Introduction to Python

A **computer program** is a collection of instructions that perform a specific task when executed by a computer. It is usually written by a computer program in a programming language.

Python is a powerful high-level, object-oriented programming language created by Guido van Rossum. It has simple easy-to-use syntax, making it the perfect language for someone trying to learn computer programming for the first time.

Python is a cross-platform programming language, meaning, it runs on multiple platforms like Windows, MacOS, Linux and has even been ported to the Java and .NET virtual machines.

To write and run Python program, we need to have Python interpreter installed in our computer.

INTRODUCTION TO PYTHON PROGRAMMING

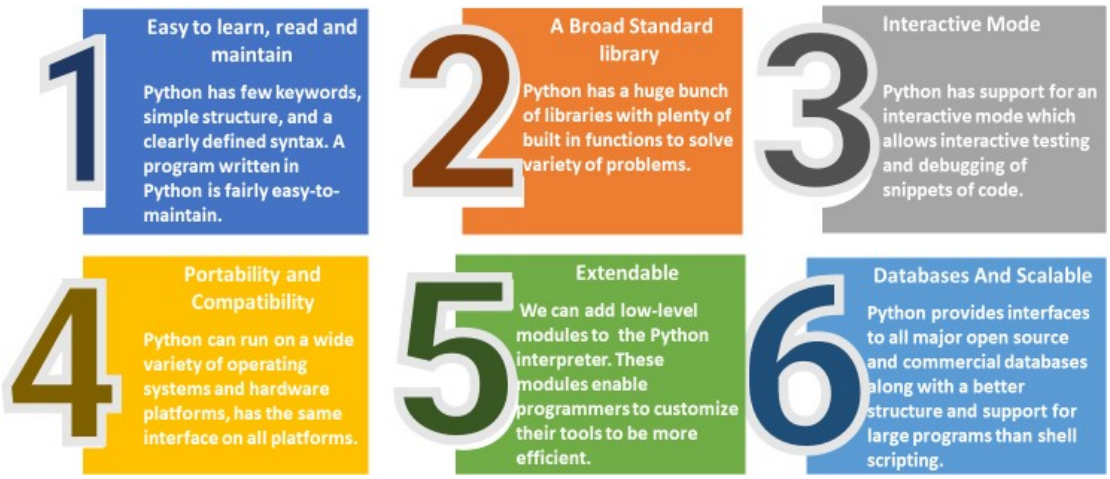
- Python is a high-level, general-purpose, interpreted, interactive and object-oriented programming language.
- It was created by Guido van Rossum during 1985- 1990.
- Like Perl, Python source code is also available under the GNU General Public License (GPL).

Why the name python?

Guido van Rossum was interested on watching a comedy show, which is telecasting on the BBC channel from 1969 to 1974 The complete Monty Python's Flying Circus.

Guido Van Rossum thought he needed a name that was short, unique, ands lightly mysterious for his programming language, so he decided to call the language Python.

Python gains a maximum popularity because of the following reasons:



Applications of Python

- Web Development
- Game Development

- Scientific and Numeric applications
- Artificial Intelligence and Machine Learning based applications
- Data Science related applications
- Desktop GUI applications
- Software Development
- Enterprise-level/Business Applications
- Education programs and training courses
- Web Scraping Applications
- Image Processing and Graphic Design Applications
- Data Analysis

Getting started with Python

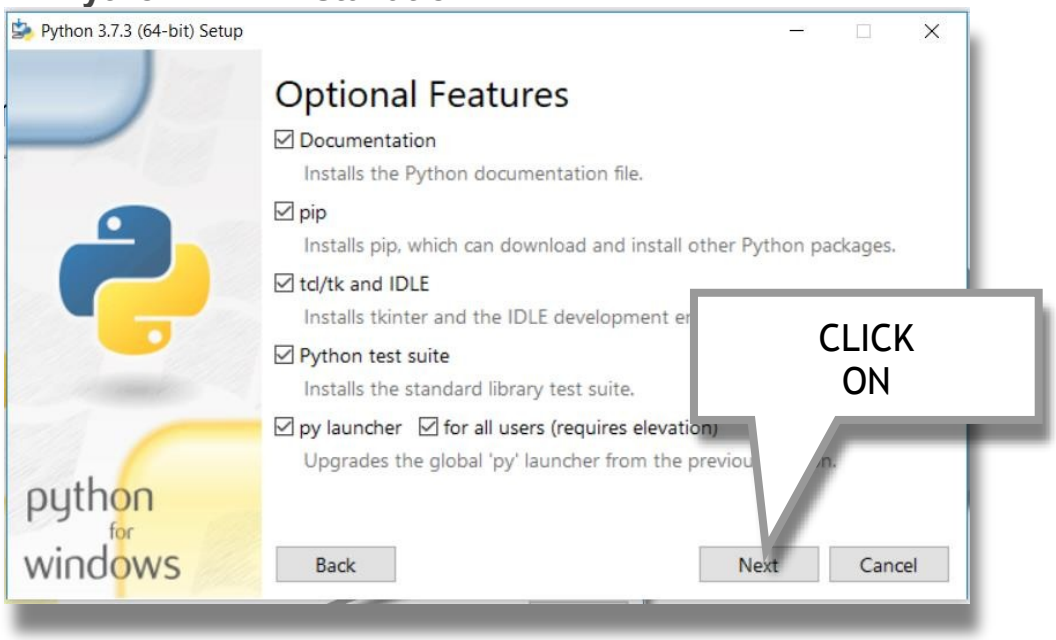
Python is a cross-platform programming language, meaning, it runs on multiple platforms like Windows, MacOS, Linux and has even been ported to the Java and .NET virtual machines.

