PYTHON PROGRAMMING

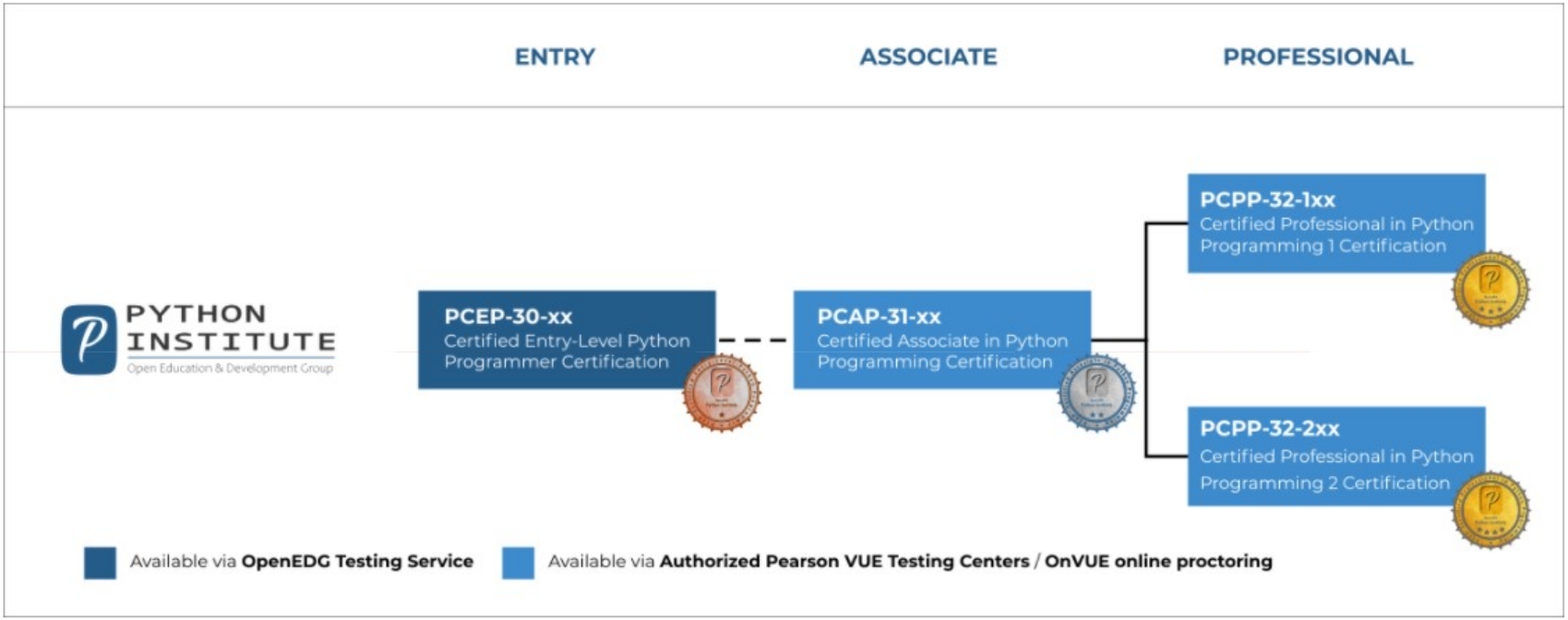
Product name:
Python Programming

Product code:
GDPY201

Duration:
6 months / 24 weeks/ 6 hours per week

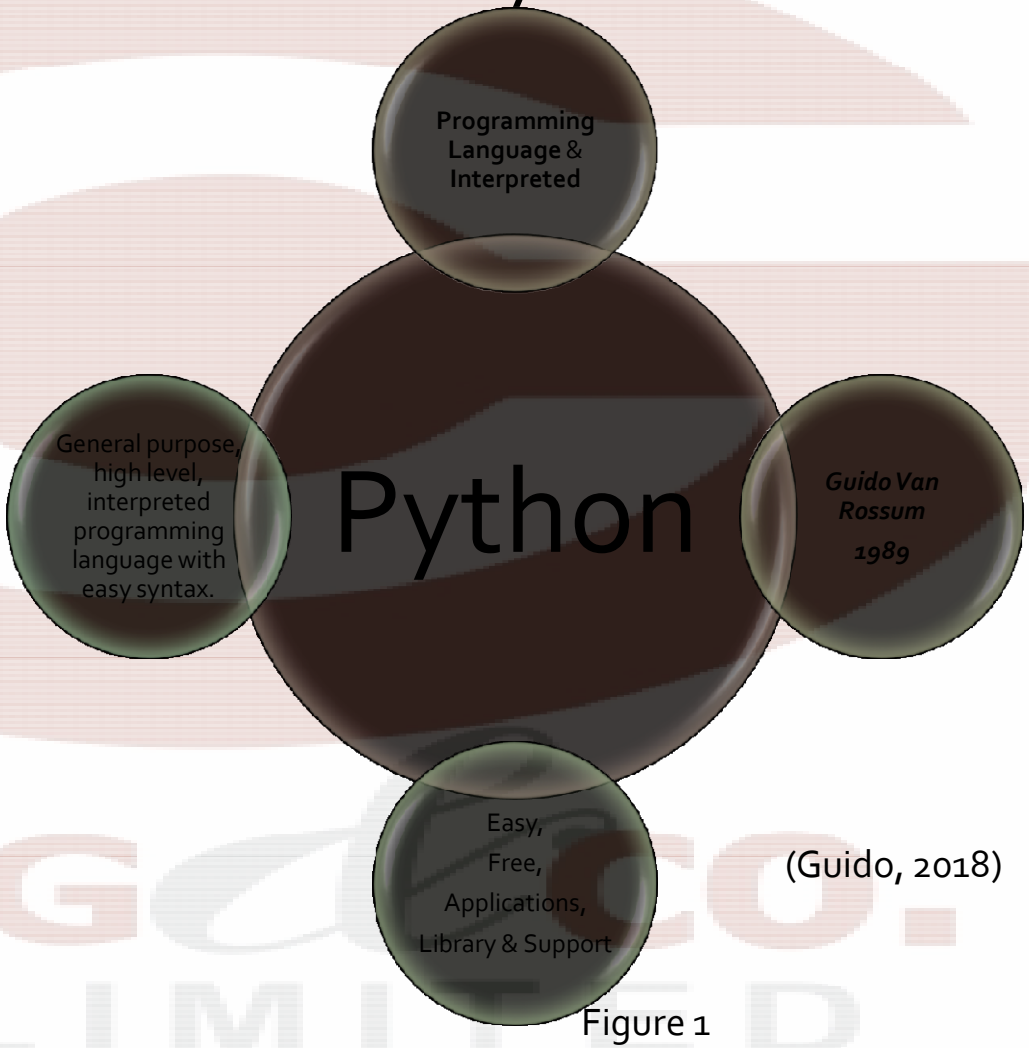
Certification:
Certified Entry-Level Python Programmer Certification(PCEP-30-xx)
Certified Associate in Python Programming Certification (PCAP-31-xx)
Certified Professional in Python Programming 1 Certification (PCPP-32-1xx)
Certified Professional in Python Programming 2 Certification (PCPP-32-2xx)

Candidates can take **certification exams at three competency levels:** Entry, Associate, and Professional.



Python Institute Certification Roadmap

About Python



(Guido, 2018)

Figure 1

Introduction

- Python programming language is dominating other programming languages such as C, C++ or Java.
- Python is an Object-Oriented, High-Level programming language with dynamic features.
- It is one of the fastest growing programming languages.
- Python is designed to be highly readable.
