UNIT - I

Prepared by: Barad Bhavna

Agenda

No.	**Topics**
1.	Features of Python
2.	Structure of Python Program
3.	Elements of Python
4.	Python Interpreter
5.	Python Shell
6.	Indentation
7.	Strongly typed features
8.	Basic Datatypes
9.	Variables, Expression
10.	Flow of Execution
11.	Input/output Statements
12.	Atoms, Identifiers and Keywords, Literals, String
	Operators
13.	Arithmetic Operator
14.	Relational Operator
15.	Logical or Boolean Operator
16.	Assignment Operator
17.	Ternary, Bit-wise and Increment/Decrement Operator
	Control Statements
18.	If If….Else Else If Ladder Match….Case(Switch Statement)
	Loop Controls
19.	While Loop For Loop Rang Function Break Continue Pass Statements with Loop

❖ Introduction of Python

- **Python** is a very popular general-purpose interpreted, interactive, object-oriented, and high-level programming language.
- Python is dynamically-typed and garbage-collected programming language.
- It was created by Guido van Rossum during 1991. Like Perl, Python source code is also available under the GNU General Public License (GPL).
- Python supports multiple programming paradigms, including Procedural, Object Oriented and Functional programming language.
- Python design philosophy emphasizes code readability with the use of significant indentation.
- Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming.

- **It is used for:**
  - web development (server-side),
  - software development,
  - mathematics,
  - system scripting.

✓ What can Python do?

- Python can be used on a server to create web applications.
- Python can be used alongside software to create workflows.
- Python can connect to database systems. It can also read and modify files.
- Python can be used to handle big data and perform complex mathematics.
- Python can be used for rapid prototyping, or for production-ready software development.

✓ Why Python?

- Python works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc).
- Python has a simple syntax similar to the English language.
- Python has syntax that allows developers to write programs with fewer lines than some other programming languages.
- Python runs on an interpreter system, meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.
- Python can be treated in a procedural way, an object-oriented way or a functional way.

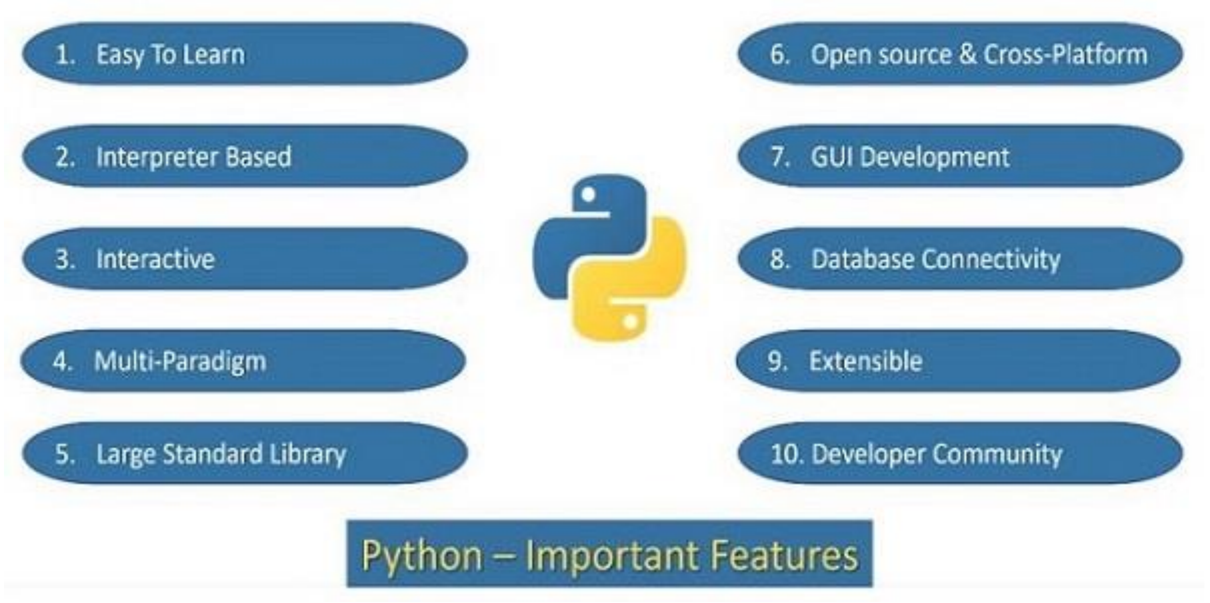
✓ Characteristics of Python

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

✓ Applications of Python

- Python is a general purpose programming language known for its readability. It is widely applied in various fields.
- In **Data Science**, Python libraries like **Numpy**, **Pandas**, and **Matplotlib** are used for data analysis and visualization.
- Python frameworks like **Django**, and **Pyramid**, make the development and deployment of Web Applications easy.
- This programming language also extends its applications to **computer vision** and image processing.
- It is also favored in many tasks like **Automation**, Job Scheduling, GUI development, etc.

❖ Features of Python



- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.

- **Easy-to-maintain** − Python's source code is fairly easy-to-maintaining.

- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

- **Interactive Mode** − Python has support for an interactive mode that allows interactive testing and debugging of snippets of code.

- **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

- **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

- **Databases** − Python provides interfaces to all major commercial databases.

- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries, and Windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

- **Standard Library -** Even though it has a very few keywords (only Thirty-Five), Python software is distributed with a standard library made of large number of modules and packages. Thus Python has out of box support for programming needs such as serialization, data compression, internet data handling, and many more. Python is known for its batteries included approach.

Some of the Python's popular modules are:

- ✓ NumPy
- ✓ Pandas
- ✓ Matplotlib
- ✓ Tkinter
- ✓ Math

- **Dynamically Typed -** Python is a dynamically typed programming language. In Python, you don't need to specify the variable time at the time of the variable declaration. The types are specified at the runtime based on the assigned value due to its dynamically typed feature.

- **Interpreter Based -** Instructions in any programming languages must be translated into machine code for the processor to execute them. Programming languages are either compiler-based or interpreter based.

In case of a compiler, a machine language version of the entire source program is generated. The conversion fails even if there is a single erroneous statement. Hence, the development process is tedious for the beginners. The C family languages (including C, C++, Java, C# etc) are compiler based. Python is an interpreter based language.

- **Interactive -** Standard Python distribution comes with an interactive shell that works on the principle of REPL (Read – Evaluate – Print – Loop). The shell presents a Python prompt >>>. You can type any valid Python expression and press Enter. Python interpreter immediately returns the response and the prompt comes back to read the next expression.

  Example: 2*3+1 O/P: 7

  Example: print ("Hello World") O/P: Hello World

- **Multi-paradigm -** Python is a completely object-oriented language. Everything in a Python program is an object. However, Python conveniently encapsulates its object orientation to be used as an imperative or procedural language – such as C. Python also provides certain functionality that resembles functional programming. Moreover, certain third-party tools have been developed to support other programming paradigms such as aspect-oriented and logic programming.

