MASTER OF COMPUTER APPLICATIONS (MCA)

Post Graduate Department of Computer Science and Technology

Sardar Patel University

Vallabh Vidyanagar, Gujarat, INDIA

PS01CMCA51 [Python Programming]

MR. BHARAT B. PATEL (ASSOCIATE PROFESSOR)

(✉) bb_patel@spuvvn.edu

Python Programming

- Brief history
- Key characteristics
- Advantages of Python
- Application Areas of Python
- Syntax overview, comments, naming conventions
- Primitive data types, data type constructors
- Operators
- Console input and output
- Control structures

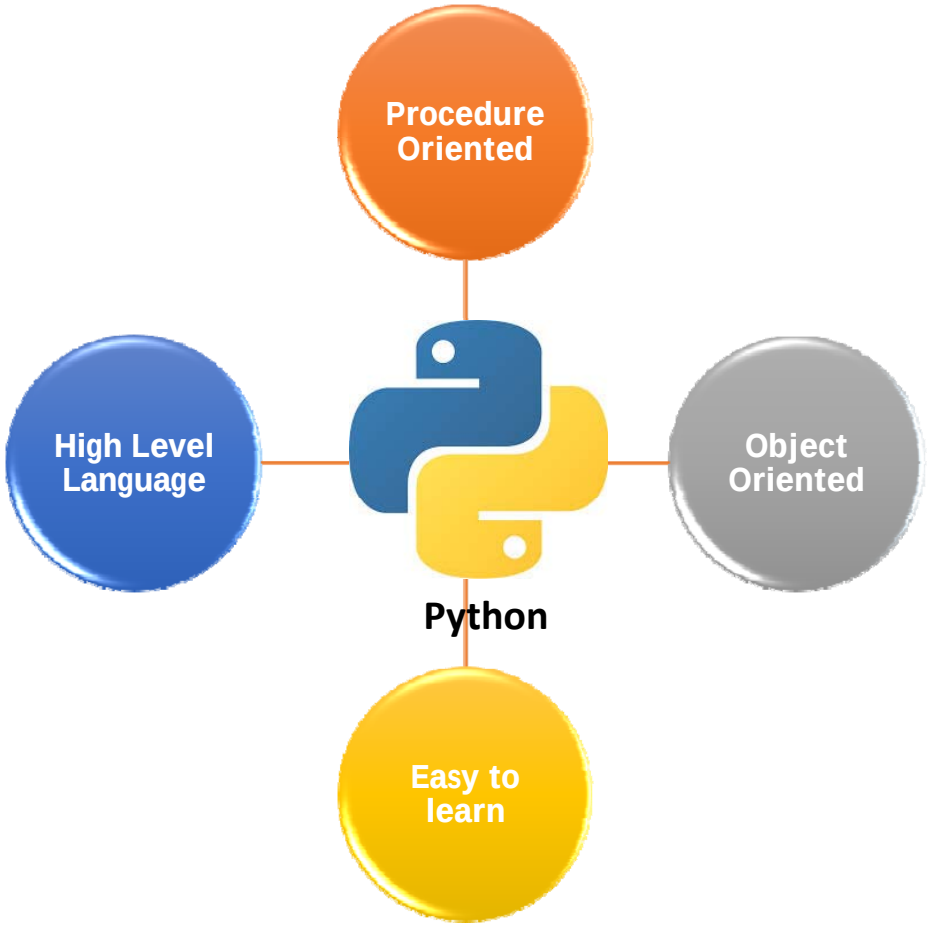
Brief History

- **Python is an interpreted, high-level and general-purpose programming language.**
- **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.**
- **Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.**
- **The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, https://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.**

Brief History

- **Python is a widely used general-purpose, high level programming language.**
- **It was created by Guido van Rossum in 1991 and further developed by the Python Software Foundation.**
- **It was designed with an emphasis on code readability, and its syntax allows programmers to express their concepts in fewer lines of code.**

Python


Procedure Oriented
High Level Language
Object Oriented
Python
Easy to learn

Key characteristics of Python

Readable	Cross Platform	Memory Management	Easy to learn and use	Free
Open Source	Large Standard Library	Exception Handling	Object Oriented	Procedure Oriented
Interpreted Langauge	Expressive Language	Dynamically Typed	GUI programming support	Extensible

Advantages of Python

- Presence of third-party modules
- Extensive support libraries (NumPy for numerical calculations, Pandas for data analytics etc)
- Open source and community development
- Easy to learn
- User-friendly data structures
- High-level language
- Dynamically typed language(No need to mention data type based on value assigned, it takes data type)
- Object-oriented language
- Portable and Interactive
- Portable across Operating systems