- Python programming language is best used for application development, web application or web development, game development, system administration and scientific computing etc.

Features of Python

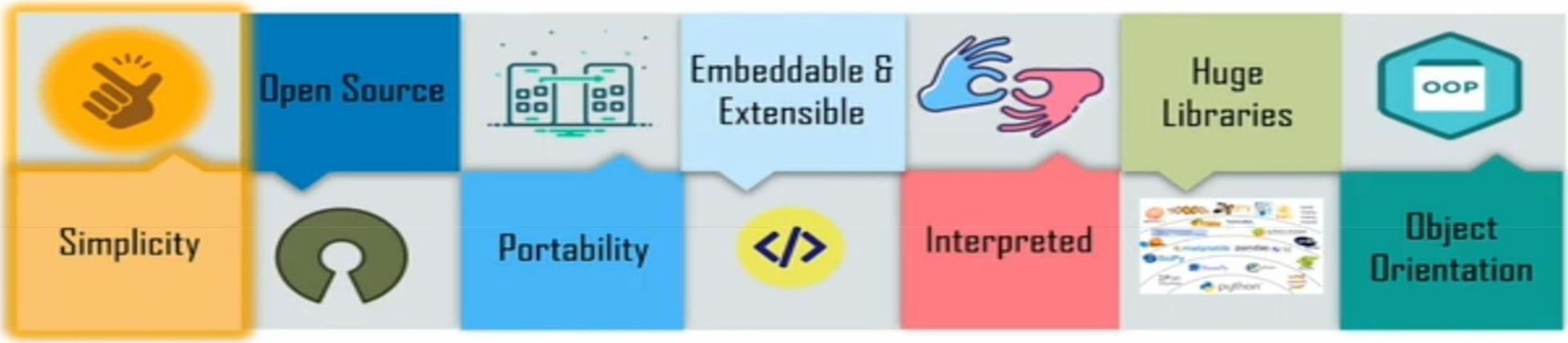


Figure 2

(Guido, 2018)

Main Features

- Python is an **interpreted**, **interactive**, **object-oriented** high-level language.
- Python is a **dynamically** typed language, and does not require variables to be declared before they are used. Variables **"appear"** when they are first used and **"disappear"** when they are no longer needed.
- Python is a **scripting language** like Tcl and Perl. Because of its interpreted nature, it is also often compared to Java. Unlike Java, Python does not require all instructions to reside inside classes.
- Python is also a **multi-platform language**, since the Python interpreter is available for a large number of standard operating systems, including MacOS, UNIX, and Microsoft Windows. Python interpreters are usually written in **C**, and thus can be ported to almost any platform which has a **C compiler**.