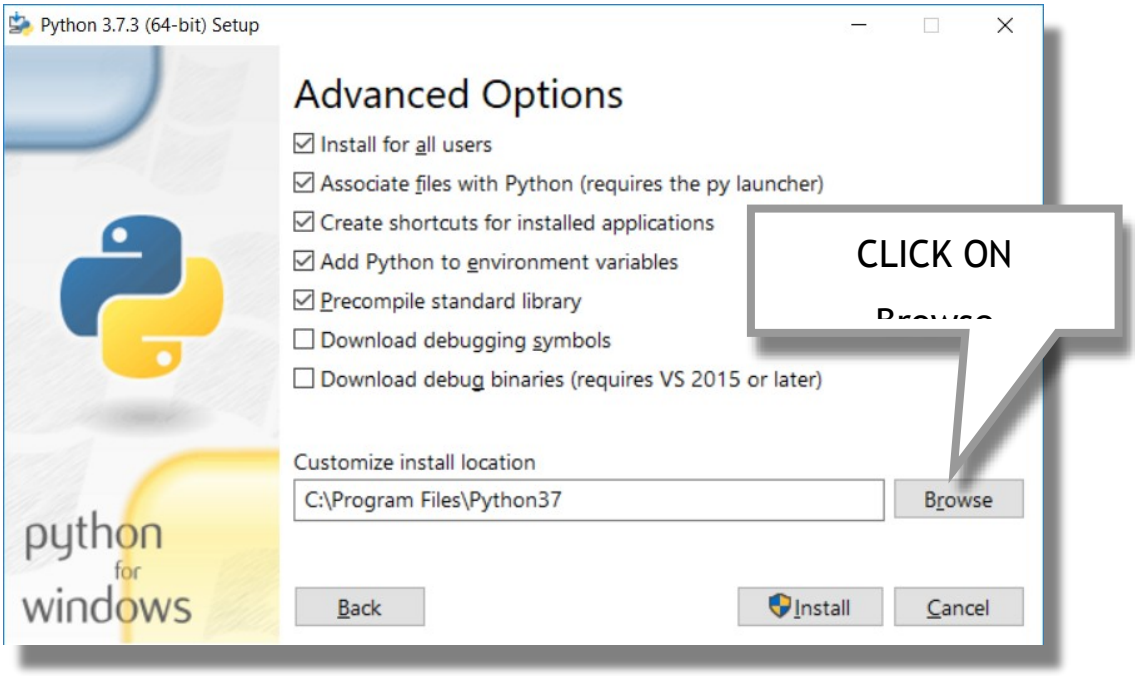
To write and run Python program, we need to have Python interpreter installed in our computer.

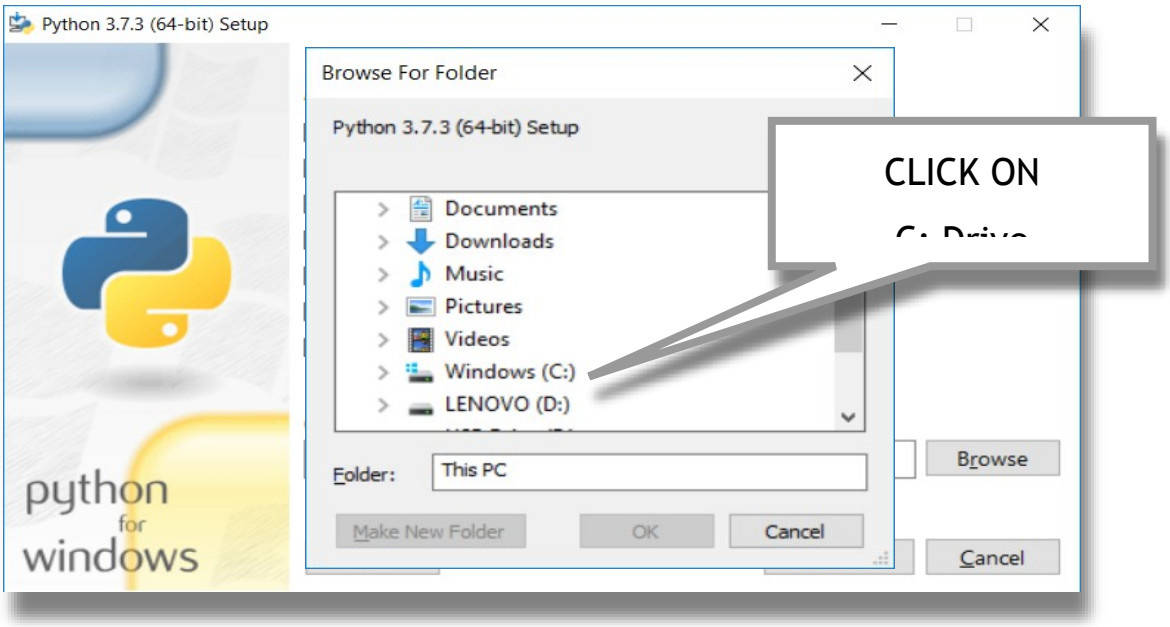
Downloading and Setting up Python for use

