (‘15’)

type(age)

Output: <class ‘str’>

Start

age = 15 value ← Assigning an ‘int’ value to the variable ‘age’

print type age ← Printing the data type of the variable (Output: <class int>)

age_new = float age ← Changing the content of the variable to ‘float’ and assigning it to a new variable ‘age_new’

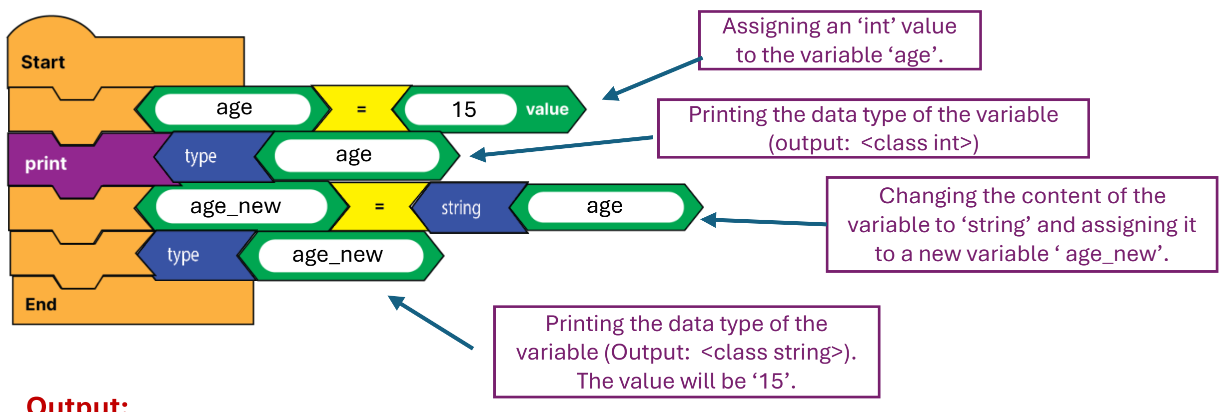
type age_new ← Printing the data type of the variable (Output: <class float>). The value will be 15.0

End

Output:

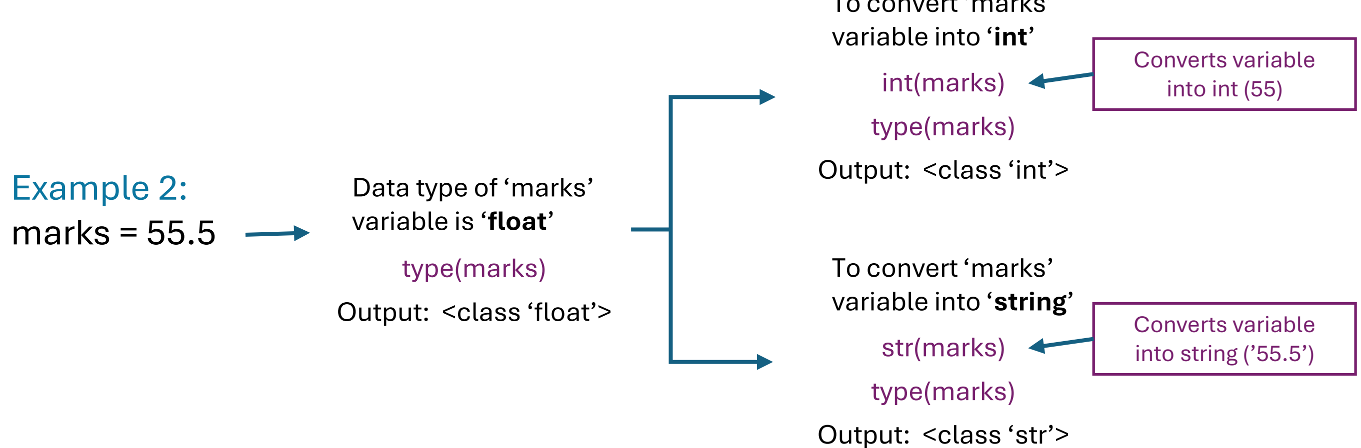
<class ‘float’>

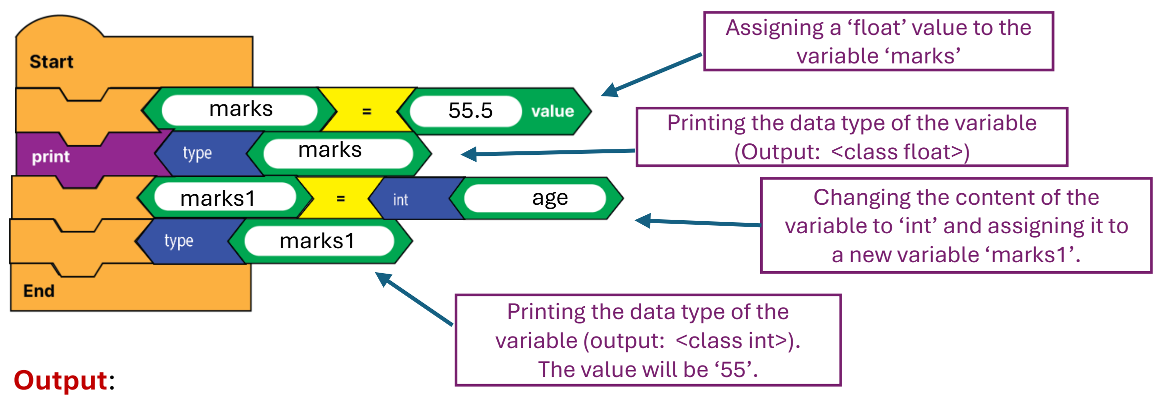
CEDURA © Copyrighted Content. All rights reserved



Output:

<class ‘str’>

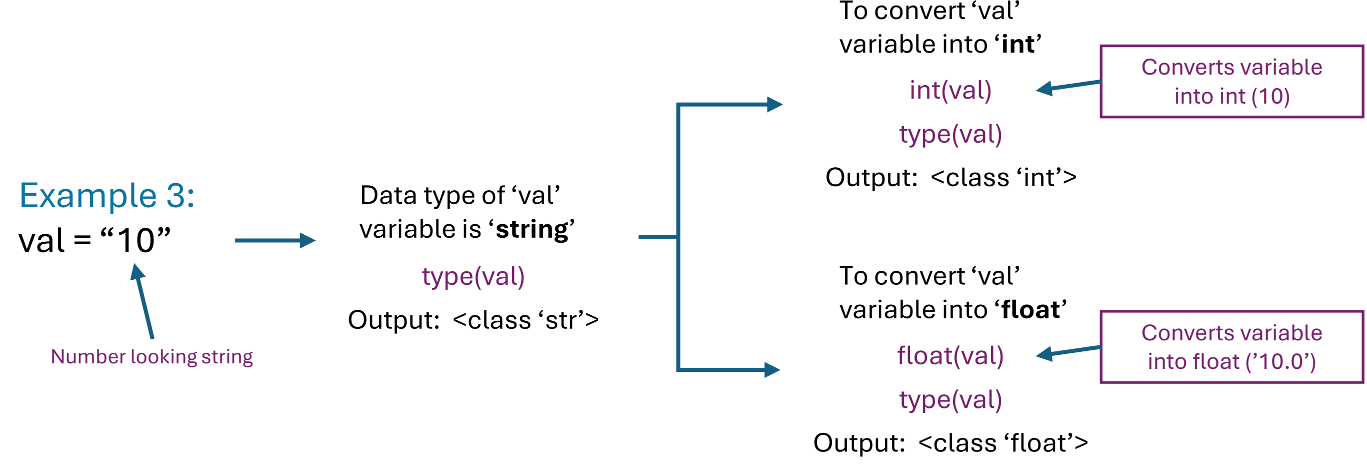


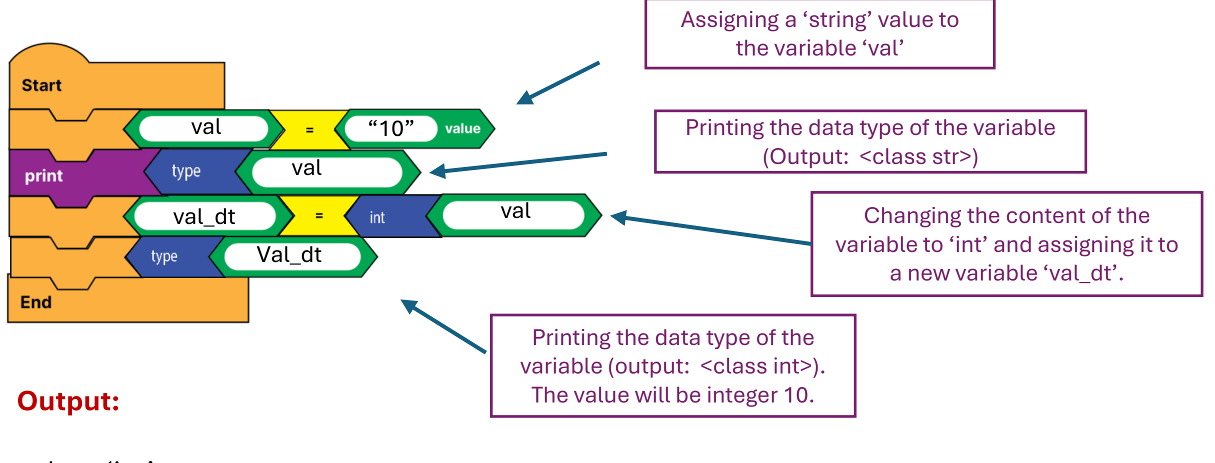


Output:

<class ‘int’>

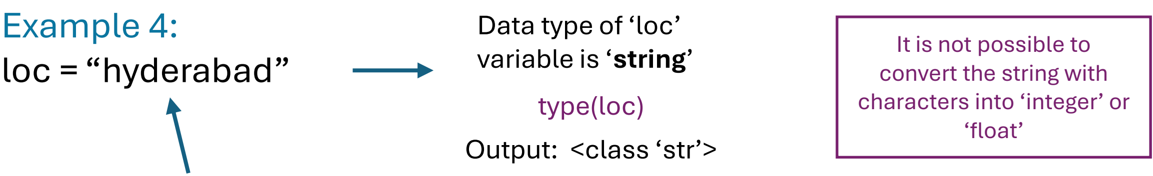
CEDURA © Copyrighted Content. All rights reserved





Output:

<class ‘int’>



Note: *There are other types of data type conversions however, for the current level of learning we will focus on these three primary type casting (integer, float, string)*

CEDURA © Copyrighted Content. All rights reserved

TOPIC ASSIGNMENT

1. Build a Python Block Code to convert the content of the variable price = 24.5 into an integer.
2. Build a Python Block Code to convert the content of the variable height = 5 into a float.
3. Build a Python Block Code to convert the content of the variable name = “56” into an integer.

CEDURA © Copyrighted Content. All rights reserved

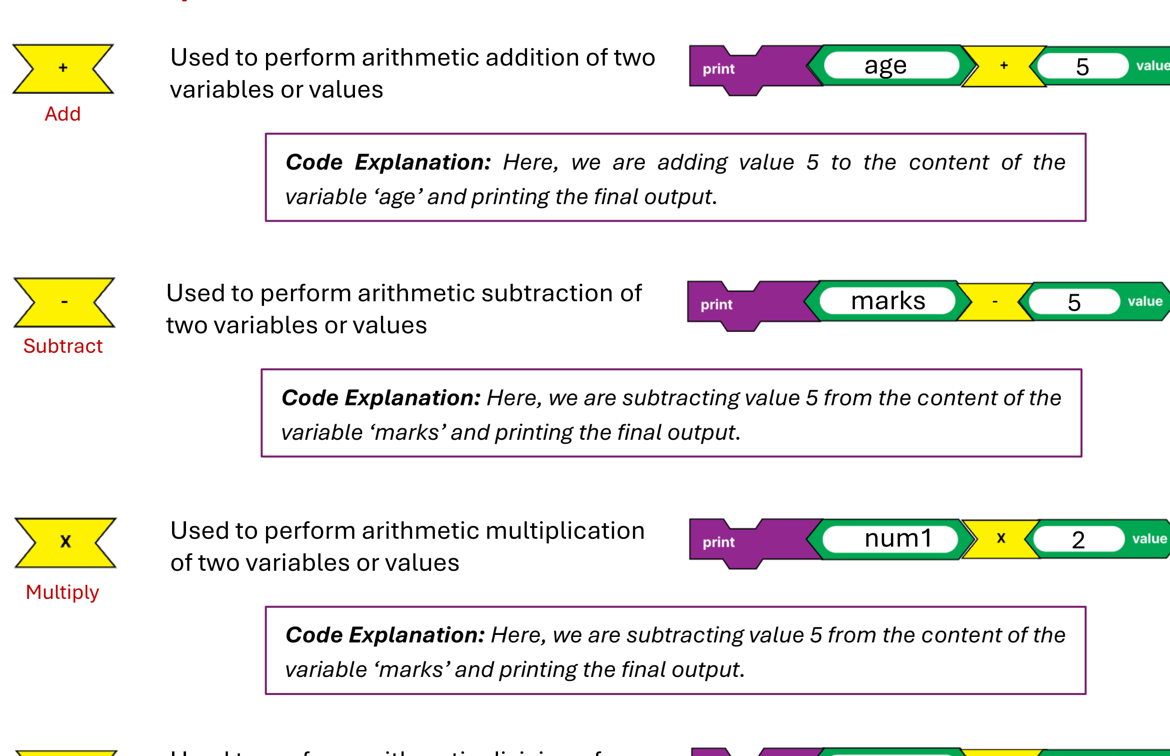
Operators in Python

Operators in Python are special symbols or keywords used to perform operations on variables and values. They are the building blocks of any programming language and help in creating expressions for calculations, comparisons, and logical decisions. Python supports several types of operators, including

- **arithmetic operators** (e.g., +, -, *, /) for mathematical operations
- **assignment operators** (e.g., =, +=, -=) to assign values to variables
- **comparison operators** (e.g., ==, !=, >, <) to compare values and return Boolean results
- **membership operators** (e.g., in, not in) to check the presence of an item in a sequence
- **logical operators** (e.g., and, or, not) to combine more than one expression
- **bitwise operators** (&, |, ^) for operations on binary numbers
- **identity operators** (is, is not) are used to compare the memory location of two objects

These operators allow developers to handle simple and complex computations efficiently. Operators make Python a versatile language for different programming tasks, from simple arithmetic to advanced data manipulation. Understanding and using operators effectively is crucial for writing efficient and readable Python programs.

Arithmetic Operators



Used to perform arithmetic addition of two variables or values

Code Explanation: *Here, we are adding value 5 to the content of the variable 'age' and printing the final output.*

Used to perform arithmetic subtraction of two variables or values

Code Explanation: *Here, we are subtracting value 5 from the content of the variable 'marks' and printing the final output.*

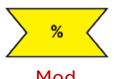
Used to perform arithmetic multiplication of two variables or values

Code Explanation: *Here, we are subtracting value 5 from the content of the variable 'marks' and printing the final output.*

Used to perform arithmetic division of a variable (or value) by another

Code Explanation: *Here, we are dividing the content of the 'month' variable with 12 and the result will be printed.*

CEDURA © Copyrighted Content. All rights reserved



Mod

Mod returns the remainder of the division operation.

Code Explanation: *Here, the content of the 'num' variable will be divided by 2. After division, the* ***remainder*** *will be printed.*

Floor Division

Floor division divides two numbers and returns the largest integer less than or equal to the result

Code Explanation: *Here, the content of the 'num' variable will be divided by 3 and outputs the value after discarding the fractional or decimal part of the division result and returning the integer part. For instance, If the value in the 'num' is 10 then, 10 // 3 will give an output as 3 (rounding to the nearest integer)*

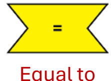
It is different from division which results in the exact quotient as a float number

Exponentiation

Exponentiation raises a number (the base) to the power of another number (exponent)

Code Explanation: *Here, the content of the 'val' variable (base) will be raised to the power of 2 (exponent) and the result is printed.*

Assignment Operator



Equal to

This operator is used to assign a value (or another variable) to a variable.

Code Explanation: *Here, we are assigning a value 5 to the variable 'age'.*

Comparison Operators

Lesser than

This operator is used to check whether a value or a variable is lesser than another value (or a variable)

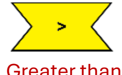
Code Explanation: *Here, we are checking whether the age is less than 15. If the age is less than 15 it will return TRUE, otherwise it will return FALSE*

CEDURA © Copyrighted Content. All rights reserved

Lesser than or Equal to

This operator is used to check whether a value or a variable is lesser than or equal to another value (or a variable)

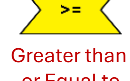
Code Explanation: *Here, we are checking whether the age is less than or equal to 15. If the age is less than or even equal to 15 it will return TRUE, otherwise it will return FALSE*



Greater than

This operator is used to check whether a value or a variable is greater than another value (or a variable)

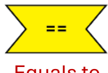
Code Explanation: *Here, we are checking whether the age is greater than 18. If the age is greater than 18 it will return TRUE, otherwise it will return FALSE*



Greater than or Equal to

This operator is used to check whether a value or a variable is greater than or equal to another value (or a variable)

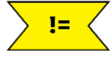
Code Explanation: *Here, we are checking whether the age is greater than or equal to 18. If the age is greater than or even equal to 18 it will return TRUE, otherwise it will return FALSE*



Equals to

This operator is used to check whether a value or a variable is equal to another value (or a variable)

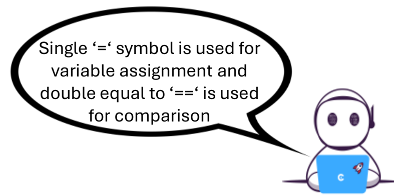
Code Explanation: *Here, we are checking whether the age is EQUAL to 18. If the age is equal to 18, it will return TRUE, otherwise it will return FALSE*



Not Equal

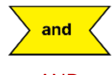
This operator is used to check whether a value or a variable is not equal to another value (or a variable)

Code Explanation: *Here, we are checking whether the age is NOT EQUAL to 18. If the age is not equal to 18, it will return TRUE, otherwise it will return FALSE*



CEDURA © Copyrighted Content. All rights reserved

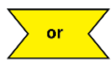
Logical Operators



AND

This operator is used to combine two or more expressions/conditions. The output will be TRUE only when both the expressions are TRUE and vice-versa.

Code Explanation: *Here, we have two expressions - age greater than 18 and height greater than 6. When these two conditions result in TRUE then only the resultant will be TRUE*



OR

This operator is used to combine two or more expressions/conditions. The output will be TRUE when one of the expressions is TRUE. The output will be FALSE only when both the expressions result FALSE

Code Explanation: *Here, we have two expressions - age greater than 18 and height greater than 6. For the output to be TRUE any of the expressions should be TRUE. The output will be FALSE only when the age is not greater than 18 and the height is not more than 6.*

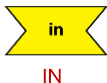
NOT

This operator is used to invert the truth value of a condition or Boolean expression. It is a unary operator, meaning it operates on a single operand. When the condition or expression TRUE, it makes it FALSE and vice-versa.

Code Explanation: *Here, we have an expression to check whether the age is greater than 18 or not. In case the age is greater than 18 it should result in TRUE. However, as we are using 'NOT' which will invert the truth value the output will be FALSE.*

CEDURA © Copyrighted Content. All rights reserved

Membership Operators



IN

This operator is used to check whether a specific value exists within a sequence or collection such as strings, lists, tuples, etc. Result TRUE if the value is present.

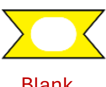
Code Explanation: *Here, we have a sequence (having a series of items or values). The intention is to check whether a value 5 exists within this sequence or not. If the value is found in the sequence the result will be TRUE, else FALSE. In this case, we have the value 5 available in the sequence hence the result will be TRUE.*

NOT IN

This operator is used to check whether a specific value **does not exist** within a sequence or collection, such as strings, lists, tuples, etc. Result TRUE if the value is **not present**.

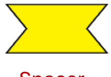
Code Explanation: *Considering the earlier example using the 'not in' operator, as the value 5 is available in the sequence the output will be FALSE. It works opposite to the 'in' operator. It is used to verify that an item is absent from a collection of items.*

Other Operators



Blank

This blank operator block is used to replace any operator. Supplied with blank space so that you could write the operator symbol or text as you like to fulfil your coding requirement.



Spacer

This spacer operator block can be used between variables or values just as a spacer. You can also use this as a blank operator and write a symbol or text when you are falling short of operator blocks.

Comma Separator

This comma separator block can be used while working on multiple variables. Can be used while assigning multiple values (input function) or while printing multiple variables.

CEDURA © Copyrighted Content. All rights reserved

TOPIC ASSIGNMENT

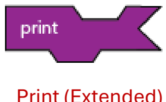
1. Build a Python Block Code to assign two values as integers to different variables and perform below athematic operations.
    a. Addition
    b. Subtraction
    c. Multiplication
    d. Division
    e. Floor Division
2. Build a Python Block Code to assign two values as integers to different variables and perform the below comparison operations.
    a. Greater than
    b. Lesser than
    c. Greater than or Equal to
    d. Lesser than or Equal to

CEDURA © Copyrighted Content. All rights reserved

Print Function

The print() function in Python is one of the most commonly used built-in functions, allowing us to display results or output. Its primary purpose is to output data, making it a vital tool for debugging and communicating results to users. The print() function takes one or more arguments, separated by commas, and displays them as a single line of output. For example, print("Hello, World!") prints the text Hello, World! to the console. Additionally, it supports formatting through tools like f-strings, concatenation, or the format() method, enabling developers to create clear and informative outputs.

Print Function



Print (Extended)

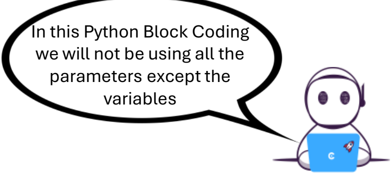
A key feature of the print() function is its flexibility. It automatically adds a new line character (\n) at the end of each output, though this behaviour can be customised with the end parameter. For instance, print("Hello", end=" ") outputs Hello (with end space) without moving to a new line.

The function also supports custom separators between multiple arguments via the sep parameter, such as print("Hello", "World", sep="-"), which outputs Hello-World.

Furthermore, the print() function can handle various data types, including strings, integers, floats, and even complex objects, making it an essential tool for Python programming. Its versatility and ease of use ensure that the print() function is a foundational element for beginners and professionals alike.

In the Python Block Coding we will learn to use print in basic format and work on with variables to get you familiar with this useful function.

Print Block



Syntax:

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

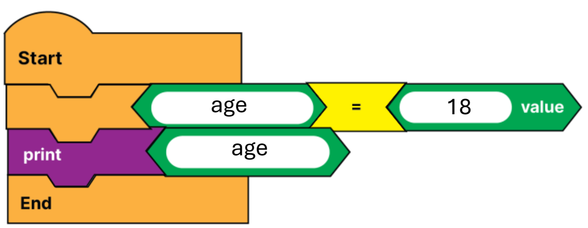
Parameters:

1. *Sep:
   • Represents one or more objects (values, variables, or expressions) to be printed.
   • Multiple objects can be separated by commas.
2. sep *(Optional)*:
   • Defines the string used to separate the objects. Default: A single space (' ').
   • Example: print("Hello", "World", sep="-") → Output: Hello-World.
3. end *(Optional)*:
   • Defines the string appended after the printed output.
   • Default: A newline character ('\n'), which moves the cursor to the next line.
   • Example: print("Hello", end="!") → Output: Hello! (without a new line).
4. file *(Optional)*:
   • Specifies the file or stream where the output is sent. Default: sys.stdout (console output).
   • Example: print("Hello", file=open('output.txt', 'w')) writes the output to a file.
5. flush *(Optional)*:
   • A Boolean value (True or False) that forces the output buffer to be flushed immediately if set to True. Default: False. Used when working with real-time streams.

CEDURA © Copyrighted Content. All rights reserved

PRACTICE 1

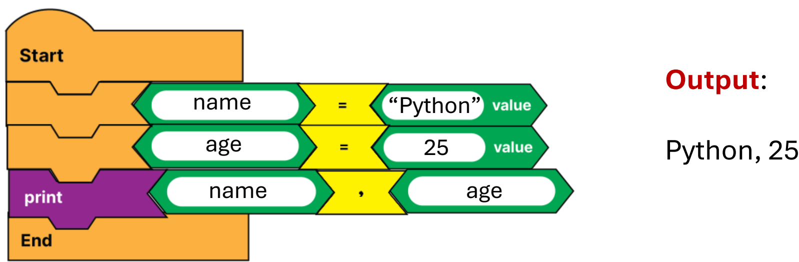
Problem Statement: *Build a Python Block Code to load a variable with a value and print it.*



> ***Code Explanation:*** *In this program, we have considered a variable 'age' and assigned value 18 to it. Now using the print function, we are printing the content of the 'age' variable.*

PRACTICE 2

Problem Statement: *Build a Python Block Code to print multiple variables.*



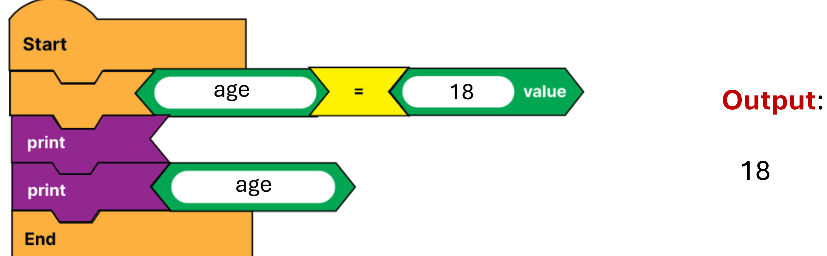
Output:

Python, 25

> ***Code Explanation:*** *In this program, we have considered two variables 'name' and 'age' and assigned them with different values. Using a single print function we have assigned both the variables as objects to it. The print function will print both variables. This is how you could handle multiple variables in print.*

PRACTICE 3

Problem Statement: *Build a Python Block Code to print a blank line.*



Output:

18

CEDURA © Copyrighted Content. All rights reserved

Code Explanation: *In this program, we have assigned a value to the variable. In the next line, we have used a blank print function. This will print a blank line. This is followed by another print function that prints the variable. This way, you could print blank lines in the output.*

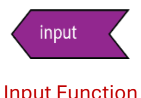
TOPIC ASSIGNMENT

1. Write the Syntax of the print function and explain all its parameters.
2. Build a Python Block Code to assign an integer value to a variable and print its data type.
3. Build a Python Block Code to assign product name, price, and quantity and print any two variables, individually.
4. Build a Python Block Code to assign product name, price, and quantity and print any two variables using a single print function.
5. Build a Python Block Code to assign salary and designation and print them individually with a blank line in between.
6. Find the total age of Raju (29 years) and Rajiya (31 years) and print it.
7. What is the total bill amount if the prices of two products are Rs.5 and Rs. 6.50

CEDURA © Copyrighted Content. All rights reserved

Input Function

The input() function in Python is a built-in function used to accept user input from the console during program execution. It allows developers to interact with users by pausing the program until the user provides the required input. The input() function takes an optional argument, which is a prompt string displayed to the user, guiding them on what to enter.



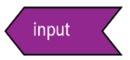
Input Function

Note that the input function always converts and stores the user's input as a string in the variable. Since the input() function always returns the input as a string, additional type conversion (e.g., using int() or float()) is required if numerical input is needed. Recap the type-casting topic we discussed earlier. Below are the important points about the input function.

- Used to seek value from the user
- Makes programs interactive
- Work on real-time values
- Accepts both numerical and string values
- Converts everything value into a string
- It is always used along with a variable

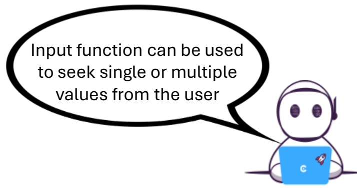
Despite its simplicity, the input() function is an essential tool for learning Python and creating basic interactive programs. Using input and print functions together will make the program very interactive and useful while developing solutions.