Python Environment Setup

- Local Environment setup
- Open a terminal window and type "python" to find out if it is already installed and which version is installed.
- If you don’t have it installed, download it from https://www.python.org/

Windows Installation

- Here are the steps to install Python on Windows machine.

1. Open a Web browser and go to https://www.python.org/downloads/.
2. Follow the link for the Windows installer *python-XYZ.msi* file where XYZ is the version you need to install.
3. To use this installer *python-XYZ.msi*, the Windows system must support Microsoft Installer 2.0. Save the installer file to your local machine and then run it to find out if your machine supports MSI.
4. Run the downloaded file. This brings up the Python install wizard, which is really easy to use. Just accept the default settings, wait until the install is finished, and you are done.

Windows Installation

- **Setting path at Windows**
- To add the Python directory to the path for a particular session in Windows –
- **At the command prompt** – type path %path%;C:\Python and press Enter.
- **Note** – C:\Python is the path of the Python directory

Running Python

- There are three different ways to start Python –
- **Interactive Interpreter:** You can start Python from Unix, DOS, or any other system that provides you a command-line interpreter or shell window.

  Enter **python** the command line.

  Start coding right away in the interactive interpreter.
- **Script Mode Programming:** Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

E.g C: >python script.py

- **Integrated Development Environment:** You can run Python from a Graphical User Interface environment as well, if you have a GUI application on your system that supports Python.
  - **Unix** – IDLE is the very first Unix IDE for Python.
  - **Windows** – PythonWin is the first Windows interface for Python and is an IDE with a GUI.
  - **Macintosh** – The Macintosh version of Python along with the

Running Python

- There are three different ways to start Python –
- **Interactive Interpreter:** You can start Python from Unix, DOS, or any other system that provides you a command-line interpreter or shell window.

  Enter **python** the command line.

  Start coding right away in the interactive interpreter.
- **Script Mode Programming:** Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

E.g C: >python script.py

- **Integrated Development Environment:** You can run Python from a Graphical User Interface environment as well, if you have a GUI application on your system that supports Python.
  - **Unix** – IDLE is the very first Unix IDE for Python.
  - **Windows** – PythonWin is the first Windows interface for Python but the IDE we will use is Pycharm.
  - **Macintosh** – The Macintosh version of Python along with the

Python Variables

- Variable is a name which is used to refer memory location.
- Variables are the example of identifiers and are used to hold values.
- **Identifiers**
- A Python identifier is a name used to identify a variable, function, class, module or other object.
- **The rules to name an identifier are given below:**

1. The first character of the variable must be an alphabet or underscore (_).
2. All the characters except the first character may be an alphabet of lower-case(a-z), upper-case (A-Z), underscore or digit (0-9).
3. Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).
4. Identifier name must not be similar to any keyword defined in the language.
5. Identifier names are case sensitive for example my name, and MyName is not the same.

- Examples of **valid identifiers** : a123, _n, n_9, etc.
- Examples of **invalid identifiers**: 1a, n%4, n 9, etc.
- Here are naming conventions for Python identifiers –
  - Class names start with an uppercase letter. All other identifiers start with a lowercase letter.
  - Starting an identifier with a single leading underscore indicates that the identifier is private.

  Starting an identifier with two leading underscores indicates a strongly private identifier.

  If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Declaring Variable and Assigning Values

- We don't need to declare explicitly variable in Python. When we assign any value to the variable that variable is declared automatically.
- The equal (=) operator is used to assign value to a variable.
- **Eg: a = 10, hobby = 'watching football'**

- **Multiple Assignment**

We can apply multiple assignments in two ways:

1. **Assigning single value to multiple variables: E.g** x=y=z=50
2. **Assigning multiple values to multiple variables: E.g** a,b,c=5,10,15

Python Data Types

- Variables can hold values of different data types.
- Python provides us the **type()** function which returns the type of the variable passed.
- E.g: A=10
- **print**(type(a));
- **Standard data types**
- Python has five standard data types –

1. **Numbers**
2. **String**
3. **List**
4. **Tuple**
5. **Dictionary**

Python Numbers

- E.g; var1 = 1

  var2 = 10
- You can also delete the reference to a number object by using the del statement. The syntax of the del statement is
- E.g; del var1[,var2[,var3[....,varN]]]]
- Python supports four different numerical types –

1. Int: signedintegers
2. Long: longintegers, they can also be represented in octal and hexadecimal
3. Float: floatingpointrealvalues
4. Complex: complexnumbers

- Python allows you to use a lowercase l with long, but it is recommended that you use only an uppercase L to avoid confusion with the number 1. Python displays long integers with an uppercase L. E.g 51924361L, 59004L
- A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x and y are the real numbers and j is the imaginary unit. E.g; 3.14j, 9.322e-36j

Python Numbers

- E.g; var1 = 1

  var2 = 10

- You can also delete the reference to a number object by using the del statement. The syntax of the del statement is
- E.g; del var1[,var2[,var3[….,varN]]]]
- Python supports three different numerical types –