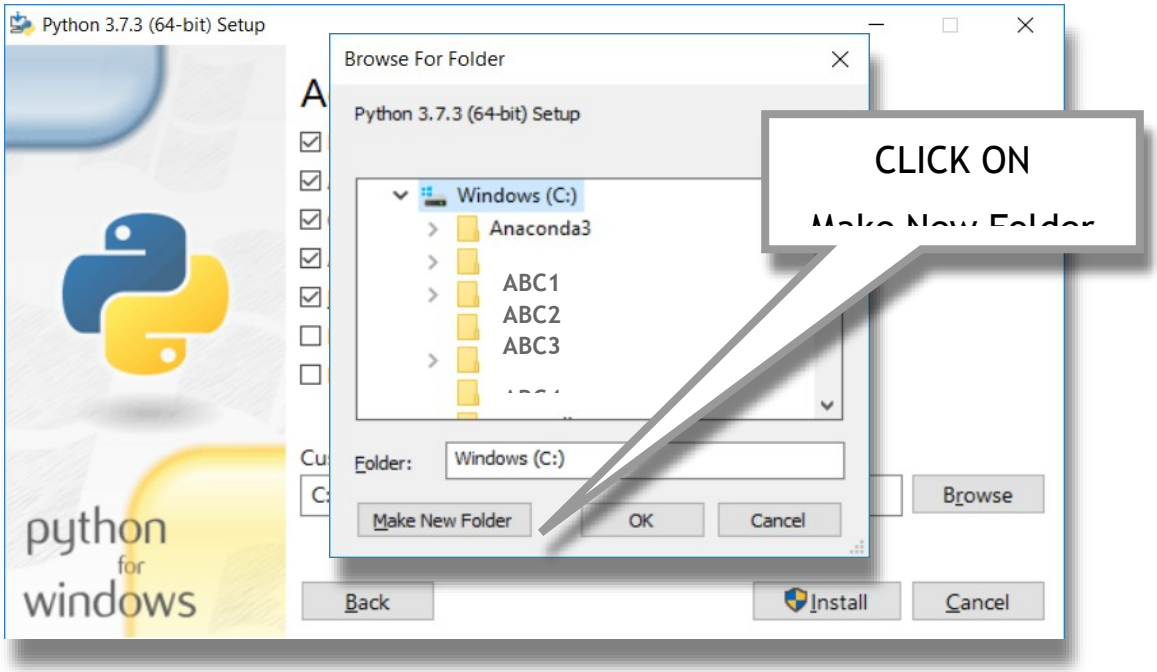
Download Python from python.org using link python.org/downloads
Select appropriate download link as per Operating System [Windows 32 Bit/64 Bit, Apple iOS]
FOR EXAMPLE, for Windows 64 Bit OS
Select the following link

