FYBCA-SEM2-204 - Programming Skills

UNIT-2: Python Fundamentals

What is Python?

- Python is a general purpose, dynamic, high-level, and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.
- Python is easy to learn yet powerful and versatile scripting language, which makes it attractive for Application Development.
- Python's syntax and dynamic typing with its interpreted nature make it an ideal language for scripting and rapid application development.
- Python supports multiple programming patterns, including object-oriented, imperative, and functional or procedural programming styles.
- Python is not intended to work in a particular area, such as web programming. That is why it is known as multipurpose programming language because it can be used with web, enterprise, 3D CAD, etc.
- We don't need to use data types to declare variable because it is dynamically typed so we can write a=10 to assign an integer value in an integer variable.
- Python makes the development and debugging fast because there is no compilation step included in Python development, and edit-test-debug cycle is very fast.

Python History

Python was invented by **Guido van Rossum in 1991** at CWI in Netherland. The idea of Python programming language has taken from the ABC programming language or we can say that ABC is a predecessor of Python language.

There is also a fact behind the choosing name Python. Guido van Rossum was a fan of the popular BBC comedy show of that time, "Monty Python's Flying Circus". So he decided to pick the name Python for his newly created programming language.

Where is Python used?

Python is a general-purpose, popular programming language and it is used in almost every technical field. The various areas of Python use are given below.

- Data Science
- Date Mining
- Desktop Applications
- Console-based Applications
- Mobile Applications
- Software Development
- Artificial Intelligence
- Web Applications
- Enterprise Applications
- 3D CAD Applications
- Machine Learning
- Computer Vision or Image Processing Applications.
- Speech Recognitions

Python Popular Frameworks and Libraries

Python has wide range of libraries and frameworks widely used in various fields such as machine learning, artificial intelligence, web applications, etc. We define some popular frameworks and libraries of Python as follows.

1. Web development (Server-side) - Django Flask, Pyramid, CherryPy

2. GUIs based applications - Tk, PyGTK, PyQt, PyJs, etc.
3. Machine Learning - TensorFlow, PyTorch, Scikit-learn, Matplotlib, Scipy, etc.
4. Mathematics - Numpy, Pandas, etc.

2.1 Concepts of Interpreter based programming language:

- An Interpreter directly executes instructions line by line written in a programming or scriptinglanguage without converting them to an object code or machine code.
- Examples of interpreted languages are Perl, Python and Matlab.

2.1.1 Structure of Python Programming language.

- Basic Structure of python programming includes following components:

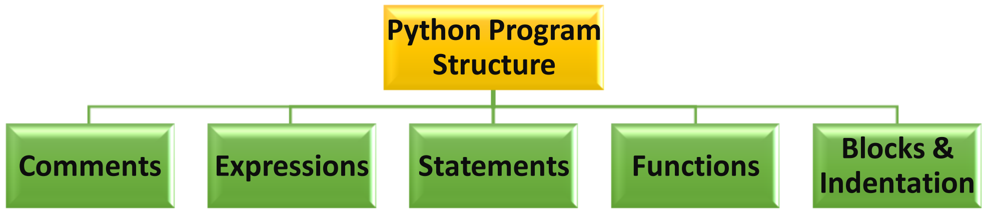


Figure: Structure of Python Program

- **Comments**

Comments are the additional readable information to get better understanding about the source code.
Comments in Python are the non-executable statements.
Comments are of 2 types:

1. Single Line Comments: which begin with a hash symbol (#).
   E.g. #This is a sample python program
2. Multiline Comments: which begins with '''and ends with ''' (3 single quotes)
   E.g. '''This is a sample python program1
   This is a sample python program2'''

- **Expression:**

An expression is any legal combination of symbols that represents a value.
An expression represents something which python evaluates and which produces a value.
E.g. 10, x + 5