1. Int: signedintegers
2. Float: floatingpointrealvalues
3. Complex: complexnumbers

- A complex number consists of an ordered pair of real floating-point numbers denoted by x + yj, where x and y are the real numbers and j is the imaginary unit. E.g; 3.14j, 9.322e-36j

Python Strings

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.
- Python allows for either pairs of **single** or **double quotes**. Subsets of strings can be taken using the **slice operator[] or [:]** with indexes starting at **0** in the beginning of the string and working their way from **-1** at the end.
- The **plus + sign** is the **string concatenation operator** and the **asterisk *** is the **repetition operator**. For example
- str = 'Hello World!'
- **print** str *# Prints complete string*
- **print** str[0] *# Prints first character of the string*
- **print** str[2:5] *# Prints characters starting from 3rd to 5th*
- **print** str[2:] *# Prints string starting from 3rd character*
- **print** str * 2 *# Prints string two times*
- **print** str + "TEST" *# Prints concatenated string*
- **Print** (str.index('o')) # Prints the index of the first occurrence of char o.

Activity

- Create a python program that collects a full email address and prints out the username and mail server's address separately.

Python Lists

- Lists are the most versatile of Python's **compound data types**. A list contains items separated by commas and enclosed within square brackets[]. To some extent, lists are **similar to arrays in C**. One difference between them is that all the items belonging to a list can be **of different data type**.
- The values stored in a list can be accessed using the **slice operator[]** 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.
- **For example –**
- list = ['abcd', 786 , 2.23, 'john', 70.2]
- tinylist = [123, 'john']

```
print list # Prints complete list
print list[0] # Prints first element of the list
print list[1:3] # Prints elements starting from 2nd till 3rd
print list[2:] # Prints elements starting from 3rd element
print tinylist * 2 # Prints list two times
print list + tinylist # Prints concatenated lists
```

Popular Lists methods

- Len(list) – to return the length of the list
- List.pop(index) - to remove item on specified index
- List.remove(item) – to remove item from the list
- List.clear() – removes all items from list
- List.append(item) – to add new item to list from the last item
- List.insert(index,item) – to add new item to list on specified index
- List.reverse() – to rearrange a list from the last item to the first item
- List.count(item) – to count the number of item in the list.

Activity

Prepare a list of 30 elements holding 15 student's names and ages. Your python script should print out 15 lines of information, each line contains student's name and corresponding age.

Python Tuple

- A tuple is another **sequence data type** that is similar to the list. A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within **parentheses ().**
- The main differences between lists and tuples are: Lists are enclosed in brackets and their elements and size can be changed, while tuples are enclosed in parentheses and cannot be updated. Tuples can be thought of as
- **read-only lists**.
- **For example –**
- tuple = ('abcd', 786 , 2.23, 'john', 70.2)
- tinytuple = (123, 'john')

```
print tuple # Prints complete list
print tuple[0] # Prints first element of the list
print tuple[1:3] # Prints elements starting from 2nd till
                 3rd
print tuple[2:] # Prints elements starting from 3rd
                element
print tinytuple * 2 # Prints list two times
print tuple + tinytuple # Prints concatenated lists
```

Popular Tuple methods

- Len(tuple) – to return the length of the tuple
- Tuple.count(item) – to count the number of item in the list.

Python Set

- The set in python can be defined as the unordered collection of various items enclosed within the curly braces. The elements of the set can not be duplicate. The elements of the python set must be immutable.
- Unlike other collections in python, there is no index attached to the elements of the set, i.e., we cannot directly access any element of the set by the index. However, we can print them all together or we can get the list of elements by looping through the set.
- **For example –**
- Days = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sun day"}
- **print**(Days)
- **print**(type(Days))
- **print**("looping through the set elements ... ")
- **for** i **in** Days:
- **print**(i)

```
Days = set(["Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday", "Sunday"])
print(Days)
print(type(Days))
print("looping through the set elements ... ")
for i in Days:
  print(i)
```

Python Dictionary

- Python's dictionaries are kind of **hash table type.** They work like associative arrays or hashes found in Perl and **consist of key-value pairs**. 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.
- Dictionaries are enclosed by **curly braces {}** and values can be assigned and accessed using **square braces []** .
- **For example –**
- d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'};
- **print**("1st name is "+d[1]);
- **print**("2nd name is "+ d[4]);
- **print** (d);
- **print** (d.keys());
- **print** (d.values());
- tinydict = {'name': 'john','code':6734,
  'dept': 'sales'}

print dict['one'] *# Prints value for 'one' key*
print dict[2] *# Prints value for 2 key*
print tinydict *# Prints complete dictionary*
print tinydict.keys() *# Prints all the keys*
print tinydict.values() *# Prints all the values*

Python Basic Operators

Arithmetic Operators

Operator	Description	Example
+	Addition	C=a+b
-	Subtraction	C=a-b
*	Multiplication	C=a*b
/	Division	C=a/b
%	Modulus	C=a%b
**	Exponent	C=a**b
//	Floor Division	C=a//b