Input Block



Syntax:
input(prompt)

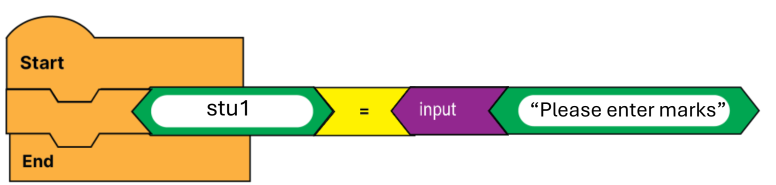
Parameters:
prompt Optional)



CEDURA © Copyrighted Content. All rights reserved

PRACTICE 1

Problem Statement: *Build a Python Block Code to seek input from the user.*



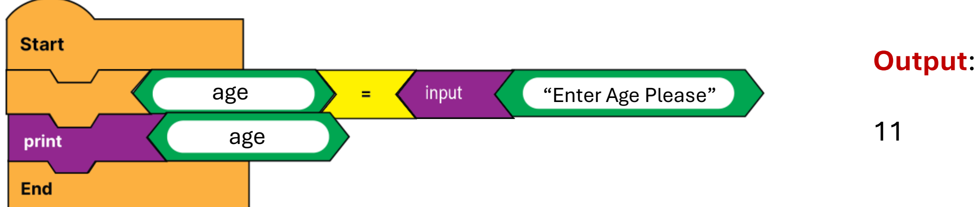
Output:

Note: As there is no print statement hence there is no output

Code Explanation: *In this program, we use the input() function to seek marks from the user. As we know, every input function should have a variable attached to it. When the user provides the value of the marks, the same would be assigned to the variable 'stu1'. This is how the input function works. Whenever a value is given, the same would be stored in a variable. Now, you can work on 'stu1' variable or print it.*

PRACTICE 2

Problem Statement: *Build a Python Block Code to seek input from the user and print it.*

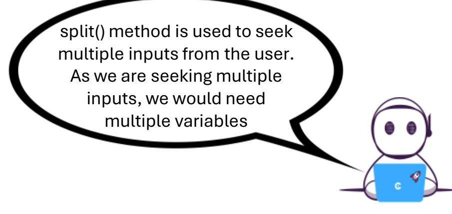


Output:

11

Code Explanation: *In this program, we use the input() function to seek marks from the user. This program is the same as above, where we are using an input function to seek age input from the user. As the user provides a value, the same would be assigned to the variable named 'age'. Now, the variable 'age' is storing the value given by the user. In the next line, we are printing the variable 'age' content using the print function.*

Split Block



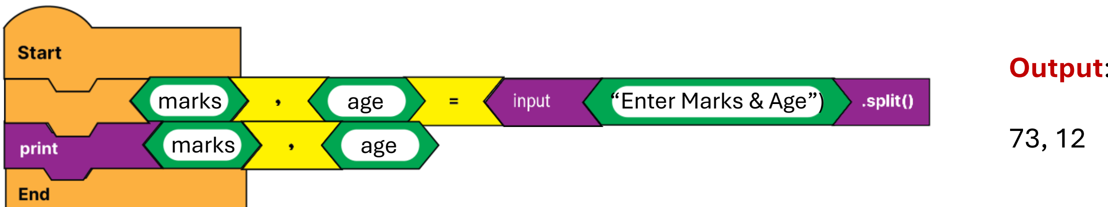
Syntax:
input(prompt).split()

split() method is always used along with the input function

CEDURA © Copyrighted Content. All rights reserved

PRACTICE 1

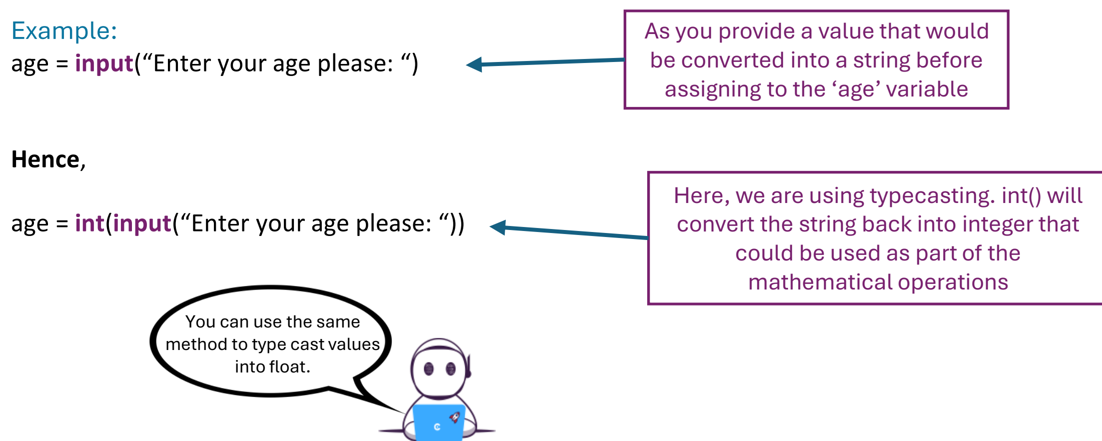
Problem Statement: *Build a Python Block Code to seek multiple values from the user and print them.*



Code Explanation: *In this program, we are using the split() method of the input function. This method is used to seek multiple values from the user and load them into multiple variables, as shown in the 2nd life. If you observe, we have two variables 'i.e., 'age' and through the input function, we are seeking two values, in the same sequence. The split() method would split the entered values and load them into the respective variables. The* ***first entry will go into the first variable*** *i.e., 'marks', and the* ***second entry will go into the second variable*** *i.e., 'age'. (Here, we are using short variables to minimize the size of the statement)*

Type Casting - int / float function Blocks

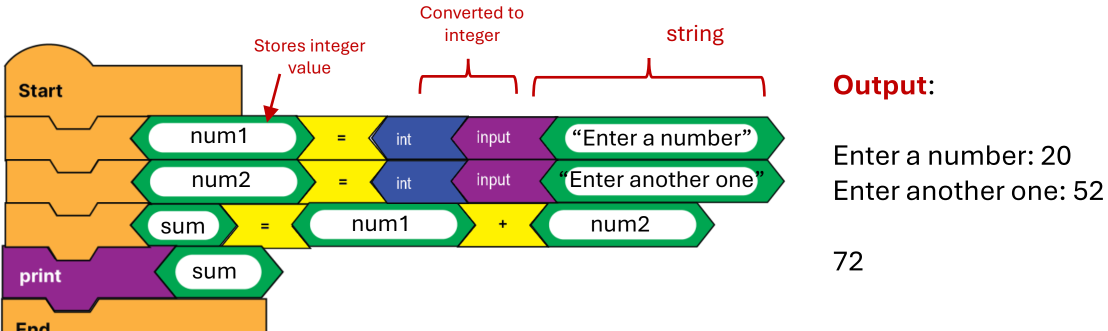
Recap type-casting, where we learned to convert from one data type to another - Type Casting. We have also learned that the input() function will always convert all the values into 'strings'. In case you enter or input a number(say 18), it will be converted into a string ('18') and then assigned to the variable. As it is a string value now, you cannot do any mathematical operations such as addition, multiplication, etc. This is a common issue we encounter while using the input() function. Hence, we use Type Casting here.



CEDURA © Copyrighted Content. All rights reserved

PRACTICE 1

Problem Statement: *Build a Python Block Code to seek two values from the user and conduct a mathematical operation.*



Output:

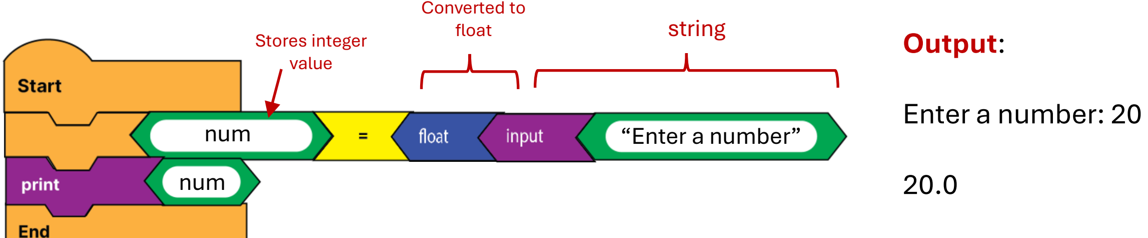
Enter a number: 20
Enter another one: 52

72

> ***Code Explanation:*** *In this program, we are using data type casting to overcome the issue we face with the input function. As the input function converts the value you enter into a string before assigning it to a variable, we are using the 'int()' function to convert the value back to an integer. We can also convert the value into a float by using the 'float()' function. Once both the input values are converted into integer values and stored in their respective variables, we can perform the arithmetic operations. Without type casting into integers or float we cannot perform arithmetic operations, as they are strings.*

PRACTICE 2

Problem Statement: *Build a Python Block Code to seek a value from the user and convert it into a float data type.*

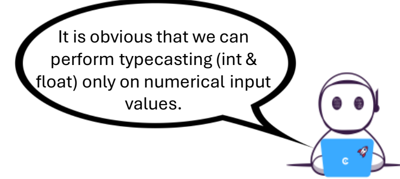


Output:

Enter a number: 20

20.0

> ***Code Explanation:*** *In this program, we are demonstrating how to convert the input value into a float data type. We are aware that the input() function will convert the value into a string, hence, we are using the float() function to convert the string into a float value and assign to the variable 'num'. In the next statement, we are printing the content of the num, which is a float value now.*



CEDURA © Copyrighted Content. All rights reserved

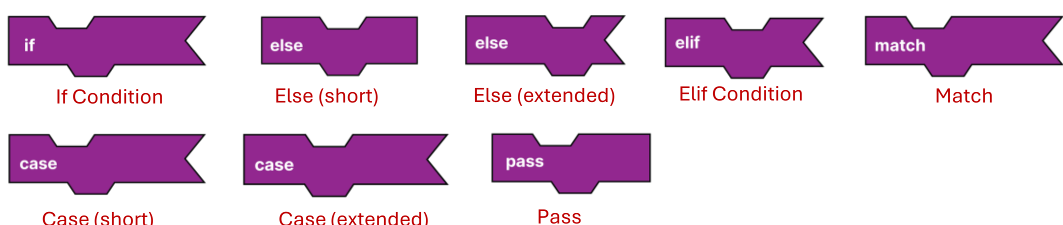
TOPIC ASSIGNMENT

1. Build a Python Block Code to seek an Employee’s name and print it.
2. Build a Python Block Code to seek the ’first name’ and ‘last name’ of a student and print them together.
3. Build a Python Block Code to seek an Employee's name and salary using a single input function and print them.
4. Build a Python Block Code to seek the price (float) value of a product, convert it into an integer and print it.
5. Build a Python Block Code to seek the Employee’s salary and add Rs.100 and print the final salary.
6. Build a Python Block Code to seek Science and Social Marks from a student and print the total marks.
7. Build a Python Block Code to seek the price of a product add Rs.100 as GST and print the final amount.

CEDURA © Copyrighted Content. All rights reserved

CONDITIONAL Blocks

Conditional blocks in Python are a fundamental feature that allows programmers to make decisions and control the flow of their code. These blocks use conditions, typically Boolean expressions (TRUE or FALSE), to determine which lines of code under the 'if' block should be executed. To teach and practice Python Conditional Blocks we have given the below blocks as part of the kit.



Python uses the if, elif, and else statements to structure conditional logic. For example, an if block checks whether a condition is True and executes the associated code; otherwise, it moves to the next block or terminates if there are no further conditions. This makes conditional blocks essential for creating dynamic programs that respond to different inputs or situations.

One of the advantages of Python's conditional blocks is their simplicity and readability. The indentation-based structure ensures that the code is visually clear, which is particularly helpful for beginners. Conditional blocks are widely used in real-world applications such as validating user input, controlling loops, or building decision-making algorithms. They also support nested conditions, allowing programmers to handle more complex logic by combining multiple conditions within the same program. With features like logical operators (and, or, not) and comparison operators (==, !=, <, >), Python's conditional blocks empower developers to write concise and efficient code for a wide variety of tasks.

Let's learn about each of these blocks in detail and use them as part of our problem-solving.

if Conditional Block

Syntax:

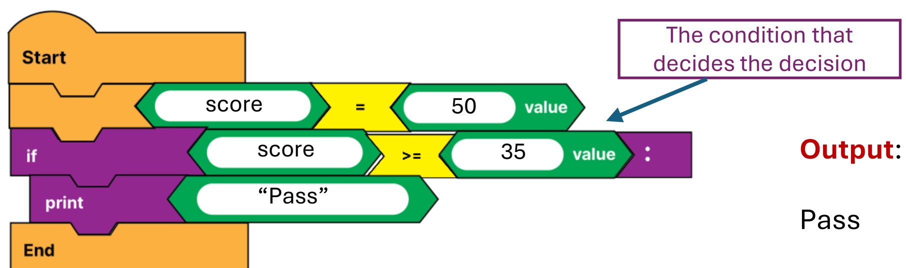
```
if <condition>:
  # Code to execute if the condition is True
```

The "if" Block is used for decision-making and to make the program dynamic. Every 'if' block should have a conditional block. This condition could be a comparison, membership, or such that would give a Boolean output (TRUE or FALSE). If the condition results TRUE, meaning the condition is satisfied, the block under the 'if' Block will be executed. In case the condition results FALSE, meaning the condition is not satisfied, then it would not execute anything.

CEDURA © Copyrighted Content. All rights reserved

PRACTICE 1

Problem Statement: *Build a Python Block Code to check whether an individual passed in an exam.*



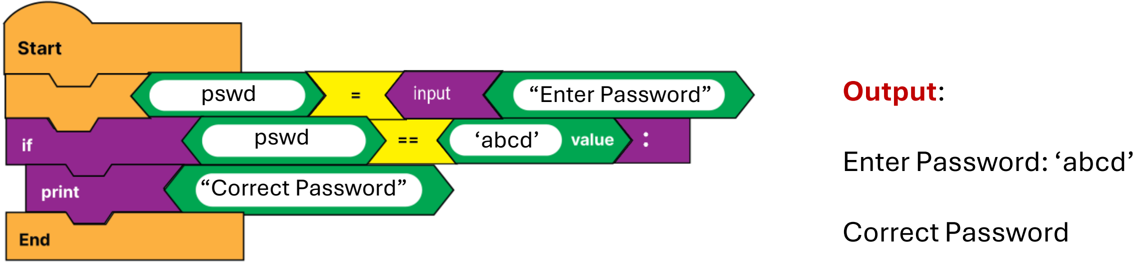
Output:

Pass

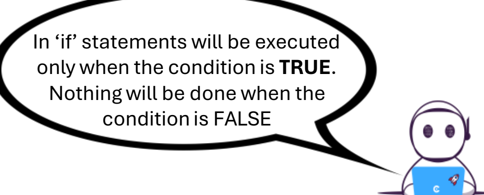
> ***Code Explanation:*** *The logic here is that the marks should be greater than or equal to 35. Hence, once we assign the marks (50) to the variable (score), we are using an 'if' block. The 'if' block will have a condition to check whether the score variables have a value that is greater than or equal to 35. If the condition output is TRUE, the control will go under the 'if' block and execute the print message "pass" and the program will END. In case the condition is FALSE, the control nothing will happen or execute and the program will END.*

PRACTICE 2

Problem Statement: *Build a Python Block Code to check the password entered by the user. (Assume the correct password as 'abcd'.)*

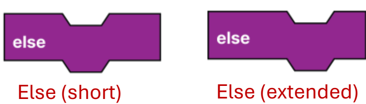


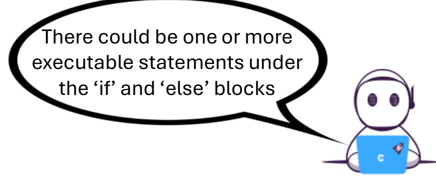
> ***Code Explanation:*** *In this program, we are using both the input() function and the if conditional statement. First, we are seeking a password from the user using the input() function. As the user enters the password, we assign it to a variable 'pswd'. Following this we are using the 'if' block with the condition where we check whether the user - given value is equal to 'abcd'? In case the user entered value is equal to 'abcd' then the if block's condition will result in TRUE and the print statement under the 'if' block will be executed. If the entered password is not 'abcd' then the condition fails and the print block under the 'if' block will not be executed.*



CEDURA © Copyrighted Content. All rights reserved

else Block





Syntax:
if condition:
  *# Code to execute if the condition is **True***
else:
  *# Code to execute if the condition is **False***

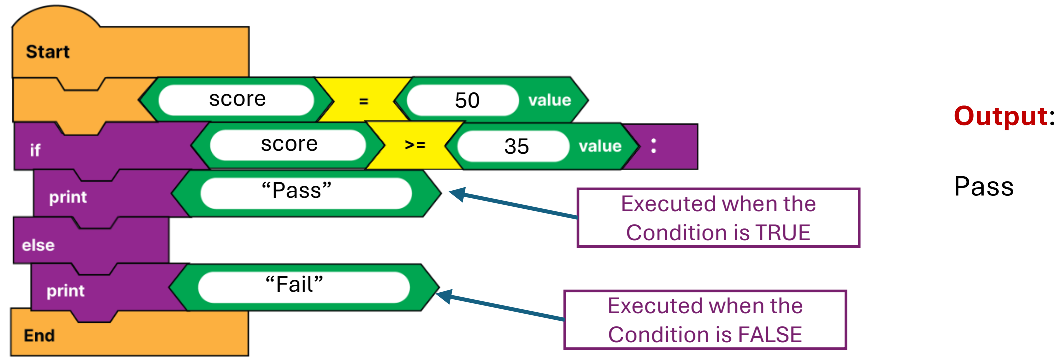
Note: Extended else block is used in the nested if examples.

While using the 'if' conditional statement we have noted that statements will be placed under the 'if' block and will be executed only with the condition is TRUE. If the condition is FALSE nothing will be executed. But in real life, we may need to execute 1 or more statements when the condition is TRUE or FALSE. This is where we use the 'if-else' blocks.

The if-else statement is an extension of 'if' conditional statement and is used to implement decision-making in programs. It allows the code to execute different blocks of code based on specific conditions (TRUE or FALSE). The 'if' block introduces the condition, which evaluates to either TRUE or FALSE. If the condition is TRUE, the code block **under the if** statement is executed. If the condition is FALSE, the code block **under the optional else** statement is executed instead. This makes if-else statements highly versatile for controlling program flow and handling diverse scenarios.

PRACTICE 1

Problem Statement: *Build a Python Block Code to declare whether an individual passes or fails an exam.*



Output:

Pass

Code Explanation: *While using the 'if' block, we executed a statement only when the condition is TRUE. In this code, we have both 'if' and 'else', each with a statement under it. This means that if the condition (whether the score is greater than or equal to 35) results in TRUE, the statement under the 'if' block will execute, and if the condition results in FALSE the statement under the 'else' block will execute. We say that depending on the condition at least one block of code will be executed (either under 'if' block or under the 'else' block)*

CEDURA © Copyrighted Content. All rights reserved

elif Block

The 'elif' block is a combination of two blocks 'else' + 'if'. You can have multiple 'elfi' blocks

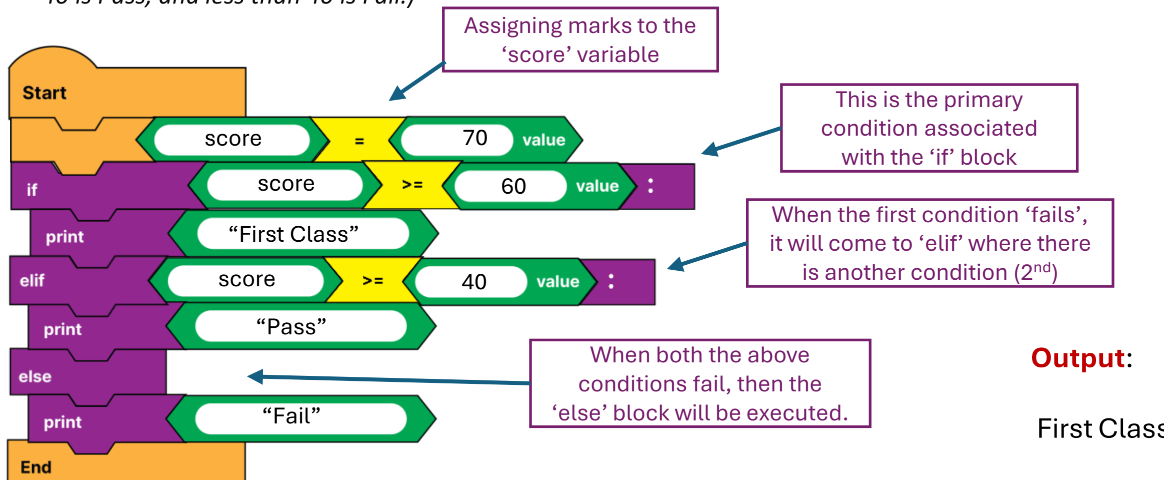
Syntax:
if condition1:
  # Code to execute if condition1 is True
elif condition2:
  # Code to execute if condition2 is True
elif condition3:
  # Code to execute if condition3 is True
  ...
  ...
else:
  # Code to execute if none of the conditions are True (optional)

We have worked with 'if' and 'else' to understand that if the condition is TRUE, the statements under the 'if' will be executed, and if the condition is FALSE, the statements under 'else' will be executed. However, in real life, the complexity increases, and you may need to check another condition when the first or primary condition results in FALSE. Meaning there is if added to the else, hence, elif (else + if).

The elif statement in Python stands for "else if" and is used to check multiple conditions in a sequential manner. It follows an if block and precedes an optional else block, allowing programs to test additional conditions if the initial if condition is not met. The 'elif' statement ensures that only the first condition that evaluates to True is executed, skipping all remaining conditions. This makes it a powerful tool for handling complex decision-making scenarios without nesting multiple if statements, which can make code less readable.

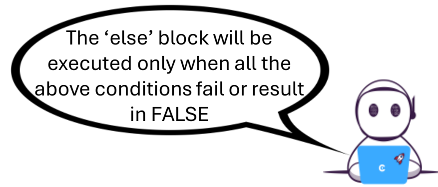
PRACTICE 1

Problem Statement: *Build a Python Block Code to declare whether an individual achieved first class, passed or failed in an exam. (Hint: Greater than or equal to 60 is First Class, Greater than or equal to 40 is Pass, and less than 40 is Fail.)*



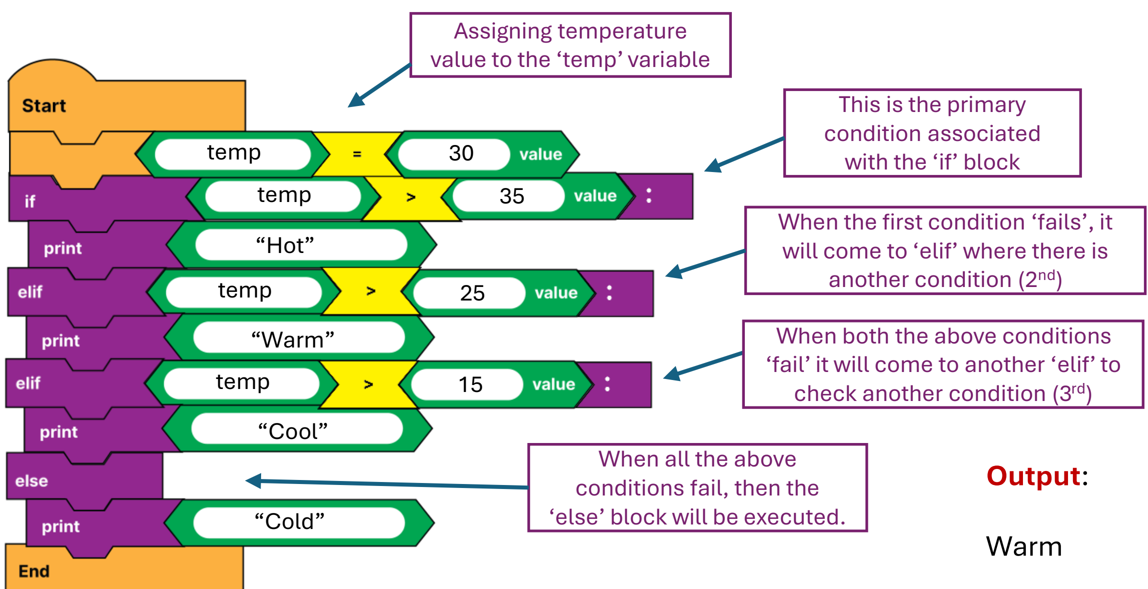
CEDURA © Copyrighted Content. All rights reserved

Code Explanation: In this program, we are using ‘elif’ in place of ‘else’ because we have an additional condition to check when the first condition fails (score >= 60). The ‘elif’ block is a combination of ’else’ and ‘if’. That means when the first condition fails, technically, the control will go to the ‘else’, but here we have ‘elif’ (an else with if condition score >=40). Here, the 2nd condition will be checked. If the 2nd condition results in TRUE then the statement under it will be executed (‘pass’). In case this condition too fails or results in FALSE, then the final ‘else’ will be executed (‘fail’).



PRACTICE 2

Problem Statement: Build a Python Block Code program using ‘elif’ to achieve the mentioned output. If the temperature is greater than 35 - Hot; if the temperature is greater than 25 - Warm; if the temperature is greater than 15 - Cool; otherwise it is Cold.

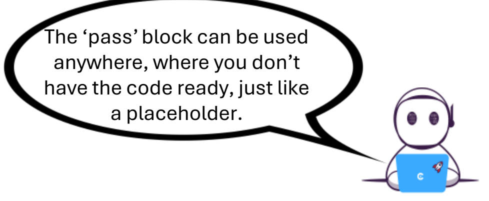


Code Explanation: This program is a good example of the usage of multiple ‘elif’ statements. When we have multiple conditions that need to be checked, one after another, to arrive at a conclusion, we can use the ’elif’ block multiple times. In this example we are checked 3 conditions - the primary one that goes with the ‘if’ block, the second one that goes with the ‘elif’ block and the final one that goes with the final ‘elif’ block. We have an ‘else’ block at the end which will only be executed when all the above conditions results in FALSE.

Even though optional, it is recommended to use an ‘else’ block at the end for some action when every condition fails

CEDURA © Copyrighted Content. All rights reserved

Pass Block



Syntax:
if condition1:
  pass

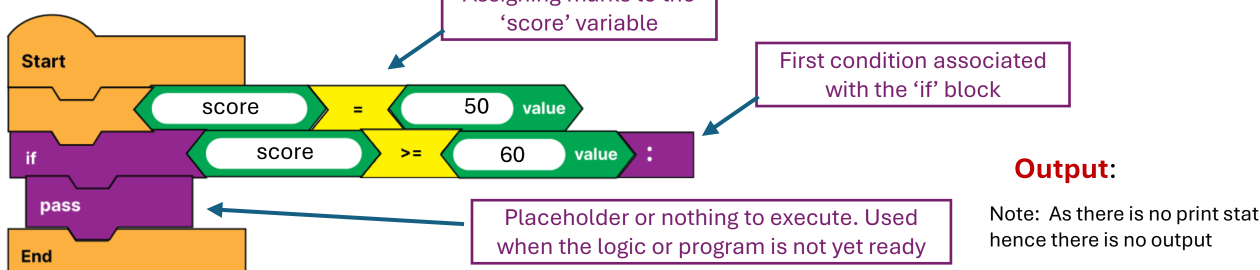
***for** var in **sequence**:*
  pass

In Python, the **pass** statement is a placeholder that does nothing when executed. It is often used in situations where code is syntactically required but no operation is desired, allowing the program to continue running without errors. The **pass** statement ensures that the program compiles and runs without any logical action taking place, providing a clean and error-free way to outline the structure of code before completing it

For example, the pass statement is useful in creating empty code blocks, such as within if, functions, classes, or loops, during the development phase when the actual implementation is not yet ready. It essentially acts as a "do nothing" operation, enabling the developer to define structures like an empty function or class without causing an IndentationError or SyntaxError.

PRACTICE 1

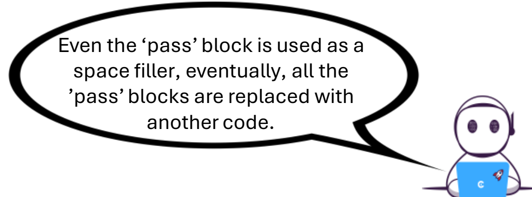
Problem Statement: Demonstrate usage of the 'pass' statement in an 'if' conditional program.



Output:

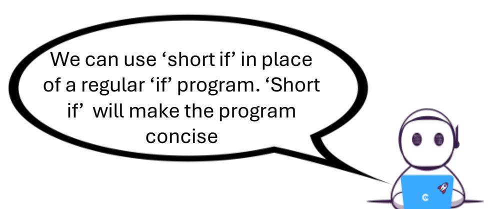
Note: As there is no print statement hence there is no output

Code Explanation: *In this program, we are using the 'pass' block under the 'if' condition. It means that even if the condition results as TRUE and the control goes under the 'if' block, it will encounter the 'pass' block which does nothing. It is just a space filler. Hence, there will no output from this program.*



CEDURA © Copyrighted Content. All rights reserved

Short if-else Statement



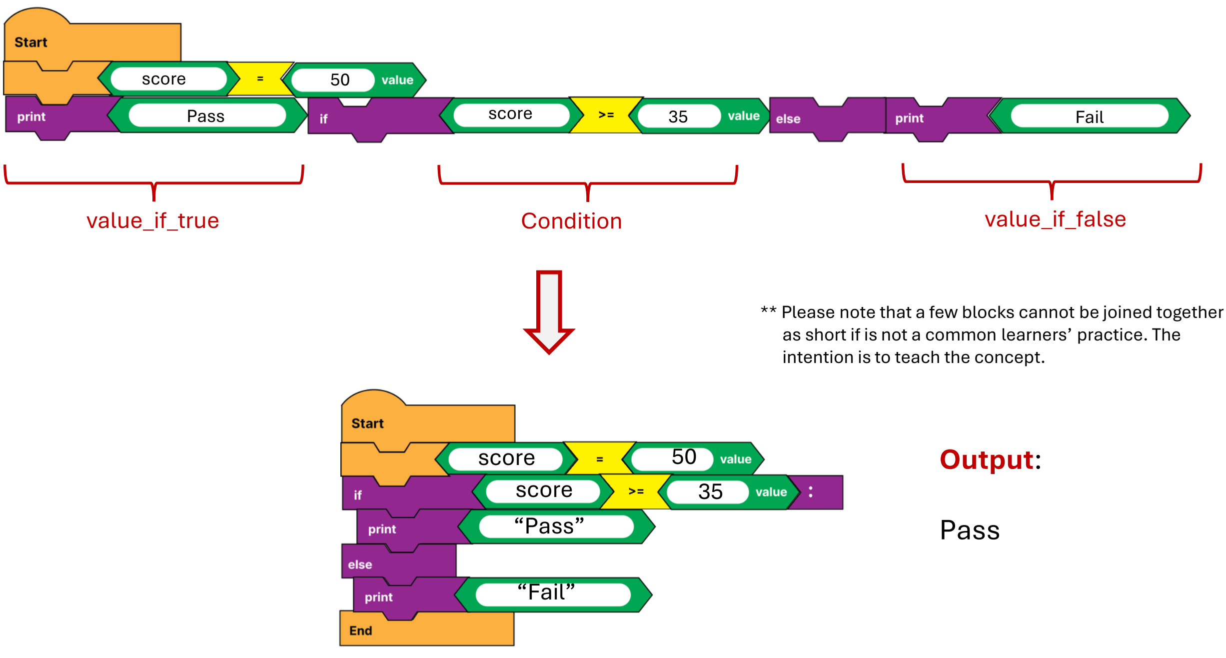
Syntax:

value_if_true ***if*** *condition* ***else*** *value_if_false*

In Python, the short if statement, also known as the *ternary conditional operator*, allows you to write concise conditional expressions in a single line. Its syntax is **value_if_true** if condition else **value_if_false**, which is both readable and compact compared to a full if-else block. This makes it ideal for simple decisions where one value is assigned based on a condition. However, it is recommended to avoid using it for complex logic, as it can become harder to read.