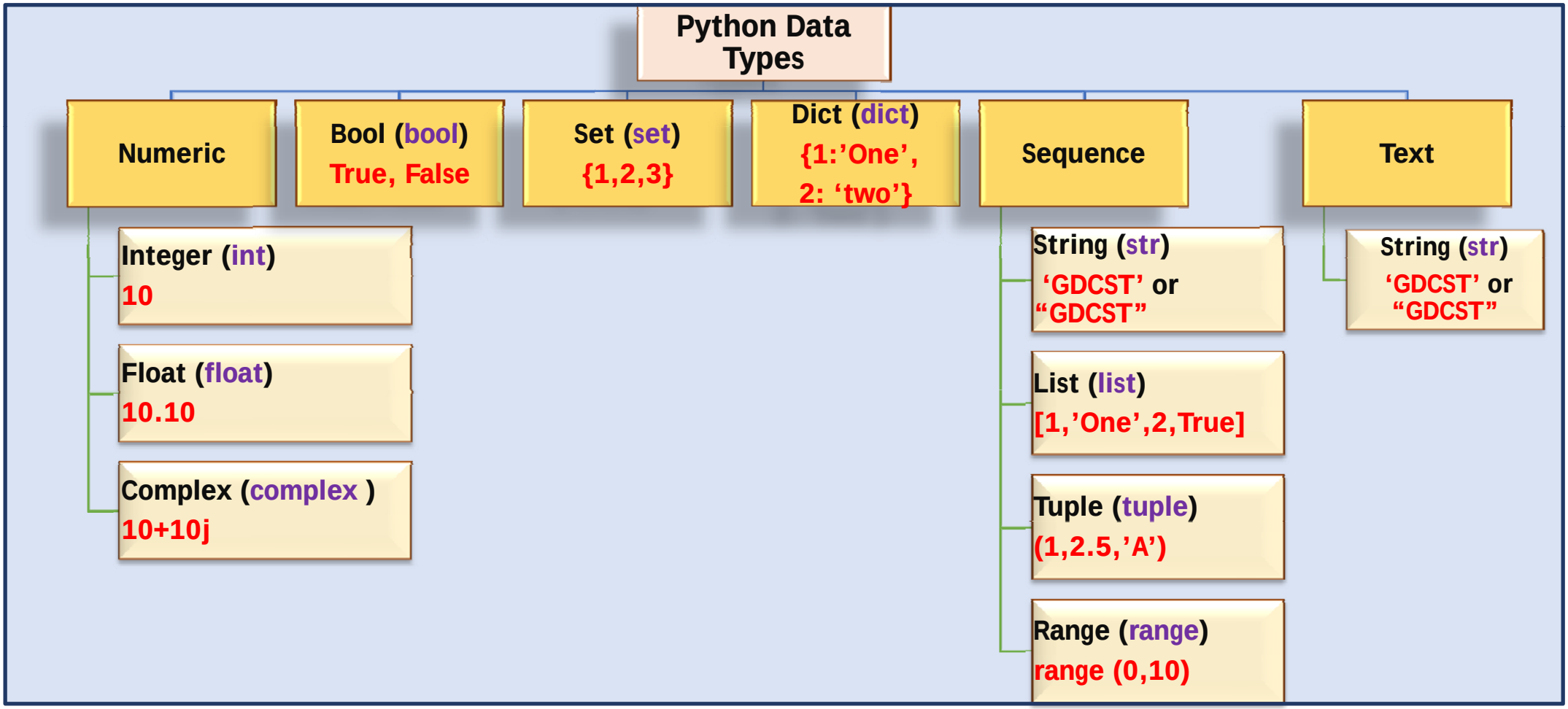
Application Areas of Python

Artificial Intelligence	Machine Learning	Data analytics	Data Science	Data Visualization	Web Development
Web Scraping Applications	Embedded Applications	Enterprise-level/Business Applications	Scientific and Numeric Applications	Software Development	Console Applications
Desktop applications	3D graphics	Audio/Video Applications	Automation	CAD Applications	Education programs and training courses
Game Development	Image Processing	Graphic Design Applications	Finance	Language Development	Network programming

Python Data Types

- **Data Type is a classification of data that we used in our computer program to process the data.**
- **Data can be integer, float, decimal, string, character, boolean, etc.**
- **Every value in Python has a datatype.**
- **Since everything is an object in Python programming, data types are actually classes and variables are instances (object) of these classes.**
- **Python is a dynamically-typed language, which means it is *not required beforehand to declare the variables explicitly to use in a program*.**
- **Python interpreter sets the variable's datatype based on the assigned value, the data type is changed if the value of the variable is changed.**
- **There are different data types in Python.**

Python Data Types



Advantages of Python

Easy to Read, Learn and Write

- Python is a high-level programming language that has English-like syntax. This makes it easier to read and understand the code.
- Python is really easy to pick up and learn, that is why a lot of people recommend Python to beginners.
- You need less lines of code to perform the same task as compared to other major languages like C/C++ and Java.

Improved Productivity

- Python is a very productive language. Due to the simplicity of Python, developers can focus on solving the problem.
- They don't need to spend too much time in understanding the syntax or behaviour of the programming language. You write less code and get more things done.

