MODULE-1

PPT-2

- Executing Python from the Command Line
- Editing Python Files (by using text editor or by using IDE)

Basic Python Syntax

- Input and output are distinguished by the presence or absence of prompts (>>> and …): to repeat the example, you must type everything after the prompt, when the prompt appears; lines that do not begin with a prompt are output from the interpreter.
- >>> is called a prompt, which means it's something the computer displays to tell you it's ready for your instructions. You can type things into that window, and the computer will obey them when it understands your commands.

Comments

- Comments in Python start with the hash character, #, and extend to the end of the physical line. A comment may appear at the start of a line or following whitespace or code, but not within a string literal. A hash character within a string literal is just a hash character.

```
C:\Users\PRIYANKA>python
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> # My name is priyanka Singh
>>>
```

- Comments starts with a #, and Python will ignore them:

```
#This is a comment
print("Hello, World!")
```

- Comments can be placed at the end of a line, and Python will ignore the rest of the line:

```
print("Hello, World!") #This is a comment
```

- Comments does not have to be text to explain the code, it can also be used to prevent Python from executing code:

```
#print("Hello, World!")
print("Cheers, Mate!")
```

- Multiple comments- If more than one consecutive line are to be commented, # symbol must be put at beginning of each line

```
##comment1
##comment2
##comment3
 print ("Hello World")
```

A triple quoted multi-line string is also treated as comment if it is not a docstring of a function or class.

'''comment1

comment2

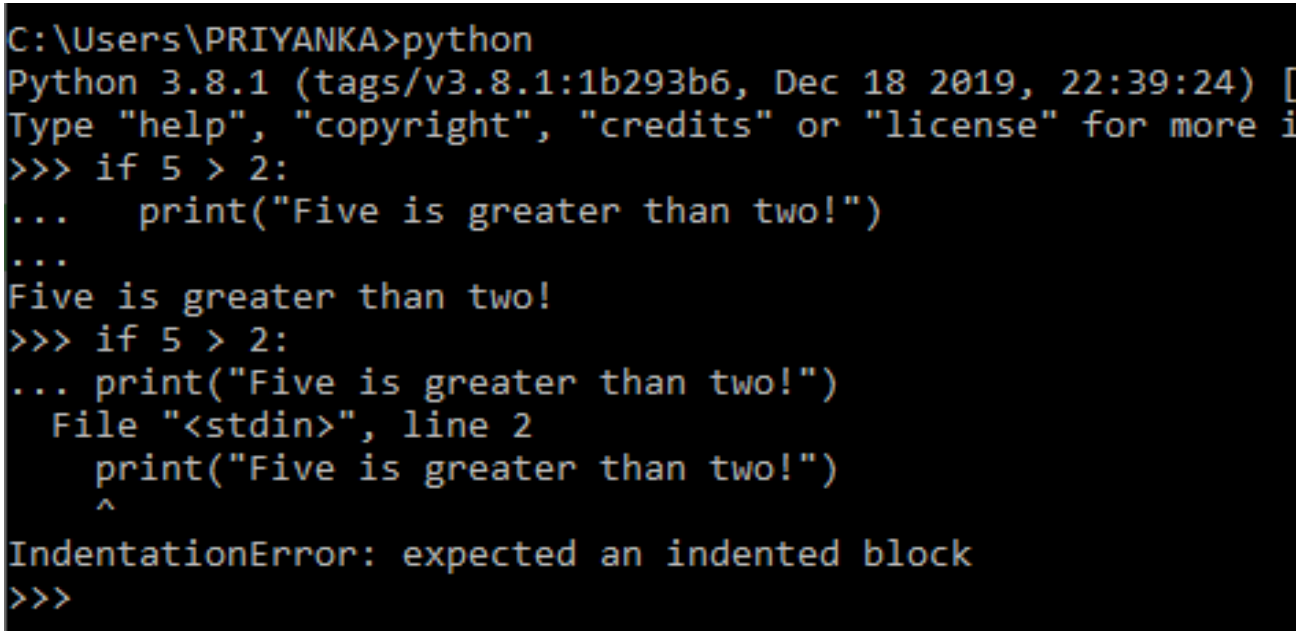
comment3'''

print ("Hello World")

```
C:\Users\PRIYANKA>python
Python 3.8.1 (tags/v3.8.1:1b293b6, Dec 18 2019, 22:39:24) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> #This is a comment
>>> print("Hello, World!")
Hello, World!
>>> print("Hello, World!") #This is a comment
Hello, World!
>>> #print("Hello, World!")
>>> print("Cheers, Mate!")
Cheers, Mate!
>>> ##comment1
>>> ##comment2
>>> ##comment3
>>> print("Hello World")
Hello World
>>> '''comment1
... comment2
... comment3'''
'comment1\ncomment2\ncomment3'
>>> print ("Hello World")
Hello World
```

Python Indentation

- 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 number of spaces is up to you as a programmer, but it has to be at least one.

```
>>> if 5>2:
...  print("hi")
...
hi
>>> if 5>2:
...     print("hello")
...
hello
```