Python Basic Operators

Comparison Operators

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true.
<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Python Basic Operators

Assignment Operators

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b assigns value of a + b into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	c += a is equivalent to c = c + a
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	c -= a is equivalent to c = c - a
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	c *= a is equivalent to c = c * a
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	c /= a is equivalent to c = c / a
%= Modulus AND	It takes modulus using two operands and assign the result to left operand	c %= a is equivalent to c = c % a
**= Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	c **= a is equivalent to c = c ** a
//= Floor Division	It performs floor division on operators and assign value to the left operand	c //= a is equivalent to c = c // a

Python Basic Operators

Logical Operators

Operator	Description	Example
And Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

Python Basic Operators

Membership Operators

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Python Basic Operators

Identity Operators

Operator	Description	Example
is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here **is** results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here **is not** results in 1 if id(x) is not equal to id(y).

Lines and Indentation

- Python provides no braces to indicate blocks of code for class and function definitions or flow control.
- Blocks of code are denoted by line indentation, which is rigidly enforced.
- The number of spaces in the indentation is variable, but all statements within the block must be indented the
- same amount. For example –
- if True:

```
        print "True"
  else:
        print "False"
```

Multi-Line Statements

- Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation
- character (\) to denote that the line should continue.

For example –

```
total = item_one + \
item_two + \
item_three
```

- Note however that Statements contained within the [], {}, or brackets do not need to use the line continuation character.
- For example –

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

Quotation in Python

- Python accepts single , double and triple quotes to denote string literals, as long as the same type of
- quote starts and ends the string.
- The triple quotes are used to span the string across multiple lines. For example, all the following are legal
- For example:

```
word = 'word'

sentence = "This is a sentence."

paragraph = """This is a paragraph. It is

                made up of multiple lines and sentences."""
```

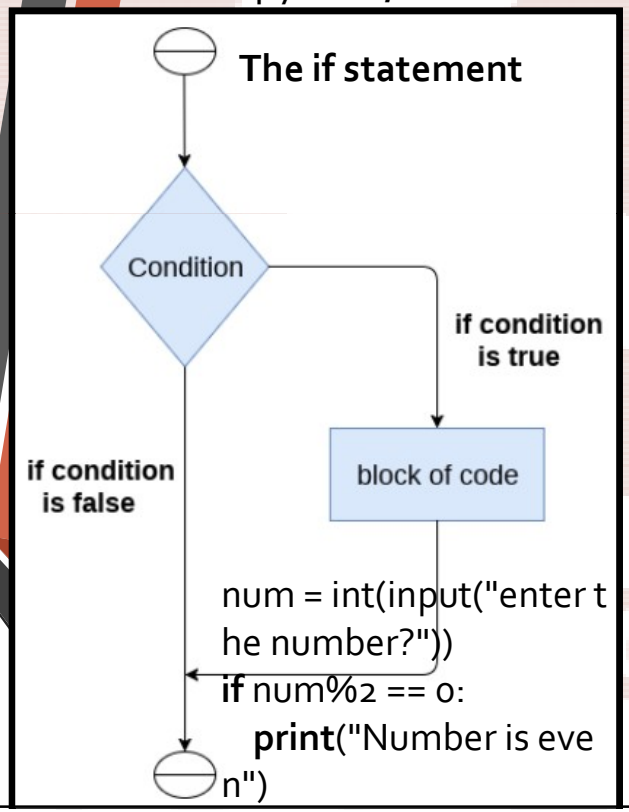
Comments in Python

- A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.
- Example:

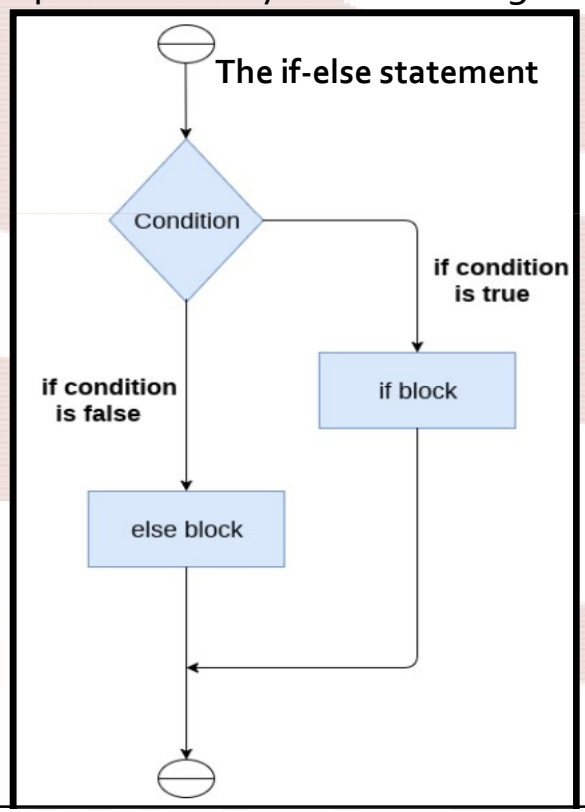
```
# First comment
print "Hello, Python!" # second comment
```