❖ Structure of Python Program

- ✓ **Steps:**

  Step 1: Install Python. Make sure that Python is installed on your system or not. If Python is not installed, then install it from here: https://www.python.org/downloads/

  Step 2: Choose Text Editor or IDE to write the code.

  Step 3: Open Text Editor or IDE, create a new file, and write the code to print Hello World.

  Step 4: Save the file with a file name and extension ".py".

  Step 5: Compile/Run the program.

- ✓ **Python Program to Print Hello World:**

```
# Python code to print "Hello World"

print ("Hello World")
```

  In the above code, we wrote two lines. The first line is the Python comment that will be ignored by the Python interpreter, and the second line is the **print()** statement that will print the given message ("Hello World") on the output screen.

- ✓ **Output**

```
Hello World
```

In Command Prompt

```
C:\Users\MAHESH>python

Python 3.12.7 (tags/v3.12.7:0b05ead, Oct  1 2024, 03:06:41) [MSC v.1941
64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license" for more information.
```

❖ Elements of Python

The elements of a Python program refer to the basic building blocks or components that make up a Python program. Understanding these elements is crucial for writing effective and efficient Python code.

1. **Variables**

Variables store data values. In Python, you don't need to declare the type of a variable beforehand; Python automatically detects the type.

Python variables are the reserved memory locations used to store values with in a Python Program. This means that when you create a variable you reserve some space in the memory.

Example:

```
x = 5  # integer
name = "Alice"  # string
pi = 3.14  # float
is_valid = True  # Boolean
```

- A variable name must start with a letter or the underscore character.
- A variable name cannot start with a number or any special character like $, (, * % etc.
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Python variable names are case-sensitive which means Name and NAME are two different variables in Python.
- Python reserved keywords cannot be used naming the variable.

If the name of variable contains multiple words, we should use these naming patterns −

- **Camel case** − First letter is a lowercase, but first letter of each subsequent word is in uppercase. For example: kmPerHour, pricePerLitre

- **Pascal case** − First letter of each word is in uppercase. For example: KmPerHour, PricePerLitre

- **Snake case** − Use single underscore (_) character to separate words. For example: km_per_hour, price_per_litre

2. **Data Types**

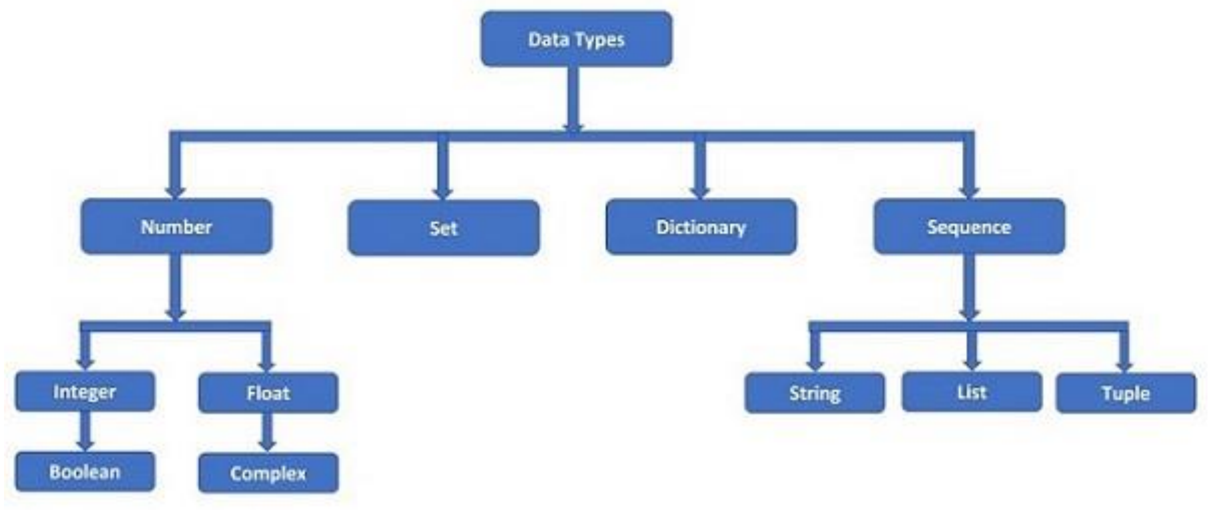
Python has the following data types built-in by default, in these categories:

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview
None Type:	NoneType

Python data types are actually classes, and the defined variables are their instances or objects. Since Python is dynamically typed, the data type of a variable is determined at runtime based on the assigned value.

In general, the data types are used to define the type of a variable. It represents the type of data we are going to store in a variable and determines what operations can be done on it.

Each programming language has its own classification of data items. With these datatypes, we can store different types of data values.



1. Numeric Data Types

Python numeric data types store numeric values. Number objects are created when you assign a value to them.

Example:

```
var1 = 1       # int data type

var2 = True    # bool data type

var3 = 10.023  # float data type

var4 = 10+3j   # complex data type
```

2. String Data Type

Python string is a sequence of one or more Unicode characters, enclosed in single, double or triple quotation marks (also called inverted commas).

Python strings are immutable which means when you perform an operation on strings, you always produce a new string object of the same type, rather than mutating an existing string.

Example:

```
>>> 'TutorialsPoint'

'TutorialsPoint'

>>> "TutorialsPoint"

'TutorialsPoint'

>>> '''TutorialsPoint'''

'TutorialsPoint'
```

3. Sequence Data Types

Sequence is a collection data type. It is an ordered collection of items. Items in the sequence have a positional index starting with 0.

It is conceptually similar to an array in C or C++. There are following three sequence data types defined in Python.

- List Data Type
- Tuple Data Type
- Range Data Type

(a) List Data Type

Python Lists are the most versatile compound data types. A Python list contains items separated by commas and enclosed within square brackets ([]).

To some extent, Python lists are similar to arrays in C.

One difference between them is that all the items belonging to a Python list can be of different data type where as C array can store elements related to a particular data type.

Example: [2023, "Python", 3.11, 5+6j, 1.23E-4]

A list in Python is an object of list class. We can check it with **type()** function.

type([2023, "Python", 3.11, 5+6j, 1.23E-4])

<class 'list'>

As mentioned, an item in the list may be of any data type. It means that a list object can also be an item in another list. In that case, it becomes a nested list.

[['One', 'Two', 'Three'], [1,2,3], [1.0, 2.0, 3.0]]

A list can have items which are simple numbers, strings, tuple, dictionary, set or object of user defined class also.

The values stored in a Python list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.

The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator. A list in Python is an object of list class. We can check it with **type()** function.

Example: type([2023, "Python", 3.11, 5+6j, 1.23E-4])

<class 'list'>

As mentioned, an item in the list may be of any data type.