PRACTICE 1

Problem Statement: Demonstrate the short if statement



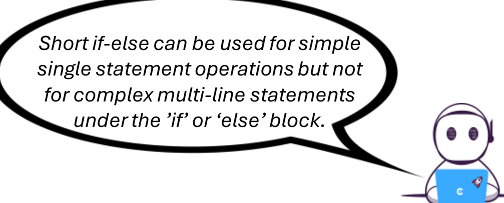
Equivalent if-else long code

** Please note that a few blocks cannot be joined together as short if is not a common learners' practice. The intention is to teach the concept.

Output:

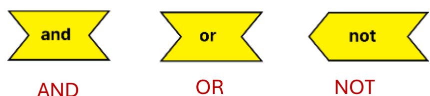
Pass

Code Explanation: *In this program, we have used the short if-else which would minimize the code and allow us to complete the code in a single line. Below that is the alternate regular or long code using the if-else condition.*

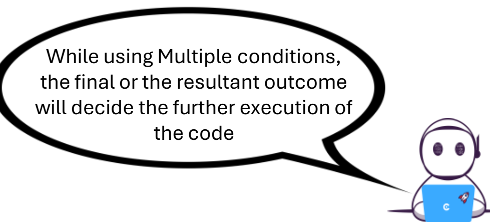


CEDURA © Copyrighted Content. All rights reserved

Multiple Conditions



AND OR NOT



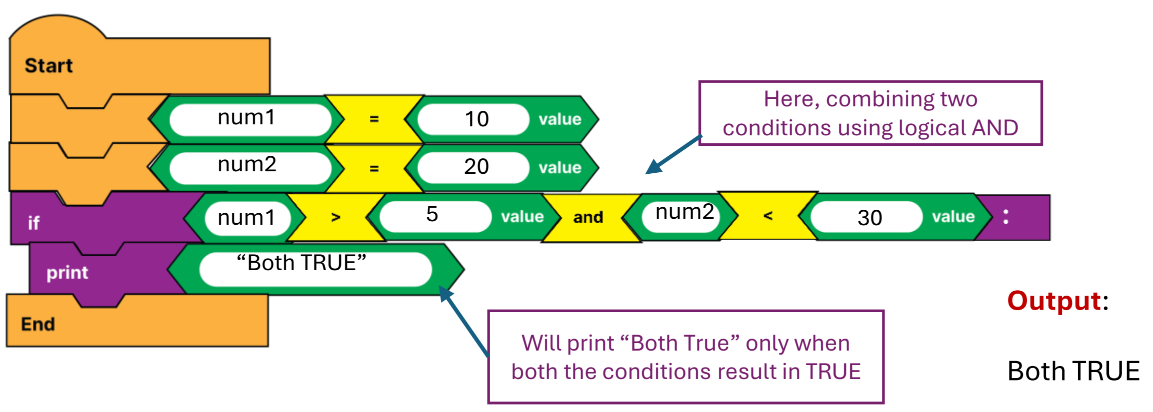
Syntax:
*if **condition1 and condition2**:*
  # Code block to execute if both conditions are True

In Python, handling multiple conditions is achieved using logical operators such as **and, or, and not**, which allow you to combine or modify conditions within an if statement. The 'and' operator ensures that all conditions must be true for the code block to execute, while the or operator executes the code if at least one condition is true. The not operator reverses the truth value of a condition, making it useful for negating expressions.

For example, if x > 5 and y < 10: check if both conditions are satisfied before proceeding. When working with complex conditions, parentheses can be used to group expressions and clarify their evaluation order, as not has the highest precedence, followed by and, and then or. This flexibility in combining conditions allows developers to implement intricate decision-making logic in a concise and readable manner.

PRACTICE 1

Problem Statement: Demonstrate the multiple condition with logical AND.



Output:

Both TRUE

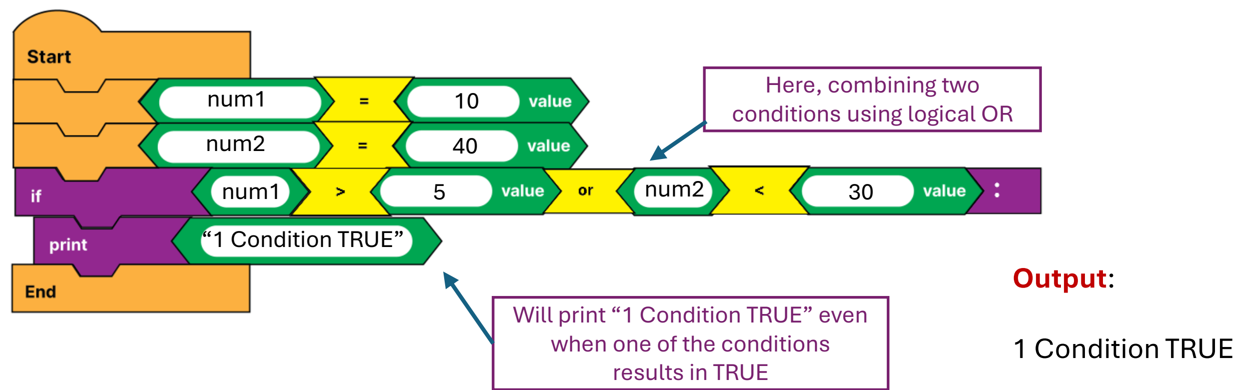
Condition1	Condition 2	Condition1 **AND** Condition2
TRUE	TRUE	TRUE
FALSE	TRUE	FALSE
TRUE	TRUE	FALSE
FALSE	FALSE	FALSE

Code Explanation: *In this program, while using logical AND, the **resultant outcome** of the multiple conditions will be TRUE only when both conditions result in TRUE. If you observe the above truth table, you will understand the **resultant outcome** for various outcomes of the first and second conditions. In this case, the print will work only when the num1 is greater than 5 and num2 is less than 30. Here, both the conditions satisfy hence the output will print "Both TRUE".*

CEDURA © Copyrighted Content. All rights reserved

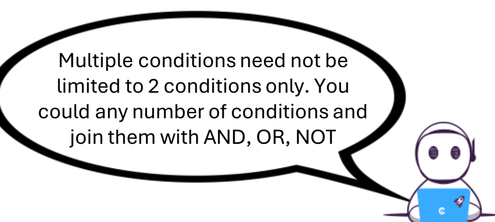
PRACTICE 2

Problem Statement: Demonstrate the multiple condition with logical OR.



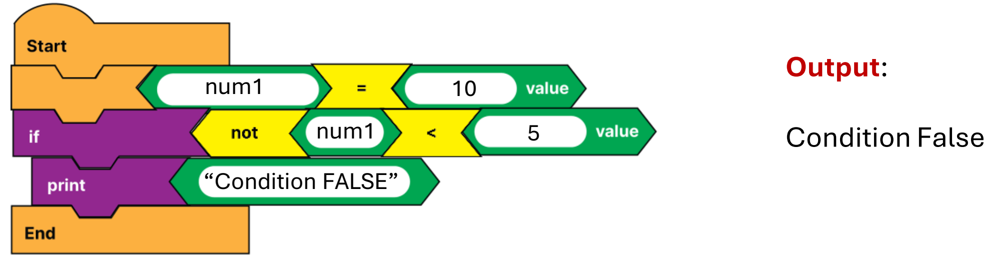
Condition1	Condition 2	Condition1 **OR** Condition2
TRUE	TRUE	TRUE
FALSE	TRUE	TRUE
TRUE	TRUE	TRUE
FALSE	FALSE	FALSE

Code Explanation: *In this program, while using logical OR, the **resultant outcome** of the multiple conditions will be TRUE even if one of both conditions results in TRUE. If you observe the above truth table, you will understand the **resultant outcome** for various outcomes of the first and second conditions. In this case, the print will work when either the num1 is greater than 5 or num2 is less than 30. Here, as at least one condition is satisfied (i.e. num1 is greater than 5) hence, the output will print "1 Condition TRUE".*



PRACTICE 3

Problem Statement: Demonstrate the usage of logical NOT.

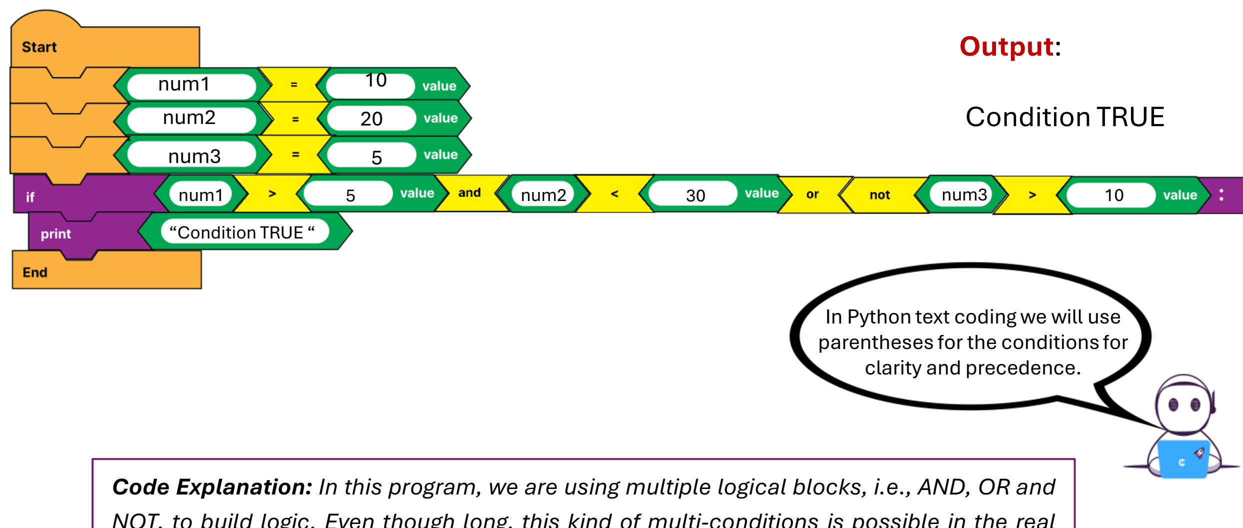


Code Explanation: *In this program, we are using NOT that negative a condition. Here, the code block under the 'if' condition will work then the condition results in FALSE. This is rarely used, unless sure, as it confuses most of the users.*

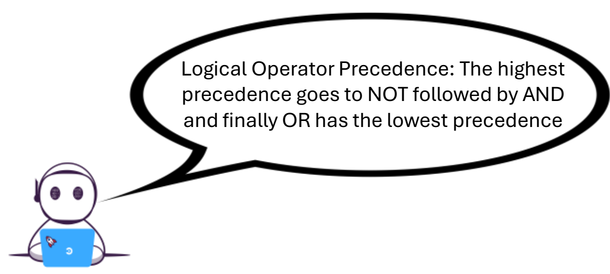
CEDURA © Copyrighted Content. All rights reserved

PRACTICE 4

Problem Statement: Demonstrate the combining multiple conditions using logical and, or and not.



> ***Code Explanation:*** *In this program, we are using multiple logical blocks, i.e., AND, OR and NOT, to build logic. Even though long, this kind of multi-conditions is possible in the real world. Here we have 3 conditions - the first two conditions are joined by logical AND and the last one (Negate condition) is joined to the first two using an OR logical block. In the Python text coding we can represent this condition as:*
>
> ***if*** *(x > 5* ***and*** *y < 30)* ***or not*** *(z > 10):*



CEDURA © Copyrighted Content. All rights reserved

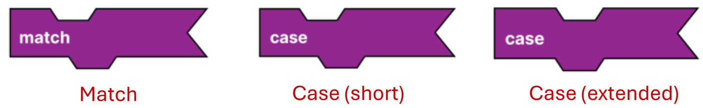
TOPIC ASSIGNMENT

1. Build a Python Block Code to check whether a given number is positive or not.
2. Build a Python Block Code to seek a number from the user and check whether a given number is even or odd.
3. Build a Python Block Code to seek the Gender of the student ("Male" or "Female") and print 1 if the gender is male and 0 if the gender is female.
4. Build a Python Block Code to seek two numbers from the user and display the bigger number.
5. Build a Python Block Code to seek two numbers from the user and display the smaller number.
6. Build a Python Block Code to seek the marks and age of a student and check the given eligibility condition: age should be less than 25 and marks should be greater than 65. If the student meets both conditions, print 'Eligible'.
7. Build a Python Block Code to check whether a person is eligible to drive or not. (**Condition**: Age of 18 and above are eligible to drive)
8. Build a Python Block Code to check whether a given number is between 1 and 9
9. Build a Python Block Code to seek a password from the user and authenticate it. (Tip: You decide on the password and check it against the user input)
10. Build a Python Block Code to check the age and display "Child", "Teenager" or "Youth". (Tip: age 1 to 12 - Child; 13 to 19 - Teenager; 20 and above - Youth)

CEDURA © Copyrighted Content. All rights reserved

Match-Case Blocks

Introduced in Python 3.10, the **match-case** statement is a powerful feature that allows for pattern matching, providing a cleaner and more intuitive alternative to complex 'if-elif' chains. It works by comparing a given value against one or more patterns, executing the corresponding code block when a match is found. In the kit, you will find 3 blocks related to match-case, one being the 'match' block and 2 'case' blocks. The 'case' block comes in two sizes to match the position. Use them as per the position of the case.



Match Case (short) Case (extended)

The syntax begins with the match keyword, followed by an expression to be compared. Within the case blocks, specific patterns can be defined, such as literals, variable bindings, wildcards, or even more complex structures like sequences and dictionaries. For example, in a match day statement, the case "Monday": block can execute code specific to Monday, while the case _: (underscore) acts as a wildcard for unmatched cases, similar to a default case in other languages.

Syntax:

```
match expression:
  case pattern1:
    # Code to execute if pattern1 matches
  case pattern2:
    # Code to execute if pattern2 matches
  case pattern3 if condition:
    # Code to execute if pattern3 matches and the condition is True
  case _:
    # Code to execute if no patterns match (default case)
```

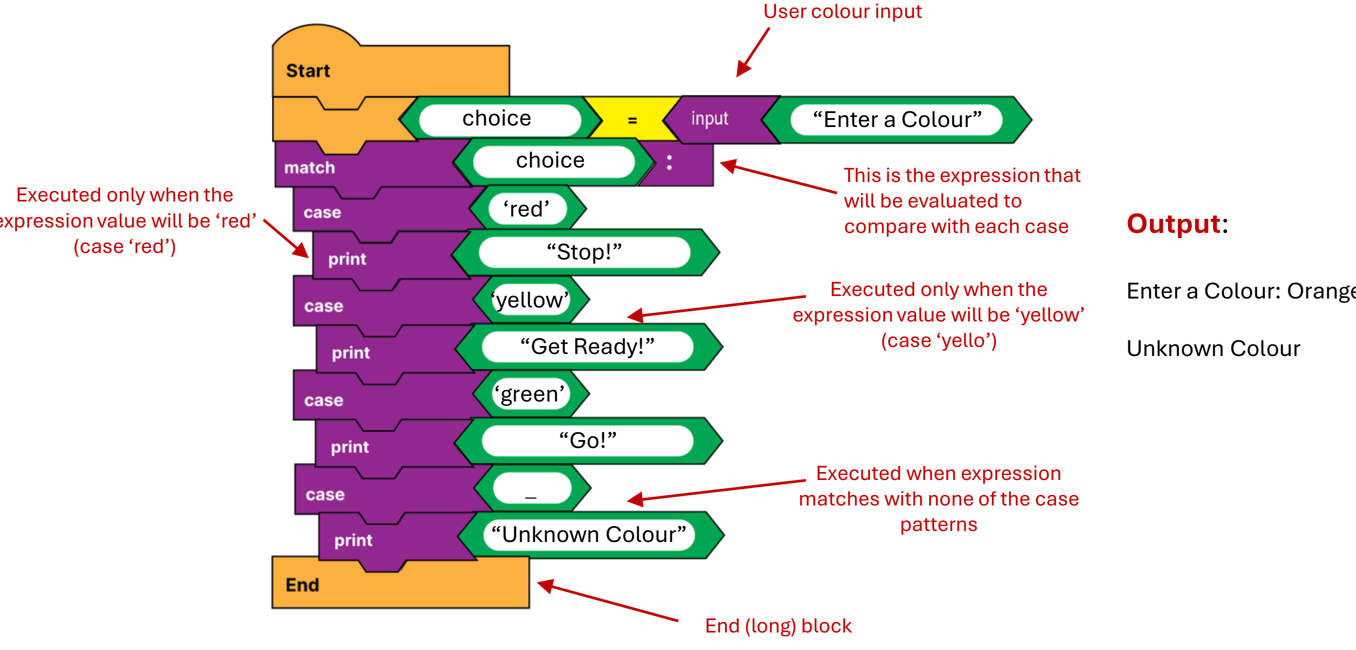
Components:

1. ***match** expression:*
    - *The expression is evaluated and compared against each case pattern.*
2. ***case** pattern:*
    - *Patterns can be values, variables, sequences, or more complex structures.*
    - *Use | to specify multiple patterns (e.g., case "A" | "B":).*
3. ***case _:***
    - *A wildcard case that matches anything. It works like a default case.*
4. ***if** condition: **(optional)***
    - *Guards add an extra condition to a case for more refined matching.*

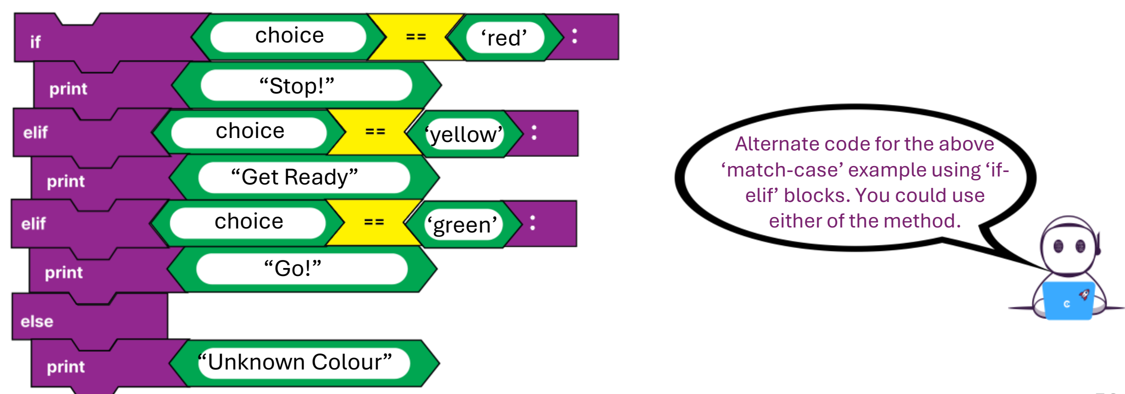
CEDURA © Copyrighted Content. All rights reserved

PRACTICE 1

Problem Statement: *Build a Python Block Code to display traffic signal based on user input*



Code Explanation: *In this program, we are using a 'match-case' where there will be an expression ('choice') that would be compared with all the patterns of the case blocks. When the user inputs colour, we are storing in the 'choice' variable. This variable is used as an expression in the 'match' block. When the input is 'red' the first case matches hence the print function under it will be executed (prints "Stop!"). When the input is 'yellow', it matches with the second case and the print function under it will be executed (prints "Get Ready"). When the user input matches with none of the patterns (colours) mentioned in the case blocks, the default case i.e. the one with the '_' pattern will be executed (prints "Unknown Colour"). The same come can be written using 'If-elif' blocks. However, the match-case method will be simpler for such applications.*



CEDURA © Copyrighted Content. All rights reserved

TOPIC ASSIGNMENT

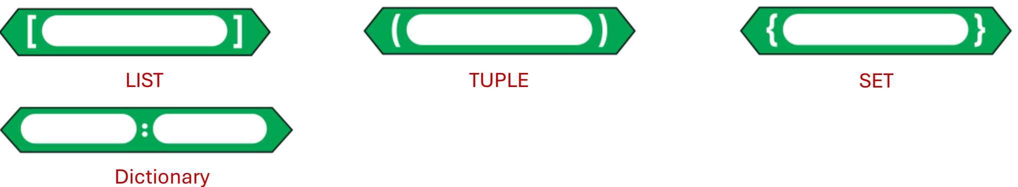
1. Build a Python Block Code (using Match-Case) to seek input from a user (0 or 1) and print the text (‘one’ or ‘two’) form of the input number. In case a user enter any other number, then print ‘invalid input’
2. Build a Python Block Code (using Match-Case) to take input from the user (0 or 1) and perform the addition of two variables if the input is 0 and subtraction if the input is 1.
3. Build a Python Block Code (using Match-Case) to take input from the user (Big or Small) and compare two variables, var1 (6) and var2 (5). If the input is Big, check whether variable var1 is bigger than var2. If the input is small, check whether variable var1 is smaller than var2.

CEDURA © Copyrighted Content. All rights reserved

Data Structures in Python

Data structures in Python are ways of organizing and storing data so that they can be accessed and manipulated efficiently. hey provide a way to manage and work with data based on specific needs and operations, such as searching, sorting, or accessing elements. They are essential for handling complex data efficiently and are a core concept in programming. Python offers a rich set of built-in data structures, which can be broadly categorized into the following:

- **Built-in Data Structures**
  These are the basic structures that come pre-defined in Python. List, Tuple, Set, Dictionary and Strings belong to this group.



- **User-defined Data Structures**
  Python allows the creation of custom data structures to solve specific problems. In this, we have Stack, Queue, Linked List, Tree, Graph and Hash Table

Choosing the right Data Structure depends on the Data Type (Homogeneous or Heterogeneous), Operations (Searching, Sorting, Indexing, etc) and Performance requirements (Time and Space complexity).

Essential Concepts

Heterogeneous Data: This is the property to have values of different data types. Sequences help to ensemble values of different data types together.

var = [1, 4,7,9] → **Homogeneous** or Same data type values

var = [1, ‘py’, 7.8] → **Heterogeneous** or Different data type values

Ordered: It is all about maintaining the sequence or a defined order which the items maintain while creating a sequence. Few Sequences maintain the order and others are unordered, meaning they will jumble the position of the values, once a sequence is created.

Iteration: It is the ability to go through all the items in a sequence. All sequences allow iterating through their items.

Duplicates: Few sequences allow duplicate values and few do not. This is a feature we need to check before using a sequence.

Indexing: Every item in a sequence in given a unique integer number to identify the item’s position in the sequence. This is used to retrieve that particular item. The indexing always starts from 0.

Slicing: This is an extension of Indexing. Here we can select more than one item. Hence, we will provide the ‘**start**’ and ‘**upto**’ value. E.g. var =“Python”. In case you want ‘th’ characters we will use slicing var[2:4].

CEDURA © Copyrighted Content. All rights reserved

SEQUENCES:
Sequences are ordered collections of items that allow access to their elements through indexing. Common sequence types include strings, lists, tuples, and ranges, each serving different purposes. Python's sequence types simplify handling collections of data with their intuitive syntax and built-in functionality, forming the backbone of many applications.

- The sequence is an ordered set
- Used to ensemble a group of items
- Every item or element in the sequence is numbered which is called indexing
- We could refer to the elements or items in the sequence using their index number

STRINGS

Syntax:
var_name = *"PYTHON"*

var_name = "PYTHON"

In Python, a **string** is a sequence of characters enclosed within single quotes ('), double quotes ("), or triple quotes (''' or """). Each character is treated as an element in the string and can be indexed. Strings are one of the most widely used data types and are immutable, meaning their content cannot be changed after they are created. Strings can store letters, numbers, symbols, or even spaces and are used extensively for handling and manipulating textual data. Python provides a variety of string operations and methods, making it easy to process and transform text effectively.

Python offers powerful tools for working with strings, such as indexing, slicing, and built-in methods. Strings support indexing, where the first character is at index 0, and slicing, allows extraction of specific portions of the string. For example, var = "Python", and var[1:4] extracts the substring 'yth'. Additionally, Python provides methods like .lower(), .upper(), .strip(), .replace(), and .split() for transforming and analysing strings.

For this course we are not using any methods but understand how to index and slice a string.

String Methods

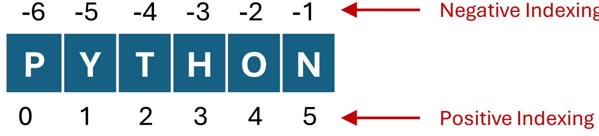
- capitalize()
- upper()
- islower()
- isupper()
- count()
- len()
- index()
- find()
- lstrip()
- split()
- replace()
- concatenate()
- ....

Indexing - Numbering the elements in the sequence

Index Position - Used to refer to a particular character or a string

Positive Indexing & Negative Indexing:

var_name = "PYTHON"

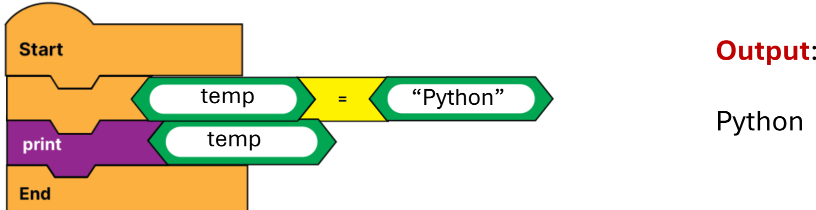


CEDURA © Copyrighted Content. All rights reserved

Positive Indexing	**Negative** Indexing
var_name[2] → T	var_name[-3] → T
var_name[0] → P	var_name[-2] → P

PRACTICE 1

Problem Statement: *Build a Python Block Code to assign a string and print it*



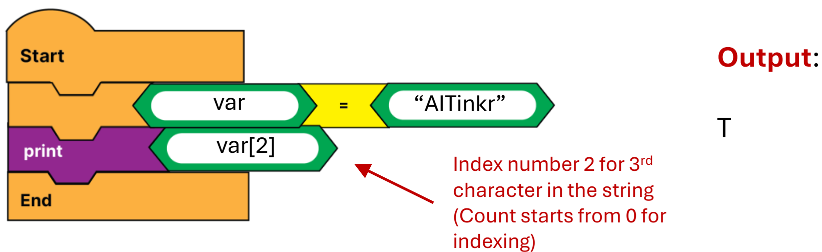
Output:

Python

Code Explanation: *In this program, we assign a string "Python" to a variable temp. Please note that we always encapsulate a string within single(') or double (") or triple ('") quotes. As the assignment is completed, we can print the content of the variable temp using a print function.*

PRACTICE 2

Problem Statement: *Build a Python Block Code to assign a string 'AITinkr' to a variable and index the 3rd character (index number 2).*



Index number 2 for 3rd character in the string (Count starts from 0 for indexing)

Output:

T

Code Explanation: *In this program, we are assigning a string "AITinkr" to a variable 'var'. As we are interested in the 3rd character, the index number would be 2. In this, the output will be character 'T'. Please note that indexing always starts with 0, unlike we humans count from 1. Hence, be careful while indexing. Each character in the string is considered as an individual element.*

CEDURA © Copyrighted Content. All rights reserved

SLICING:

Slicing in Python is a powerful technique used to extract a portion of a sequence, such as a string, list, tuple, or range. It allows you to access specific elements or a range of elements using the syntax sequence [start:stop:step]. The start index specifies where the slice begins (inclusive), the stop determines where it ends (exclusive), and the step defines the interval between elements. Slicing provides flexibility in manipulating sequences, enabling tasks like reversing sequences, extracting subsets, or skipping elements. If any parameter is omitted, Python uses default values: start=0, stop=len(sequence), and step=1. Importantly, slicing does not modify the original sequence but instead creates a new one, making it a safe and efficient way to work with data.

Syntax:

var_name [start : end : step]

Up to position (Not included)

Positive Slicing

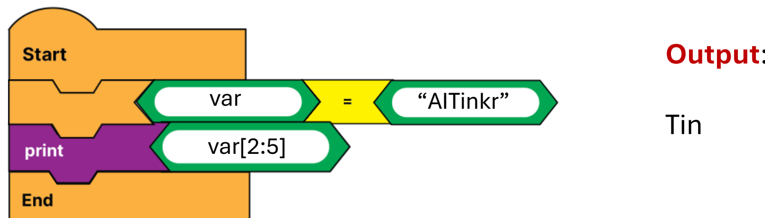
var_name[1:3] → HE
var_name[1:2] → H
var_name[:3] → AHE
var_name[1:] → HEAD
var_name[:] → AHEAD

Negative Slicing

var_name[-4:-2] → HE
var_name[-4:-3] → H
var_name[:-2] → AHE
var_name[-4:] → HEAD
var_name[:] → AHEAD

PRACTICE 3

Problem Statement: *Build a Python Block Code to assign a string 'AITinkr' to a variable and slice characters 'Tin' (index numbers 2, 3, and 4)*



Output:

Tin

Code Explanation: *In this program, we are assigning a string "AITinkr" to a variable 'var'. As we are interested in characters 'Tin' we must slice them indicating the starting (2) and upto value (5). Please note that it is an 'upto' value hence, we will always consider one value higher than the last item index number (in our case 5). This way we will get the output as 'Tin' as desired.*

CEDURA © Copyrighted Content. All rights reserved

List Block

Syntax:

list_name = [element1, element2, element3, ...]