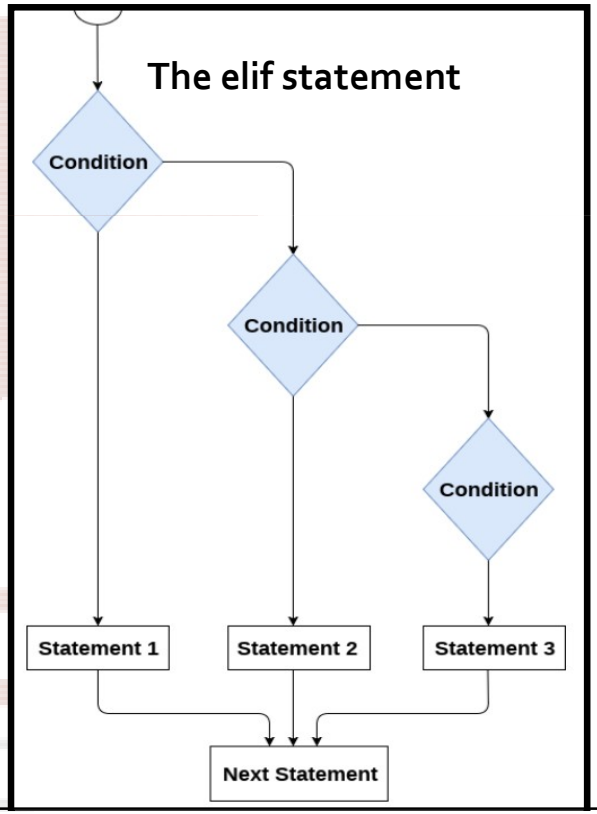
Python If-else statements

- Decision making allows us to run a particular block of code for a particular decision. Here, the decisions are made on the validity of the particular conditions. Condition checking is the backbone of decision making.
- In python, decision making is performed by the following statements.




```
num = int(input("enter the number?"))
if num%2 == 0:
    print("Number is even")
```



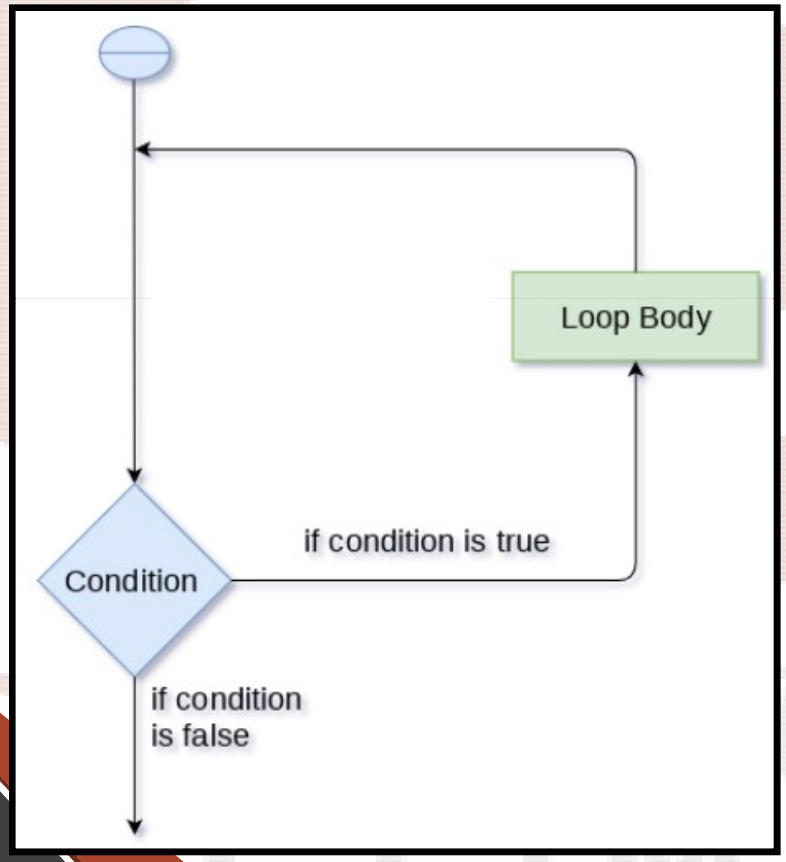


Implementations of IF-else statements

1. Program to check whether a person is eligible to vote or not.
2. Program to print the largest of the three numbers.
3. Program to check if an email address entered is valid or not.

Python Loops

- The programming languages provide various types of loops which are capable of repeating some specific code several numbers of times. Consider the following diagram to understand the working of a loop statement.



Advantages of loops

There are the following advantages of loops in Python.

1. It provides code re-usability.
2. Using loops, we do not need to write the same code again and again.
3. Using loops, we can traverse over the elements of data structures (array or linked lists).