- You have to use the same number of spaces in the same block of code, otherwise Python will give you an error:

```
>>> if 5>2:
...  print("hi")
...     print("hello")
  File "<stdin>", line 3
    print("hello")
    ^
IndentationError: unexpected indent
```

Python Variables

- Variables are containers for storing data values.
- Unlike other programming languages, Python has no command for declaring a variable.
- A variable is created the moment you first assign a value to it.

```
x = 5
y = "John"
print(x)
print(y)
```

- Variables do not need to be declared with any particular type and can even change type after they have been set.

```
x = 4
x = "Sally"
print(x)
```

Variable Names

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

#Legal variable names:

myvar = "John"

my_var = "John"

_my_var = "John"

myVar = "John"

MYVAR = "John"

myvar2 = "John"

#Illegal variable names:

2myvar = "John"

my-var = "John"

my var = "John"

```
>>> #Legal variable names:
>>> myvar = "John"
>>> my_var = "John"
>>> _my_var = "John"
>>> myVar = "John"
>>> MYVAR = "John"
>>> myvar2 = "John"
>>>
>>> #Illegal variable names:
>>> 2myvar = "John"
  File "<stdin>", line 1
    2myvar = "John"
     ^
SyntaxError: invalid syntax
>>> my-var = "John"
  File "<stdin>", line 1
SyntaxError: cannot assign to operator
>>> my var = "John"
  File "<stdin>", line 1
    my var = "John"
       ^
SyntaxError: invalid syntax
```

Assign Value to Multiple Variables

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

```
x, y, z = "Mukesh", "Vivek", "Mamta"
print(x)
print(y)
print(z)
```

- And you can assign the same value to multiple variables in one line:

```
x = y = z = "Priyanka"
print(x)
print(y)
print(z)
```

Output Variables

- The Python print statement is often used to output variables.
- To combine both text and a variable, Python uses the + character
- You can also use the + character to add a variable to another variable:

```
>>> x = "Priyanka Singh"
>>> print("My name is " + x)
My name is Priyanka Singh
>>> x="Priyanka"
>>> y="Singh"
>>> z=x+y
>>> print(z)
PriyankaSingh
```

- For numbers, the + character works as a mathematical operator:

```
x = 5
y = 10
print(x + y)
```

- If you try to combine a string and a number, Python will give you an error:

```
x = 5
y = "Priyanka"
print(x + y)  # error
```

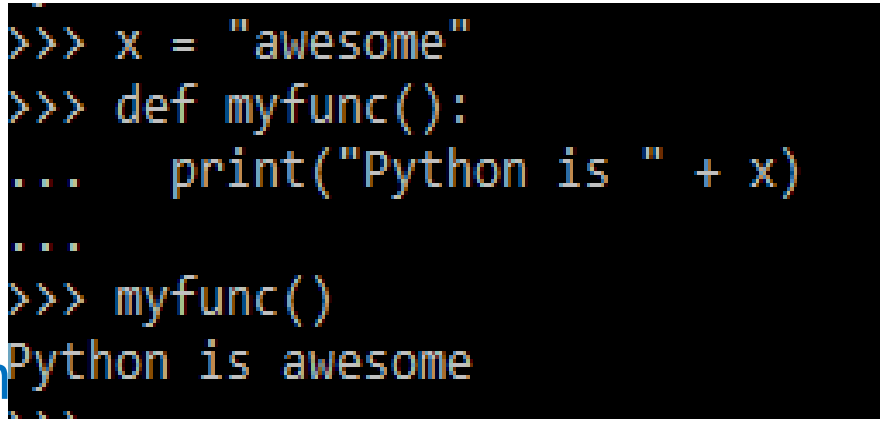
Global Variables

- Variables that are created outside of a function (as in all of the examples above) are known as global variables.
- Global variables can be used by everyone, both inside of functions and outside.
- Example
- Create a variable outside of a function, and use it inside the function

```
x = "awesome" # global variable

def myfunc():
  print("Python is " + x)


myfunc() # execute function nam
```



- 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
- Create a variable inside a function, with the same name as the global variable

```
x = "awesome"
def myfunc():
  x = "fantastic"
  print("Python is " + x)
myfunc()
print("Python is " + x)
```

```
>>> x = "awesome"
>>> def myfunc():
...   x = "fantastic"
...   print("Python is " + x)
...
>>> myfunc()
Python is fantastic
>>> print("Python is " + x)
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 = "Priyanka Singh"
myfunc()
print("My name is " + x)
```

```
>>> def myfunc():
...     global x
...     x = "Priyanka Singh"
...
>>> myfunc()
>>> print("My name is " + x)
My name is Priyanka Singh
>>>
```

- Also, use the global keyword if you want to change a global variable inside a function.

```
x = "awesome"
def myfunc():
  global x
  x = "fantastic"
myfunc()
print("My name is " + x)
```

```
>>> x = "awesome"
>>>
>>> def myfunc():
...   global x
...   x = "fantastic"
...
>>> myfunc()
>>> print("My name is " + x)
My name is fantastic
>>>
```