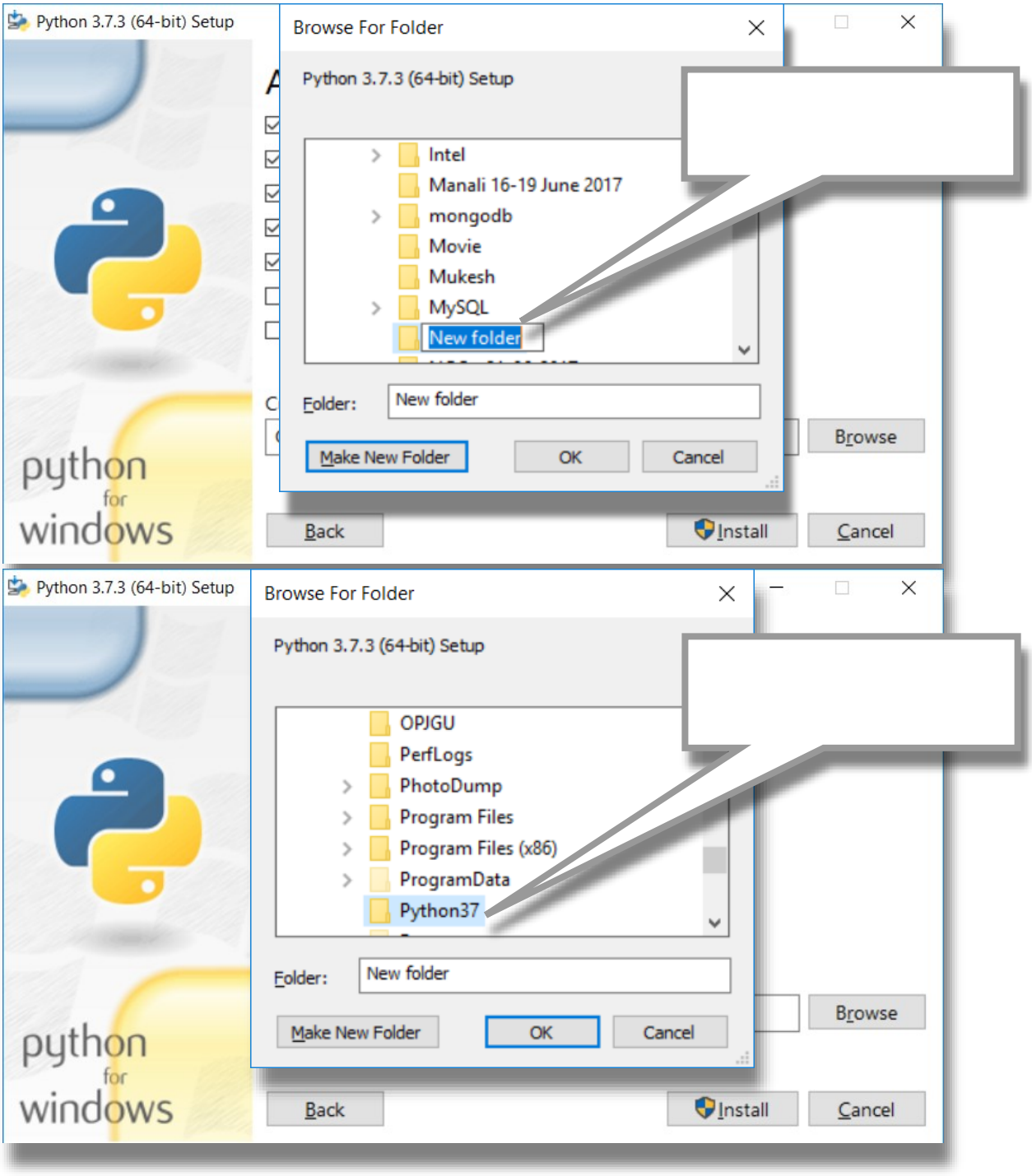
Python IDLE installation

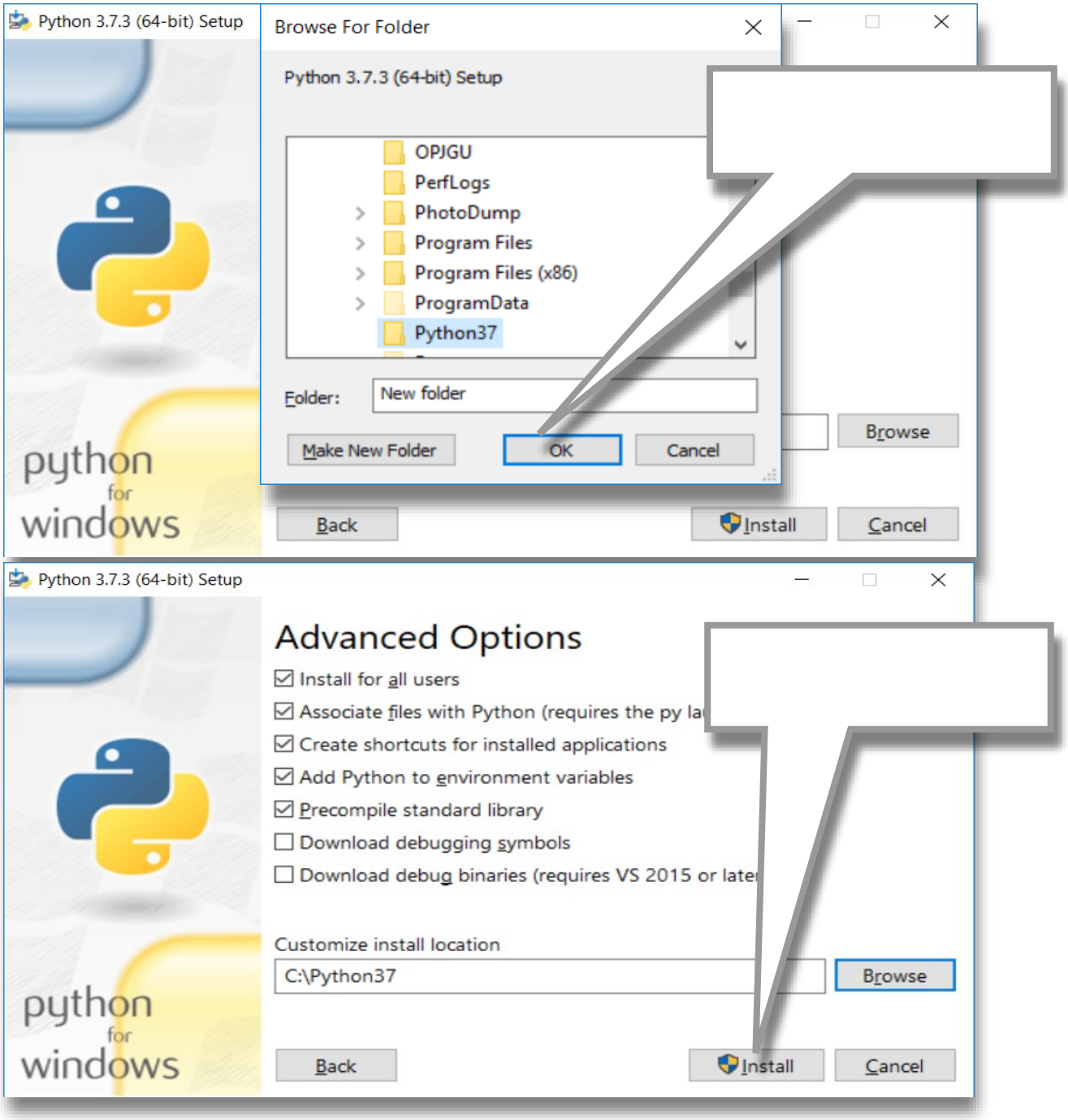




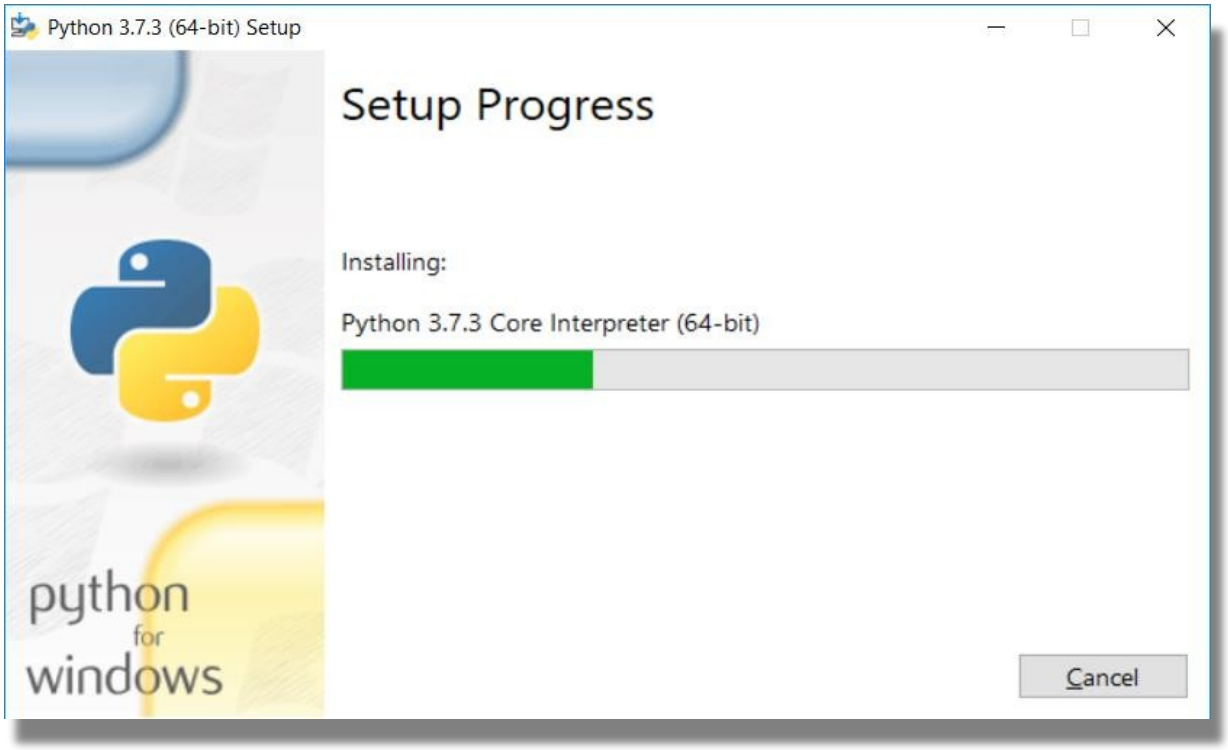

Python 3.7.3 (64-bit) Setup
Browse For Folder
Python 3.7.3 (64-bit) Setup
Documents
Downloads
Music
Pictures
Videos
Windows (C:)
LENOVO (D:)
CLICK ON
Folder:
This PC
Make New Folder
OK
Cancel
Browse
Cancel
python
for
windows


Python 3.7.3 (64-bit) Setup
Browse For Folder
Python 3.7.3 (64-bit) Setup
Windows (C:)
Anaconda3
ABC1
ABC2
ABC3
CLICK ON
Folder:
Windows (C:)
Make New Folder
OK
Cancel
Browse
Back
Install
Cancel
python
for
windows


Python 3.7.3 (64-bit) Setup
Browse For Folder
Python 3.7.3 (64-bit) Setup
Intel
Manali 16-19 June 2017
mongodb
Movie
Mukesh
MySQL
New folder
Folder:
New folder
Make New Folder
OK
Cancel
Browse
Back
Install
Cancel
python
for
windows
Python 3.7.3 (64-bit) Setup
Browse For Folder
Python 3.7.3 (64-bit) Setup
OPJGU
PerfLogs
PhotoDump
Program Files
Program Files (x86)
ProgramData
Python37