OR

list_name = list(iterable)   # constructor method

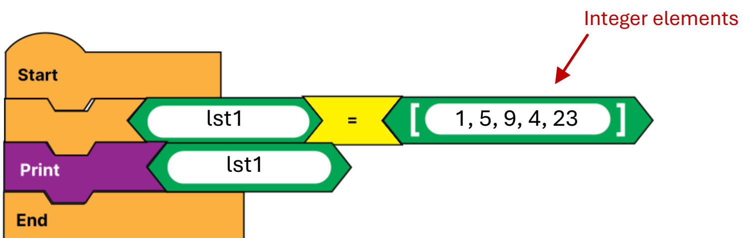
A **list** in Python is a built-in data structure that represents an ordered, mutable collection of elements. Lists can store items of various data types, such as integers, floats, strings, or even other lists, making them highly versatile for handling diverse datasets. They are defined using square brackets [], with elements separated by commas. Lists are indexed, meaning you can access individual elements using their position (starting from 0).

Properties:

1. *Ordered*
2. *Mutable*
3. *Heterogeneous*
4. *Dynamic*
5. *Indexable*
6. *Allows Duplicates*
7. *Supports Nesting*
8. *Extensive Built-in Methods*
9. *Symbol - []*

PRACTICE 1

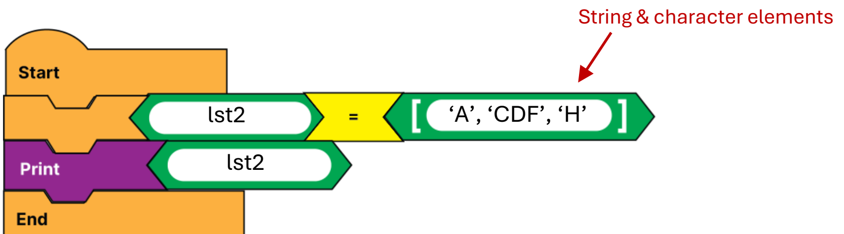
Problem Statement: *Write a Python Block Code to create a list of integers.*



> ***Code Explanation:*** *In this program, we have a series of integer elements which are used to create a list and assigned to the variable 'lst1'. A List is created by simply sequencing the items and assigning them to an object. As we are using square brackets ([ ]), Python will create a list with the given elements.*

PRACTICE 2

Problem Statement: *Write a Python Block Code to create a list of Strings.*

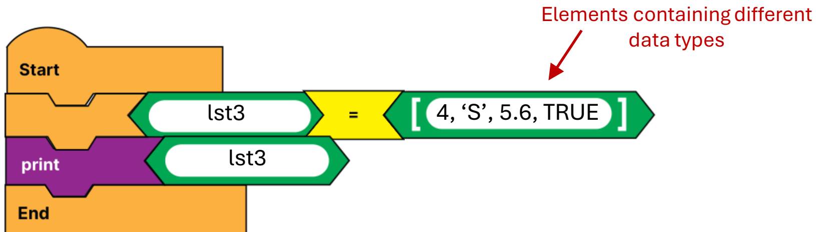


CEDURA © Copyrighted Content. All rights reserved

Code Explanation: *In this program, we have a series of string elements which are used to create a list and assigned to the variable 'lst2'. A list is created by simply sequencing the items and assigning them to an object. As we are using square brackets ([ ]) , Python will create a list with the given elements.*

PRACTICE 3

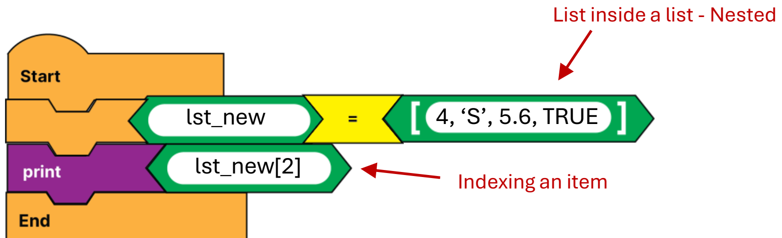
Problem Statement: *Build a Python Block Code to create a list of mixes data types (Heterogeneous).*



Code Explanation: *In this program, we have a series of mixed data type elements (Integer, String, Float, and Boolean) that are used to create a list and assigned to the variable 'lst3'. A list is created by simply sequencing the items and assigning them to an object. As we are using square brackets ([ ]) , Python will create a list with the given elements.*

PRACTICE 4

Problem Statement: *Build a Python Block Code to create a list and print the third item in it.*

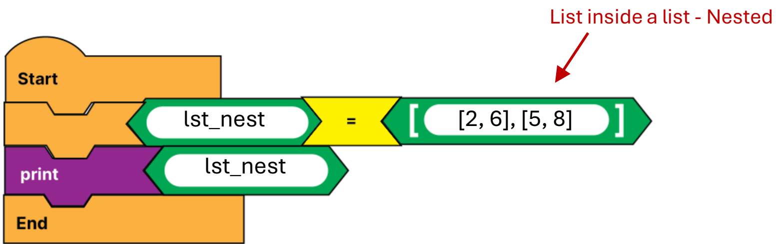


Code Explanation: *In this program, we have created a list with items. As per the problem statement we are expected to print the 3rd item i.e. 'TRUE'. We have learning that items in a sequence are indexed with a unique integer value starting with zero. Hence the index number of the item 'TRUE' will be '2'. To access the item with index number 2 we will use the square brackets and indicate the desired index number. When we code lst_new[2], it will print the item 'TRUE' as its index position is 2.*

CEDURA © Copyrighted Content. All rights reserved

PRACTICE 5

Problem Statement: *Build a Python Block Code to create a nested list.*



> ***Code Explanation:*** *In this program, we have two lists nested inside the main list or outer list. We have assigned this to an object 'lst_nest'. This is how we create nested lists. Each list inside the outer list is indexed so that we can access them individually. Suppose, you are interested in list item [5,8] you will index it directly as we do for any item in a sequence (lst_nest[1]). Nested lists are widely used when each element is made of another list.*

Tuple Block

Syntax:

tuple_name = *(element1, element2, element3, ...)*

OR

tuple_name = *tuple(iterable)* *# constructor method*

A **tuple** in Python is a built-in data structure that represents an ordered, immutable collection of elements. Similar to lists, tuples can store elements of different data types, such as integers, strings, floats, or even other tuples. They are defined using parentheses () with elements separated by commas. Since tuples are immutable, their elements cannot be modified, added, or removed after they are created, making them ideal for storing fixed collections of data.

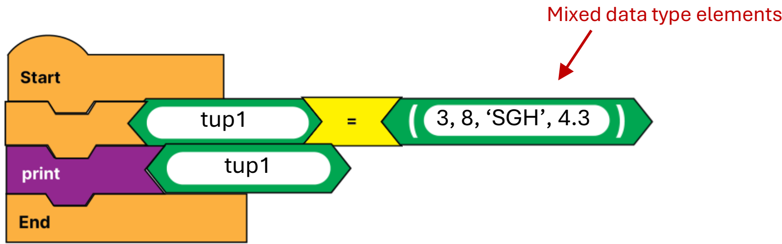
Properties:

1. *Ordered*
2. *Immutable*
3. *Heterogeneous*
4. *Iterable*
5. *Indexable*
6. *Allows Duplicates*
7. *Supports Nesting*
8. *Hashable*
9. *Symbol - ( )*

CEDURA © Copyrighted Content. All rights reserved

PRACTICE 1

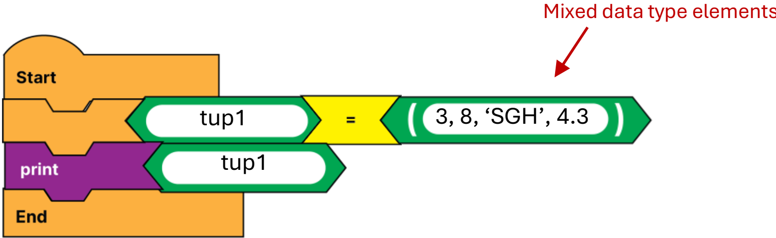
Problem Statement: *Build a Python Block Code to create a tuple with multiple elements*



> ***Code Explanation:*** *In this program, we have elements of different data types used to build a tuple, tup1. We used mixed data type elements (Integer, String, Float, and Boolean). A tuple is created by simply sequencing the items and assigning them to an object. As we are using parenthesis ('( )'), Python will create a tuple with the given elements.*

PRACTICE 2

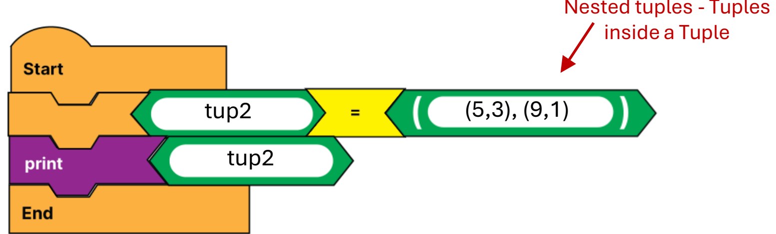
Problem Statement: *Build a Python Block Code to create a tuple and index 2nd item (index number 1)*



> ***Code Explanation:*** *In this program, we have elements of different data types used to build a tuple, tup1. To index 2nd item (index number 1), we have to use the code tup1[1], which will index the item '8' and print it. This is the same as an indexing list or any other sequence. Always use square brackets to indicate the index number of the item.*

PRACTICE 3

Problem Statement: *Build a Python Block Code to create a nested tuple*



> ***Code Explanation:*** *In this program, we have two tuples nested inside the main tuple or outer tuple. This is how we create a nested tuple. Each tuple inside the outer tuple is indexed so that we can access them individually. Nested tuples are widely used when each element is made of another tuple.*

CEDURA © Copyrighted Content. All rights reserved

Set Block

Syntax:

set_name = {element1, element2, element3, ...}

OR

set_name = set(iterable)   # constructor method

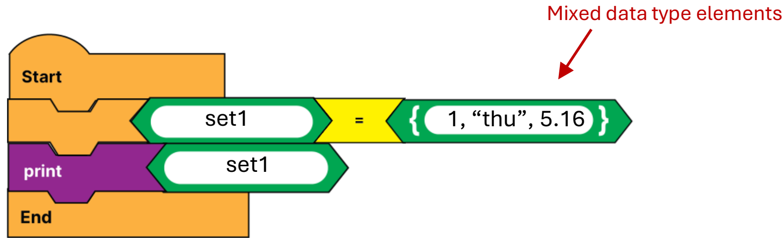
A **set** in Python is a built-in data structure that represents an unordered collection of unique elements. Unlike lists and tuples, sets do not allow duplicate elements and are unindexed, meaning the order of elements is not guaranteed. Sets are mutable, so you can add or remove elements, but they can only contain immutable (hashable) objects like numbers, strings, or tuples.

Properties:

1. *Unordered*
2. *Mutable*
3. *Unindexed*
4. *Iterable*
5. *Supports Set Operations*
6. *No Duplicates*
7. *Dynamic Size*
8. *Unique Elements*
9. *Symbol - { }*

PRACTICE 1

Problem Statement: *Build a Python Block Code to create a set with multiple elements*



Code Explanation: *In this program, we have elements of different data types used to build a set, set1. We used mixed data type elements (Integer, String, Float, and Boolean). A set is created by simply sequencing the items and assigning them to an object. As we are using braces ('{ }'), Python will create a set with the given elements.*

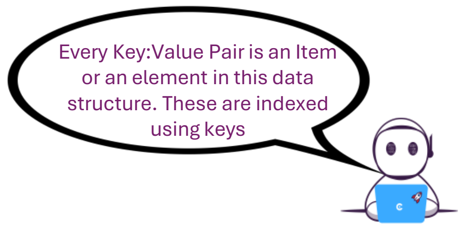
Learning TIP: All the sequences behave as per their properties i.e. ordered/unordered, mutable/immutable, duplicate items, etc. Hence few methods apply to a particular data structure, whereas the same may not apply to another type of data structure. Recommend you learn all the methods of a LIST and then differentiate which methods could be applied or not applied for others

CEDURA © Copyrighted Content. All rights reserved

Dictionary Block

Syntax:

dictionary_name = {
key1: value1,
key2: value2,
key3: value3,
… }



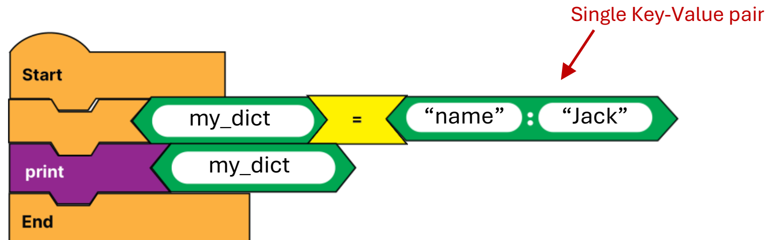
A **dictionary** in Python is a built-in data structure that stores data in the form of **key-value pairs**. It is a versatile and efficient way to organise and retrieve data based on unique keys rather than positional indices, as with lists or tuples. It is a collection of key-value pairs where the value can be any python object, and the keys are used to index values and keys can be any immutable data type; strings and numbers. Dictionaries are defined using curly braces {} and allow for quick lookups, additions, and updates.

Properties:

1. *Unique Key-value Pair Structure*
2. *Unique Keys*
3. *Immutable Keys*
4. *Heterogeneous Values*
5. *Ordered*
6. *Mutable*
7. *No Duplicate Keys*
8. *Dynamic Size*
9. *Supports Nesting*
10. *Symbol - { }*

PRACTICE 1

Problem Statement: *Build a Python Block Code to create a tuple with multiple elements*

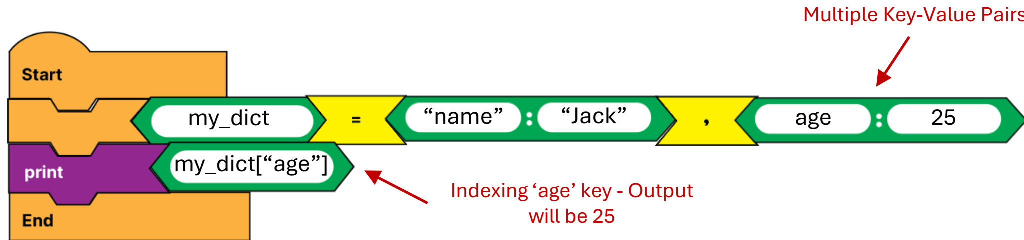


Code Explanation: *In this program, we are creating a single key:value pair dictionary. Note that every element in the dictionary is a key:value pair. We can index the value using the key of the element. We could use multiple key:value pairs by separating them with comma (,). Dictionaries are widely used data structure in python.*

CEDURA © Copyrighted Content. All rights reserved

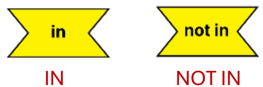
PRACTICE 2

Problem Statement: *Build a Python Block Code to create a tuple with multiple elements*



Code Explanation: *In this program, we have created a dictionary with multiple key:value pairs (name and age). We used a comma separator to use multiple key:value pairs. After creating the dictionary, we are trying to print the age value by indexing its key ("age"). This is the way we can use the key to index in a dictionary. The dictionary provides efficiency for storing and retrieving data based on unique keys.*

Membership Operators



Syntax:

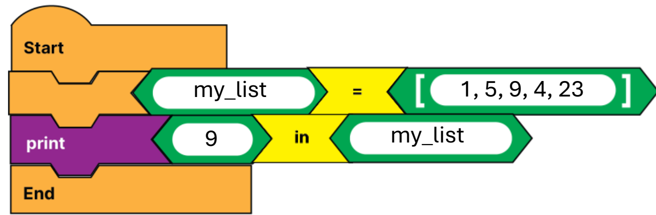
*value **in** sequence*

*value **not in** sequence*

Membership operators in Python are used to check whether a specific value or variable is present in a sequence (e.g., string, list, tuple, set, dictionary). The result of the operation is a Boolean value (True or False). Python provides two membership operators: in and not in. The in operator returns True if the value is present in the sequence, while not in returns True if the value is absent.

PRACTICE 1

Problem Statement: *Build a Python Block Code to check the existence of a value in a sequence.*

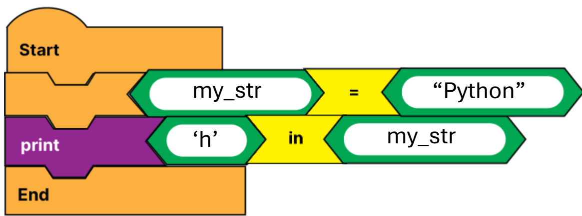


CEDURA © Copyrighted Content. All rights reserved

> ***Code Explanation:*** *In this program, we are creating a list and assigning it to an object 'my_list'. In the next statement, we are checking whether the value '9' is available in the above list or not using the membership operator 'in'. As the value '9' exists in the sequence the output will result in TRUE. The same method could be used for any sequence.*

PRACTICE 2

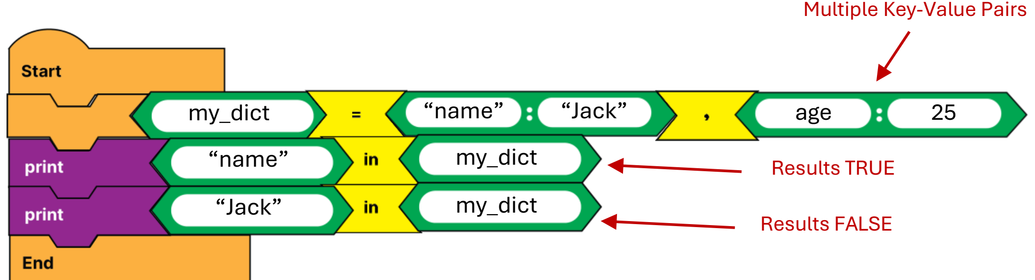
Problem Statement: *Build a Python Block Code to check whether a particular character exists in a string.*



> ***Code Explanation:*** *In this program, we are assigning a string "Python" to a variable 'my_str'. In the next statement, we are checking whether the character 'h' is available in the string or not using the 'in' membership operator. As the character 'h' is available in the string, the output would result in TRUE.*

PRACTICE 3

Problem Statement: *Build a Python Block Code to demonstrate usage of membership operator on a dictionary.*



> ***Code Explanation:*** *In this program, we have created a multiple key:value paired dictionary. In the next statement, we are checking whether the key "name" is available in the dictionary and it would result in TRUE. However, when we check whether the value "Jack" is available, it results in FALSE. We can check for key membership but not the value membership.*

Note: We can use the 'not in' operator the same way, however, it is the negation of the 'in' operator

CEDURA © Copyrighted Content. All rights reserved

TOPIC ASSIGNMENT

1. Build a Python Block Code to create a LIST of student marks and print them.
2. Build a Python Block Code to create a LIST and print its data type.
3. Build a Python Block Code to create a TUPLE of student ages and print them.
4. Build a Python Block Code to create a SET of Employee Details and print them.
5. Build a Python Block Code to create a SET with duplicate items in it. What will be the output when you print it?
6. Build a Python Block Code to create a DICTIONARY of Employee Details and print it.
7. Build a Python Block Code to create a DICTIONARY using keys (Name, age) and values (Akram, 10).
8. Build a Python Block Code to create a DICTIONARY of Employee Details and print any one value.
9. Build a Python Block Code to create a nested list.
10. Write a Python Block code to create a list based on the given values and print the 3rd Item in the list. (Hint: Consider 10, 20.3, True, "Hello", -56, 22.5 as values)
11. Write a Python Block code to create a list using the values 22, 45, 66, 77, 22, "78.67", and 22. After creating find out whether a value 77 is in the list or not.
12. Write a Python Block code to assign a string "Sequence" to a variable and print the 4th character and the characters "uen", individually.

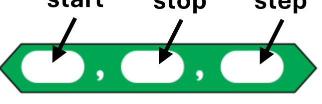
CEDURA © Copyrighted Content. All rights reserved

Range in Python

The range() function in Python is a versatile tool used to generate sequences of numbers, often in combination with loops. You can **generate a sequence of numbers** without depending on a List or Tuple. For example, you can generate 1,2,3,4,5 a sequence of numbers using range and convert them into a list to get a sequence, instantly.

Syntax:
range(start, stop, step)

Range



Range - Parameters

Parameters:

1. ***start (optional):*** *The starting value of the sequence. The default value is 0.*
2. ***stop:*** *The endpoint of the sequence. The 'up to' value. Must for a range.*
3. ***step (optional):*** *The difference between each number generated in the sequence. The default value is 1. Can be negative too to generate a sequence in reverse order.*

Example:

list(**range**(1,5,1)) → *Output: [1,2,3,4]* *'1' being start | '5' being stop (up to) | '1' being step*
list(**range**(1,5,2)) → *Output: [1,3]* *'1' being start | '5' being stop (up to) | '2' being step*
list(**range**(5,1)) → *Output: [0,1,2,3,4]* *No Start value hence default 0*
list(**range**(1, 5)) → *Output: [1,2,3,4]* *No Step value hence default 1*
list(**range**(5)) → *Output: [0,1,2,3,4]* *No Start or Step hence '0' and '1', respectively*

The basic syntax is range(start, stop, step), where start is the beginning of the sequence (default is 0), stop is the endpoint (exclusive), and step specifies the increment (default is 1). If two values are given in the parameters, it will consider them as start and stop values, and step as default 1. If only one value is provided as a parameter it will consider it as the stop value, and start as default '0', step as default '1'. If you want a decreasing sequence, you can use a negative step, such as range(5, 0, -1) which produces 5, 4, 3, 2, 1. This flexibility makes the range() function essential for repetitive tasks and algorithmic problem-solving in Python.

It is commonly employed in for loops to iterate over a specific range of values without manually specifying each number. One of the key benefits of the range() function is its memory efficiency. Instead of creating a list of numbers outright, range() generates numbers on demand, making it suitable for handling large sequences without consuming excessive memory. It works seamlessly with loops, enabling concise and readable code for iterating over numerical ranges.

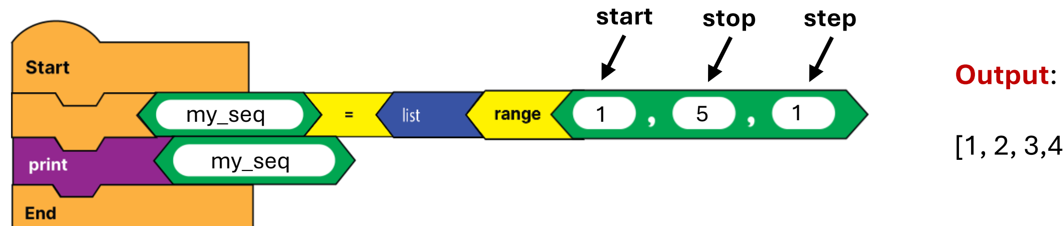
We use the list() function along with the range() function in Python to explicitly convert the range object into a list. The range() function returns a range **object**, which is an iterable that generates numbers on demand, but not an actual list. The list() function creates a list from this range object, allowing us to view or manipulate the generated sequence directly.

The 'range' can be used to generate numerical sequences only.

CEDURA © Copyrighted Content. All rights reserved

PRACTICE 1

Problem Statement: *Build a Python Block Code to demonstrate the usage of the range() function*



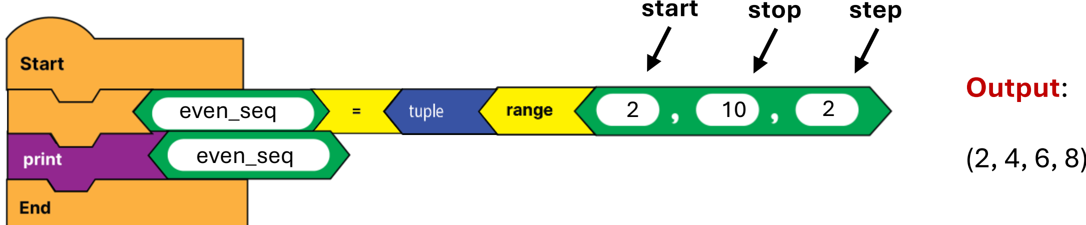
Output:

[1, 2, 3,4]

> ***Code Explanation:*** *In this program, we are generating a sequence of numbers with a 'start' value of 1, a 'stop' value of 5 and a 'step' value of 1. This will generate a range object which is not an actual list. Hence, we are using the 'list' function in front of it to create a list from this range object. This allows us to iterate and manipulate the generated sequence directly. We are storing the list in the variable 'my_seq'. In the next line, we are printing the list. The output will be [1,2,3,4]. This is how you can generate a sequence of numbers using the range function in python.*

PRACTICE 2

Problem Statement: *Build a Python Block Code to generate a sequence of even numbers from 1 to 10.*



Output:

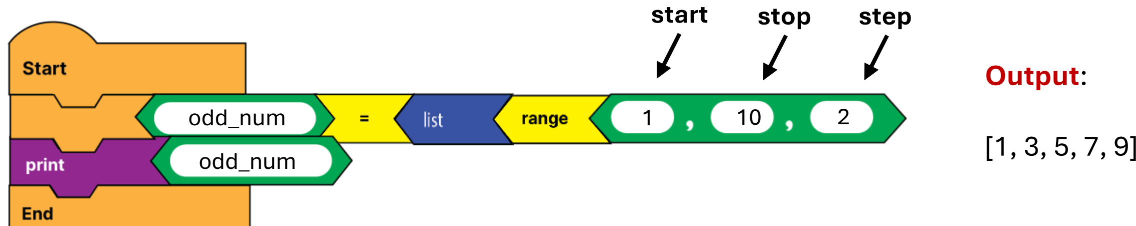
(2, 4, 6, 8)

> ***Code Explanation:*** *In this program, we are expected to generate even numbers from 1 to 10. The output would be 2,4,6,8. Here the starting value is 2 (which is the smallest even number possible) and the ending value is 8 (which is the largest even number possible before 10) and the difference between the numbers is 2. Hence, we will consider 2 as the 'start' parameter, 10 as the 'stop' parameter and finally 2 as the 'step' as we want every alternate number. After that, we use the tuple() function to convert the range object into a tuple. The same method could be used to generate odd numbers too.*

CEDURA © Copyrighted Content. All rights reserved

PRACTICE 3

Problem Statement: *Build a Python Block Code to print a list of odd numbers from 1 to 10.*



Output:

[1, 3, 5, 7, 9]

Code Explanation: *In this program, we are expected to print a list of odd numbers from 1 to 10. As 1 is an odd number we can consider it as our 'start' value. As the target is up to 10, we will consider the 'stop' value as 10. However, as we need odd numbers the step size should be 2. This way we are adding 2 to the start value 1 so that the next number will be 3 and then again add 2 to the result so that the next number will be 5 and so on. This is an important concept to learn to use the 'step' value to have the desired output. This way the output will be 1,3,5,7,9. Mind that it will never cross the 'stop' value or include it in the output.*

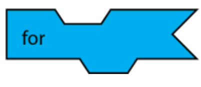
TOPIC ASSIGNMENT

1. Build a Python Block Code to print a list of numbers [1,2,3,4,5], using the Range function.
2. Build a Python Block Code to print a tuple of numbers (7,8,9,10), using the Range function.
3. Build a Python Block Code to print even numbers from 51 to 60, using the Range function.
4. Build a Python Block Code to print three values only between 1 to 10, using the Range function. (Hint: Should print 1, 4, 7)

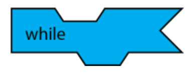
CEDURA © Copyrighted Content. All rights reserved

Control Loops in Python

Control loops are fundamental in Python programming as they enable repetitive execution of code, making tasks more efficient and reducing redundancy. By automating repetitive operations, loops eliminate the need for manual intervention, saving time and effort. Control loops also enhance code readability and maintainability by replacing lengthy, repetitive code blocks with concise looping constructs. They play a pivotal role in implementing algorithms, enabling tasks like searching, sorting, and iterative computations. Python supports two main types of loops: for **loops** and while **loops**.

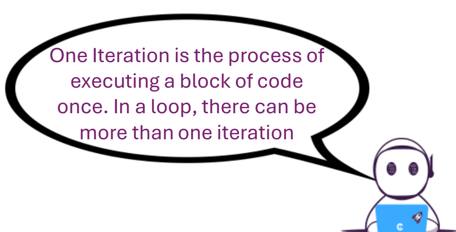


For Control Loop



While Control Loop

A **for loop** iterates over a sequence, such as a list, tuple, string, or range, allowing the programmer to process each element sequentially. For example, for i in range(5): iterates through numbers from 0 to 4. On the other hand, a **while loop** repeatedly executes a block of code as long as its condition evaluates to True, making it ideal for scenarios where the number of iterations is not predetermined. Python also provides loop control statements like **break** (to exit a loop prematurely), **continue** (to skip the current iteration), and **pass** (to create a placeholder for future code). These loops are invaluable for automating repetitive tasks, processing data structures, or implementing algorithms efficiently.



For Loop Block

Syntax:

for *variable in* ***sequence****:*
  # Code block to execute for each element in the sequence

Components:

1. ***variable:*** *Represent each element in the sequence during the iteration.*
2. ***sequence:*** *The collection or iterable object to iterate over (list, range, tuple, string)*
3. ***Indentation:*** *The block of code under the for loop must be indented.*

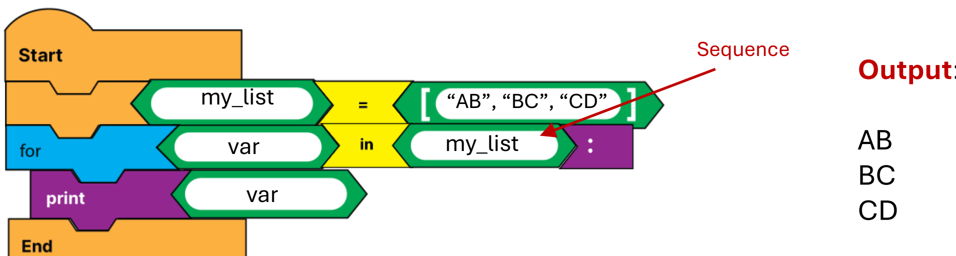
The **for loop** in Python is a powerful and versatile control structure used to iterate over a sequence, such as a list, tuple, string, dictionary, or range, and execute a block of code for each element. It is especially useful for automating repetitive tasks, processing data, or iterating through collections efficiently. Unlike traditional loops in other programming languages, Python's for loop directly accesses elements of a sequence rather than relying on index counters.