It means that a list object can also be an item in another list. In that case, it becomes a nested list.

Example: [['One', 'Two', 'Three'], [1,2,3], [1.0, 2.0, 3.0]]

A list can have items which are simple numbers, strings, tuple, dictionary, set or object of user defined class also. A list in Python is an object of list class. We can check it with type() function.

Example: type([2023, "Python", 3.11, 5+6j, 1.23E-4])

<class 'list'>

As mentioned, an item in the list may be of any data type. It means that a list object can also be an item in another list. In that case, it becomes a nested list.

Example: [['One', 'Two', 'Three'], [1,2,3], [1.0, 2.0, 3.0]]

A list can have items which are simple numbers, strings, tuple, dictionary, set or object of user defined class also.

(b) Tuple Data Type

Python tuple is another sequence data type that is similar to a list. A Python tuple consists of a number of values separated by commas. Tuples are enclosed within parentheses (...).

A tuple is also a sequence, hence each item in the tuple has an index referring to its position in the collection. The index starts from **0**.

Example: (2023, "Python", 3.11, 5+6j, 1.23E-4)

In Python, a tuple is an object of **tuple** class. We can check it with the type() function.

Example: type((2023, "Python", 3.11, 5+6j, 1.23E-4))

<class 'tuple'>

(c) Range Data Type

A Python range is an immutable sequence of numbers which is typically used to iterate through a specific number of items.

It is represented by the **Range** class. The constructor of this class accepts a sequence of numbers starting from 0 and increments to 1 until it reaches a specified number. Following is the syntax of the function –

Syntax: range(start, stop, step)

Example: for i in range(5):

```
  print(i)
```

Here is the description of the parameters used −

- **start**: Integer number to specify starting position, (Its optional, Default: 0)
- **stop**: Integer number to specify ending position (It's mandatory)
- **step**: Integer number to specify increment, (Its optional, Default: 1)

4. Binary Data Types

A binary data type in Python is a way to represent data as a series of binary digits, which are 0's and 1's.

It is like a special language computers understand to store and process information efficiently.

This type of data is commonly used when dealing with things like files, images, or anything that can be represented using just two possible values.

So, instead of using regular numbers or letters, binary sequence data types use a combination of 0s and 1s to represent information.

5. Python Dictionary Data Type

Python dictionaries are kind of hash table type.

A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.

Python dictionary is like associative arrays or hashes found in Perl and consist of **key:value** pairs.

The pairs are separated by comma and put inside curly brackets {}. To establish mapping between key and value, the semicolon':' symbol is put between the two.

Example: {1:'one', 2:'two', 3:'three'}

6. Python Set Data Type

Set is a Python implementation of set as defined in Mathematics.

A set in Python is a collection, but is not an indexed or ordered collection as string, list or tuple.

An object cannot appear more than once in a set, whereas in List and Tuple, same object can appear more than once.

Comma separated items in a set are put inside curly brackets or braces {}. Items in the set collection can be of different data types.

Example: {2023, "Python", 3.11, 5+6j, 1.23E-4}

{(5+6j), 3.11, 0.000123, 'Python', 2023}

Note that items in the set collection may not follow the same order in which they are entered.

The position of items is optimized by Python to perform operations over set as defined in mathematics.

Python's Set is an object of built-in set class, as can be checked with the **type()** function.

Example: type({2023, "Python", 3.11, 5+6j, 1.23E-4})

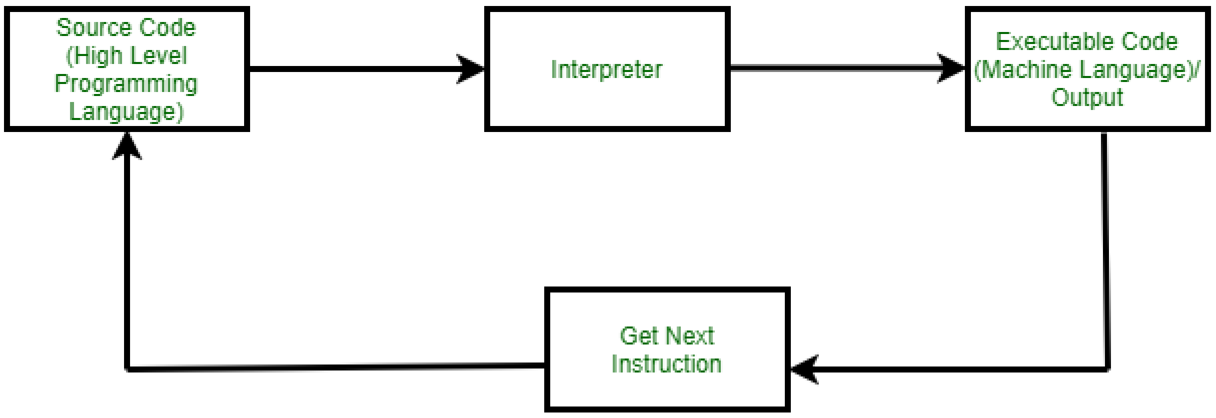
<class 'set'>

❖ Python Interpreter

As we all know Python is one of the most high-level languages used today because of its massive versatility and portable library & framework features. It is an interpreted language because it executes line-by-line instructions. There are actually two way to execute python code one is in Interactive mode and another thing is having Python prompts which is also called script mode. Python does not convert high level code into low level code as many other programming languages do rather it will scan the entire code into something called bytecode.

Interpreted Languages: Perl, BASIC, Python, JavaScript, Ruby, PHP. Compiled Languages: C, C++, C#, COBOL and CLEO.

What are Interpreters?
Interpreters are the computer program that will convert the source code or an high level language into intermediate code (machine level language). It is also called translator in programming terminology. Interpreters executes each line of statements slowly. This process is called Interpretation. For example, Python is an interpreted language, PHP, Ruby, and JavaScript.


Source Code (High Level Programming Language)
Interpreter
Executable Code (Machine Language)/ Output
Get Next Instruction

Difference between Compilers and Interpreters

Compiler	Interpreter
It translates a program on a single run	It executes a program line by line.
It is an Fast Process	It is an Slow Process
It generates output in the form of .(exe)	It does not generate any form of outputs.
It utilizes CPU more.	It takes less utilization of CPU
It is used in Production environment	It is maximum used in Programming and development environment.
C, C++, C# are the compiled languages	Python, Ruby, PHP are the interpreted languages.
It takes lot of time to analyze the code structure	It takes less time compared to compilers.

❖Python Indentation

Indentation is a critical part of the syntax. It is used to define blocks of code that belong together. Unlike many other programming languages that use braces {} to delimit blocks, Python uses indentation to group statements.

Whitespace is used for indentation in Python. Unlike many other programming languages which only serve to make the code easier to read, Python indentation is mandatory.