Folder:
New folder
Make New Folder
OK
Cancel
Browse
Back
Install
Cancel
python
for
windows


Python 3.7.3 (64-bit) Setup
Browse For Folder
Python 3.7.3 (64-bit) Setup
OPJGU
PerfLogs
PhotoDump
Program Files
Program Files (x86)
ProgramData
Python37
Folder:
New folder
Make New Folder
OK
Cancel
Browse
Back
Install
Cancel
python
for
windows
Python 3.7.3 (64-bit) Setup
Advanced Options
Install for all users
Associate files with Python (requires the py la
Create shortcuts for installed applications
Add Python to environment variables
Precompile standard library
Download debugging symbols
Download debug binaries (requires VS 2015 or late
Customize install location
C:\Python37
Browse
Back
Install
Cancel
python
for
windows

In



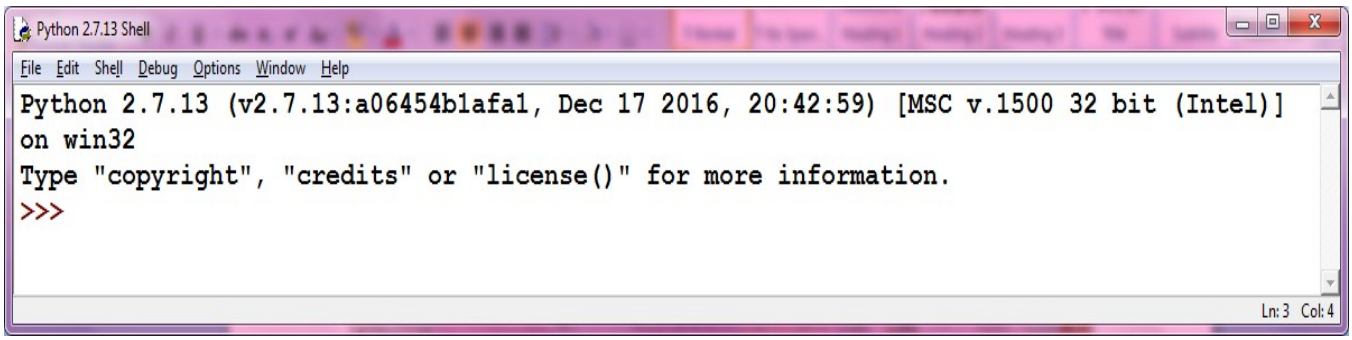
Python shell can be used in two ways, viz.,

- Interactive mode
- Script mode

Where Interactive Mode, as the name suggests, allows us to interact with OS; script
mode lets us create and edit Python source file.