Interpreted Language

- Python is an interpreted language which means that Python directly executes the code line by line. In case of any error, it stops further execution and reports back the error which has occurred.
- Python shows only one error even if the program has multiple errors. This makes debugging easier.

Dynamically Typed

- Python doesn't know the type of variable until we run the code. It automatically assigns the data type during execution. The programmer doesn't need to worry about declaring variables and their data types.

Advantages of Python

Free and Open-Source

- Python comes under the OSI approved open-source license. This makes it free to use and distribute. You can download the source code, modify it and even distribute your version of Python. This is useful for organizations that want to modify some specific behaviour and use their version for development.

Vast Libraries Support

- The standard library of Python is huge, you can find almost all the functions needed for your task. So, you don't have to depend on external libraries. But even if you do, a Python package manager (pip) makes things easier to import other great packages from the Python package index (PyPi). It consists of over 200,000 packages.

Portability

- In many languages like C/C++, you need to change your code to run the program on different platforms. That is not the same with Python. You only write once and run it anywhere. However, you should be careful not to include any system-dependent features.

Disadvantages of Python

Slow Speed	• We discussed above that Python is an interpreted language and dynamically-typed language. The line by line execution of code often leads to slow execution. The dynamic nature of Python is also responsible for the slow speed of Python because it has to do the extra work while executing code. So, Python is not used for purposes where speed is an important aspect of the project.
Not Memory Efficient	• To provide simplicity to the developer, Python has to do a little tradeoff. The Python programming language uses a large amount of memory. This can be a disadvantage while building applications when we prefer memory optimization. • Any doubts in Python advantages and disadvantages till now? Mention in the comment section. We will be happy to help you!!
Weak in Mobile Computing	• Python is generally used in server-side programming. We don't get to see Python on the client-side or mobile applications because of the following reasons. Python is not memory efficient and it has slow processing power as compared to other languages.
Database Access	• Programming in Python is easy and stress-free. But when we are interacting with the database, it lacks behind. The Python's database access layer is primitive and underdeveloped in comparison to the popular technologies like JDBC and ODBC. Huge enterprises need smooth interaction of complex legacy data and Python is thus rarely used in enterprises.
Runtime Errors	• As we know Python is a dynamically typed language so the data type of a variable can change anytime. A variable containing integer number may hold a string in the future, which can lead to Runtime Errors. Therefore Python programmers need to perform thorough testing of the applications.

Python Operators

- **Operators are special symbols that perform some operation on operands and returns the result.**
- **Operators are used to perform operations on variables and values.**
- **For example, 11 + 10 is an expression where + is an operator that performs arithmetic add operation on numeric left operand 11 and the right side operand 10 and returns a sum of two operands as a result.**
- **Python includes the operator module that includes underlying methods for each operator. For example, the + operator calls the operator.add(a,b) method.**

Example of Operator Methods

```
>>> import operator
>>> operator.mul ( 3,4)
12
>>> operator.eq ( 23,34)
False
>>>
```

Python Operators

- **Python includes the following categories of operators :**

Arithmetic Operators	• Are used to perform the common mathematical operation on the numeric operands.
Assignment Operators	• Are used to assign values to variables.
Comparison Operators	• Are used to compare two operands and return a Boolean either True or False.
Logical Operators	• Are used to combine two Boolean expressions.
Identity Operators	• Are used to check whether the two objects have the same id value i.e. both the objects point to the same memory location.
Membership Test Operators	• Are used to test whether the sequence has a given item or not.
Bitwise Operators	• Are used to perform operations on binary operands.

[1] Python Arithmetic Operators

Operator	Meaning	Example
+	Add two operands or unary plus	A+b+3
-	Subtract right operand from the left or unary minus	A-b-2
*	Multiply two operands	a * b
/	Divide left operand by the right one (always results into float)	a / b
%	Modulus - remainder of the division of left operand by the right	a% b (remainder of a/b)
//	Floor division - division that results into whole number adjusted to the left in the number line	a//b
**	Exponent - left operand raised to the power of right	x**y (x to the power y)

[2] Python Assignment Operators

Operator	Meaning	Example
=	Assign value of right side of expression to left side operand	a = 5
+=	Add AND: Add right-side operand with left side operand and then assign to left operand	a += 5
-=	Subtract AND: Subtract right operand from left operand and then assign to left operand	a -= 5
*=	Multiply AND: Multiply right operand with left operand and then assign to left operand	a *= 5
/=	Divide AND: Divide left operand with right operand and then assign to left operand	a /= 5
%=	Modulus AND: Takes modulus using left and right operands and assign the result to left operand	a %= 5

[2] Python Assignment Operators (contd..)