One can understand it better by looking at an example of indentation in Python.

Role of Indentation in Python

A block is a combination of all these statements. Block can be regarded as the grouping of statements for a specific purpose.

Most programming languages like C, C++, and Java use braces { } to define a block of code for indentation.

One of the distinctive roles of Python is its use of indentation to highlight the blocks of code.

All statements with the same distance to the right belong to the same block of code. If a block has to be more deeply nested, it is simply indented further to the right.

Example:

```
# Python indentation

site = 'gfg'

if site == 'gfg':
    print('Logging on to geeksforgeeks...')
else:
    print('retype the URL.')
print('All set !')
```

❖ Strongly Typed Features

Python is both a strongly typed and dynamically typed language:

✓ Strong typing

strong typing refers to the concept that variables have a specific type, and the language enforces type correctness in operations. This means that once a variable is assigned a particular type, it cannot be implicitly coerced into another type (unless explicitly converted). This is a characteristic of strongly-typed languages.

Example:

```
a = 10              # Integer
b = "5"             # String
c = a + b
        # TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

✓ Dynamic typing

The type of a variable is determined only during runtime. This allows for flexibility in programming, but can impact performance.

Python is also **dynamically typed**, meaning that the type of a variable is determined at runtime, and you don’t have to explicitly declare the type when you create a variable.

Despite being dynamically typed, Python is considered a strongly typed language because it doesn't allow you to perform operations between incompatible types without explicit type conversion.

❖ Expressions

An expression is a combination of operators and operands that is interpreted to produce some other value.

In any programming language, an expression is evaluated as per the precedence of its operators.

So that if there is more than one operator in an expression, their precedence decides which operation will be performed first. We have many different types of expressions in Python,

1. Constant Expressions: These are the expressions that have constant values only.

Example: # Constant Expressions

```
x = 15 + 1.3
print(x)    Output:16.3
```

2. Arithmetic Expressions: An arithmetic expression is a combination of numeric values, operators, and sometimes parenthesis.

The result of this type of expression is also a numeric value.

The operators used in these expressions are arithmetic operators like addition, subtraction, etc.

Example: # Arithmetic Expressions

```
x = 5
y = 10
 add = x + y
sub = x - y
pro = x * y
div = x / y

print(add)
print(sub)
print(pro)
print(div)
```

3. Integral Expressions: These are the kind of expressions that produce only integer results after all computations and type conversions.

Example:
```
# Integral Expressions
a = 13
b = 12.0

c = a + int(b)
print(c)
```

4. Floating Expressions: These are the kind of expressions which produce floating point numbers as result after all computations and type conversions.

Example:
```
# Floating Expressions

a = 13
b = 5

c = a / b
print(c)
```

5. Relational Expressions: In these types of expressions, arithmetic expressions are written on both sides of relational operator (> , < , >= , <=).

Those arithmetic expressions are evaluated first, and then compared as per relational operator and produce a boolean output in the end.

These expressions are also called *Boolean expressions*.

Example:
```
# Relational Expressions
a = 21
b = 13
c = 40
d = 37

p = (a + b) >= (c - d)
print(p)
```

❖ Comments in Python

- Python comments start with the hash symbol # and continue to the end of the line.
- Comments in Python are useful information that the developers provide to make the reader understand the source code.
- It explains the logic or a part of it used in the code.
- Comments in Python are usually helpful to someone maintaining or enhancing your code when you are no longer around to answer questions about it.

Types of comments in Python

- ✓ Single-line comment in Python
- ✓ Multiline comment in Python

Example:

```
# This is a comment
# This is a single line comment
# This is multi line comment

'''
This is a multi-line comment using triple-quoted strings.
It can span multiple lines and is often used for doc-strings.
'''
```

❖Python Statement

A **Python statement** is an instruction that the Python interpreter can execute.

There are different types of **statements in Python** language as Assignment statements, Conditional statements, Looping statements, etc.

The token character NEWLINE is used to end a statement in Python.

It signifies that each line of a Python script contains a statement. These all help the user to get the required output.

Types of statements in Python?

- Multi-Line Statements
- Python Conditional and Loop Statements
    - Python If-else
    - Python for loop
    - Python while loop
    - Python try-except
    - Python with statement
- Python Expression statements
    - Python pass statement
    - Python del statement
    - Python return statement
    - Python import statement
    - Python continue and
    - Python break statement

❖ Execution of Python Program

✓ Compilation

The program is converted into **byte code.** Byte code is a fixed set of instructions that represent arithmetic, comparison, memory operations, etc.

It can run on any operating system and hardware. The byte code instructions are created in the **.pyc** file.

The .pyc file is not explicitly created as Python handles it internally but it can be viewed with the following command

python -m py_complie DefineVariable.py

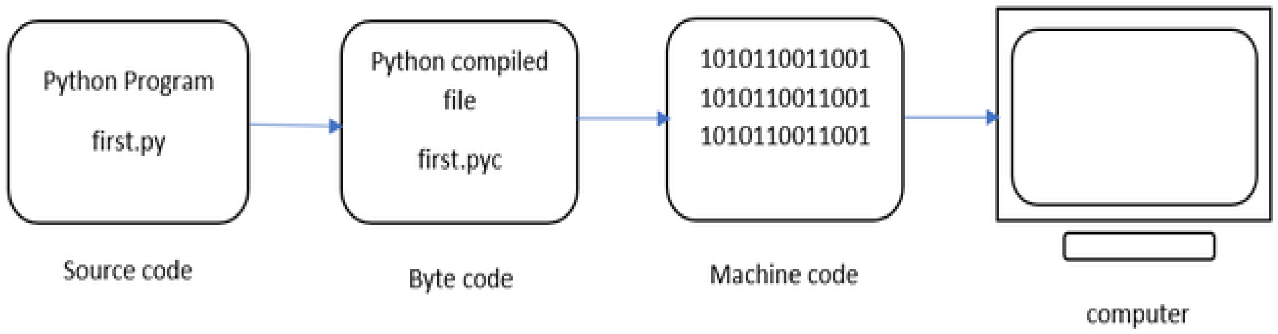
✓ Interpreter

The next step involves converting the byte code (.pyc file) into machine code.

This step is necessary as the computer can understand only machine code (binary code).

Python Virtual Machine (PVM) first understands the operating system and processor in the computer and then converts it into machine code.

Further, these machine code instructions are executed by a processor and the results are displayed.



Execution of Python Program

❖ Input and Output in Python

- Understanding input and output operations is fundamental to Python programming.

- With the print() function, you can display output in various formats, while the input() function enables interaction with users by gathering input during program execution.

- In this introductory guide, we'll explore the essentials of Python's basic input and output functionalities, empowering you to efficiently interact with users and display information.

- At its core, printing output in Python is straightforward, thanks to the print() function.
- This function allows us to display text, variables, and expressions on the console. Let's begin with the basic usage of the print() function:

Example 1:

```
print("Hello, World!")
```

Example 2:

```
name=input("Enter your name : ")
print("Hello, ", name)
```

❖ Atoms Python

There are multiple matches for atoms in Python, including a Python package for machine learning, a framework for creating Python objects, and the basic elements of expressions in Python

ATOM

An open-source Python package for machine learning that helps users:

- Train multiple models with one line of code
- Analyze data and model performance with plots
- Avoid refactoring to test new pipelines

Atom framework

A framework for creating memory-efficient Python objects with features like:

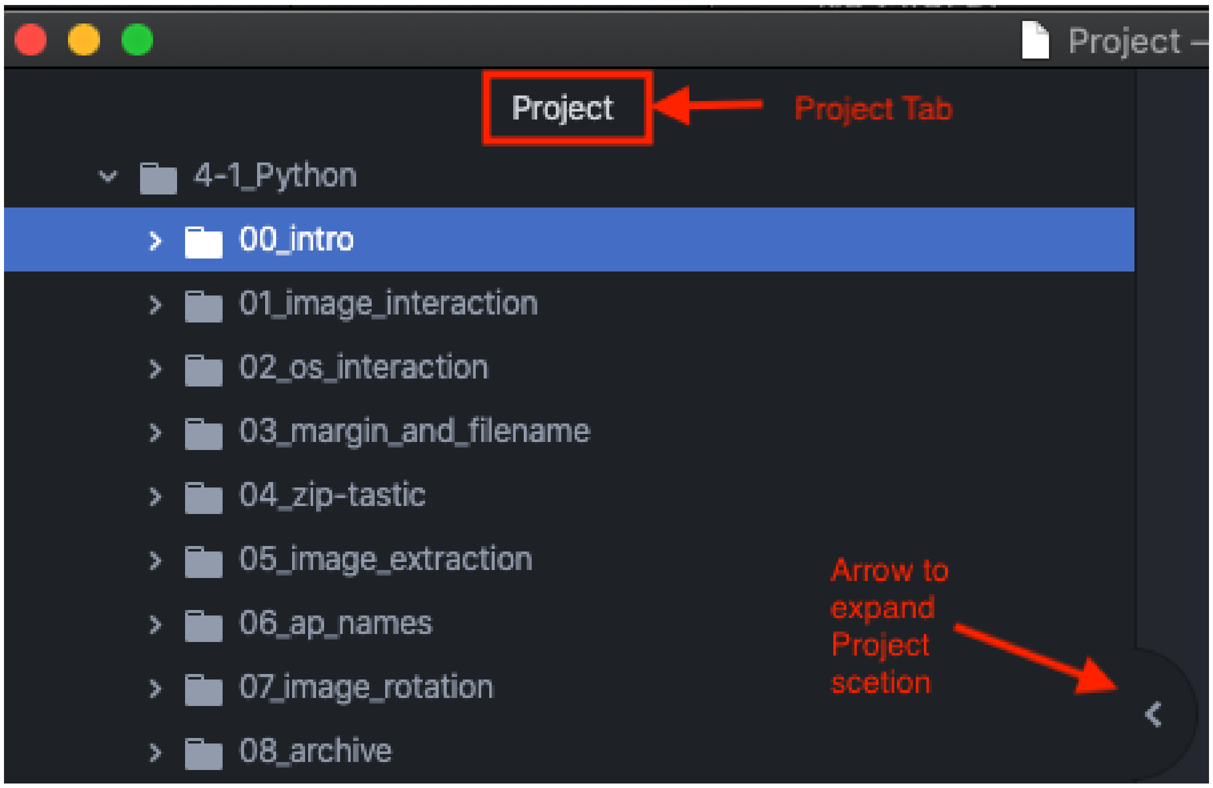
- Dynamic initialization
- Validation
- Change notification for object attributes

Atoms in expressions

The basic elements of expressions in Python, including:

- Identifiers or literals
- Forms enclosed in parentheses, brackets, or braces

Atom can also refer to Atom, an open-source, cross-platform text editor and integrated development environment (IDE) for Python:


Project –
Project
Project Tab
4-1_Python
00_intro
01_image_interaction
02_os_interaction
03_margin_and_filename
04_zip-tastic
05_image_extraction
06_ap_names
07_image_rotation
08_archive
Arrow to expand Project scetion

❖Identifiers and Keywords Python

✓ Keywords

Reserved words with special meanings that cannot be used as variable names.

Examples: if, else, for, while, def, class, import, True, False

✓ Identifiers:

Python Identifier is the name we give to identify a variable, function, class, module or other object.

That means whenever we want to give an entity a name, that's called identifier. A python identifier is a name given to various entities like variables, functions, and classes.

Rules for creating identifiers:

- Can contain letters (a-z, A-Z), numbers (0-9), and underscores (_).
- Cannot start with a number.
- Case-sensitive (myVariable and MyVariable are different).
- Cannot use Python keywords (e.g., if, else, for, while, etc.) as identifiers.

Example:

```
my_variable = 10
def calculate_sum(a, b):
    return a + b

class MyClass:
    pass
```

# having an empty function definition like this, would raise an error without the pass statement

Here, my_variable, calculate_sum, and MyClass are identifiers.

Checking if a string is a valid identifier:

You can use the isidentifier() method to check if a string is a valid identifier:

```
print("my_variable".isidentifier())  # True
print("1my_variable".isidentifier()) # False (starts with a number)
print("for".isidentifier())          # False (keyword)
```

❖ Literals in Python

Literals are notations representing fixed values in the source code. They are the raw data that is assigned to variables or constants.

Here's a breakdown of different types of literals in Python:

✓ **Numeric Literals:**

- **Integers:** Whole numbers, e.g., 42, -10, 0b1010 (binary), 0x1A (hexadecimal)
- **Floats:** Decimal numbers, e.g., 3.14, -2.7, 1e-5 (scientific notation)

✓ **String Literals:**

- **Single-quoted:** 'Hello, world!'
- **Double-quoted:** "Python is fun"
- **Triple-quoted:** '''Multi-line string''' or """Multi-line string"""

✓ **Boolean Literals:**

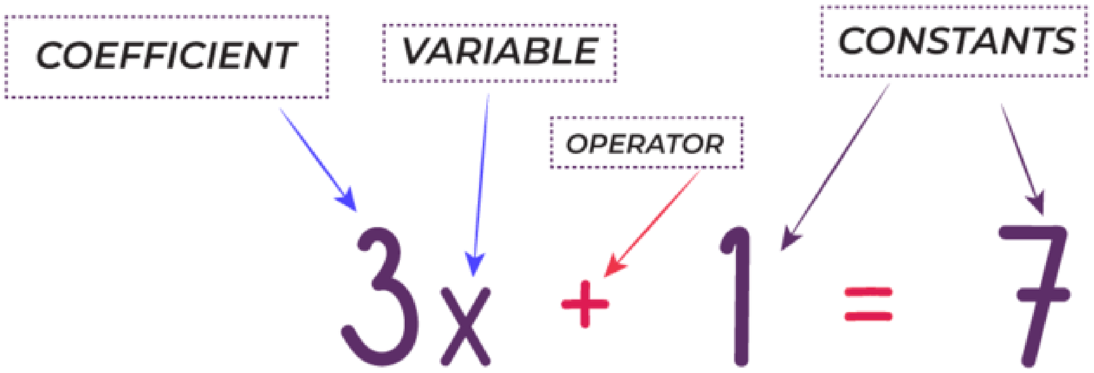
- True
- False

✓ **None Literal:**

- None (represents the absence of a value)

✓ **Collection Literals:**

- **List:** [1, 2, 3]
- **Tuple:** (1, 2, 3)
- **Dictionary:** {'name': 'Alice', 'age': 25}
- **Set:** {1, 2, 3}



❖ Python Strings

Strings in python are surrounded by either single quotation marks, or double quotation marks.

'hello' is the same as "hello".

You can display a string literal with the print() function:

Example:

```
a = "Hello, World!"

print(a[1])
```

❖ Python Operators

Operators in general are used to perform operations on values and variables. These are standard symbols used for logical and arithmetic operations. In this article, we will look into different types of **Python operators.**

- OPERATORS: These are the special symbols. Eg- + , * , /, etc.
- OPERAND: It is the value on which the operator is applied.

Types of Operators in Python

1. Arithmetic Operators
2. Relational/Comparison Operators
3. Logical/Boolean Operators
4. Bitwise Operators
5. Assignment Operators
6. Ternary Operator
7. Increment and Decrement Operators

1. Arithmetic Operators

- Python Arithmetic operators are used to perform basic mathematical operations like **addition, subtraction, multiplication**, and **division**.

Operator	Description	Syntax
+	Addition: adds two operands	x + y
–	Subtraction: subtracts two operands	x – y
*	Multiplication: multiplies two operands	x * y
/	Division (float): divides the first operand by the second	x / y
//	Division (floor): divides the first operand by the second	x // y
%	Modulus: returns the remainder when the first operand is divided by the second	x % y
**	Power: Returns first raised to power second	x ** y

2. Comparison Operators

Comparison of Relational operators compares the values. It either returns **True** or **False** according to the condition.

Operator	**Description**	**Syntax**
>	Greater than: True if the left operand is greater than the right	x > y
<	Less than: True if the left operand is less than the right	x < y
==	Equal to: True if both operands are equal	x == y
!=	Not equal to – True if operands are not equal	x != y
>=	Greater than or equal to True if the left operand is greater than or equal to the right	x >= y
<=	Less than or equal to True if the left operand is less than or equal to the right	x <= y

= is an assignment operator and == comparison operator.

3. Logical Operators

Python Logical operators perform **Logical AND**, **Logical OR**, and **Logical NOT** operations. It is used to combine conditional statements.

Operator	Description	Syntax
and	Logical AND: True if both the operands are true	x and y
or	Logical OR: True if either of the operands is true	x or y
not	Logical NOT: True if the operand is false	not x

4. Bitwise Operators

Python Bitwise operators act on bits and perform bit-by-bit operations. These are used to operate on binary numbers.

Operator	Description	Syntax
&	Bitwise AND	x & y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x ^ y
>>	Bitwise right shift	x>>
<<	Bitwise left shift	x<<

5. Assignment Operators

Assignment operators are used to assign values to the variables.

Operator	Description	Syntax
=	Assign the value of the right side of the expression to the left side operand	x = y + z
+=	Add AND: Add right-side operand with left-side operand and then assign to left operand	a+=b a=a+b
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	a-=b a=a-b
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a*=b a=a*b
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a/=b a=a/b
%=	Modulus AND: Takes modulus using left and right operands and assign the result to left operand	a%=b a=a%b
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a//=b a=a//b

Operator	Description	Syntax
**=	Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand	a**=b a=a**b
&=	Performs Bitwise AND on operands and assign value to left operand	a&=b a=a&b
|=	Performs Bitwise OR on operands and assign value to left operand	a|=b a=a|b
^=	Performs Bitwise xOR on operands and assign value to left operand	a^=b a=a^b
>>=	Performs Bitwise right shift on operands and assign value to left operand	a>>=b a=a>>b
<<=	Performs Bitwise left shift on operands and assign value to left operand	a <<= b a= a << b

6. Ternary Operator

Ternary operators also known as conditional expressions are operators that evaluate something based on a condition being true or false.

It was added to Python in version 2.5.

It simply allows testing a condition in a **single line** replacing the multiline if-else making the code compact.

Syntax : *[on_true] if [expression] else [on_false]*

7. Increment or Decrement Operators

you would have known Increment and Decrement operators (both pre and post) are not allowed in it.

Python is designed to be consistent and readable.

One common error by a novice programmer in languages with ++ and — operators are mixing up the differences (both in precedence and in return value) between pre and post-increment/decrement operators.

❖ Arrays

Python Arrays are a data structure that can hold a collection of items.

Unlike lists, which can hold items of different data types, arrays in Python are typically used to store items of the same data type.

This makes them more efficient for numerical computations.

Python does not have a built-in array data type, but the ‘array’ module provides an array type that can be used for this purpose.

Example:

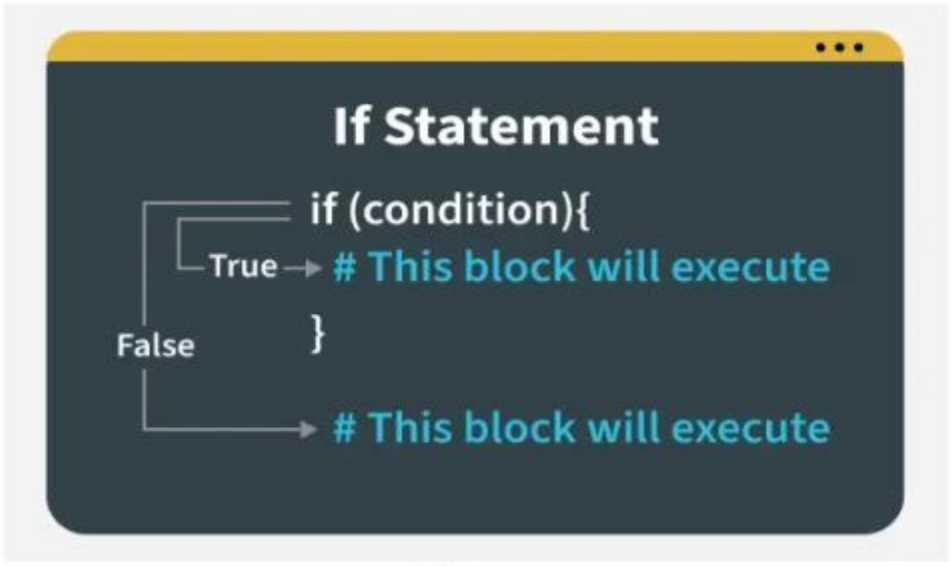
```
import array as arr