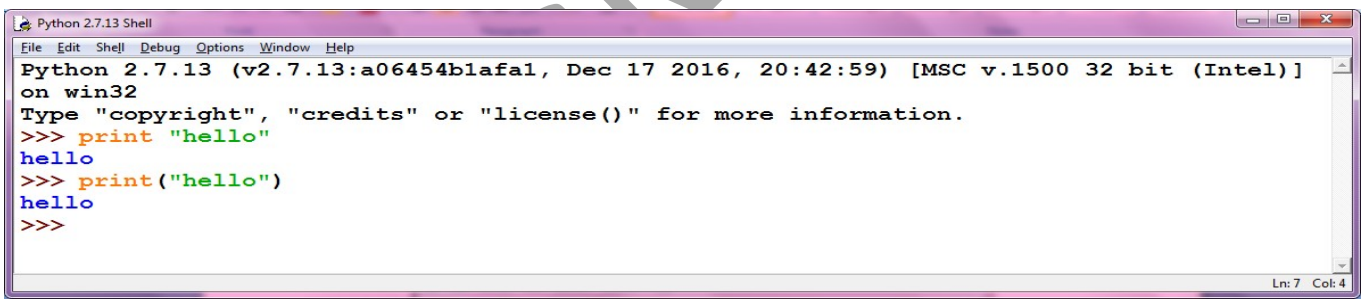
Running Python Interpreter:

- Python comes with an interactive interpreter.
- When you type python in your shell or command prompt, the python interpreter becomes active with a
  >>> (REPL) and waits for your commands.



In

- Now you can type any valid python expression at the prompt. Python reads the typed expression, evaluates it and prints the result.



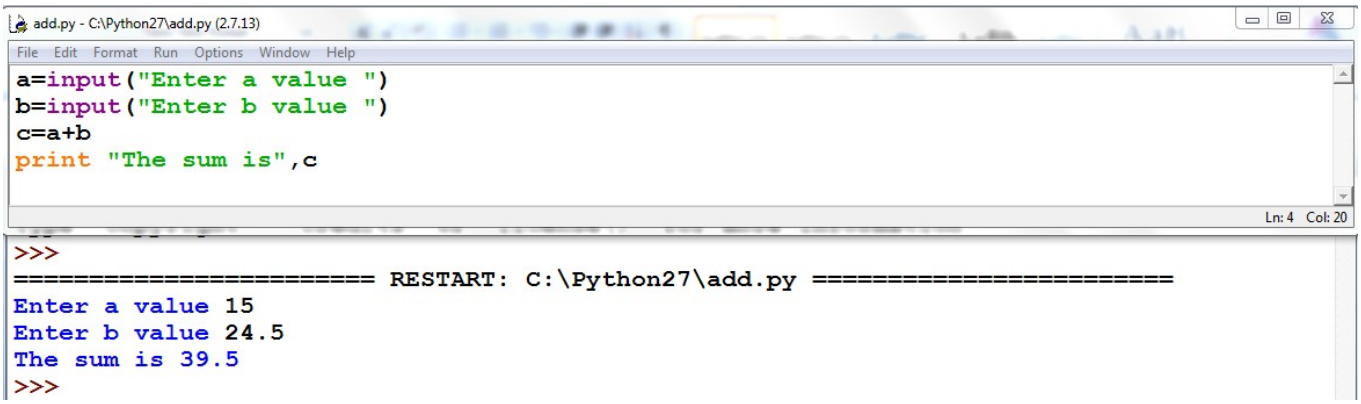
Script mode

- IDLE is the standard Python development environment.
- Its name is an acronym of "Integrated Development Environment". It works well on both Unix and Windows platforms.
- It has a Python shell window, which gives you access to the Python interactive mode. It also has a file editor that lets you create and edit existing Python source files.
- Goto File menu click on New File (CTRL+N) and write the code and save add.py

```
a=input("Enter
value ")
b=input("Enter
value")
        c=a+b
        print ("The sum is",c)
```

- Then run the program by pressing F5 or Run ==> Run Module.

In [ ] Running Python scripts in Command Prompt:



Indentation

- Code blocks are identified by indentation rather than using symbols like curly braces.
- Indentation clearly identifies which block of code a statement belongs to. Of course, code blocks canconsist of single statements, too.
- When one is new to Python, indentation may come as a surprise. Humans generally prefer to avoid change, so perhaps after many years of coding with brace delimitation, the first impression of using pureindentation may not be completely positive.
- However, recall that two of Python's features are that it is
- simplistic in nature and easy to read. Python does not support braces to indicate blocks of code for class and function definitions or flowcontrol. Blocks of code are denoted by line indentation.
- All the continuous lines indented with same number of spaces would form a block.

Python strictlyfollow indentation rules to indicate the blocks.