- **Statements:**

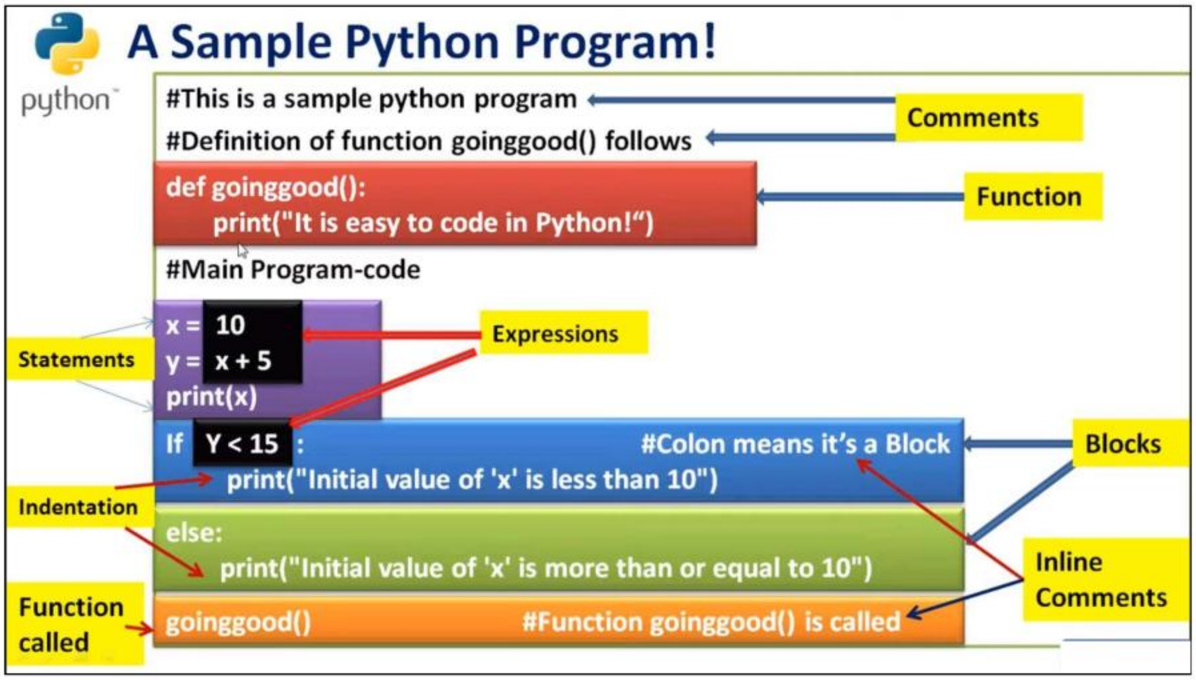
A statement is a programming instruction that does something i.e. some action takes place.
A statement executes and may or may not results in a value.
E.g. print(x + 2), y = x + 5, x = 10

- **Functions:**

A function is a code that has a name and it can be reused (executed again) by specifying its name in the program, where needed.
A function begin with ‘def’ statement
E.g. defCalculate_Sum():

❖ **Block and Indentation:**

A group of statements which are part of another statement or a function are called block or Code-block or suite in python.
Indentation is used to show blocks in python. Four spaces together mark the next indent-level.



```
#This is a sample python program
#Definition of function goinggood() follows
def goinggood():
    print("It is easy to code in Python!")
#Main Program-code
x = 10
y = x + 5
print(x)
If Y < 15 :                                   #Colon means it's a Block
    print("Initial value of 'x' is less than 10")
else:
    print("Initial value of 'x' is more than or equal to 10")
goinggood()                  #Function goinggood() is called
```

❖ **Python code Indention and execution**

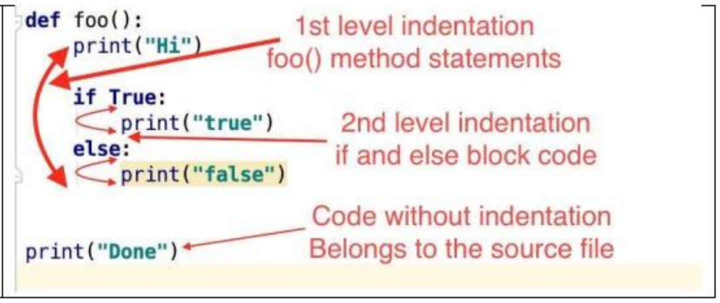
➢ Indentation refers to the spaces at the beginning of a code line.
➢ Where in other programming languages the indentation in code is for readability only, the indentation in Python is very important.
➢ Python uses indentation to indicate a block of code.
➢ The leading whitespaces (space and tabs) at the start of a line is used to determine the indentation level of the line.
➢ Increase the indent level to group the statements for that code block. Similarly, reduce the indentation to close the grouping.

Example:

```
def foo():
    print("Hi")

    if True:
        print("true")
    else:
        print("false")

print("Done")
```



```
def foo():
    print("Hi")
    if True:
        print("true")
    else:
        print("false")

print("Done")
```

❖ **Python Indentation Rules:**

➢ We can't split indentation into multiple lines using backslash.
➢ The first line of Python code can't have indentation, it will throw IndentationError.

- You should avoid mixing tabs and whitespaces to create indentation.
- It is preferred to use whitespaces for indentation than the tab character.
- The best practice is to use **4 whitespaces** for first indentation and then keep adding additional 4 whitespaces to increase the indentation.

2.2 Python Variables

- **Variables:**

Variables are containers for storing data values. Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.

Rules for Python Variable:

- A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).
- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables).

Valid Variable Name:	Invalid Variable Name:
`myvar = "John"`	`2myvar = "John"`
`my_var = "John"`	`my-var = "John"`
`_my_var = "John"`	`my var = "John"`
`myVar = "John"`	
`MYVAR = "John"`	
`myvar2 = "John"`	

- **Creating Variables**

- Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.
- Python is completely object oriented, and not "statically typed". You do not need to declare variables before using them, or declare their type. Every variable in Python is an object.
- Variables do not need to be declared with any particular type, and can even change type after they have been set.

Example:

```
x = 4       # x is of type int
x = "John" # x is now of type str
print(x)
```

Note: String variables can be declared either by using single or double quotes.