# creating array of integers

a = arr.array('i', [1, 2, 3])


# accessing Araay

print("First Element is", a[0])


# Adding element to array

a.append(5)

print(a)
```

❖ Conditional Statements

If-Else is a fundamental conditional statement used for decision-making in programming. If…Else statement allows to execution of specific blocks of code depending on the condition is *True* or *False*.

1. if Statement

if statement is the most simple decision-making statement. If the condition evaluates to True, the block of code inside the if statement is executed.



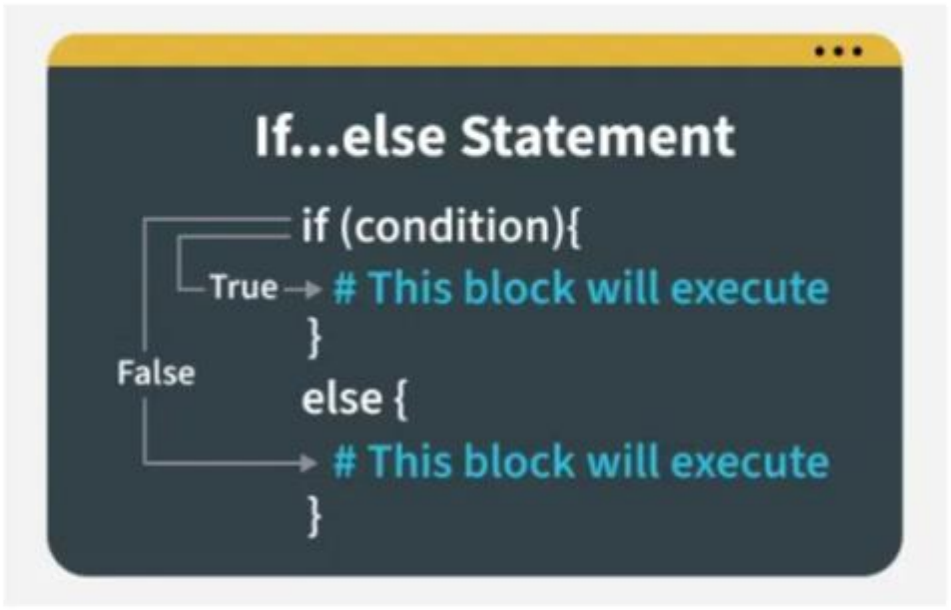
If Statement

Example of If Statement:

```
i = int(input('Enter the value of I = '))

 # Checking if i is greater than 15

if (i > 50):

    print("10 is less than 50")
```

2. if….else Statement

if…else statement is a control statement that helps in decision-making based on specific conditions. When the if condition is **False**. If the condition in the if statement is not true, the else block will be executed.



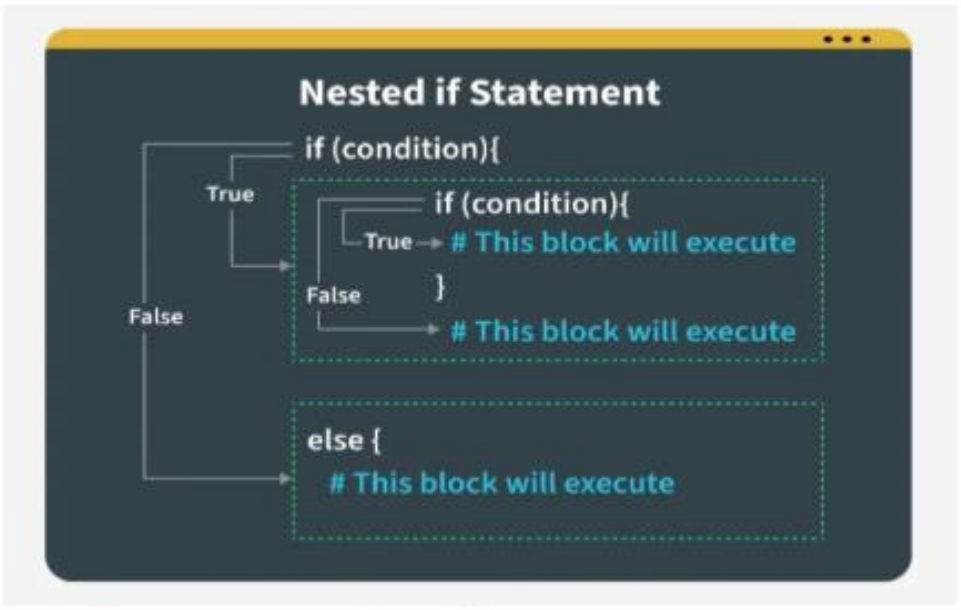
If....else Statement

Example of If….else Statement:

```
i = int(input('Enter the value of I = '))

if (i > 0):

    print("i is positive")

else:

    print("i is 0 or Negative")
```

3. Nested If Else Statement

Nested if…else statement occurs when **if…else** structure is placed inside another **if** or **else** block. Nested If..else allows the execution of specific code blocks based on a series of conditional checks.



```
if (condition){
    if (condition){
        # This block will execute
    }
    # This block will execute