In

```
statement
if condition:
    if condition:
        statement
    else:
        statement
statement
```

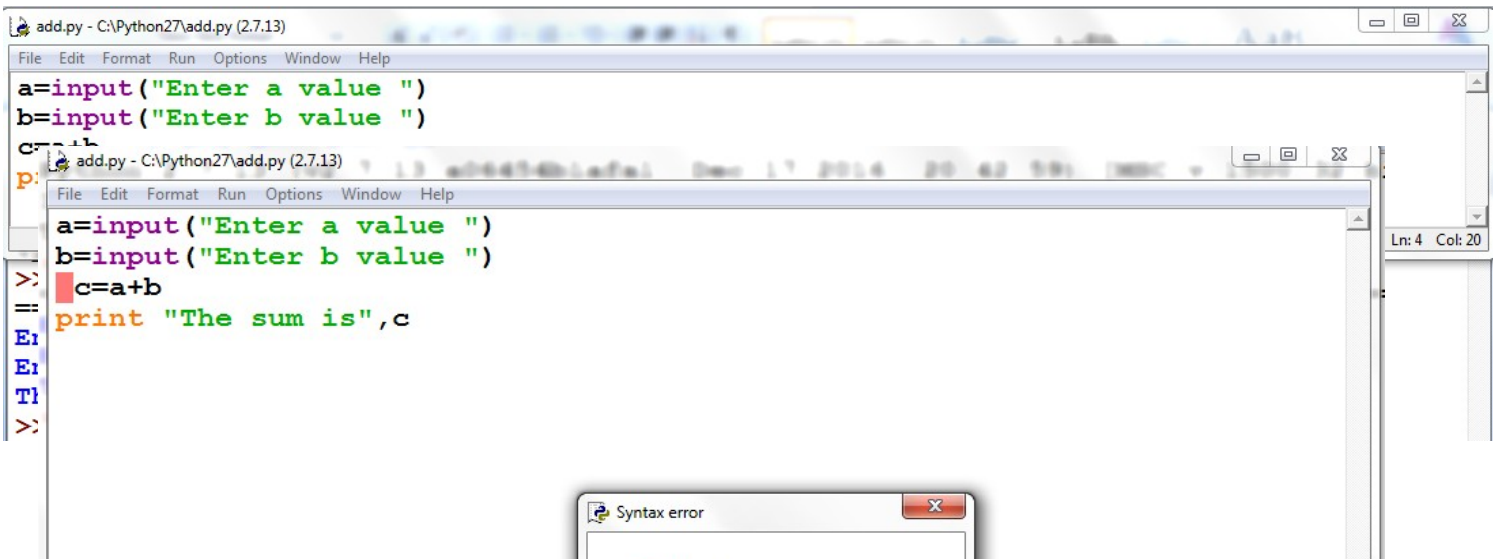
```
code block 1
code block 1
    code block 2
        code block 3
    code block 2
        code block 3
code block 1
```

Example1

```
name = "Robotics"
if name==" Robotics ":
print("Welcome back Robotics ")
else:
print("Welcome to the AI Class Robotics ")
print("Hello")
```

Example2

```
ptr = 2
while ptr*ptr<=100:
    if 100%ptr==0:
        print("One of the divisor of 100 is", ptr)
    ptr+=1
```



In

Python Statement and Comments

In this section we will learn about Python statements, why indentation is important and howto use comments in programming.

Python Statement

Instructions written in the source code for execution are called statements. There are different types of statements in the Python programming language like Assignment statement, Conditional statement, Looping statements etc. These help the user to get the required output. For example, n = 50 is an assignment statement.

Multi-line statement

In Python, end of a statement is marked by a newline character. However, Statements in Python can be extended to one or more lines using parentheses (), braces {}, square brackets [], semi-colon (;), continuation character slash (\). When we need to do long calculations and cannot fit these statements into one line, we can make use of these characters.

Examples:

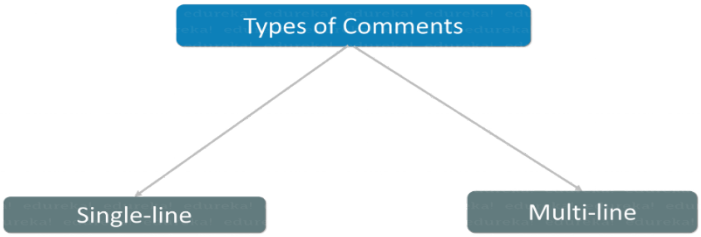
Type of Multi-line Statement	Usage
Using Continuation Character (/)	s = 1 + 2 + 3 + \ 4 + 5 + 6 + \ 7 + 8 + 9
Using Parentheses ()	n = (1 * 2 * 3 + 4 – 5)
Using Square Brackets []	footballer = ['MESSI', 'NEYMAR', 'SUAREZ']
Using braces {}	x = {1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9}
Using Semicolons (;)	flag = 2; ropes = 3; pole = 4

Python Comments

A comment is text that doesn't affect the outcome of a code, it is just a

In

piece of text to let someone know what you have done in a program or what is being done in a block of code.



Single Line Comments:

They can appear either in an individual line or in line with some other code.

#Comments in Python start like this
Print ("Comments in Python start with a #")

Multiline Comments:

Multi-line comments appear in more than one line. All the lines to be commented are to be prefixed by a #. If you don't do so, you will encounter an error.

#Comments in Python
#start with this character
print("Comments in Python")

Docstring Comments:

Docstrings are not actually comments, but, they are documentation strings. These docstrings are within triple quotes. They are not assigned to any variable and therefore, at times, serve the purpose of comments as well.