Many Values to Multiple Variables

Python allows you to assign values to multiple variables in one line:

x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)
	Output:
	Orange
	Banana
	Cherry

One Value to Multiple Variables

We can assign the same value to multiple variables in one line:

Example:

x, y, z = "Orange"
print(x)
print(y)
print(z)
	Output:
	Orange
	Orange
	Orange

2.3 Python Datatypes:

2.3.1 Text (str), Numeric Type(int, float, complex), Boolean (bool)

- In programming, data type is an important concept
- Variables can store data of different types, and different types can do different things.
- Python has the following data types built-in by default, in these categories:

Data Types	Keywords
Text Types:	str
Numeric Types:	int, float, complex
Boolean Type:	Bool

Getting the Data Type

You can get the data type of any object by using the type() function:

Example:

x = 5
print(type(x))
	Output:
	<class 'int'>

2.3.2 Setting Datatypes

Setting the Data Type

In Python, the data type is set when you assign a value to a variable:

Example:	**Data Types**
x = "Hello World"	str
x = 20	int
x = 20.5	float
x = 1j	complex
x = True	bool

2.3.3 Type conversion (int, float, complex), casting (int, float,str)

Type Conversions and Casting:

If you want to specify the data type of a variable, this can be done with casting.

```
x = str(3) # x will be '3'
y = int(3) # y will be 3
z = float(3) # z will be 3.0
```

Other Examples:

Example	**Data Type**
x = str("Hello World")	Str
x = int(20)	Int
x = float(20.5)	Float
x = complex(1j)	Complex
x = bool(5)	Bool

Python Casting

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using functions as follows:

1. int() – an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)
2. float() - a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)
3. str() - a string from a wide variety of data types, including strings, integer literals and float literals

Example:

```
int()
x = int(1) # x will be 1
y = int(2.8) # y will be 2
z = int("3") # z will be 3
float()
x = float(1) # x will be 1.0
y = float(2.8) # y will be 2.8
z = float("3") # z will be 3.0
w = float("4.2") # w will be 4.2
str()
x = str("s1") # x will be 's1'
y = str(2) # y will be '2'
z = str(3.0) # z will be '3.0'
```

2.4 User defined function (UDF)

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- A function can return data as a result.

2.4.1 Defining function, Function with Parameters

Creating a Function

In Python a function is defined using the def keyword:

Syntax:

```
deffunction_name(Parameter List):
        function body
```

Example:

```
def  my_function():
        print("Hello from a function")
```

Calling a Function

To call a function, use the function name followed by parenthesis:

Example:

```
def   my_function():
        print("Hello from a function")


my_function() #function call(add two enter key after function body)
```

2.4.2 Parameter with default value, Function with return value

Parameter (Arguments)

Information can be passed into functions as arguments.

- Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.
- The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

Example:

```
def my_function(fname):
        print(‟ Welcome ” + fname )


my_function("Amar")
my_function("Parth")
my_function("Dhruvi")
```

Output:

```
Welcome Amar
Welcome Parth
Welcome Dhruvi
```

Number of Arguments

By default, a function must be called with the correct number of arguments. Meaning that if your function expects 2 arguments, you must call the function with 2 arguments, not more, and not less

Example : (This function expects 2 arguments, and gets 2 arguments:)

```
def my_function(fname, lname):
        print(fname + " " + lname)


my_function("Amar ", " Patel")
```

Output:

```
Amar Patel
```

Function with Default Parameter Value

- The following example shows how to use a default parameter value.
- If we call the function without argument, it uses the default value:

Example:

```
def my_function(country = "Norway"):
        print("I am from " + country)


my_function("Sweden")
my_function("India")
my_function()        #this function will take default value
my_function("Brazil")
```

Output:

```
I am from Sweden
I am from India
I am from Norway
I am from Brazil
```

Function with Return Values

To let a function return a value, use the return statement:

Example:

```
def my_function(x):
        return 7 * x


print(my_function(3))
print(my_function(5))
print(my_function(9))
```

Output:

```
21
35
63
```

Global Variables

- Variables that are created outside of a function are known as global variables.
- Global variables can be used by everyone, both inside of functions and outside.

Example:

```
x = "Object Oriented Programming language" #Global Variable

def myfunc():
        print("Python is " + x)# Printing global Variable value


myfunc()            #function call
```

Output:

```
Python is Object Oriented Programming language
```

Note: If you create a variable with the same name inside a function, this variable will be local, and can only be used inside the function. The global variable with the same name will remain as it was, global and with the original value.

Example:

```
x = "awesome" #global variable

def myfunc():
        x = "fantastic"
        print("Python is " + x) #local variable

myfunc() #function call
print("Python is " + x) #printing global variable
```

Output:

```
Python is fantastic
Python is awesome
```

The global Keyword

- Normally, when you create a variable inside a function, that variable is local, and can only be used inside that function.
- To create a global variable inside a function, you can use the global keyword.

Example: If you use the global keyword, the variable belongs to the global scope:

`def myfunc():`
`        global x`
`        x = "fantastic"`
 	
`myfunc() #function call`
`print("Python is " + x)`
	Output:
	Python is fantastic

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

Example:

`x = "awesome"`
 	
`def myfunc():`
`        global x`
`        x = "fantastic"`
 	
`myfunc() #function call`
`print("Python is " + x)`
	Output:
	Python is fantastic