For example, for items in [1, 2, 3]: iterates through the list and processes each element. Python also allows the use of else with for loops, executing the else block when the loop completes naturally without encountering a break. The for loop's simplicity and readability make it an essential construct for tasks such as traversing arrays, manipulating strings, or implementing algorithms efficiently.

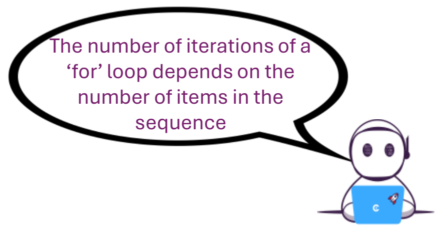
CEDURA © Copyrighted Content. All rights reserved

PRACTICE 1

Problem Statement: *Build a Python Block Code to demonstrate iteration through a sequence using for loop.*

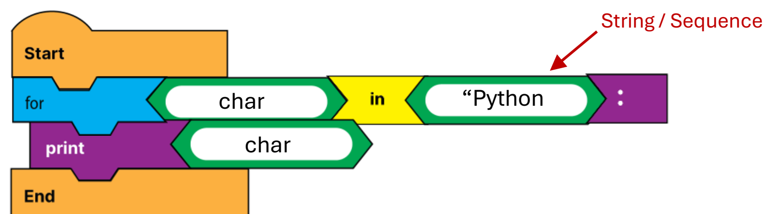


Code Explanation: *In this program, we have created a list with a bunch of strings. Using 'for' loop we intend to iterate through the sequence. As we are aware, 'for' loop fetches one by one item from the list and loads into the variable 'var'. Once an item is loaded into the variable, the control would go under the 'for' loop and execute the statements. Here, we have only one statement which is to print the content of the variable 'var'. Once it finishes printing the control goes back to the 'for' loop. Now, the second item will be loaded into the 'var' variable and the steps will repeat again. This repetition or iteration continues until the last item in the sequence is executed. This is how the for loop iterates through all the items in the sequence.*



PRACTICE 2

Problem Statement: *Build a Python Block Code to demonstrate iteration over a String.*



Output:

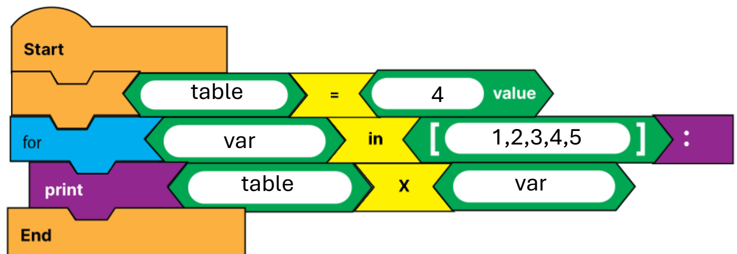
P
y
t
h
o
n

Code Explanation: *In this program, we are iterating over a string, which is a sequence. Hence, the for loop will go through one character at a time and print one character at a time. We can observe this in the output.*

CEDURA © Copyrighted Content. All rights reserved

PRACTICE 3

Problem Statement: *Build a Python Block Code to print the 4th table up to 5.*

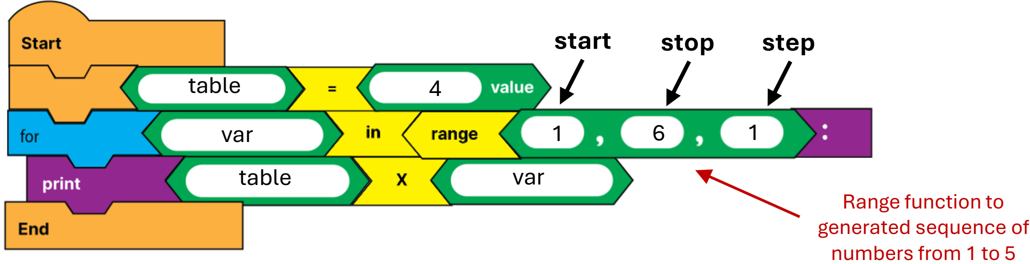


Output:
4
8
12
16
20

Code Explanation: *In this program, we have considered the variable 'table' as 4. Under that, we have a for loop which will iterate through the sequence 1,2,3,4,5. For every iteration of the for loop, we multiply the table value with the item in the sequence. This way, we are generating output, which is a 4th table. Once the for loop iterates through all the items, it will terminate automatically.*

PRACTICE 4

Problem Statement: *Build a Python Block Code to print the 4th table up to 5 using the range function.*



Output:
4
8
12
16
20

Code Explanation: *In this program, we have considered the variable 'table' as 4. Under that, we have a for loop with a range function. The range function will generate a sequence of numbers 1 to 5 (start and stop) with 1 as the step size. Now, the for loop will iterate through the sequence 1,2,3,4,5. For every iteration of the for loop, we multiply the table value with the item in the sequence. This way, we are generating output, which is a 4th table. Once the for loop iterates through all the items, it will terminate automatically.*

CEDURA © Copyrighted Content. All rights reserved

Nested for Loop

A **nested for loop** in Python is a loop inside another loop, where the **outer loop** controls the number of times the **inner loop** executes. The inner loop runs completely for each iteration of the outer loop, making it useful for handling multi-dimensional data structures like matrices, nested lists, and tables. It is commonly used in pattern printing, matrix operations, sorting algorithms, and simulations.

Syntax:

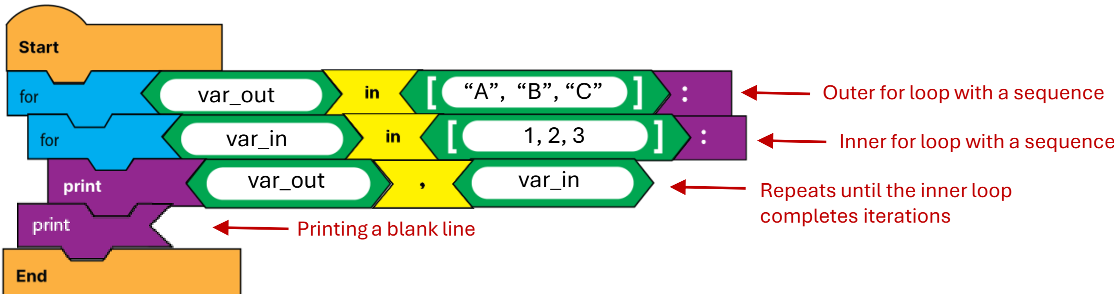
```
for outer_variable in outer_sequence: :            # Outer loop
  for inner_variable in inner_sequence: :          # Inner loop
      # Inner loop body
```

How it works:

1. *The **outer loop** runs as long as there are items to iterate.*
2. *Inside the outer loop, the **inner loop** executes repeatedly as long it has items to iterate.*
3. *After the inner loop completes iterating through all its items, the control will go back to the outer loop. Now the outer loop iterates through the next item.*
4. *This way, for every item of the outer loop, the inner loop runs completely, i.e. it iterates through all its items.*

PRACTICE 1

Problem Statement: *Build a Python Block Code to demonstrate nested for loop.*



Output:
A 1
A 2
A 3

B 1
B 2
B 3

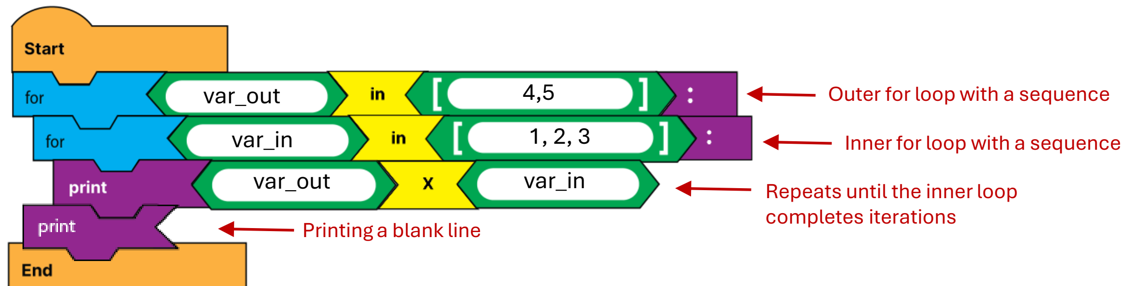
C 1
C 2
C 3

Code Explanation: *In this program, we have two 'for' loops - Outer and inner. For every outer loop iteration, the inner loop will complete all the iterations. Hence, the outer loop in its first iteration loads the value "A" into the variable var_out. Then, it would go under the for loop to encounter another for loop (inner). Here, this for loop is iterating over the numbers. For every iteration of the outer loop the inner loop with iterate through all its items. Hence, we will get A1, A2, and A3. After this, the inner for loop will terminate and print a blank line. Now, the control will go back to the outer for loop, where it iterates through the next item, i.e. "B". Again for "B" the inner for loop will run through all the items in the sequence. This way, we get B1, B2, and B3 in the output. After this, the process again repeats until the outer loop iterates through all the items.*

CEDURA © Copyrighted Content. All rights reserved

PRACTICE 1

Problem Statement: *Build a Python Block Code to print the 4th and 5th tables up to 3.*



Output:
4
8
12

5
10
15

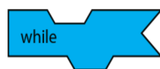
Code Explanation: *In this program, we have two 'for' loops - Outer and inner. For every outer loop iteration, the inner loop will complete all the iterations. Hence, first, the outer loop will consider 4 into the variable 'var_out'. Then it enters under the outer for loop where there is another for loop (inner). The inner for loop will iterate through all the items in the sequence i.e. 1,2,3 and load the items one by one into the variable 'var_out'. Under the inner for loop, we have an arithmetic operation to multiply var_out and var_in. Once the inner loop is done with iterating through its sequence, We are printing a blank space using a print block. After this, the control will go back to the outer loop where it iterates to the next item in the sequence. This way we will get 4th and 5th tables.*

TOPIC ASSIGNMENT

1. Build a Python Block Code to create a list of numbers and print them individually using the for loop.
2. Build a Python Block Code to demonstrate iteration over the string "Hello" using the for loop.
3. Build a Python Block Code to demonstrate iteration through the given sequence (45, 'Py", -5).
4. Build a Python Block Code to print 1 to 5 numbers using the Range function and For loop.
5. Build a Python Block Code to print 3 tables up to 5 using the for flop.

CEDURA © Copyrighted Content. All rights reserved

while Loop Block



Syntax:

```
while condition:
    # Code block to execute as long as the condition is True
```

Components:

1. ***condition:*** *A Boolean expression that is evaluated before each iteration. If the condition is TRUE, the loop executes. If it results in FALSE, the loop terminates*
2. ***code block:*** *The indented block of code under the while statement that is repeatedly executed.*

The while loop in Python is a control structure that allows repetitive execution of a block of code as long as a specified condition evaluates to True. It is particularly useful when the number of iterations is not predetermined and depends on dynamic conditions, such as user input or real-time data. For example, while count < 10: repeatedly executes the code block until the condition count < 10 becomes False. Proper care must be taken to ensure the condition eventually becomes False to avoid infinite loops. With its flexibility and simplicity, the while loop is ideal for scenarios like waiting for a specific event, real-time monitoring, or condition-based tasks.

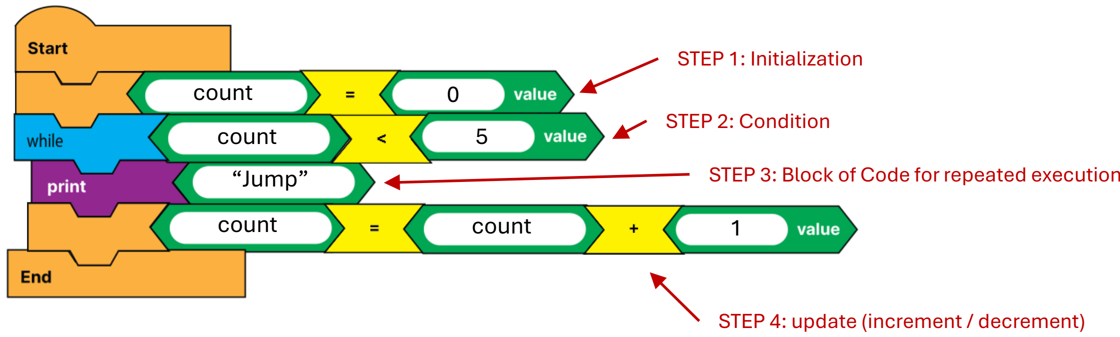
We can understand the functioning of a while loop through 4 essential steps for proper functioning.

Example:

```
count = 0                ← STEP 1: Initialization
while count < 5:         ← STEP 2: Condition
    print(count)         ← STEP 3: Block of Code for repeated execution
    count = count + 1    ← STEP 4: update (increment / decrement)
```

PRACTICE 1

Problem Statement: *Build a Python Block Code to print 'Jump" for 5 times*



CEDURA © Copyrighted Content. All rights reserved

Code Explanation: *In this program, we are using a 'while' loop to print a statement 'Jump' 5 times. From this, we can understand that the condition is 5. Following the 4 steps in writing a while loop, STEP1: initialize a counter 'count' with 0, STEP2: check whether the 'count' is less than 5, STEP3: execute the block of code which is printing 'Jump' text in this case, STEP4: for every iteration increase the 'count' variable by 1. This way, the 'while' loop will repeat the execution of a block of code 5 times (until the condition is TRUE).*

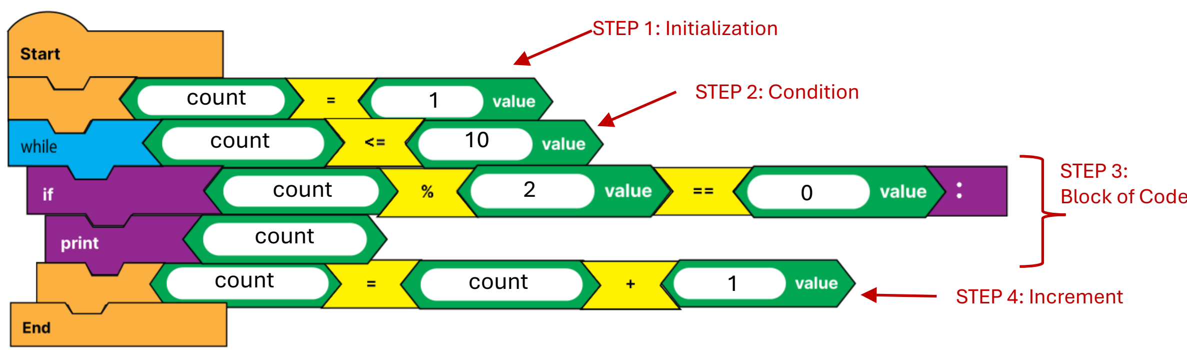
Output:

Jump
Jump
Jump
Jump
Jump

Follow the 4 steps and check them thoroughly to avoid an infinite loop or non-execution of the while loop.

PRACTICE 2

Problem Statement: *Build a Python Block Code to identify EVEN and ODD numbers from 1 to 10*



Code Explanation: *In this program, we have two parts to this code - a while loop to repeat a block of code 10 times, incrementing the count for every iteration; and an 'if' block of code to check whether the count value is an EVEN number or not, for iteration. Meaning, we are starting the count value with 1, which satisfies the condition in the 'while' loop.*

Hence, the control will go under the 'while' where it encounters a 'if' statement. This would check whether the value in the 'count' is exactly divisible by 2. If any number is exactly divisible by 2 without leaving a remainder, then it is an EVEN number. To check them, we are using a mod (%) operator. Whenever a number is exactly divisible by 2 then the if condition will result in TRUE, allowing the control to go under the 'if' block and print the count.

This means the current count value is an EVEN number. In case the value in the count is not exactly divisible by 2 then it will not be printed. After that, we increment the count value by 1 and the loop continues. Hence, by the end of the 'while' loop, we will have all the EVEN numbers printed.

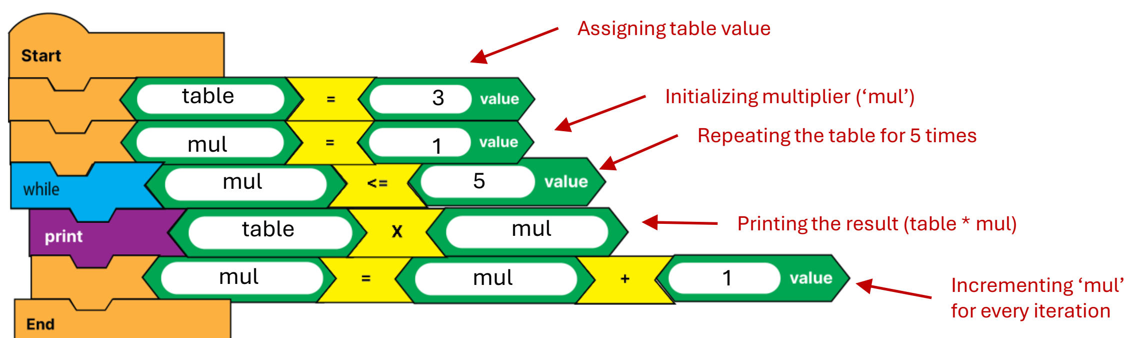
Output:

2
4
6
8
10

CEDURA © Copyrighted Content. All rights reserved

PRACTICE 3

Problem Statement: *Build a Python Block Code to print 3 tables up to 5.*



Output:
3
6
9
12
15

Code Explanation: *In this program, we are using a 'while' loop to print the 3 tables up to 5. For this, we have initialized the 'table' and 'mul' variables with 3 and 1, respectively. The 'table' variable will stay constant i.e. 3, whereas the 'mul' variable will be incremented for every iteration until it reaches value 5. This way we will be able to multiply the 'table' variable with the 'mul' variable in every iteration to get a table output. If you look at the code the 'while' loop repeats the block of code 5 times until the condition is TRUE. Once the 'mul' value crosses 5, the condition fails and the loop will be terminated.*

Nested while Loop

A **nested** while **loop** in Python refers to a while loop inside another while loop. The inner loop executes completely for each iteration of the outer loop. This structure is useful for tasks that require multiple levels of looping, such as working with multidimensional data, implementing algorithms with layered iterations, or creating patterns.

Syntax:

```
while condition1:                # Outer loop
  # Code block for the outer loop
  while condition2:              # Inner loop
    # Code block for the inner loop
```

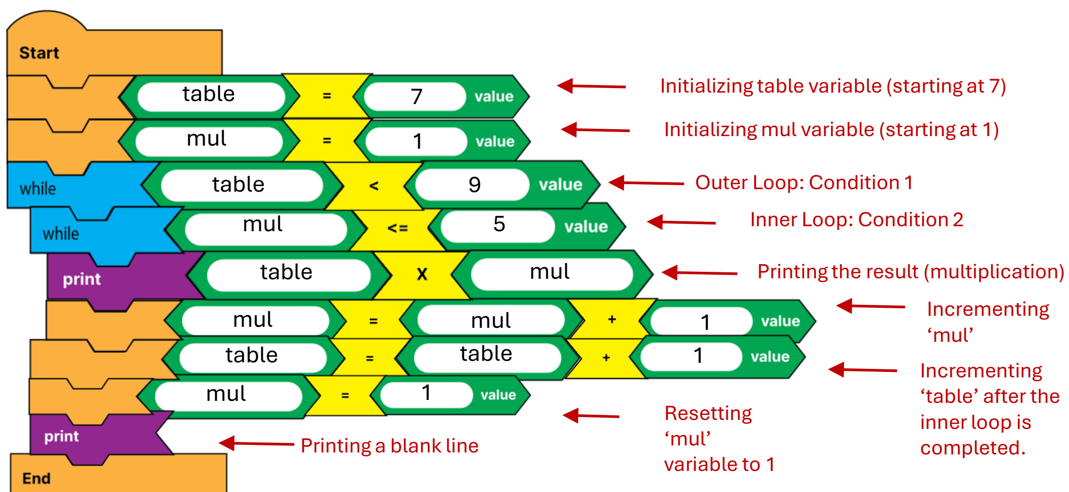
How it works:

1. *The **outer loop** runs as long as the condition1 is TRUE*
2. *Inside the outer loop, the **inner loop** executes repeatedly as long as condition2 is TRUE*
3. *After the inner loop finishes, the outer loop continues with its next iteration.*

CEDURA © Copyrighted Content. All rights reserved

PRACTICE 1

Problem Statement: *Build a Python Block Code to print 7 and 8 tables up to 5, one after another*



Output:

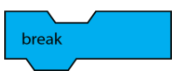
7
14
21
28
35

8
16
24
32
40

Code Explanation: *In this program, we intend to print 7 and 8 tables up to 5 each. This calls for a nested 'while' loop as we need the first 'while' loop to iterate through the tables (7 and 8) and the second 'while' loop to iterate through the multiplier (1 to 5). So, the first 'while' loop's condition (outer loop), i.e. condition 1 will check whether the table value is less than 9 so that it can repeat the block or code. The second 'while' loop's condition (inner loop) will check whether the multiplier reached value 5 or not. Both the loops continue to iterate until their respective conditions result in TRUE.* ***However, you should note that for every single iteration of the outer loop, the inner loop will iterate 5 times (condition).*** *This is how for every table we get output up to value 5.*

CEDURA © Copyrighted Content. All rights reserved

Break Block



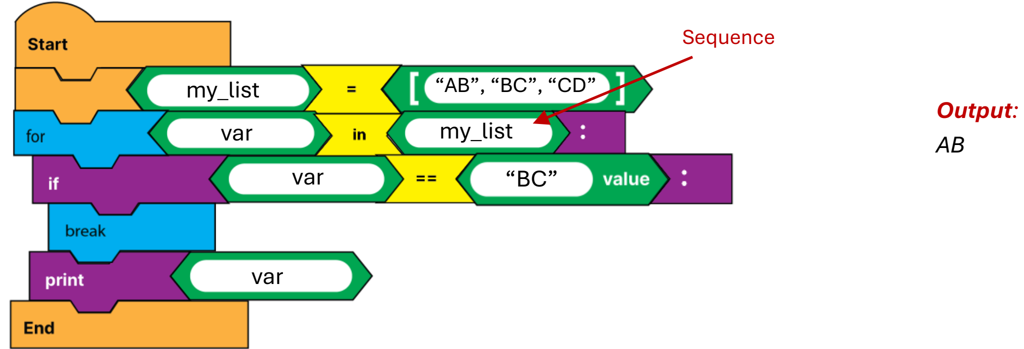
Syntax:
```
if condition:
    break
```

The **break** statement in Python is a control flow tool used to **exit a loop prematurely** when a specific condition is met. It is typically employed within 'for' or 'while' loops to interrupt their execution before they complete their normal iteration. When the break statement is encountered, the program immediately exits/terminates the loop. Even the code after the break statement but inside the loop will not be executed.

This is particularly useful for situations where continuing the loop is unnecessary after an event or a value occurs. For example, in a loop iterating through numbers, a break can stop the loop as soon as the desired number is found. It helps in optimizing performance by avoiding unnecessary iterations, making programs more efficient and easier to manage. However, care must be taken to ensure that break is used appropriately to avoid unexpected behaviour or incomplete tasks within the loop.

PRACTICE 1

Problem Statement: *Build a Python Block Code to demonstrate the BREAK statement using a for loop.*



Output:

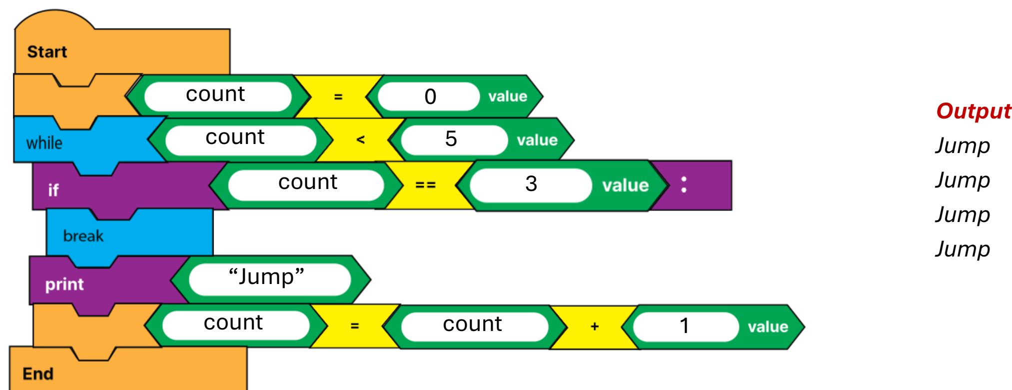
AB

Code Explanation: *Recap the 'for' loop practice problem statement where we are expected to iterate through all the items in the sequence and print them. Here, we are using the same block code however, we introduced a 'break' statement under an 'if' statement. When the code runs, the first item is loaded into the 'var' variable. After that, we are checking whether the content in the 'var' is 'BC'? As the 'if' condition results in FALSE, it will not go under the 'if' block but to the next executable statement in the loop, i.e. the printing of the variable. In the second iteration, BC will be loaded into the 'var'. When the control reaches the 'if' block, the condition will result in TRUE as the content in 'var' matches with "BC". Hence, the control will go under the 'if' block and execute the 'break' statement. Once the 'break' statement is executed, the entire 'for' loop will be terminated prematurely (no further iterations or executing of the code blocks under the 'if'). Hence, you will find only a single-item output. This is how a 'break' statement is used (always under an 'if' conditional block) to terminate the loop prematurely. Note that blocks under the 'if' condition (here, the print block) will not be executed even for the current iteration, as the loop terminates at once when it executes a break statement.*

CEDURA © Copyrighted Content. All rights reserved

PRACTICE 2

Problem Statement: *Build a Python Block Code to demonstrate the BREAK statement using while loop.*

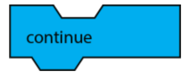


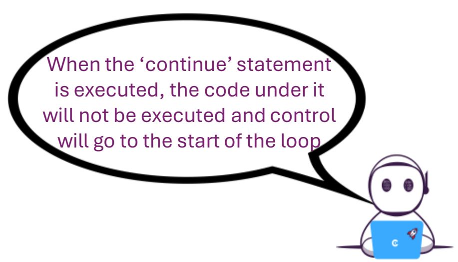
Output:

Jump
Jump
Jump
Jump

Code Explanation: *Recap the 'while' loop practice problem statement where we are expected to print 'Jump' 5 times. Here, we are using the same block code however, we introduced a 'break' statement under an 'if' statement. When the code runs, it will check, for each iteration, whether the count reached '3' or not. In the first iteration, the 'count value is '0' hence, the 'if' condition will result in FALSE, so it will not go under the 'fi' statement and execute the break statement. However, the statements below the 'if' block will execute (i.e. print and the increment) to complete the first iteration. In the second iteration, again, it checks whether the 'count' value is '3'. As the 'if' condition results in FALSE, it will not proceed like the first iteration. However, when the 'count' value reaches 3, then the 'if' conditional statement will result in TRUE and the control will go under the 'if' and execute the 'break' statement. Once the 'break' statement is executed the entire 'while' loop will be terminated prematurely (no further iterations or executing of the code blocks under the 'if').*

Continue Block





Syntax:

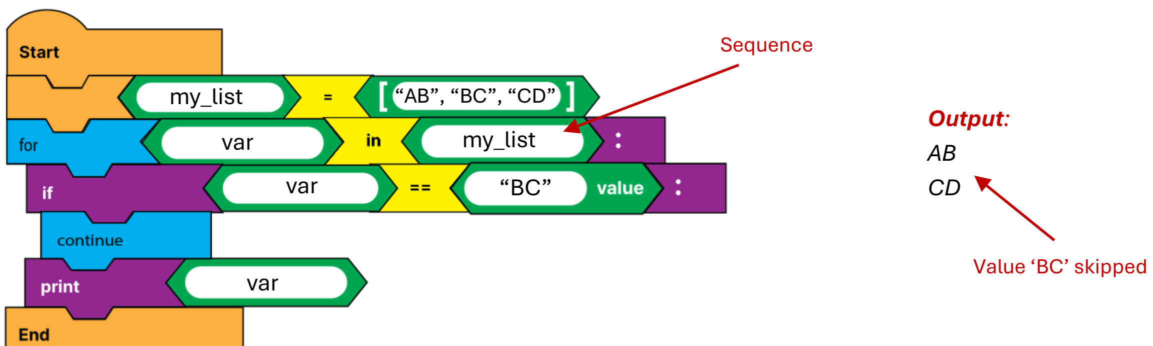
```
if condition:
    continue
```

The **continue** statement in Python is used to skip the **current iteration** of a loop and move to the next iteration, without exiting the loop entirely. The 'continue' statement, like the 'break' statement, is always used under a conditional 'if' block. When the continue statement is encountered, even the remaining code in the current iteration is ignored, and the loop proceeds with the next cycle. You should be mindful of this and should not place essential code such as count increment after the continue statement. This feature is particularly useful for filtering data or managing special conditions within loops, making the code more concise and efficient. However, like the break statement, it should be used thoughtfully to avoid unintended behaviour or skipped tasks.