else {
  # This block will execute
```

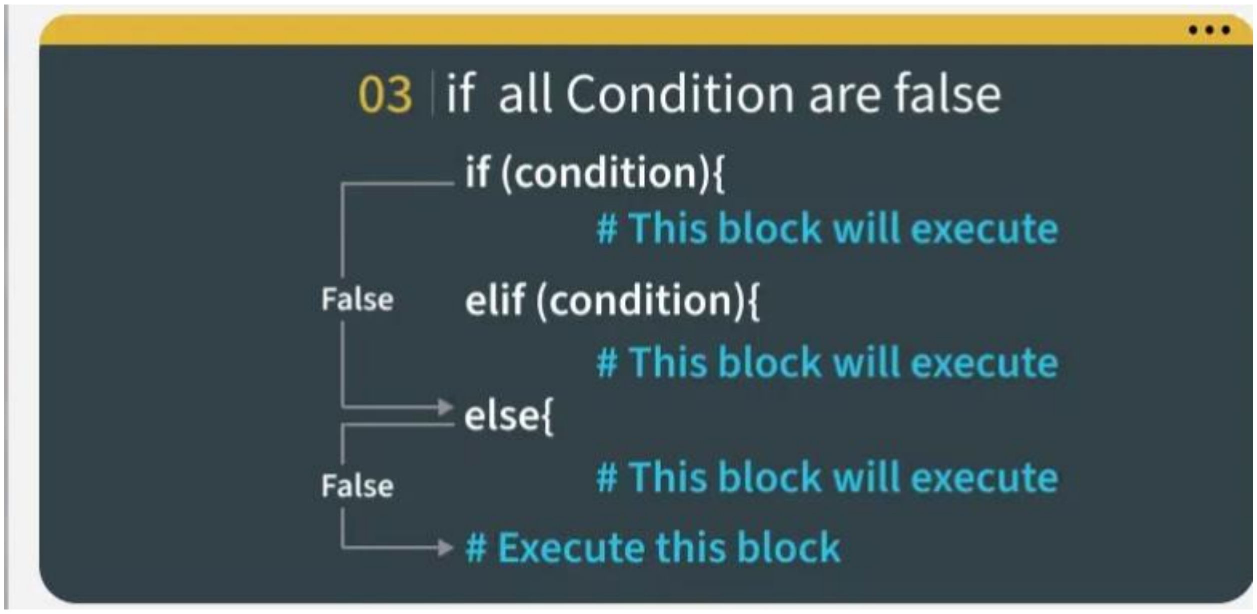
Nested if Statement

Example of Nested If Else Statement:

```
age = int(input('Enter Your Age = '))
if (age >= 18):

    ac = str(input('Is Aadhar Card Available (Yes/No)? ' ))
    if (ac == 'Yes' or ac == 'Y' or ac == 'y'):
        print("Aadhar Card Available")
        vc = str(input('Is Your Election Card Available (Yes/No)? '))
        if (vc == 'Yes' or vc == 'Y' or vc == 'y'):
             print("Eligible for Voting")
        else:
            print("Not Valid for Voting")

    else:
        print('Aadhar Card Not Available')
else:
  print("You are not eligible for Voting")
```

4. if…elif…else Statement

if-elif-else statement in Python is used for multi-way decision-making. This allows you to check multiple conditions sequentially and execute a specific block of code when a condition is True. If none of the conditions are true, the **else** block is executed.



```
if (condition){
        # This block will execute
elif (condition){
        # This block will execute
else{
        # This block will execute
# Execute this block
```

Example of Else..If…Ladder Statement:

```
subject1 = int(input('Enter the marks for C Language = '))

subject2 = int(input('Enter the marks for Networking = '))

subject3 = int(input('Enter the marks for PHP = '))

# Defining passing marks

passing_marks = 40

# Total marks and percentage calculation

total_marks = subject1 + subject2 + subject3

percentage = (total_marks / 300) * 100

# Using the elif ladder to check the conditions

if subject1 >= passing_marks and subject2 >= passing_marks and subject3
>= passing_marks:

    if percentage > 75:

        print("Result: Distinction")

    elif percentage > 50:

        print("Result: First Class")

    elif percentage > 40:

        print("Result: Second Class")

    else:

        print("Result: Fail")

else:

        print("Result: Fail")
```

5. Match Case Statement

Developers coming from languages like C/C++ or Java know that there is a conditional statement known as a **Switch Case.**

This **Match Case** is the Switch Case of Python which was introduced in Python 3.10. Here we have to first pass a parameter and then try to check with which case the parameter is getting satisfied.

If we find a match we will execute some code and if there is no match at all, a default action will take place.

Syntax:

```
match parameter:

    case pattern1:

        # code for pattern 1

    case pattern2:

        # code for pattern 2

    .

    .

    case patterN:

        # code for pattern N

    case _:

        # default code block
```

Example Match Case Statement:

```
day_number = int(input("Enter a number between 1 and 7 to get the day of the week: "))
match day_number:
    case 1:
        print("Monday")
    case 2:
        print("Tuesday")
    case 3:
        print("Wednesday")
    case 4:
        print("Thursday")
    case 5:
        print("Friday")
    case 6:
        print("Saturday")
    case 7:
        print("Sunday")
    case _:
        print("Invalid input! Please enter a number between 1 and 7.")
```

❖ Loops Controls

1. While Loop

A while loop is used to execute a block of statements repeatedly until a given condition is satisfied.

When the condition becomes false, the line immediately after the loop in the program is executed.

Syntax:

```
while expression:
    statement(s)
```

Example:

```
i = int(input("Enter Value of I =  "))

count = 0
while (count < i):
    count = count + 1
    print(count)
```

2. For Loop

For loops are used for sequential traversal. For example: traversing a list or string or array etc. In Python, there is “for in” loop which is similar to foreach loop in other languages. Let us learn how to use for loops in Python for sequential traversals with examples.

Syntax:

```
for iterator_var in sequence:
    statements(s)
```

Example:

```
i = int(input("Enter Value of I =  "))
for i in range(0, i):
    print(i)
```

❖range() Function

The range() function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

Syntax:

range(*start, stop, step*)

- start Optional. An integer number specifying at which position to start. Default is 0
- stop Required. An integer number specifying at which position to stop (not included).
- step Optional. An integer number specifying the incrementation. Default is 1

Example:

```
x = range(6)
for n in x:
  print(n)
```

❖ break, continue and pass

Break: The break statement in Python is used to exit a loop prematurely, regardless of the condition of the loop. When Python encounters a break statement, it immediately terminates the loop and proceeds with the next line of code outside the loop.

Continue: The continue statement in Python is used to skip the remainder of the code inside a loop for the current iteration only. The loop does not terminate but proceeds to the next iteration.

Pass: The pass statement in Python is a null operation; nothing happens when it is executed. It is used as a placeholder in a block where Python expects an expression.

1. break Statement:

The break statement is used to exit a loop prematurely when a certain condition is met.

Example: Breaking out of a loop when a number is found.

```
# Example of break statement
for num in range(1, 11):
    if num == 5:
        print("Breaking the loop as we found 5")
        break
    print(num)
```

Output:

```
1
2
3
4
Breaking the loop as we found 5
```

2. continue Statement:

The continue statement is used to skip the current iteration of the loop and move to the next iteration when a condition is met.

Example: Skipping the number 5

```
# Example of continue statement
for num in range(1, 11):
    if num == 5:
        print("Skipping 5")
        continue
    print(num)
```

Output:

```
1
2
3
4
Skipping 5
6
7
8
9
10
```

3. pass Statement:

The pass statement is a null operation. It’s a placeholder that does nothing. It is commonly used when you need a syntactically correct block of code but don’t want to do anything yet.

Example: Using pass in an empty if block

```
# Example of pass statement
for num in range(1, 6):
    if num == 3:
        pass  # Do nothing for 3
    else:
        print(f"Processing {num}")
```

Output:

```
Processing 1
Processing 2
Processing 3
Processing 4
Processing 5
```

Thank You