Operator	Meaning	Example
//=	Divide(floor) AND: Divide left operand with right operand and then assign the value(floor) to left operand	a //= 5
**=	Exponent AND: Calculate exponent(raise power) value using operands and assign value to left operand	a **= 5
&=	Performs Bitwise AND on operands and assign value to left operand	a &= 5
|=	Performs Bitwise OR on operands and assign value to left operand	a |= 5
^=	Performs Bitwise xOR on operands and assign value to left operand	a ^= 5
>>=	Performs Bitwise right shift on operands and assign value to left operand	a >>= 5
<<=	Performs Bitwise left shift on operands and assign value to left operand	a <<= 5

[3] Python Comparison Operators

Operator	Meaning	Example
>	Greater than - True if left operand is greater than the right	x > y
<	Less than - True if 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 left operand is greater than or equal to the right	x >= y
<=	Less than or equal to - True if left operand is less than or equal to the right	x <= y

[4] Python Logical Operators

Operator	Meaning	Example
and	True if both the operands are true	x and y
or	True if either of the operands is true	x or y
not	True if operand is false (complements the operand)	not x

[5] Python Bitwise Operators

Operator	Meaning	Example
&	Bitwise AND	x & y = 0 (0000 0000)
|	Bitwise OR	x | y = 14 (0000 1110)
~	Bitwise NOT	~x = -11 (1111 0101)
^	Bitwise XOR	x ^ y = 14 (0000 1110)
>>	Bitwise right shift	x >> 2 = 2 (0000 0010)
<<	Bitwise left shift	x << 2 = 40 (0010 1000)

[6] Python Identity Operators

Operator	Meaning	Example
is	True if the operands are identical (refer to the same object)	a is True
is not	True if the operands are not identical (do not refer to the same object)	a is not True
is	True if the operands are identical (refer to the same object)	a is True

[7] Python Membership Operators

Operator	Meaning	Example
in	True if value/variable is found in the sequence	5 in List1
not in	True if value/variable is not found in the sequence	5 not in List1

Console input and output

- **The interactive shell in Python is treated as a console. We can take the user entered data the console using built-in input() function. User enters the values in the Console and that value is then used in the program as it was required.**
- **The syntax for input() is:**

input([prompt])

- **Example :**

```
# input
input1 = input()
# output
print(input1)
```

- **We can also type cast this input to integer, float or string by specifying the input() function inside the type.**

Typecasting the input to Integer

Typecasting the input to Integer

- **There might be conditions when you might require integer input from user/Console, the following code takes two input(integer/float) from console and typecasts them to integer then prints the sum.**

```
# input
num1 = int(input())
num2 = int(input())
# printing the sum in integer
print(num1 + num2)
```

- **Typecasting the input to Float: To convert the input to float the following code will work out.**

Typecasting the input to Float

Typecasting the input to Float

- **To convert the input to float the following code will work out.**

```
# input
num1 = float(input())
num2 = float(input())
# printing the sum in float
print(num1 + num2)
```

Typecasting the input to String

Typecasting the input to String

- **All kind of input can be converted to string type whether they are float or integer. We make use of keyword str for typecasting.**

```
# input
string = str(input())
# output
print(string)
```

Console output

- **We use the print() function to output data to the standard output device (screen).**
- **The print() function contains all the functionality needed to output text to the console.**
- **When the interpreter encounters the print() statement, it will try to print everything we write between the parentheses ( ) to the console.**
- **The actual syntax of the print() function is:**

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

Control structures

- **In Python programming, flow control is the order in which statements or blocks of code are executed at runtime based on a condition.**
- **There are four types of control structures.**

  [1] Sequential

  [2] Selection

  [3] Looping or Iteration

  [4] Jumping
- **A language should have the ability to control the flow of execution so that at different intervals different statements can be executed.**

Control structures

- **Structured programming is a paradigm aims at controlling the flow of execution of statements in a program by using control structures.**
- **A language which supports the control structures is called as structured programming language.**

If statement

- **if statement is the most simple decision-making statement.**
- **It is used to decide whether a certain statement or block of statements will be executed or not i.e if a certain condition is true then a block of statement is executed otherwise not.**
- **if statement can be written using following different forms:**