Types of loops

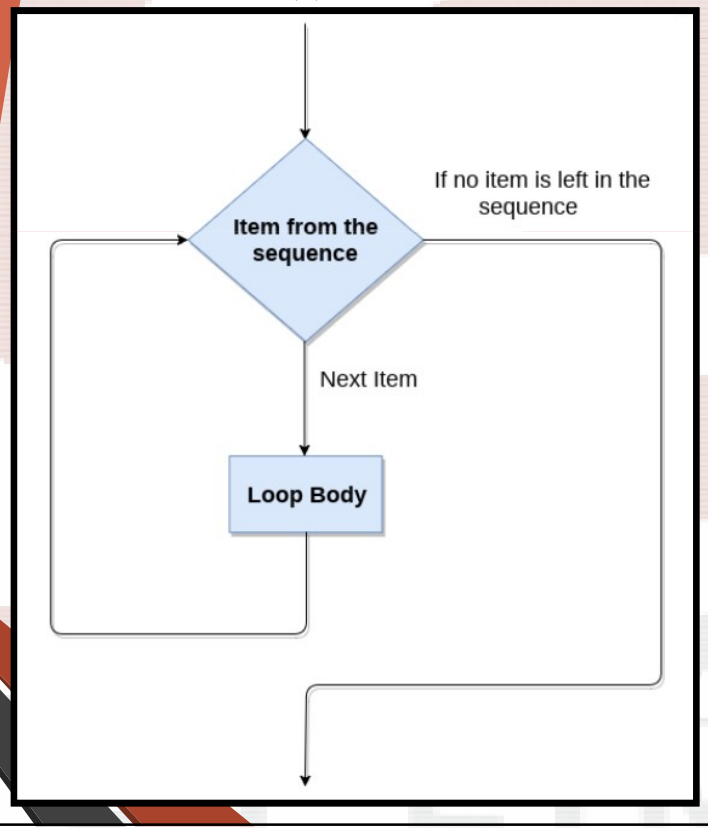
We have three types of loops which include:

1. For-loops
2. While loops
3. Do –while loops

Python For-Loops

- The for **loop in Python** is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple, or dictionary.
- The syntax of for loop
- **for** iterating_var **in** sequence:

  statement(s)



Example

```
i=1
n=int(input("Enter the number up to which you want to print the natural numbers?"))
for i in range(0,10):
  print(i,end = ' ')
```

Example 2

```
i=1;
num = int(input("Enter a number:"));
for i in range(1,11):
  print("%d X %d = %d"%(num,i,num*i));
```

Nested for loop

- Python allows us to nest any number of for loops inside a for loop. The inner loop is executed n number of times for every iteration of the outer loop. The syntax of the nested for loop in python is given below.
- **for** iterating_var1 **in** sequence:
- **for** iterating_var2 **in** sequence:
- #block of statements
- #Other statements

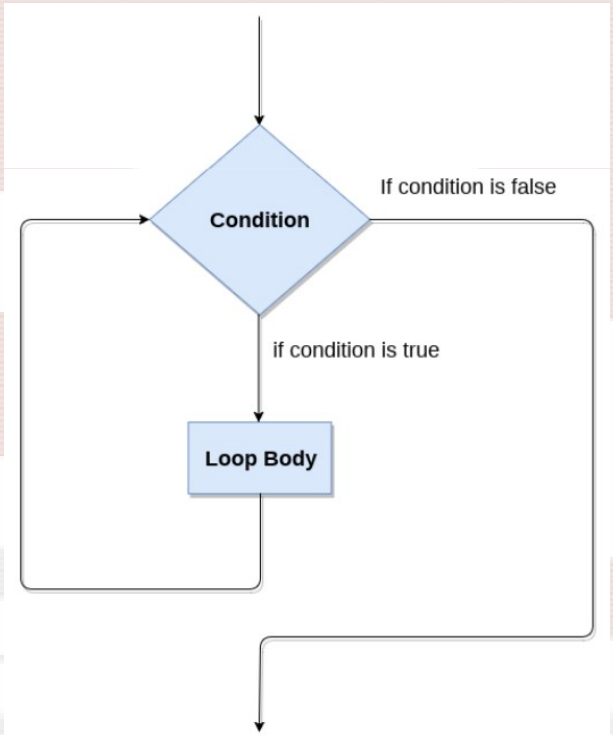
Activity

- Write a python script to create a list of at least 10 students in a department and another list of at least 5 courses offered in the department. Allocate 3 courses to the first 3 students on the list and 4 courses to the rest of students in the list.

Python while loop

- The while loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed as long as the given condition is true. It can be viewed as a repeating if statement. The while loop is mostly used in the case where the number of iterations is not known in advance.
- The syntax is given below.
- **while** expression:

  statements



Example 1:

```
i=1
while i<=10:
   print(i)
   i=i+1
```

Example 2:

```
i=1
number = int(input("Enter the number?"))
while i<=10:
   print("%d X %d = %d \n"%(number,i,number*i));
   i = i+1;
```

Using else with Python while loop

- Python enables us to use the while loop with the while loop also. The else block is executed when the condition given in the while statement becomes false. Like for loop, if the while loop is broken using break statement, then the else block will not be executed and the statement present after else block will be executed.
- Example:
- i=1;
- **while** i<=5:
- **print**(i)
- i=i+1;
- **else**:**print**("The while loop exhausted");