CEDURA © Copyrighted Content. All rights reserved

PRACTICE 1

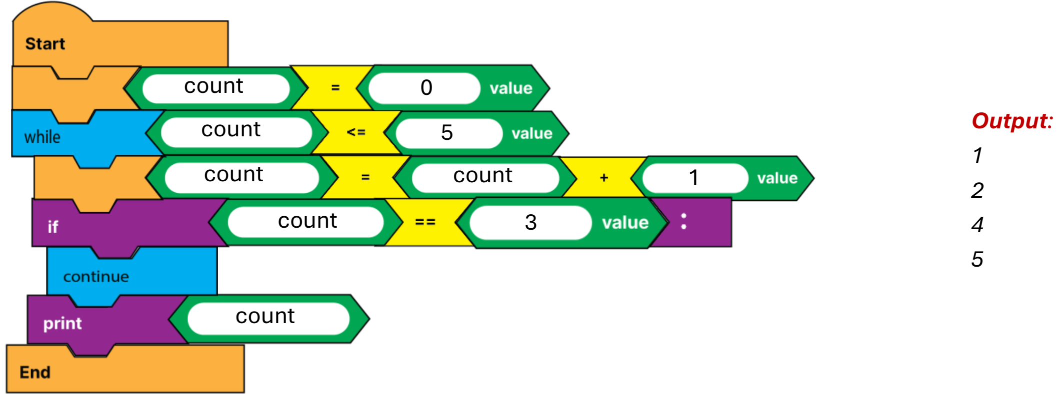
Problem Statement: *Build a Python Block Code to demonstrate the CONTINUE statement using a for loop.*



Code Explanation: *In this code we are using a ‘continue’ block under the ‘if’ conditional statement. For every iteration of the ‘for’ loop, the content of the variable ‘var’ is checked against the value“BC”. In the first iteration, the value in the ‘var’ is ‘AB’ hence, the ‘if’ condition will result in FALSE and the print function is executed. However, in the second iteration, the value in the ‘var’ variable will be ’BC’. In this case, the ‘if’ condition will result in TRUE and the control will go under the ‘if’ statement and execute the ‘continue’ statement. Once the ‘continue’ statement is executed the loop skips the current iteration, including the execution of the print function that is after the ‘continue’ statement. Hence, BC will not be printed. The final iteration will be executed without any hurdles. Hence, the output will have all the items except the second item i.e. “BC”. It is like we are filtering out this item.*

PRACTICE 2

Problem Statement: *Build a Python Block Code to demonstrate the CONTINUE statement using a while loop.*



Code Explanation: *In the above code, we are expected to print the numbers 1 to 5, except 3. To skip value 3, we are using an ‘if’ block along with ‘continue’. The while will work fine for the first and second*

CEDURA © Copyrighted Content. All rights reserved

Iterations, where the count will be 1 and 2. However, for the third iteration, the count value will be 3. In this case, the 'if' block will result in TRUE, executing the 'continue' block under it. Once the 'continue' block is executed, the current iteration will be stopped, and the control will return to the 'while' loop for the next iteration. While doing so, even the 'print' block after the 'continue' block will not be executed. Hence, the value 3 will not be printed, as observed in the output. This way, we can skip any particular event or iteration inside a control loop, using the 'continue' statement.

TOPIC ASSIGNMENT

1. Build a Python Code Block to print numbers from 1 to 5 using the while loop.
2. Build a Python Code Block to print EVEN numbers from 1 to 10 using the while loop.
3. Build a Python Code Block to print numbers from 5 to 1 in reverse order using the while loop.
4. Build a Python Code Block to print 7 table up to 5 using the while loop
5. Build a Python Code Block to print both 3 and 4 tables up to 4 using nested while loop.
6. Build a Python Code Block to print all the EVEN numbers from the give list [10,21,33,98,67,4] using for loop and continue statement.
7. Build a Python Code Block to print only NON-VOWELS(not a vowel) characters in the given string 'House' using for loop and continue statement.
8. Build a Python Code Block to print numbers from 1 to 10 but break at 7, using the while loop and break statement,
9. Build a Python Code Block to break the loop if any vowel is found in the given string "Spring" using the for loop and break statement.

CEDURA © Copyrighted Content. All rights reserved

NOTES

CEDURA © Copyrighted Content. All rights reserved

SOLUTIONS - ASSIGNMENTS

Understanding Data & Data Types

1. Identify the below data types.

i. 23 **Integer**

ii. -26 **Integer**

iii. 0 **Integer**

iv. -12.5 **Float**

v. true **Boolean**

vi. false **Boolean**

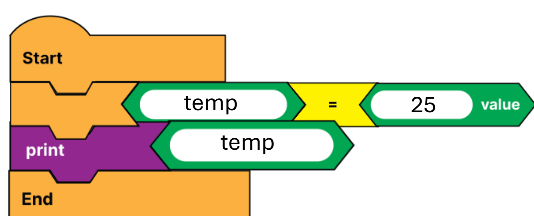
vii. 3+4j **Complex**

viii. “Hii” **String**

ix. “35” **String**

Exploring Variables

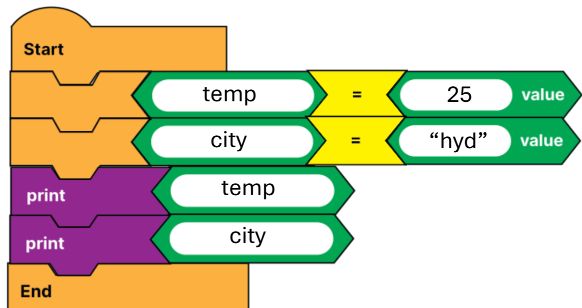
1. Build a Python Block Code to assign a value to a variable and print it.



Input1:	**Input2:**	**Input3:**
temp = 25	temp = 6.7	temp = “python”
Output:	**Output:**	**Output:**
25	6.7	python

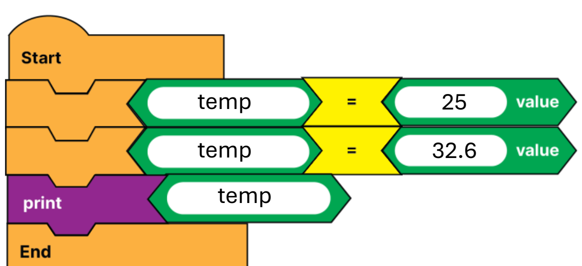
CEDURA © Copyrighted Content. All rights reserved

2. Build a Python Block Code to assign two variables with different values.



Input1:	**Input2:**	**Input3:**
temp = 25 city = "hyd"	temp = 92.5 city = "sequel"	temp = 25 temp = True
Output:	**Output:**	**Output:**
25 *hyd*	*92.5* *sequel*	25 True

3. Build a Python Block Code to assign a value to a variable and then change/update the value.

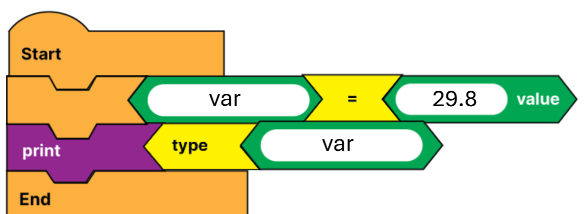


Input1:	**Input2:**	**Input3:**
temp = 25 temp = 32.6	temp = 25 temp = "keys"	temp = 25 temp = True
Output:	**Output:**	**Output:**
32.6	*keys*	True

** Assigned Boolean value

CEDURA © Copyrighted Content. All rights reserved

4. Build a Python Block Code to assign a Float value to a variable and find its data type.



Input1:

var = 29.8

Output:

<class ‘float’>

Input2:

var = 1.3

Output:

<class ‘float’>

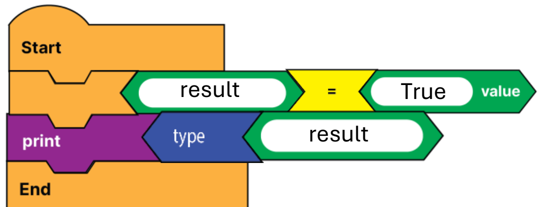
Input3:

var = 28498.33

Output:

<class ‘float’>

5. Build a Python Block Code to assign a Boolean value to a variable and find its data type.



Input1:

result = True

Output:

<class ‘bool’>

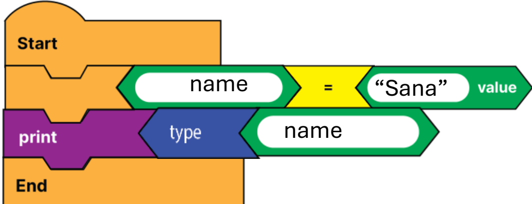
Input2:

result = False

Output:

<class ‘bool’>

6. Build a Python Block Code to assign a String value to a variable and find its data type.



Input1:

name = “Sana”

Output:

<class ‘str’>

Input2:

name = “Abhi”

Output:

<class ‘str’>

Input3:

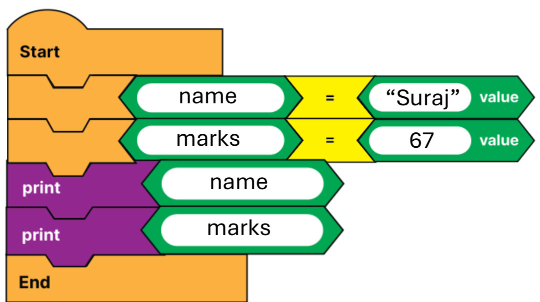
name = “123”

Output:

<class ‘str’>

CEDURA © Copyrighted Content. All rights reserved

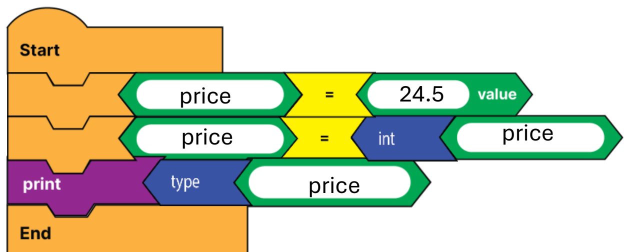
7. Build a Python Block Code to assign a name and a subject mark of a student and print them. Also, print its data type.



Input1:	Input1:	Input1:
name = "Suraj" marks = 67	name = "Pradeep" marks = 55	name = "Gireesh" marks = 92
Output:	**Output:**	**Output:**
Suraj *67*	*Pradeep* *55*	*Gireesh* *92*

Type Casting

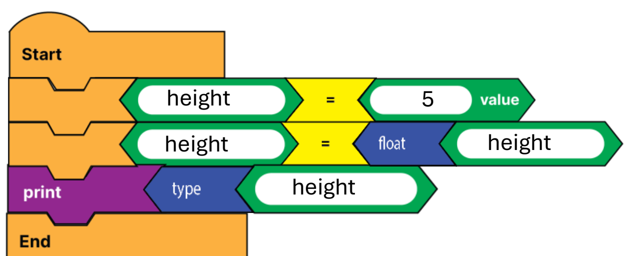
1. Build a Python Block Code to convert the content of the variable price = 24.5 into an integer.



Input1:	Input2:	Input3:
price = 24.5	price = 36.05	price = 77.19
Output:	**Output:**	**Output:**
<class "int">	<class "int">	<class "int">

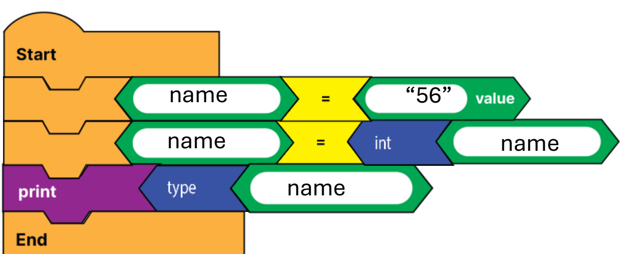
CEDURA © Copyrighted Content. All rights reserved

2. Build a Python Block Code to convert the content of the variable height = 5 into a float.



Input1:	**Input2:**	**Input3:**
height = 5	height = 32	height = 67
Output:	**Output:**	**Output:**
<class "float">	<class "float">	<class "float">

3. Build a Python Block Code to convert the content of the variable name = "56" into an integer.



Input1:	**Input2:**	**Input3:**
name = "56"	name = "63"	name = "82"
Output:	**Output:**	**Output:**
<class "int">	<class "int">	<class "int">

CEDURA © Copyrighted Content. All rights reserved

Operators

1. Build a Python Block Code to assign two values as integers to different variables and perform below athematic operations.

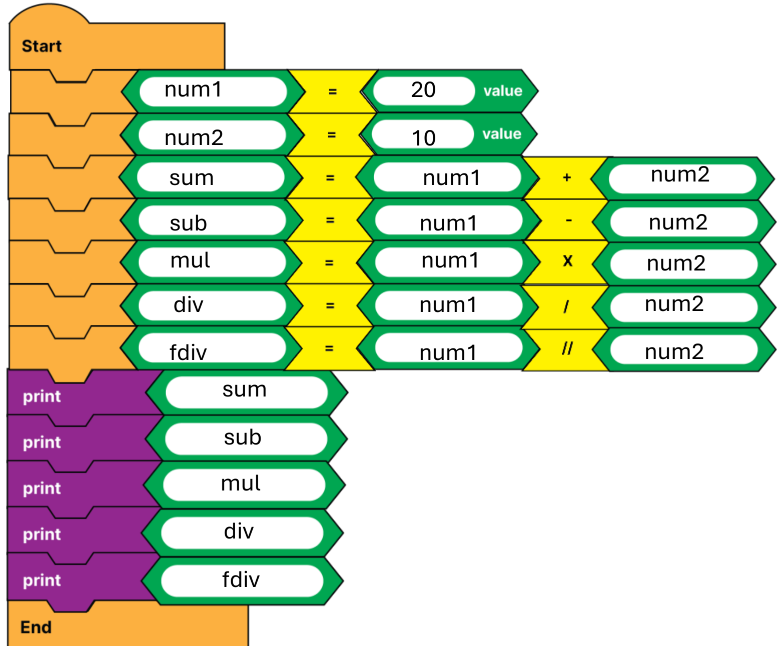
a. **Addition**

b. **Subtraction**

c. **Multiplication**

d. **Division**

e. **Floor Division**



Input1:

num1 = 20
Num2 = 10

Output:

30
10
200
2
2.0

Input2:

num1 = 36
num2 = 6

Output:

36
30
108
6
6.0

Input3:

num1 = 27
num2 = 3

Output:

30
24
81
9
9.0

CEDURA © Copyrighted Content. All rights reserved

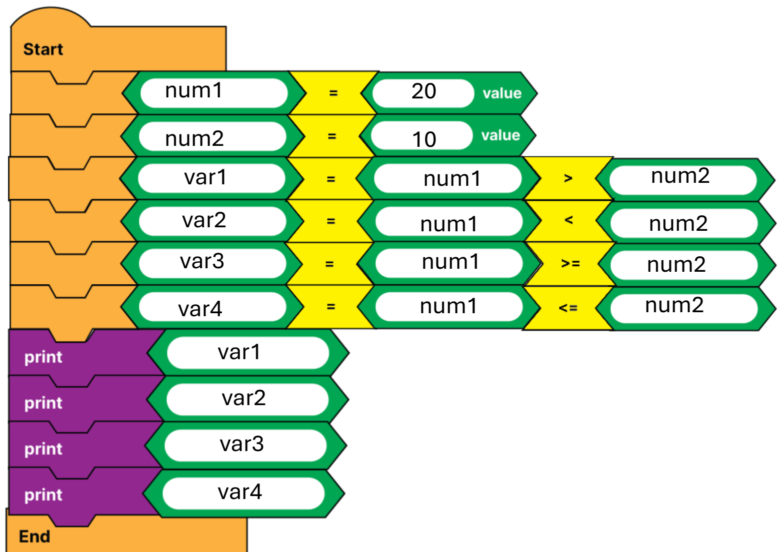
2. Build a Python Block Code to assign two values as integers to different variables and perform the below comparison operations.

a. **Greater than**

b. **Lesser than**

c. **Greater than or Equal to**

d. **Lesser than or Equal to**



Input1:

num1 = 20
Num2 = 10

Output:

True
False
True
False

Input2:

num1 = 32
num2 = 75

Output:

False
True
False
True

Input3:

num1 = 64
num2 = 64

Output:

False
False
True
True

CEDURA © Copyrighted Content. All rights reserved

Print

1. Write the Syntax of the print function and explain all its parameters.

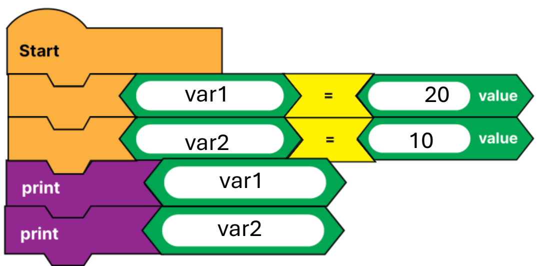
Syntax:

```
print(*objects, sep=' ', end='\n', file=sys.stdout, flush=False)
```

Parameters:

1. sep: Represents one or more objects (values, variables, or expressions) to be printed. Multiple objects can be separated by commas.
2. sep *(Optional)*: Defines the string used to separate the objects. Default: A single space (' ').
3. end *(Optional)*: Defines the string appended after the printed output. Default: A newline character ('\n'), which moves the cursor to the next line.
4. file *(Optional)*: Specifies the file or stream where the output is sent. Default: sys.stdout (console output).
5. flush *(Optional)*: A Boolean value (True or False) that forces the output buffer to be flushed immediately if set to True. Default: False. Used when working with real-time streams.

2. Build a Python Block Code to assign two variables with different value.



Input1:

var1 = 20
var2 = 10

Output:

20
10

Input2:

var1 = 37
var2 = 56

Output:

37
56

Input3:

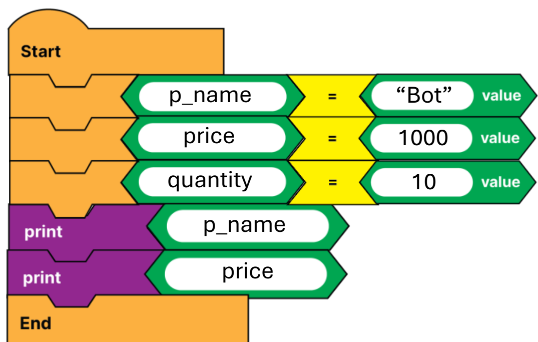
var1 = 74
var2 = 43

Output:

74
43

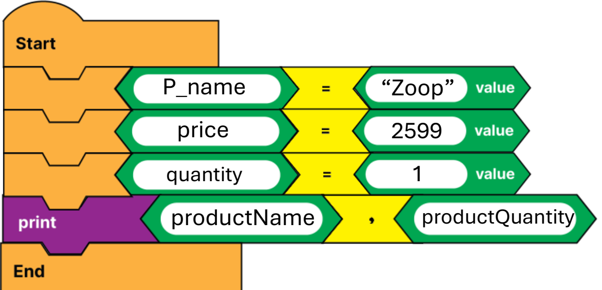
CEDURA © Copyrighted Content. All rights reserved

3. Build a Python Block Code to assign product name, price, and quantity and print any two variables, individually.



Input1: p_name = "Bot" price =1000 quantity = 10 **Output:** Bot 1000	**Input2:** p_name = "RobArm" price =15999 quantity = 10 **Output:** RobArm 15999	**Input3:** p_name = "Zoop" price =2599 quantity = 1 **Output:** Zoop 2599

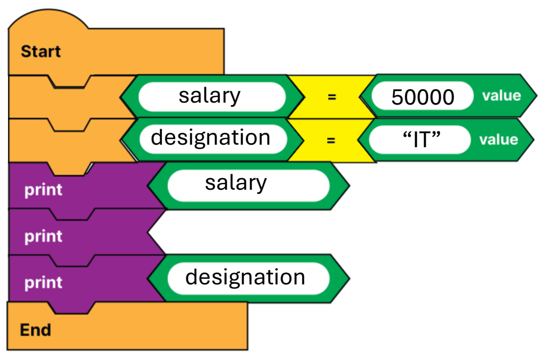
4. Build a Python Block Code to assign product name, price, and quantity and print any two variables using a single print function.



Input1: P_name = "Zoop" price =2599 quantity = 1 **Output:** Zoop, 1	**Input2:** p_name = "Bot" price =1000 quantity = 10 **Output:** Bot, 10	**Input3:** p_name = "RobArm" price =15999 quantity = 10 **Output:** RobArm, 10

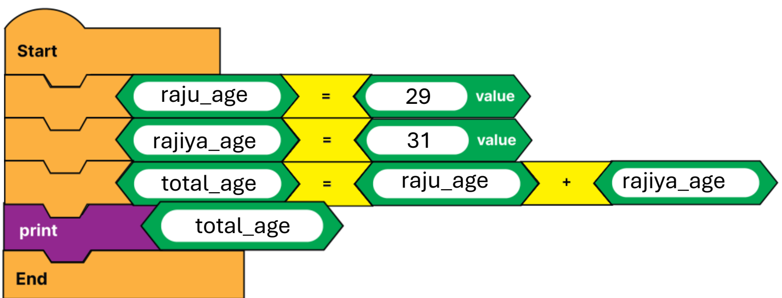
CEDURA © Copyrighted Content. All rights reserved

5. Build a Python Block Code to assign salary and designation and print them individually with a blank line in between.



Input1: salary= 50000 designation ="IT" **Output:** 50000 IT	**Input2:** salary= 75000 designation ="HR" **Output:** 75000 HR	**Input3:** salary= 25000 designation ="Marketing" **Output:** 25000 Marketing

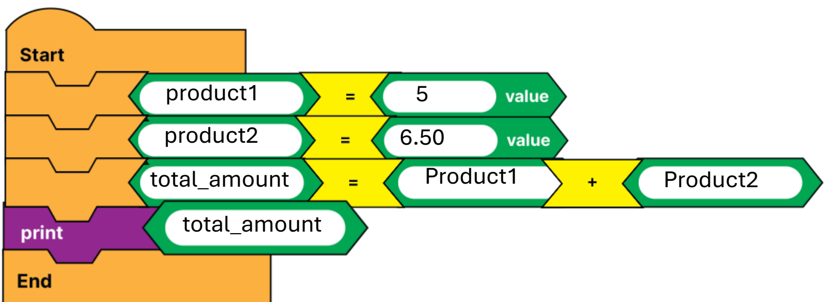
6. Find the total age of Raju (29 years) and Rajiya (31 years) and print it.



Input1: raju_age = 29 rajiya_age =31 **Output:** 60	**Input2:** raju_age = 43 rajiya_age =54 **Output:** 97	**Input3:** raju_age = 8 rajiya_age =40 **Output:** 48

CEDURA © Copyrighted Content. All rights reserved

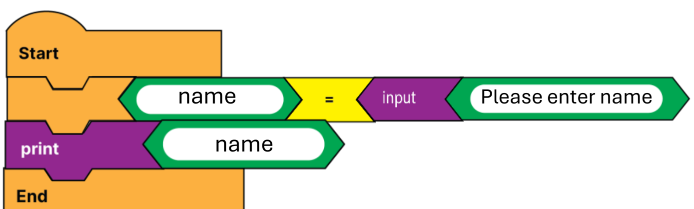
7. What is the total bill amount if the prices of two products are Rs.5 and Rs. 6.50



Input1:	Input2:	Input3:
product1 = 5 product2 = 6.50	product1 = 83.2 product2 = 39	product1 = 73.8 product2 = 56.7
Output:	**Output:**	**Output:**
11.50	122.2	130.5

Input

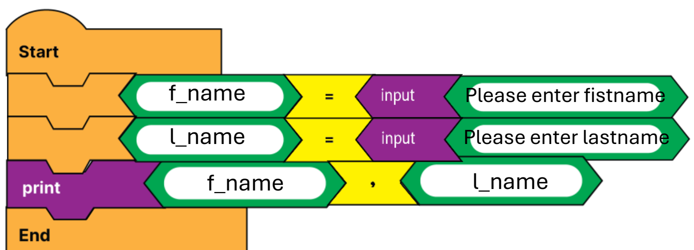
1. Build a Python Block Code to seek an Employee's name and print it.



Input1:	Input2:	Input3:
Please enter name: Suresh	Please enter name: Ramesh	Please enter name: Mukesh
Output:	**Output:**	**Output:**
Suresh	Ramesh	Mukesh

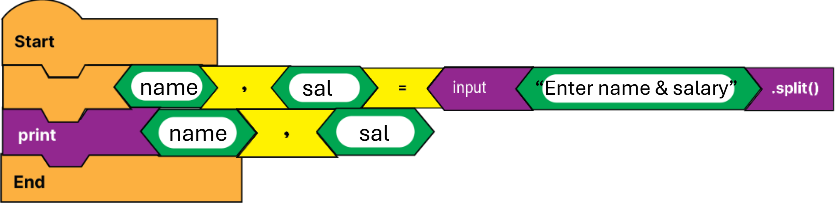
CEDURA © Copyrighted Content. All rights reserved

2. Build a Python Block Code to seek the 'first name' and 'last name' of a student and print them together.



Input1:	**Input1:**	**Input1:**
Please enter fistname :Marrapu Please enter lastname: Kiran	Please enter fistname :Siva Please enter lastname: Parvathi	Please enter fistname :Ai Please enter lastname: Tinkr
Output:	**Output:**	**Output:**
Marrapu, Kiran	Siva, Parvathi	Ai, Tinkr

3. Build a Python Block Code to seek an Employee's name and salary using a single input function and print them.



Input1:	**Input2:**
Enter name & salary: Kiran, 50000	Enter name & salary: Suma, 100000
Output:	**Output:**
Kiran, 50000	Suma, 100000

Input3:

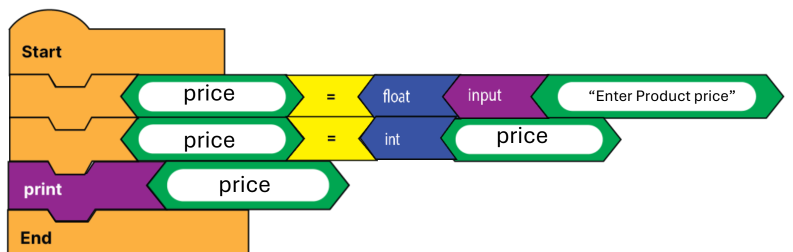
Please enter name & salary: Hyma, 75500

Output:

Hyma, 75500

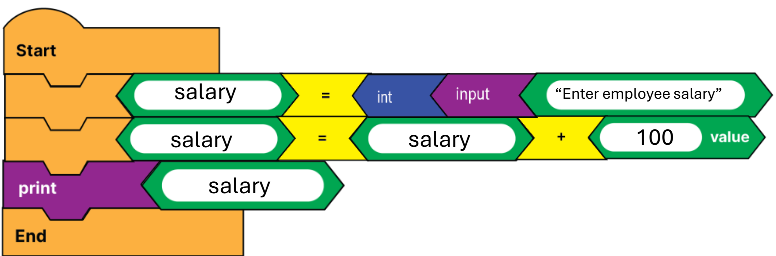
CEDURA © Copyrighted Content. All rights reserved

4. Build a Python Block Code to seek the price (float) value of a product, convert it into an integer and print it.



Input1: Enter product price: 2000.25 **Output:** 2000	**Input2:** Enter product price: 26.36 **Output:** 26	**Input3:** Enter product price: 891.4 **Output:** 891

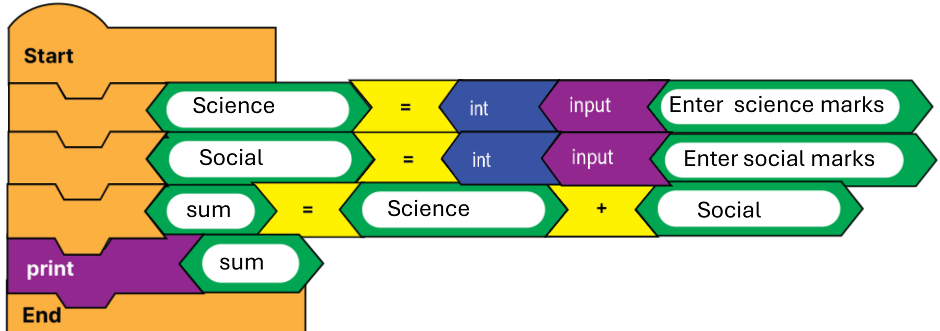
5. Build a Python Block Code to seek the Employee’s salary and add Rs.100 and print the final salary.