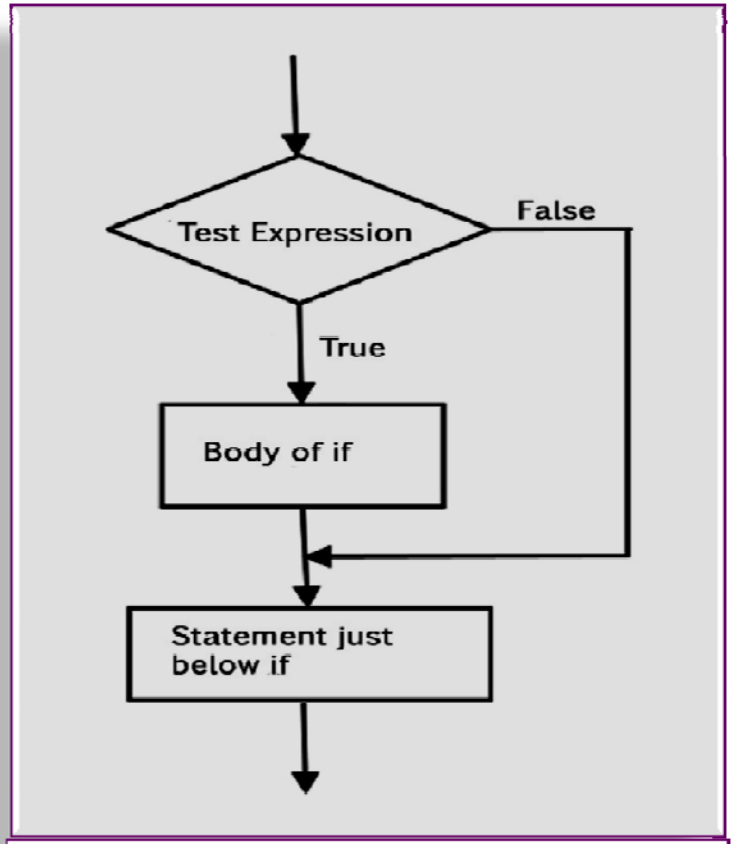
  [1] simple-if

  [2] if-else

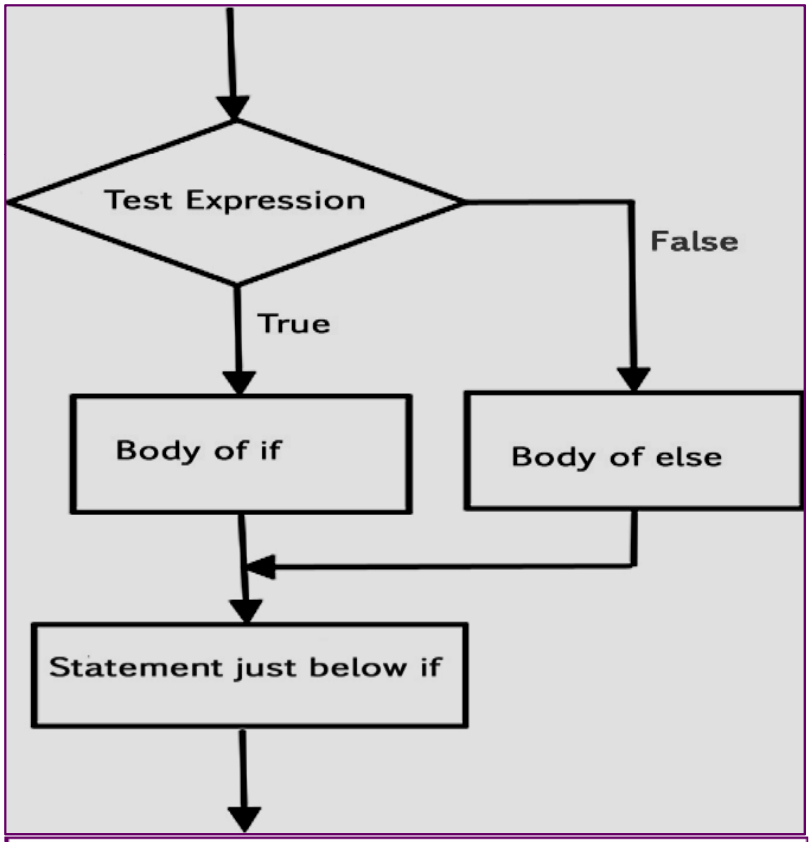
  [3] nested if

  [4] if-elif-else ladder

Different forms of if statement

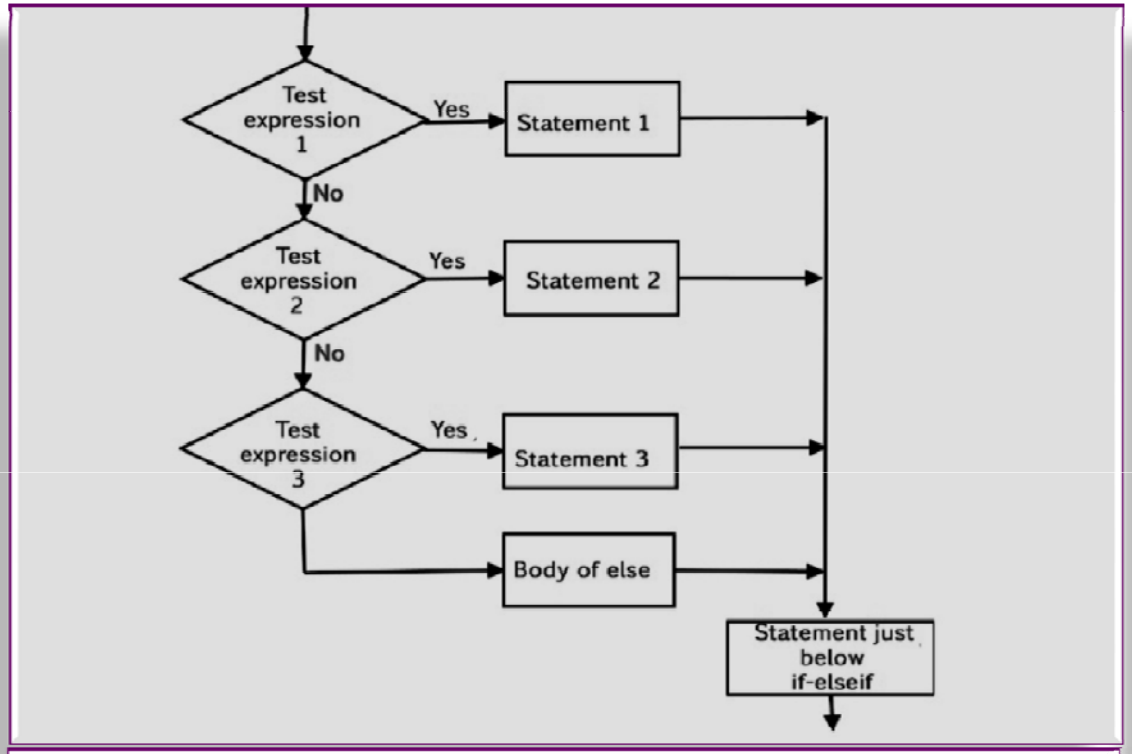


simple-if

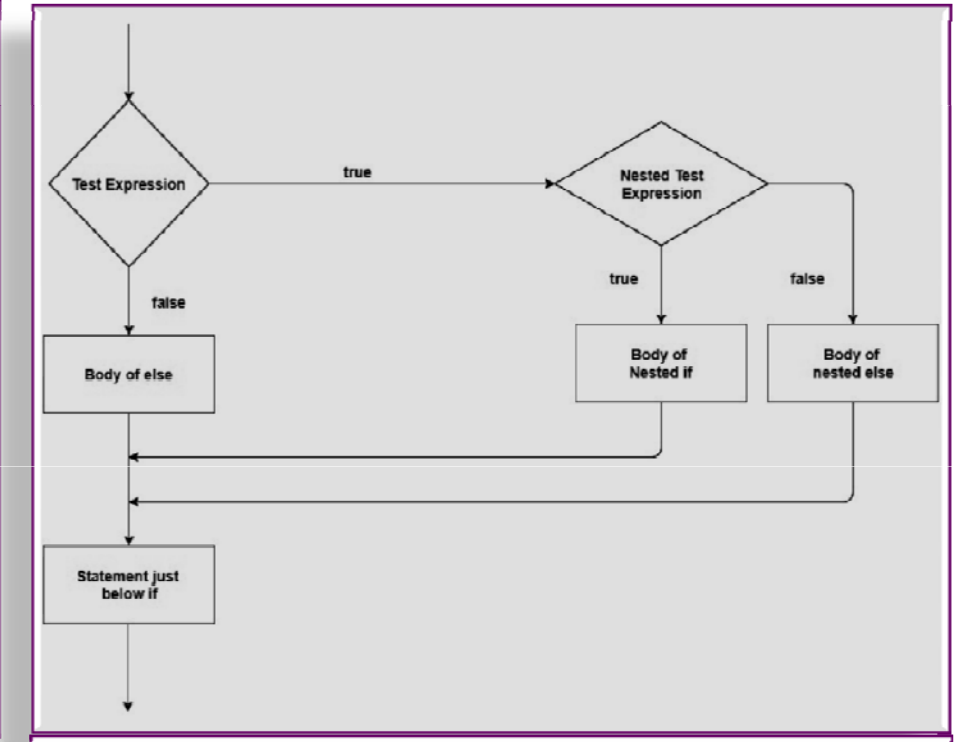


if-else statement

If statement



nested-if



if-elif-else ladder

Syntax of if statement

simple-if

```
if condition:
    # Statements to execute if
    # condition is true
```

if-else statement

```
if (condition):
    # Executes this block if
    # condition is true
else:
    # Executes this block if
    # condition is false
```

Syntax of if statement

nested if

```
if (condition1):
    # Executes when condition1 is true
    if (condition2):
        # Executes when condition2 is true
    # if Block is end here
# if Block is end here
```

if-elif-else ladder

```
if (condition1):
    # Executes when condition1 is true
    if (condition2):
        # Executes when condition2 is true
    # if Block is end here
# if Block is end here
```

If statement

- **In if statement, the condition after evaluation will be either true or false.**
- **if statement accepts boolean values – if the value is true then it will execute the block of statements below it otherwise not.**
- **One can also write the condition with bracket i.e. using ( ).**