Python break statement

- The break is a keyword in python which is used to bring the program control out of the loop. The break statement breaks the loops one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. In other words, we can say that break is used to abort the current execution of the program and the control goes to the next line after the loop. The break is commonly used in the cases where we need to break the loop for a given condition. The syntax of the break is given below.
- #loop statements
- **break**;

Example:

```
i = 0;
while 1:
   print(i," ",end=""),
   i=i+1;
   if i == 10:
      break;
print("came out of while loop");
```

Python continue Statement

- The continue statement in python is used to bring the program control to the beginning of the loop. The continue statement skips the remaining lines of code inside the loop and start with the next iteration. It is mainly used for a particular condition inside the loop so that we can skip some specific code for a particular condition.
- The syntax of Python continue statement is given below.
- #loop statements
- **continue**;
- #the code to be skipped

Example:

```
i=1; #initializing a local variable
#starting a loop from 1 to 10
for i in range(1,11):
   if i==5:
      continue;
   print("%d"%i);
```

Python Functions

- Functions are the most important aspect of an application. A function can be defined as the organized block of reusable code which can be called whenever required.
- Python allows us to divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the python program.
- **Advantage of functions in python**
- There are the following advantages of C functions.
- By using functions, we can avoid rewriting same logic/code again and again in a program.
- We can call python functions any number of times in a program and from any place in a program.
- We can track a large python program easily when it is divided into multiple functions.
- Reusability is the main achievement of python functions.
- However, Function calling is always overhead in a python program.

Creating a function

In python, we can use **def** keyword to define the function. The syntax to define a function in python is given below.

```
def my_function():
    function-suite
    return <expression>
```

- **Function calling**
- In python, a function must be defined before the function calling otherwise the python interpreter gives an error. Once the function is defined, we can call it from another function or the python prompt. To call the function, use the function name followed by the parentheses.

Example 1

```
def hello_name():
        Myname = Input("what is your name")
        print("hello  ", Myname)
  hello_name()
```

Example 2

```
#defining the function
def func (name):   #parameter
   print("Hi ",name);

#calling the function
func("Ayush")
```

Example 3

```
#python function to calculate the sum of two variables
#defining the function
def sum (a,b):
   return a+b;  #returns a value

#taking values from the user
a = int(input("Enter a: "))
b = int(input("Enter b: "))

#printing the sum of a and b
print("Sum = ",sum(a,b))
```

- **Python Built-in Functions**
- The Python built-in functions are defined as the functions whose functionality is pre-defined in Python. The python interpreter has several functions that are always present for use. These functions are known as Built-in Functions. There are several built-in functions in Python which are listed below:

The python **all()** function accepts an iterable object (such as list, dictionary, etc.). It returns true if all items in passed iterable are true. Otherwise, it returns False. E.g;

```
# all values true
k = [1, 3, 4, 6]
print(all(k))
```

The python **bin()** function is used to return the binary representation of a specified integer. A result always starts with the prefix ob. E.g;

```
x = 10
y = bin(x)
print (y)
```

The python **compile()** function takes source code as input and returns a code object which can later be executed by exec() function.

```
# compile string source to code
code_str = 'x=5\ny=10\nprint("sum =",x+y)'
code = compile(code_str, 'sum.py', 'exec')
print(type(code))
exec(code)
```

The python **exec()** function is used for the dynamic execution of Python program which can either be a string or object code and it accepts large blocks of code, unlike the eval() function which only accepts a single expression.

```
x = 8
exec('print(x==8)')
```

- **Python Built-in Functions cont.**

As the name says, python **sum()** function is used to get the sum of numbers of an iterable, i.e., list.

```
s = sum([1, 2,4 ])
print(s)
```

The python **float()** function returns a floating-point number from a number or string.

The python **format()** function returns a formatted representation of the given value.

```
# d, f and b are a type
 # integer
print(format(123, "d"))
 # float arguments
print(format(123.4567898, "f"))
 # binary format
print(format(12, "b"))
```

The python **len()** function is used to return the length (the number of items) of an object.

```
strA = 'Python'
print(len(strA))
```

The python **list()** creates a list in python.

```
# string
String = 'abcde'
print(list(String))
 # tuple
Tuple = (1,2,3,4,5)
print(list(Tuple))
```

The python **open()** function opens the file and returns a corresponding file object.

```
# opens python.text file of the current directory
f = open("python.txt")
# specifying full path
f = open("C:/Python33/README.txt")
```

Python **help()** function is used to get help related to the object passed during the call. It takes an optional parameter and returns help information. For e.g;

```
Help(str)
```

Python **min()** function is used to get the smallest element from the collection. E.g;

```
small = min(2225,325,2025)
Print(small)
```

Python **sorted()** function is used to sort elements. By default, it sorts elements in an ascending order but can be sorted in descending also

References

- Guido V. R. et al(2018), Python Tutorial. Python Software Foundation.
- https://www.tutorialspoint.com/history-of-python
- https://www.guru99.com/variables-in-python.html#:~:text=A%20Python%20variable%20is%20a,to%20the%20computer%20for%20processing.

Thanks for viewing