Input1: Enter employee salary:13000 **Output:** 13100	**Input2:** Enter employee salary: 22000 **Output:** 22100	**Input3:** Enter employee salary: 22000 **Output:** 53600

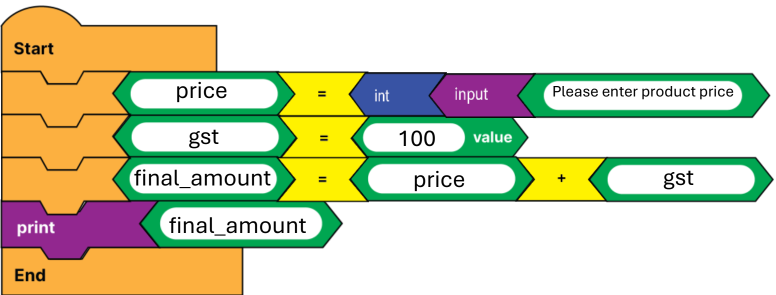
CEDURA © Copyrighted Content. All rights reserved

6. Build a Python Block Code to seek Science and Social Marks from a student and print the total marks.



Input1:	Input2:	Input3:
Enter science marks: 78 Enter social marks: 84	Enter science marks: 54 Enter social marks: 98	Enter science marks: 87 Enter social marks: 98
Output:	**Output:**	**Output:**
162	152	185

7. Build a Python Block Code to seek the price of a product add Rs.100 as GST and print the final amount.

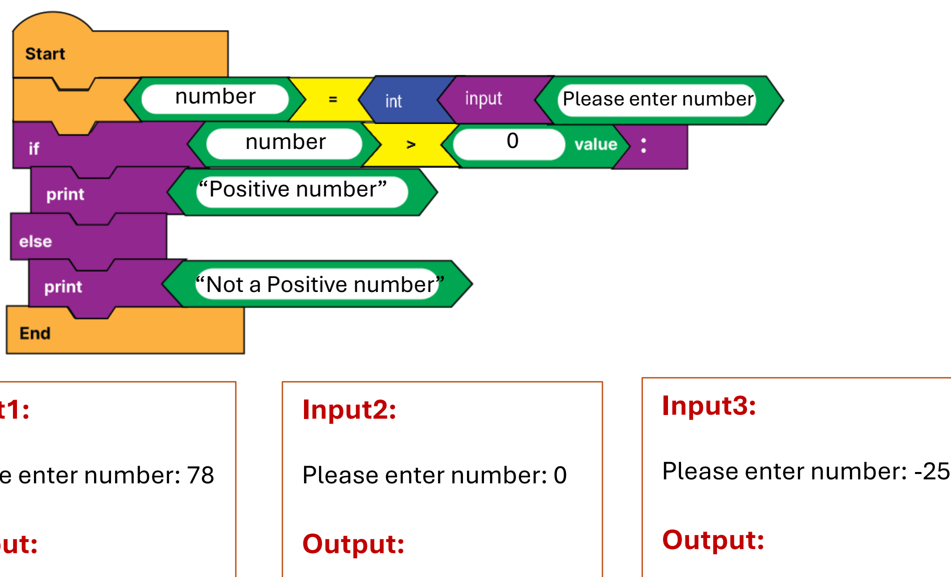


Input1:	Input2:	Input3:
Please enter product price: 1000	Please enter product price: 1800	Please enter product price: 1780
Output:	**Output:**	**Output:**
1100	1900	1880

CEDURA © Copyrighted Content. All rights reserved

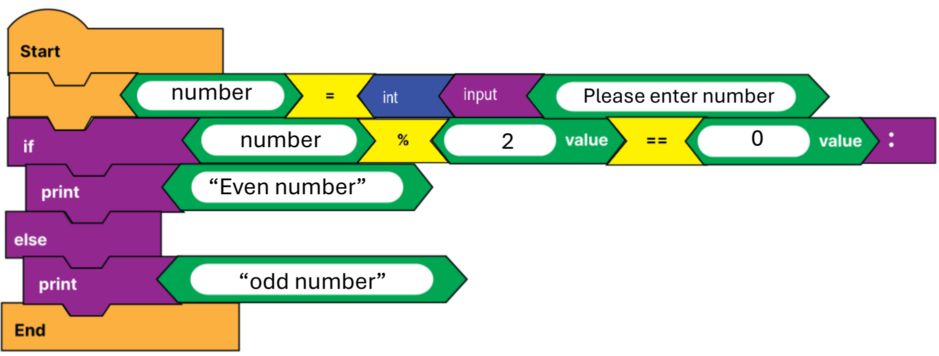
CONDITIONAL Blocks

1. Build a Python Block Code to check whether a given number is positive or not.



Input1:	**Input2:**	**Input3:**
Please enter number: 78	Please enter number: 0	Please enter number: -25
Output:	**Output:**	**Output:**
Positive number	Positive number	Not a Positive number

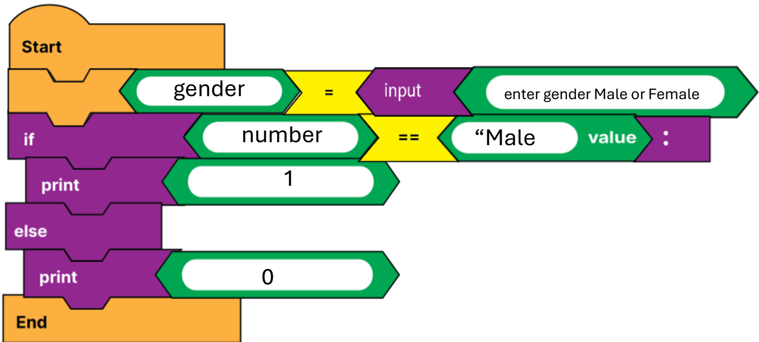
2. Build a Python Block Code to seek a number from the user and check whether a given number is even or odd.



Input1:	**Input2:**	**Input3:**
Please enter number: 78	Please enter number: 55	Please enter number: 69
Output:	**Output:**	**Output:**
odd number	Even number	odd number

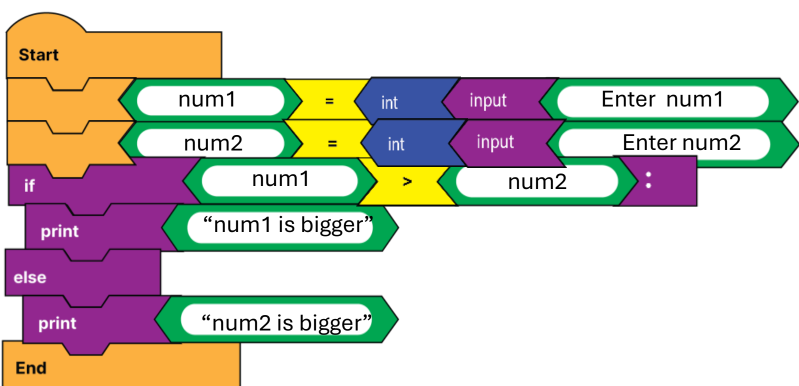
CEDURA © Copyrighted Content. All rights reserved

3. Build a Python Block Code to seek the Gender of the student (“Male” or ”Female”) and print 1 if the gender is male and 0 if the gender is female.



Input1: Enter gender Male or Female: “Male” **Output:** 1	**Input2:** Enter gender Male or Female: “Female” **Output:** 0

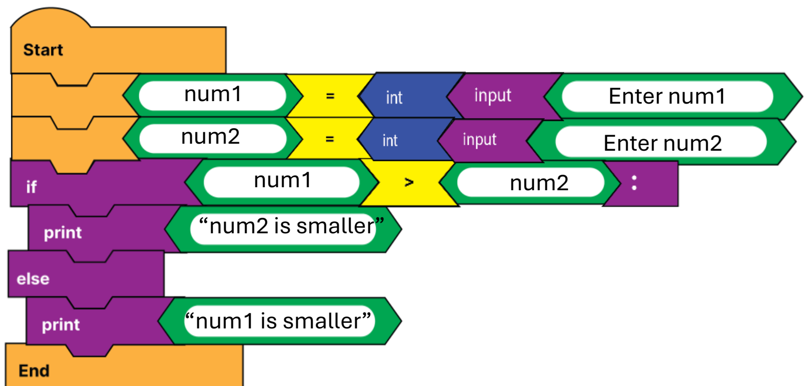
4. Build a Python Block Code to seek two numbers from the user and display the bigger number.



Input1: Enter num1: 56 Enter num2: 73 **Output:** num2 is bigger	**Input2:** Enter num1: 84 Enter num2: 29 **Output:** num1 is bigger	**Input3:** Enter num1: 56 Enter num2: 87 **Output:** num2 is bigger

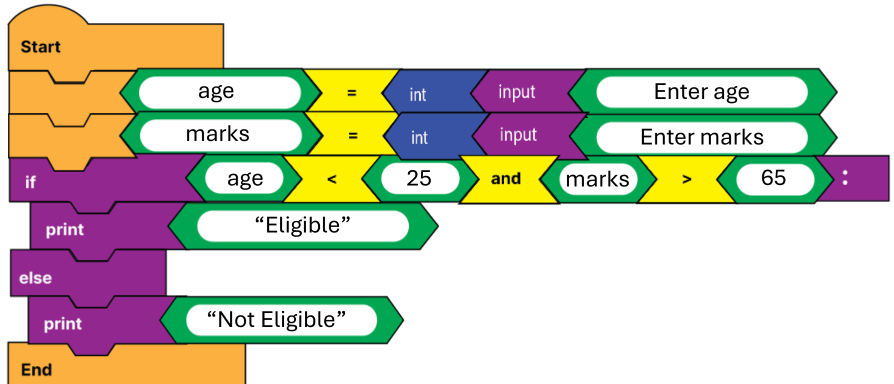
CEDURA © Copyrighted Content. All rights reserved

5. Build a Python Block Code to seek two numbers from the user and display the smaller number.



Input1: Enter num1: 56 Enter num2: 73 **Output:** num1 is smaller	**Input2:** Enter num1: 86 Enter num2: 28 **Output:** num2 is smaller	**Input3:** Enter num1: 34 Enter num2: 92 **Output:** num1 is smaller

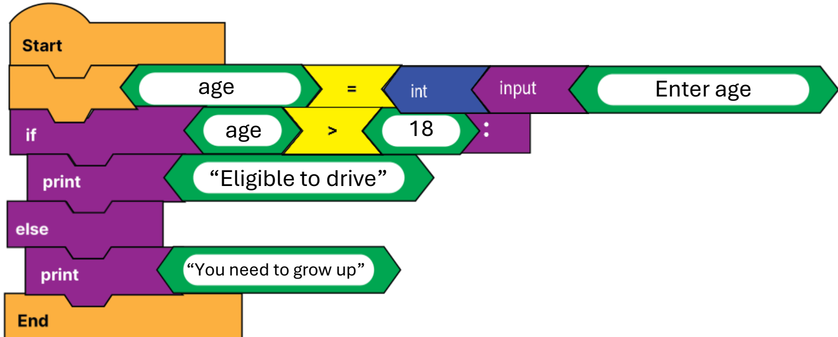
6. Build a Python Block Code to seek the marks and age of a student and check the given eligibility condition: age should be less than 25 and marks should be greater than 65. If the student meets both conditions, print 'Eligible'.



Input1: Enter age: 14 Enter marks: 73 **Output:** Eligible	**Input2:** Enter age: 26 Enter marks: 73 **Output:** Not Eligible	**Input3:** Enter age: 14 Enter marks: 54 **Output:** Not Eligible

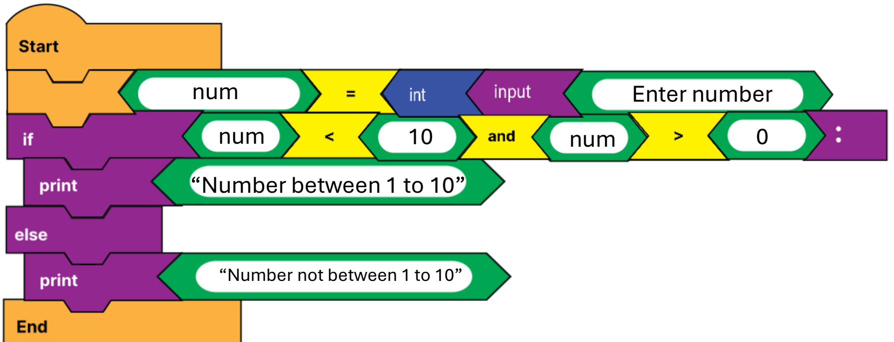
CEDURA © Copyrighted Content. All rights reserved

7. Build a Python Block Code to check whether a person is eligible to drive or not. (Condition: Age of 18 and above are eligible to drive)



Input1:	**Input2:**	**Input3:**
Enter age: 19	Enter age: 17	Enter age: 35
Output:	**Output:**	**Output:**
Eligible to drive	You need to grow up	Eligible to drive

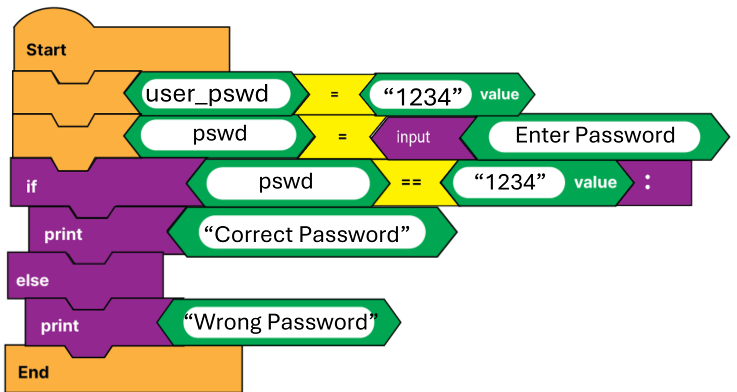
8. Build a Python Block Code to check whether a given number is between 1 and 9



Input1:	**Input2:**	**Input3:**
Enter number: 9	Enter number: 2	Enter number: 29
Output:	**Output:**	**Output:**
Number between 1 to 10	Number between 1 to 10	Number not between 1 to 10

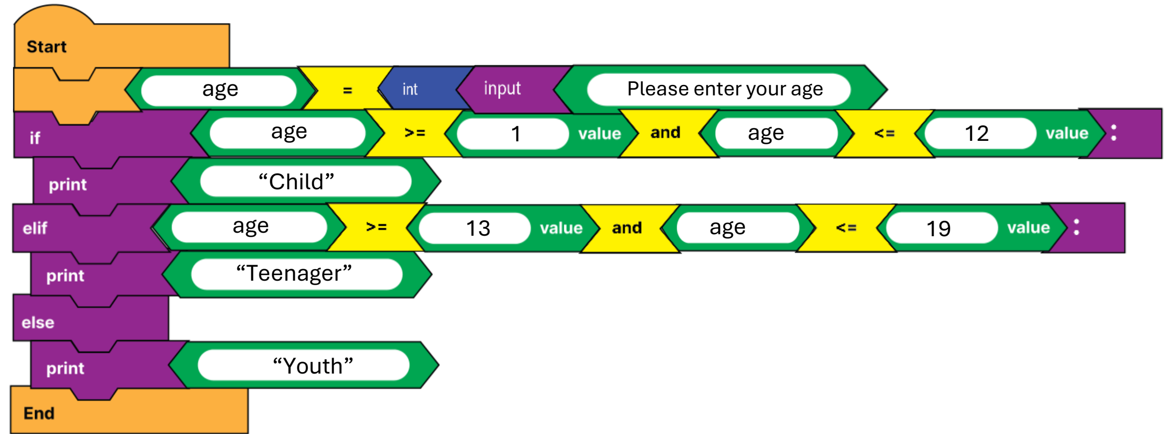
CEDURA © Copyrighted Content. All rights reserved

9. Build a Python Block Code to seek a password from the user and authenticate it. (Tip: You decide on the password and check it against the user input)



Input1:	**Input2:**	**Input3:**
Enter password: "1234"	Enter password: "124"	Enter password: "123"
Output:	**Output:**	**Output:**
Correct Password	Wrong Password	Wrong Password

10. Build a Python Block Code to check the age and display "Child", "Teenager" or "Youth". (Tip: age 1 to 12 - Child; 13 to 19 - Teenager; 20 and above - Youth)

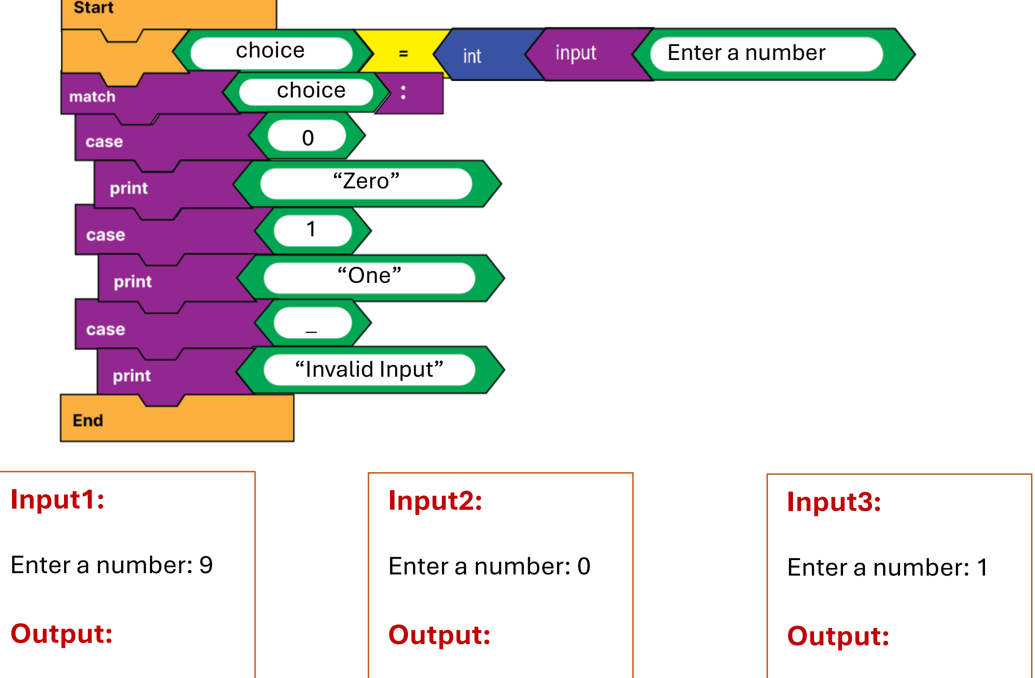


Input1:	**Input2:**	**Input3:**
Please enter your age: 9	Please enter your age: 15	Please enter your age: 23
Output:	**Output:**	**Output:**
Child	Teenager	Youth

CEDURA © Copyrighted Content. All rights reserved

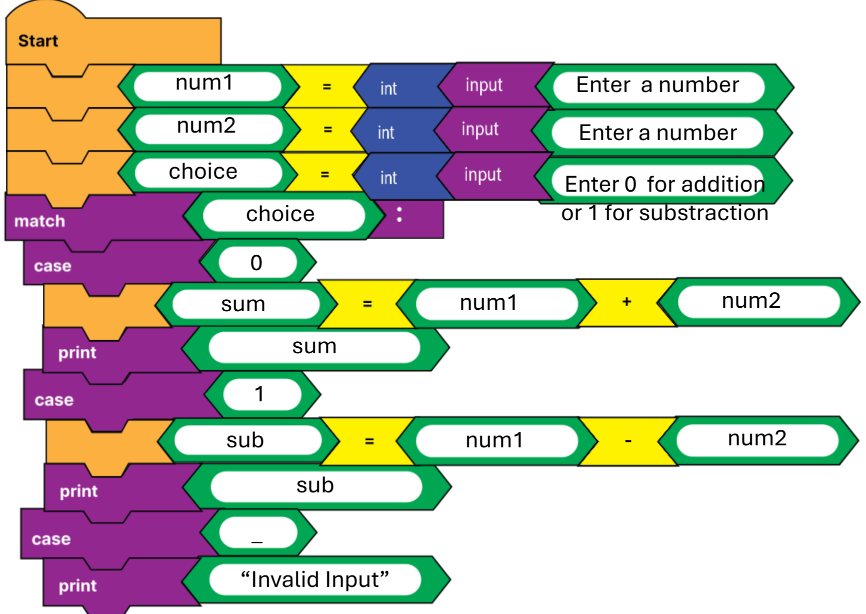
Match Case

1. Build a Python Block Code (using Match-Case) to seek input from a user (0 or 1) and print the text ('Zero' or 'One') form the input number. In case a user enter any other number, then print 'invalid input'



Input1:	**Input2:**	**Input3:**
Enter a number: 9	Enter a number: 0	Enter a number: 1
Output:	**Output:**	**Output:**
Invalid Input	Zero	One

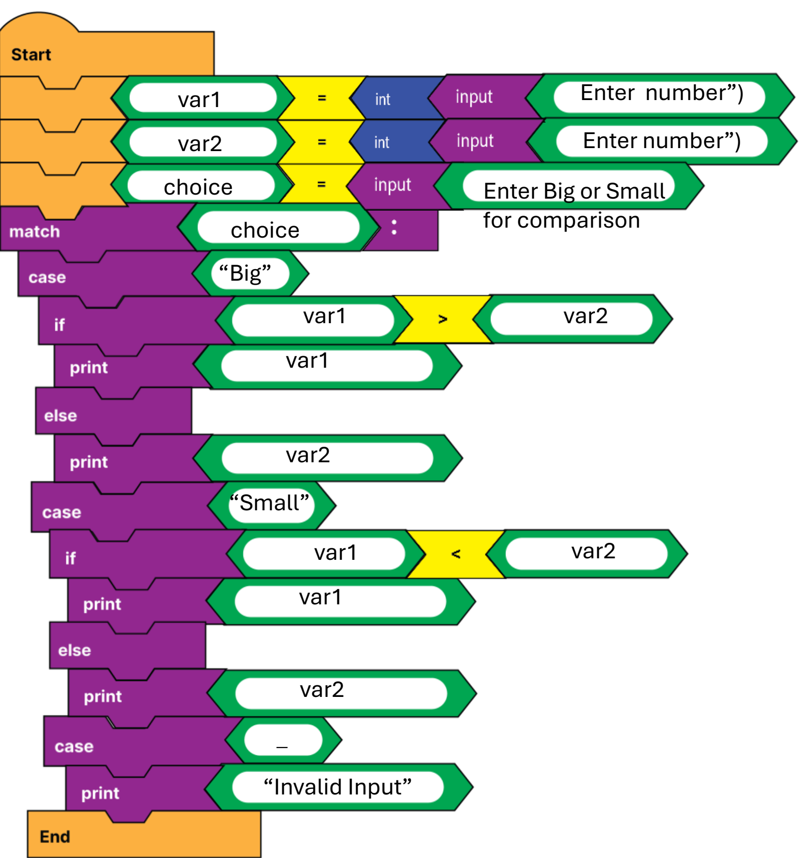
2. Build a Python Block Code (using Match-Case) to take input from the user (0 or 1) and perform the addition of two variables if the input is 0 and subtraction if the input is 1.



CEDURA © Copyrighted Content. All rights reserved

Input1: Enter a number: 9 Enter a number:10 Enter 0 for addition or 1 for substraction:0 **Output:** 19	**Input1:** Enter a number: 25 Enter a number:10 Enter 0 for addition or 1 for subtraction: 1 **Output:** 15

3. Build a Python Block Code (using Match-Case) to take input from the user (Big or Small) and compare two variables, var1 (6) and var2 (5). If the input is Big, check whether variable var1 is bigger than var2. If the input is small, check whether variable var1 is smaller than var2.

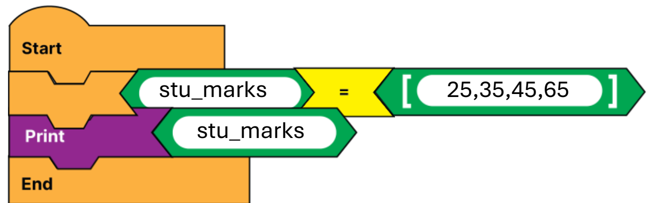


Input1: Enter number: 6 Enter number: 5 Enter Big or Small for comparison: Big **Output:** 6	**Input1:** Enter number: 14 Enter number: 25 Enter Big or Small for comparison: Small **Output:** 14

CEDURA © Copyrighted Content. All rights reserved

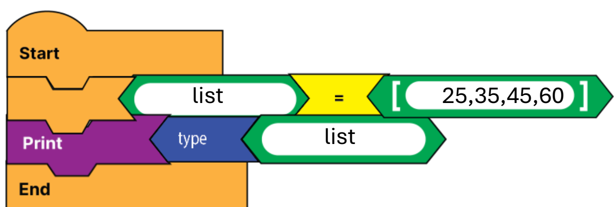
Sequences

1. Build a Python Block Code to create a LIST of student marks and print them.



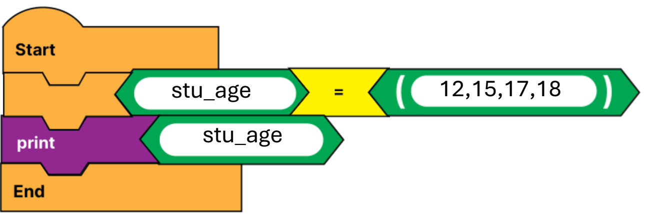
Input1:	Input2:	Input3:
stu_marks = [25,35,45,65]	stu_marks = [53,73,45,96]	stu_marks = [29,76,85,77]
Output:	**Output:**	**Output:**
[25,35,45,65]	[53,73,45,96]	[29,76,85,77]

2. Build a Python Block Code to create a LIST and print its data type.



Input1:	Input2:	Input3:
list = [53,73,45,96]	list = [29,76,85,77]	list = [25,35,45,65]
Output:	**Output:**	**Output:**
<class ‘list’>	<class ‘list’>	<class ‘list’>

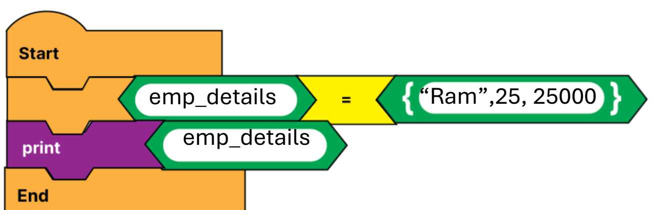
3. Build a Python Block Code to create a TUPLE of student ages and print them.



Input1:	Input2:	Input3:
stu_age = (12,15,17,18)	stu_age = (11,9,13,8)	stu_age = (10,19,14,15)
Output:	**Output:**	**Output:**
(12,15,17,18)	(11,9,13,8)	(10,19,14,15)

CEDURA © Copyrighted Content. All rights reserved

4. Build a Python Block Code to create a SET of Employee Details and print them.



Input1:

emp_details ={"Ram", 25, 25000}

Output:

{"Ram", 25, 25000}

Input2:

emp_details = {"Siva", 24, 13000}

Output:

{"Siva", 24, 13000}

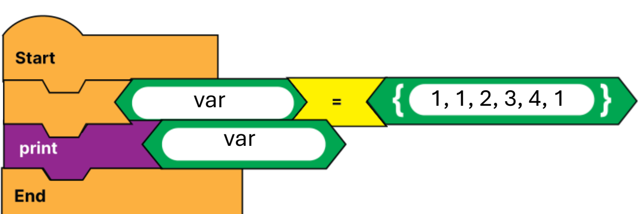
Input3:

emp_details ={"Priya", 28, 29000}

Output:

{"Priya", 28, 29000}

5. Build a Python Block Code to create a SET with duplicate items in it. What will be the output when you print it?



Input1:

var = {1, 1, 2, 3, 4, 1}

Output:

{1,2,3,4}

Input2:

var = {4,1, 2, 3, 4, 5, 1}

Output:

{1,2,3,4,5}

Input3:

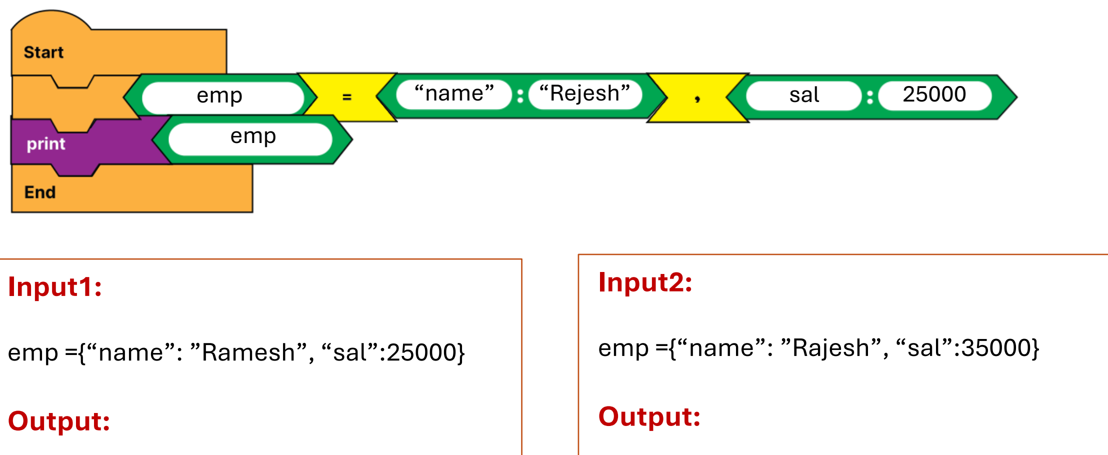
var = {1, 5, 2, 1, 4, 1}

Output:

{1,2,4, 5}

CEDURA © Copyrighted Content. All rights reserved

6. **Build a Python Block Code to create a DICTIONARY of Employee Details and print it.**



Input1:

emp ={“name”: ”Ramesh”, “sal”:25000}

Output:

{“name”: ”Ramesh”, “sal”:25000}

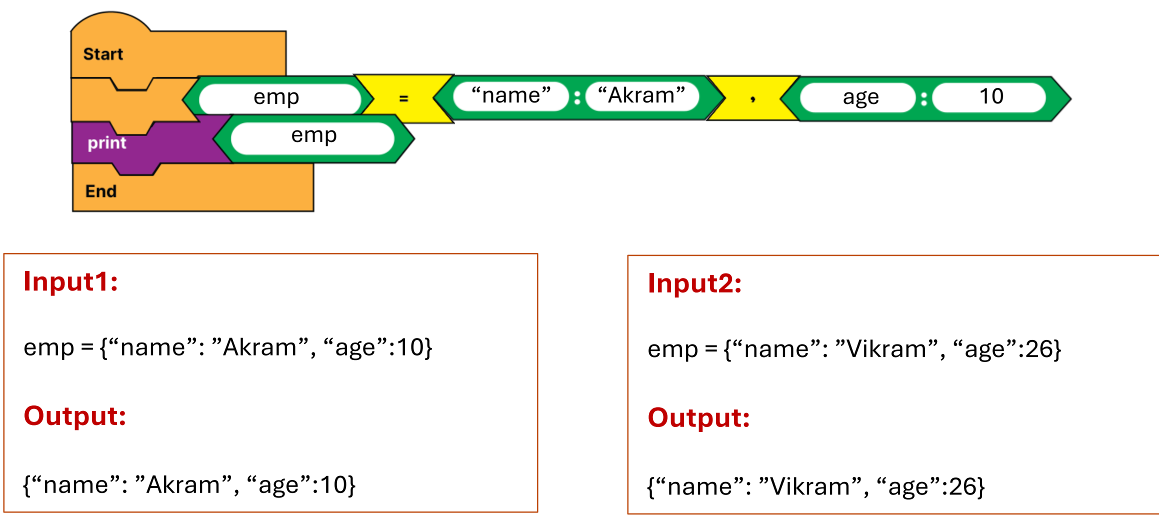
Input2:

emp ={“name”: ”Rajesh”, “sal”:35000}

Output:

{“name”: ”Rajesh”, “sal”:35000}

7. **Build a Python Block Code to create a DICTIONARY using keys (Name, age) and values (Akram, 10).**



Input1:

emp = {“name”: ”Akram”, “age”:10}

Output:

{“name”: ”Akram”, “age”:10}

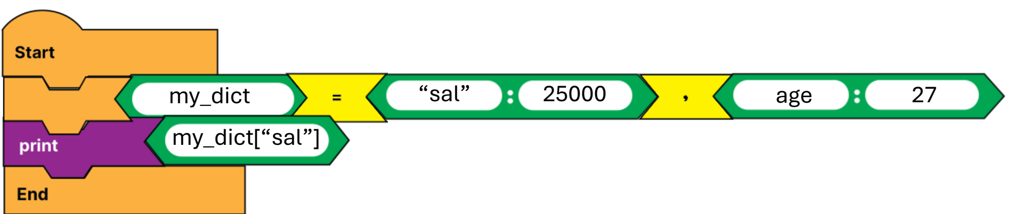
Input2:

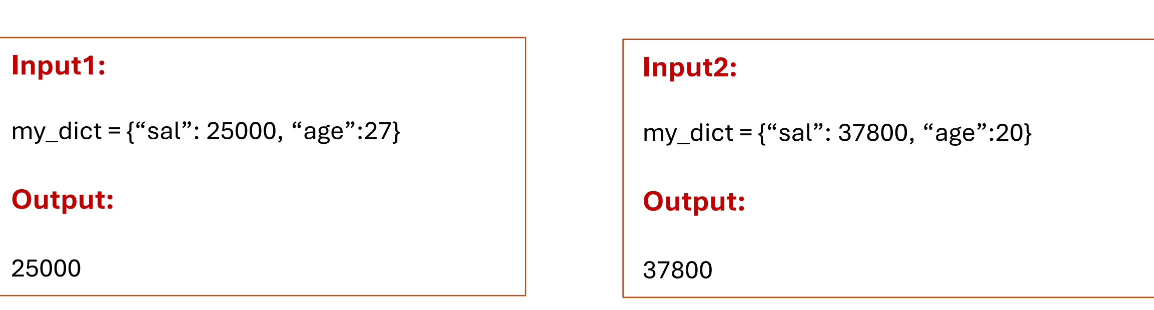
emp = {“name”: ”Vikram”, “age”:26}

Output:

{“name”: ”Vikram”, “age”:26}

8. **Build a Python Block Code to create a DICTIONARY of Employee Details and print any one value.**





Input1:

my_dict = {“sal”: 25000, “age”:27}

Output:

25000

Input2:

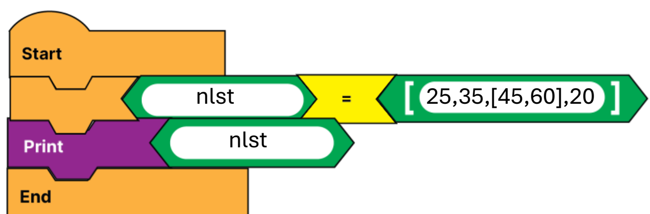
my_dict = {“sal”: 37800, “age”:20}

Output:

37800

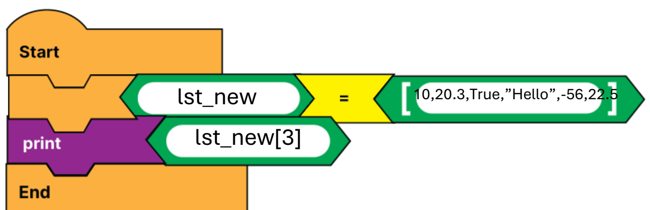
CEDURA © Copyrighted Content. All rights reserved

9. Build a Python Block Code to create a NESTED list.



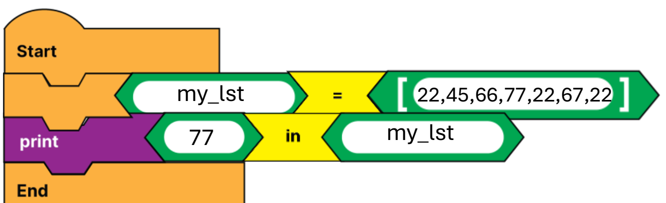
Input1:	**Input2:**	**Input3:**
nlst = [25,35,[45,60],20]	nlst = [45,55,[22,40],25]	nlst = [25, [45,60],35, 20]
Output:	**Output:**	**Output:**
[25,35,[45,60],20]	[45,55,[22,40],25]	[25, [45,60],35, 20]

10. Write a Python Block code to create a list based on the given values and print the 3rd Item in the list. (Hint: Consider 10, 20.3, True, "Hello", -56, 22.5 as values)



Input1:	**Input2:**
lst_new = [10,20.3,True,"Hello",-56,22.5]	lst_new = [10,"Hello",-56,22.5 20.3,True]
Output:	**Output:**
Hello	22.5

11. Write a Python Block code to create a list using the values 22, 45, 66, 77, 22, "78.67", and 22. After creating find out whether a value 77 is in the list or not.

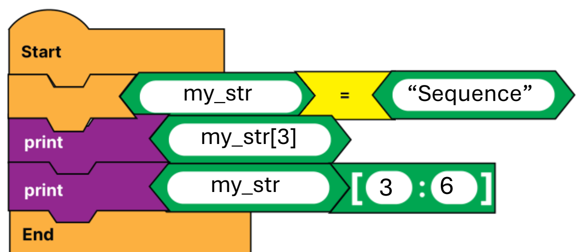


Input1:	**Input2:**	**Input3:**
my_lst = [22,45,66,77,22,67,22]	my_lst = [22,45,66,77,22,67,22]	my_lst = [22,45,66,77,22,67,22]
Output:	**Output:**	**Output:**
True	False	True

Note: Searching value: 77	Note: Searching value: 75	Note: Searching value: 22

CEDURA © Copyrighted Content. All rights reserved

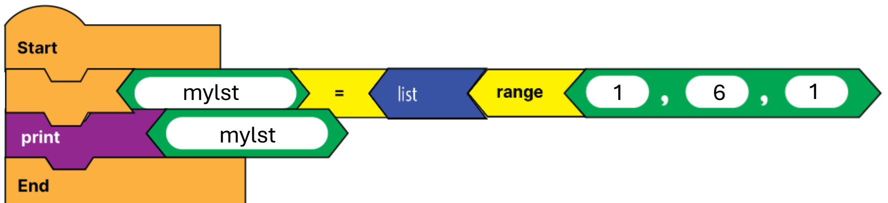
12. Write a Python Block code to assign a string “Sequence” to a variable and print the 4th character and the characters “uen”, individually.



Input1:	**Input2:**	**Input3:**
my_str = “Sequence”	my_str = “Slicing”	my_str = “Variable”
Output:	**Output:**	**Output:**
u uen	c cin	i iab

Range

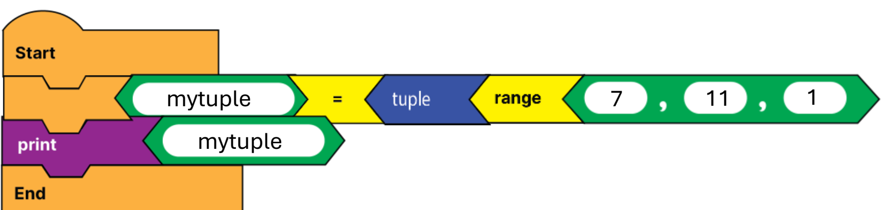
1. **Build a Python Block Code to print a list of numbers [1,2,3,4,5], using the Range function.**



Input1:	**Input2:**	**Input3:**
mylst = list(range(1,6,1))	mylst = list(range(1,5,2))	mylst = list(range(-4,-8,-1))
Output:	**Output:**	**Output:**
[1, 2, 3, 4, 5]	[1, 3]	[-4, -5, -6, -7]

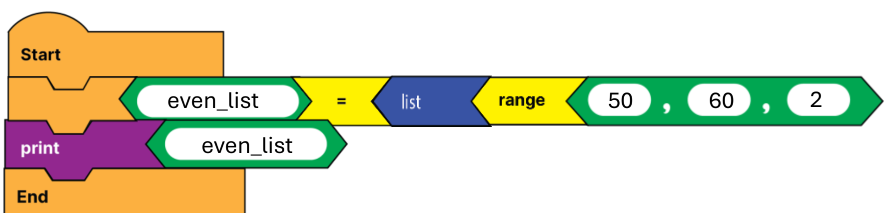
CEDURA © Copyrighted Content. All rights reserved

2. Build a Python Block Code to print a tuple of numbers (7,8,9,10), using the Range function.



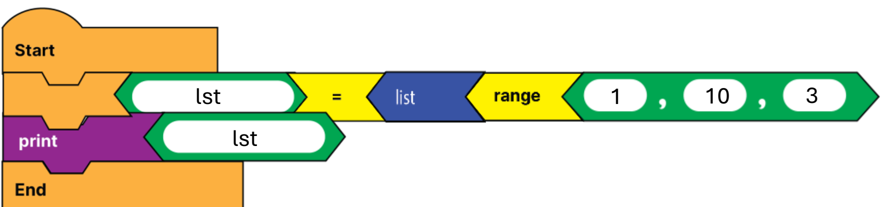
Input1:	**Input2:**	**Input3:**
mytuple= tuple(range(7, 11, 1))	mytuple= tuple(range(4, 10, 2))	mytuple= tuple(range(-3, -7, -1))
Output:	**Output:**	**Output:**
(7,8,9,10)	(4, 6, 8)	(-3, -4, -5, -6)

3. Build a Python Block Code to print even numbers from 51 to 60, using the Range function.



Input1:	**Input2:**	**Input3:**
even_list= list(range(50, 60, 2))	even_list = list(range(50, 40, -2))	even_list = list(range(8, 16, 2))
Output:	**Output:**	**Output:**
[50,52, 54, 56, 58]	[50, 48, 46, 44, 42]	[8, 10, 12, 14]

4. Build a Python Block Code to print three values only between 1 to 10, using the Range function. (Hint: Should print 1, 4, 7).

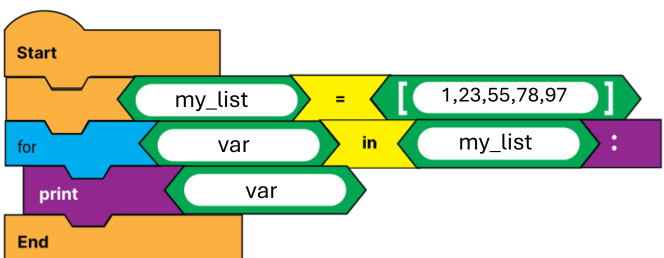


Input1:	**Input2:**	**Input3:**
lst= list(range(1, 10, 3))	lst= list(range(2, 10, 3))	lst= list(range(30, 21, -3))
Output:	**Output:**	**Output:**
[1, 4, 7]	[2, 5, 8]	[30, 27, 24]

CEDURA © Copyrighted Content. All rights reserved

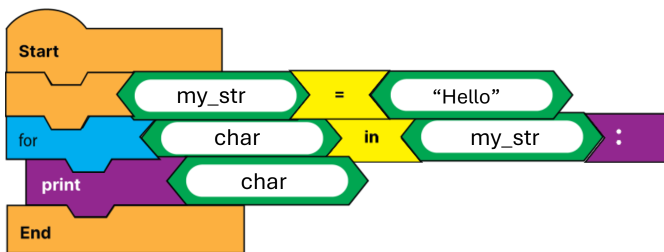
For Loops

1. **Build a Python Block Code to create a list of numbers and print them individually using the for loop.**



Input1: my_list= [1,23, 55, 78, 97] **Output:** 1 23 55 78 97	**Input2:** my_list= [27, 59, 45, 89, 31] **Output:** 27 59 45 89 31	**Input3:** my_list= [51,75, 65, 87, 67] **Output:** 51 75 65 87 67

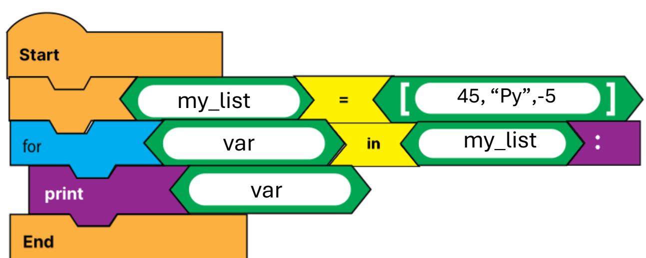
2. **Build a Python Block Code to demonstrate iteration over the string "Hello" using the for loop.**



Input1: my_str = "Hello" **Output:** H e l l o	**Input2:** my_str = "Welcome" **Output:** W e l c o m e	**Input3:** my_str = "String" **Output:** S t r i n g

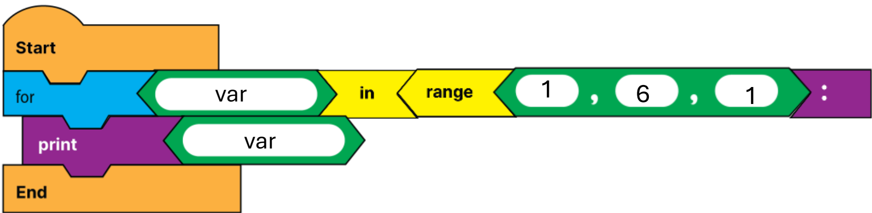
CEDURA © Copyrighted Content. All rights reserved

3. Build a Python Block Code to demonstrate iteration through the given sequence (45, ‘Py”, -5).



Input1:	**Input2:**	**Input3:**
my_list= [45, “Py”,-5]	my_list= [“Code”, 59, 6, “12”]	my_list= [22.5, -9, “Hii”]
Output:	**Output:**	**Output:**
45 Py -5	Code 59 6 12	22.5 -9 Hii

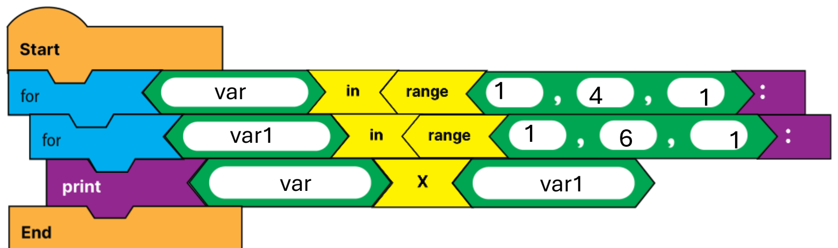
4. Build a Python Block Code to print 1 to 5 numbers using the Range function and For loop.



Input1:	**Input1:**	**Input1:**
start=1 , stop= 6, step= 1	start=2 , stop= 9, step= 2	start=10 , stop= 1, step= -2
Output:	**Output:**	**Output:**
1 2 3 4 5	2 4 6 8	10 8 6 4 2

CEDURA © Copyrighted Content. All rights reserved

5. **Buil da Python Block Code to print 3 tables up to 5 using the for loop.**



Input1:

start=1 , stop= 4, step= 1
start= 1, stop=6, step=1

Output:

1	2	3
2	4	6
3	6	9
4	8	12
5	10	15

Input2:

start=2 , stop= 5, step= 1
start= 5, stop=0, step=-1

Output:

10	15	20
8	12	16
6	9	12
4	6	8
2	3	4

Input3:

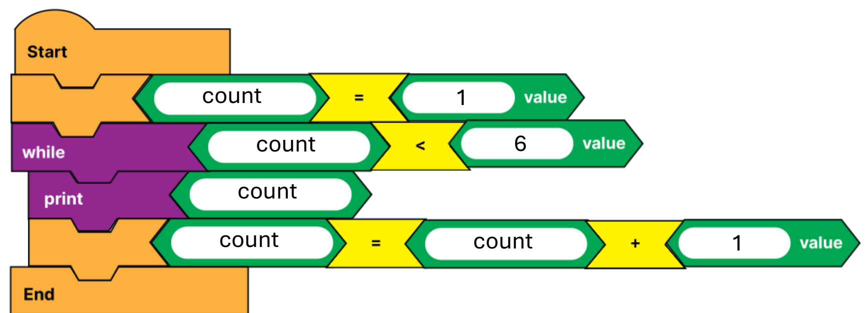
start=4 , stop= 7, step= 1
start= 1, stop=6, step=1

Output:

4	5	6
8	10	12
12	15	18
16	20	24
20	25	30

While Loops

1. **Build a Python Code Block to print numbers from 1 to 5 using the while loop.**



Input1:

count = 1
count < 6

Output:
1
2
3
4
5

Input2 (Additional - Try it)

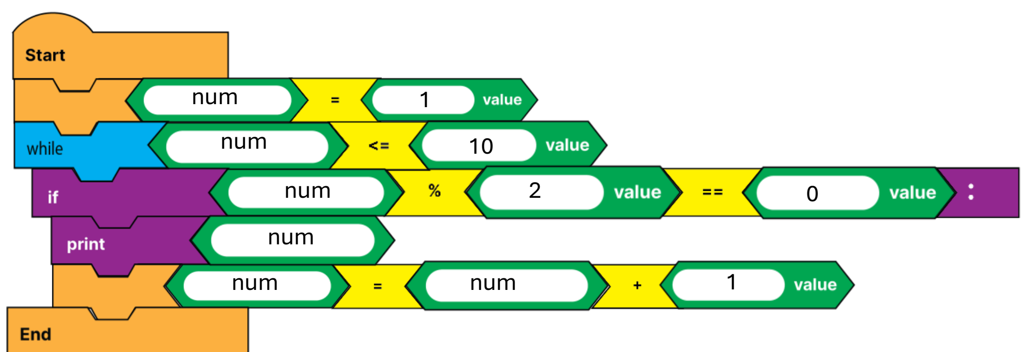
count = 0
count < 5

Output:
0
1
2
3
4

* Generates 5 numbers starting with 0

CEDURA © Copyrighted Content. All rights reserved

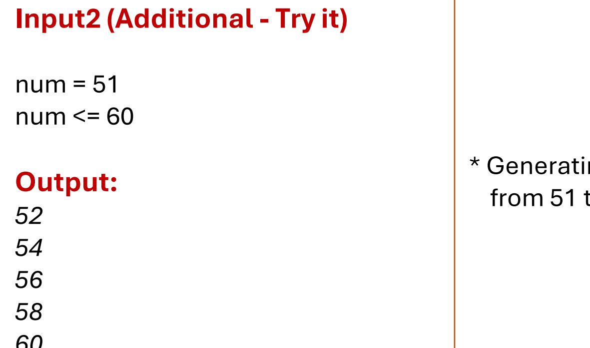
2. **Build a Python Code Block to print EVEN numbers from 1 to 10 using the while loop.**



Input1:

num = 1
num <= 10

Output:
2
4
6
8
10



Input2 (Additional - Try it)

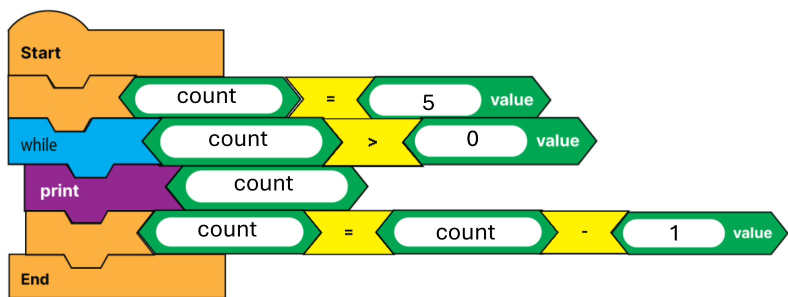
num = 51
num <= 60

Output:
52
54
56
58
60

* Generating EVEN numbers from 51 to 60

3. **Build a Python Code Block to print numbers from 5 to 1 in reverse order using the while loop.**

.



Input1:

count = 5
count > 0

Output:
5
4
3
2
4
1

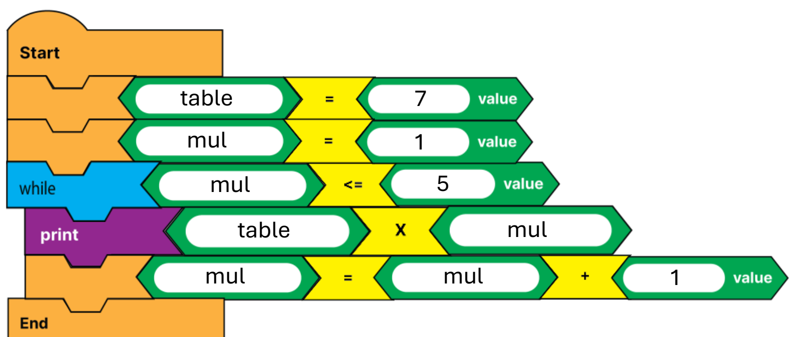
Input2 (Additional - Try it)
Printing 6 numbers in reverse order starting at 26

count = 26
count > 20

Output:
26
25
24
23
22
21

CEDURA © Copyrighted Content. All rights reserved

4. Build a Python Code Block to print 7 table up to 5 using the while loop.



Input1:

table = 7
mul = 1
mul <= 5

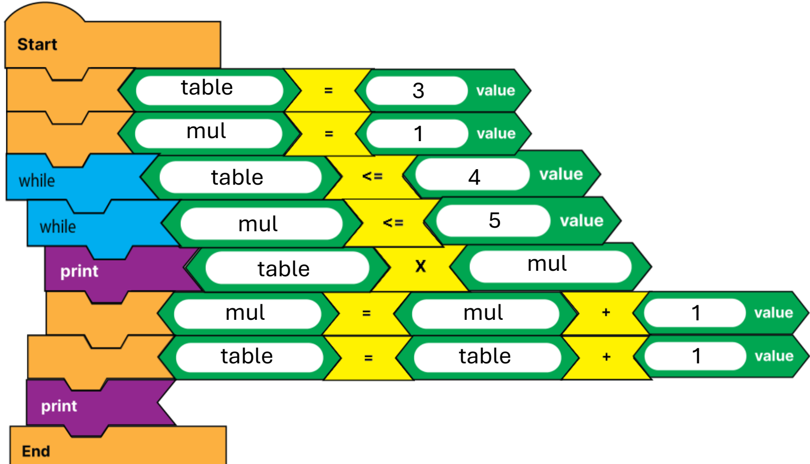
Output:
7
14
21
28
35

Input2 (Additional - Try it)
Printing 6 table from 5 to 10.

table = 6
mul = 5
mul <= 10

Output:
30
36
42
48
54
60

5. Build a Python Code Block to print both 3 and 4 tables up to 4 using nested while loop.



CEDURA © Copyrighted Content. All rights reserved

Input1:

table = 3
mul = 1
table <= 4
mul <= 5

Output:
3
6
9
12
15

4
8
12
16
20

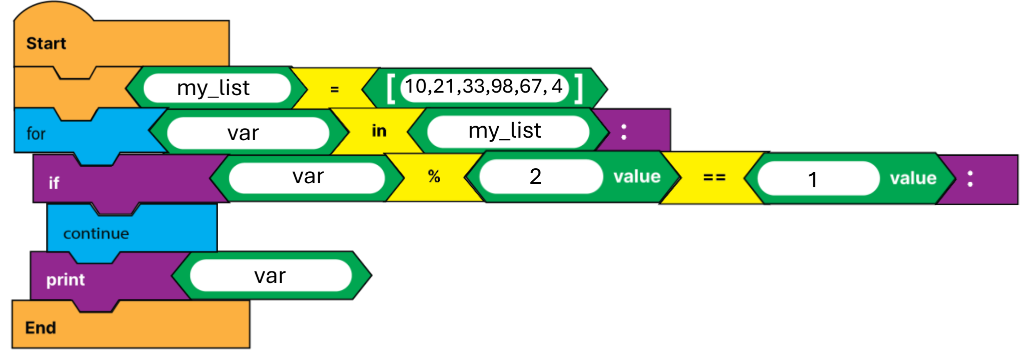
Input2 (Additional - Try it)
Printing 6 & 7 tables from 5 to 10.

table = 6
mul = 5
table <= 4
mul <= 10

Output:
30
36
42
48
60

35
42
49
56
63
70

6. Build a Python Code Block to print all the EVEN numbers from the give list [10,21,33,98,67,4] using for loop and continue statement.



Input1:

my_list = [10,21,33,98,67, 4]
if var % 2 == 1:

Output:
10
98
4

Input2 (Additional - Try it)
Print the ODD numbers from the list

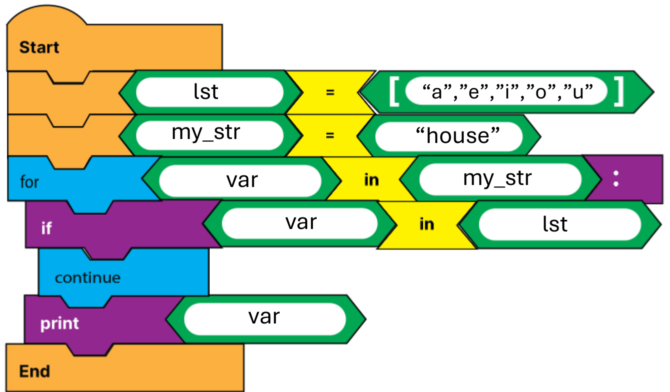
my_list = [10,21,33,98,67, 4]
if var % 2 == 0:

Output:
21
33
67

- *Logic is to find the even numbers. When an even number is found, the continue statement will be executed and the iteration skipped.*

CEDURA © Copyrighted Content. All rights reserved

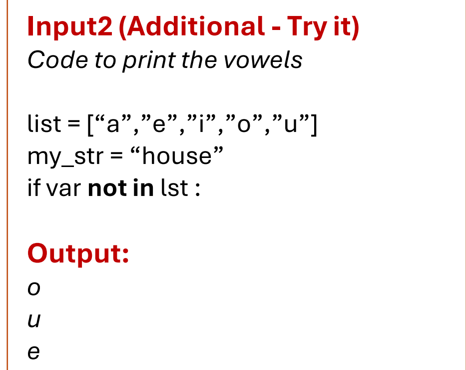
7. Build a Python Code Block to print NON-VOWELS (not a vowel) characters in the given string ‘House’ using the for loop and continue statement.



Input1:

list = [“a”,”e”,”i”,”o”,”u”]
my_str = “house”
if var **in** lst :

Output:
h
u
s

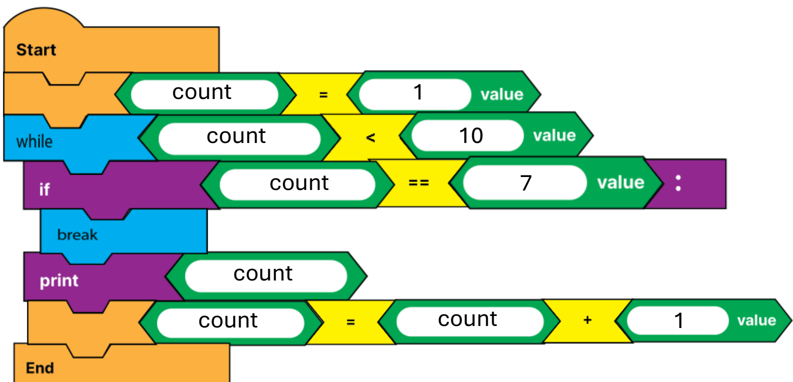


Input2 (Additional - Try it)
Code to print the vowels

list = [“a”,”e”,”i”,”o”,”u”]
my_str = “house”
if var **not in** lst :

Output:
o
u
e

8. Build a Python Code Block to print numbers from 1 to 10 but break at 7, using the while loop and break statement,



CEDURA © Copyrighted Content. All rights reserved

Input1:

count = 1
count < 10
if count == 7:

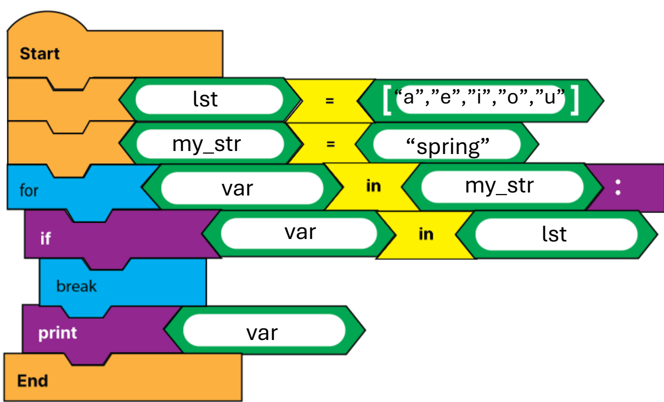
Output:
1
2
3
4
5
6

Input2 (Additional - Try it)
Code to break at 5

count = 1
count < 10
if count == 7:

Output:
1
2
3
4

9. Build a Python Code Block to break the loop if any vowel is found in the given string "Spring" using the for loop and



Input1:

lst = ["a","e","i","o","u"}
my_str - "spring"

Output:

s
p

Input2:

lst = ["a","e","i","o","u"}
my_str - "python"

Output:

P
y
t
h

Input3:

lst = ["a","e","i","o","u"}
my_str - "education"

Output:

Note: *No output as the starting character is a Vowel. Hence, the loop breaks in the first iteration.*

CEDURA © Copyrighted Content. All rights reserved

NOTES

CEDURA © Copyrighted Content. All rights reserved

PYTHON

Block Coding

hello@schoolforai.com
+91 99492 96431

Copyright CEDURA