While statement

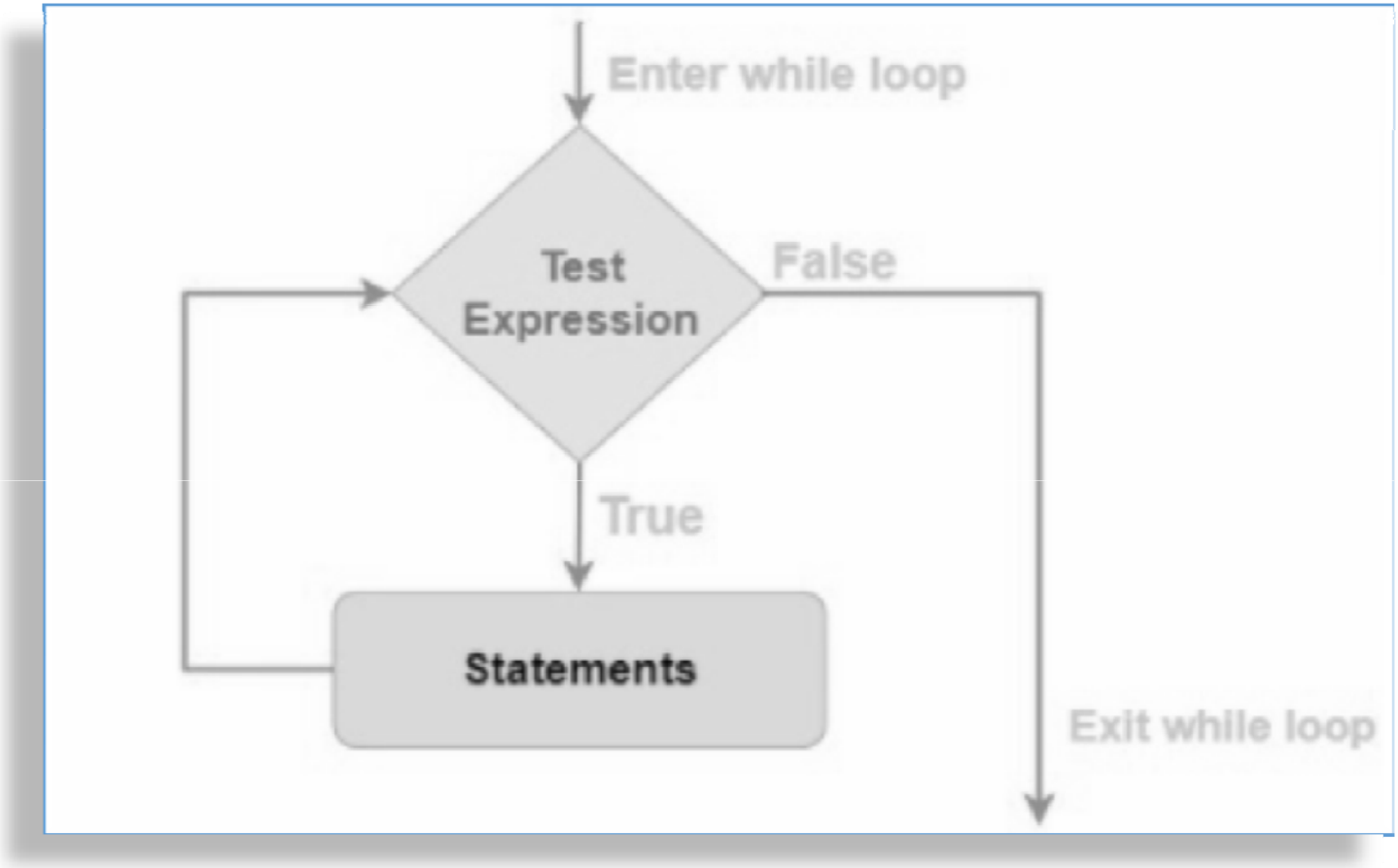
- It is used to execute a block of statements repeatedly until a given condition is satisfied.
- When the condition becomes false, the line immediately after the loop in the program is executed.
- While loop falls under the category of indefinite iteration.
- Indefinite iteration means that the number of times the loop is executed isn't specified explicitly in advance.
- Syntax : while expression:

  statement(s)

While statement

- **Statements represent all the statements indented by the same number of character spaces after a programming construct are considered to be part of a single block of code.**
- **Python uses indentation as its method of grouping statements.**
- **When a while loop is executed, expr is first evaluated in a Boolean context and if it is true, the loop body is executed.**
- **Then the expr is checked again, if it is still true then the body is executed again and this continues until the expression becomes false.**

Flow chart of While statement



While statement

- **Example :**

```
>>> cnt=0
>>> while ( cnt<5) :
        print ( "count = ",cnt)
        cnt=cnt+1
```

- **Output :**

```
count =  0
count =  1
count =  2
count =  3
count =  4
```

for statement

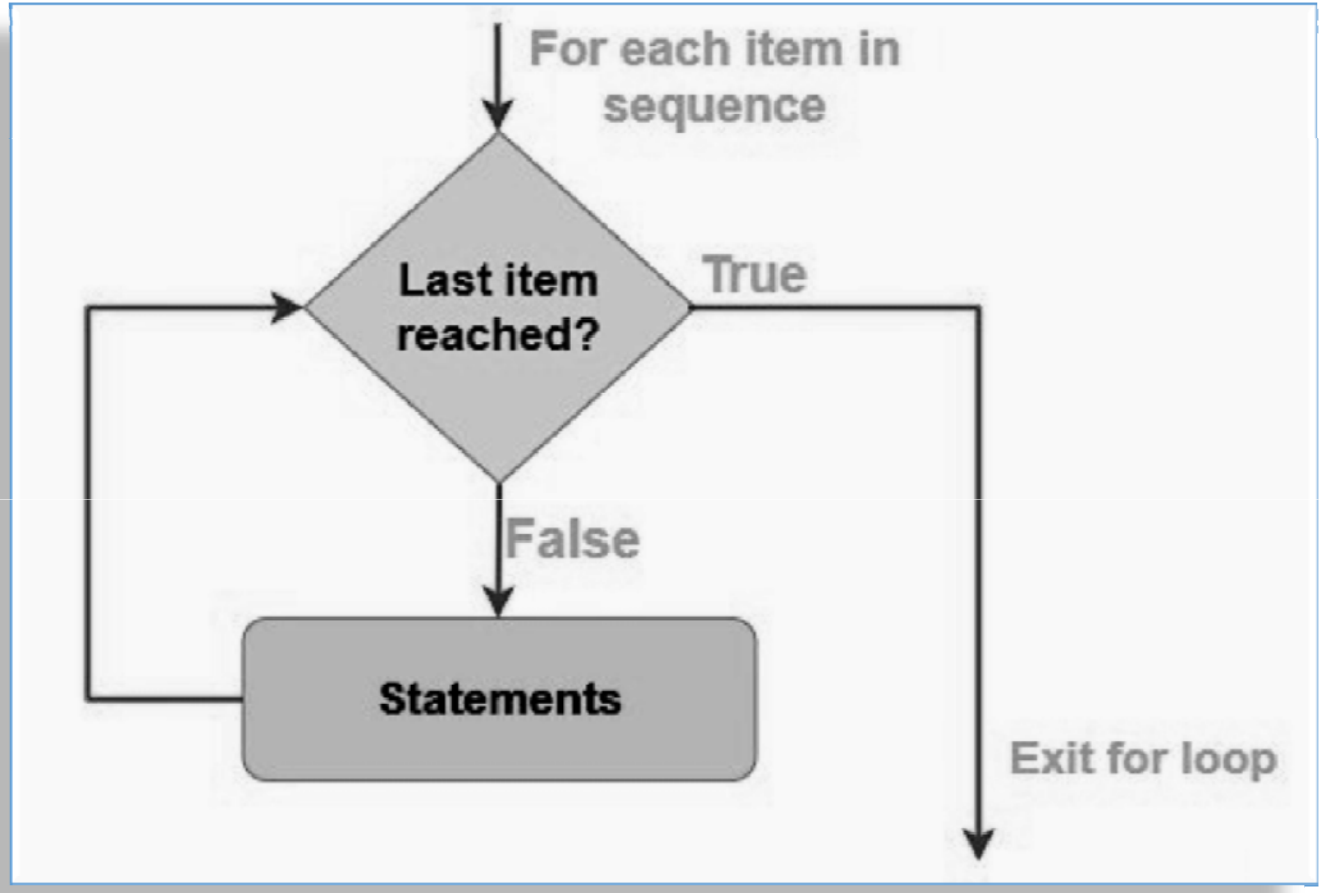
- **It is used for sequential traversal i.e. it is used for iterating over an iterable like string, tuple, list, etc. It falls under the category of definite iteration. Definite iterations mean the number of repetitions is specified explicitly in advance. In Python, there is no C style for loop, i.e., for (i=0; i<n; i++). There is "for in" loop which is similar to for each loop in other languages. Let us learn how to use for in loop for sequential traversals.**
- **Syntax : for var in iterable:**

```
    # statements
```

for statement

- **In syntax, the iterable is a collection of objects like lists, tuples.**
- **The indented statements inside the for loops are executed once for each item in an iterable.**
- **The variable var takes the value of the next item of the iterable each time through the loop.**

Flow chart of for statement



While statement

- **Example :**

```
>>> lst1=[12,23,44]
>>> for ele in lst1:
        print ( ele,end=", ")
```

- **Output :**

```
12, 23, 44,
>>>
```

continue statement

- **When the program encounters a continue statement, it will skip the statements which are present after the continue statement inside the loop and proceed with the next iterations.**
- **Syntax : continue**
- **Example : for char in 'SPU':**

```
for char in 'SPU':
    if (char == 'P'):
        continue
    print("Current character: ", char)
```

- **Output : Current character: S**
  Current character: U

break statement

- **The break statement is used to terminate the loop containing it, the control of the program will come out of that loop.**
- **Syntax : break**
- **Example : for char in 'SPU':**

```
for char in 'SPU':
    if (char == 'P'):
        break
    print("Current character: ", char)
```

- **Output : Current character: S**

Pass statement

- **Pass statement is python is a null operation, which is used when the statement is required syntactically.**
- **Syntax : pass**
- **Example :**

```
for char in 'SPU':
        if (char == 'P'):
            pass
        print("Current character: ", char)
```

- **Output : Current character: S**

  Current character: P

  Current character: U