They are used particularly when you need to affiliate some documentation related to a class or a function, etc.

"""
Using docstring as a comment.
This code divides 2 numbers
"""
x=8
y=4

```
In
z=x/y
print(z)
```

Output: 2.0

the output does not contain the docstring, therefore, it has been omitted as it appears before the code has started

```
"""
Using docstring as a comment.
This code divides 2 numbers
"""
```

Output:

```
Using docstring as a comment.
This code divides 2 numbers
```

In the above output, the docstring has been printed since it is not followed by any code.

Now, in case it would be present after the code was written, the docstring will still be printed after the result.

```
x=8
y=4
z=x/y
print(z)
"""
Using docstring as a comment.
This code divides 2 numbers
"""
```

Output:

```
2.0
Using docstring as a comment.
This code divides 2 numbers
```

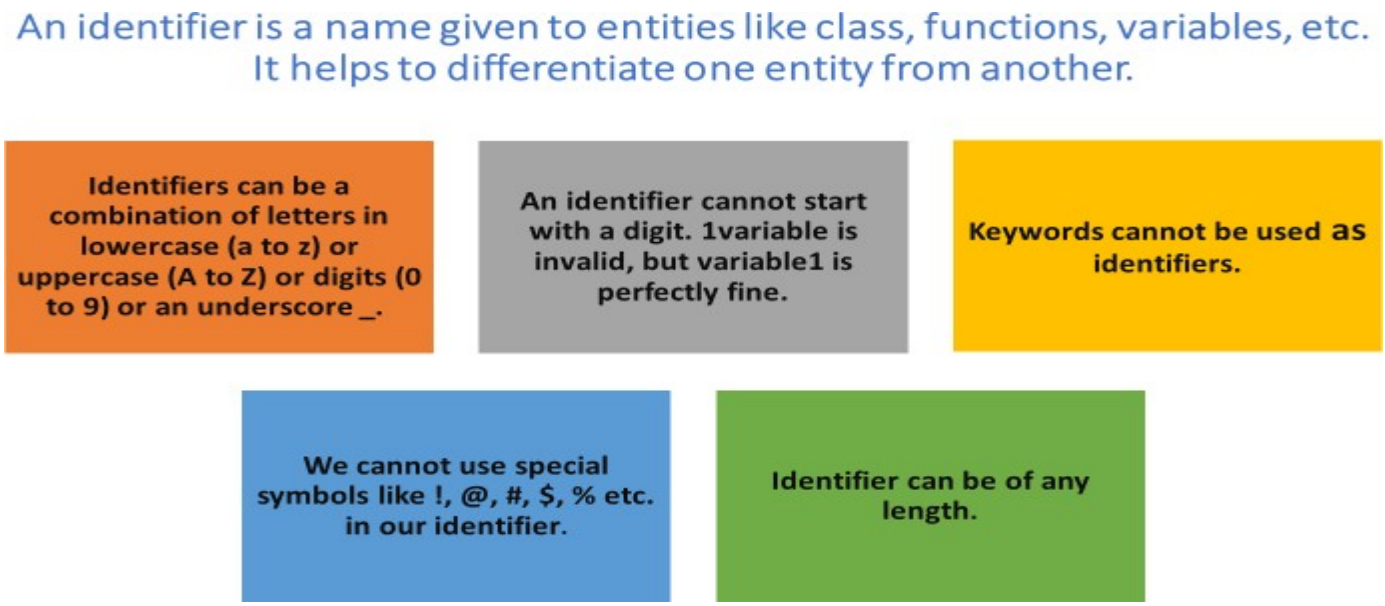
Python Keywords and Identifiers

Keywords are the reserved words in Python used by Python interpreter to recognize the structure of the program.

In

Keywords in Python

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	



Python is a case-sensitive language. This means, Variable and variable are not the same. Always name identifiers that make sense.

While, c = 10 is valid. Writing count = 10 would make more sense and it would be easier to figure out what it does even when you look at your code after a long gap.

Multiple words can be separated using an underscore, for example this_is_a_long_variable

Variables and Datatypes

Variables

A variable is a named location used to store data in the memory. It is

In

helpful to think of variables as a container that holds data which can be changed later throughout programming. For example,

x = 42

y = 42

These declarations make sure that the program reserves memory for two variables with the names x and y. The variable names stand for the memory location.

Examples on Variables:

Task	Sample Code	Output
Assigning a value to a variable	Name = “Krish” print(Name)	Krish
Changing value of a variable	Name = “Krish” print(Name) Name = “Chaitu” print(Name)	Krish Chaitu
Assigning different valuesto different variables	a,b,c=5, 3.2, “Hello” print(a) print(b)	5 3.2 Hello
	print(c)	
Assigning same value to different variable	x=y=z= "Same" print(x) print(y) print(z)	SameSameSame

Constants:

A constant is a type of variable whose value cannot be changed. It is helpful to think of constants as containers that hold information which cannot be changed later.

Non technically, you can think of constant as a shoe box with a fixed size of shoe kept inside which cannot be changed after that.

Assigning Value to a constant in Python

In

In Python, constants are usually declared and assigned on a module. Here, the module means a new file containing variables, functions etc. which is imported to the main file. Inside the module, constants are written in all capital letters and underscores separating the words.

Example: Declaring and assigning value to a constant
Create a info.py

```
NAME="Krish"
AGE = 9
```

Create a main.py

```
import info

print(info.NAME)
print (info. AGE)
```

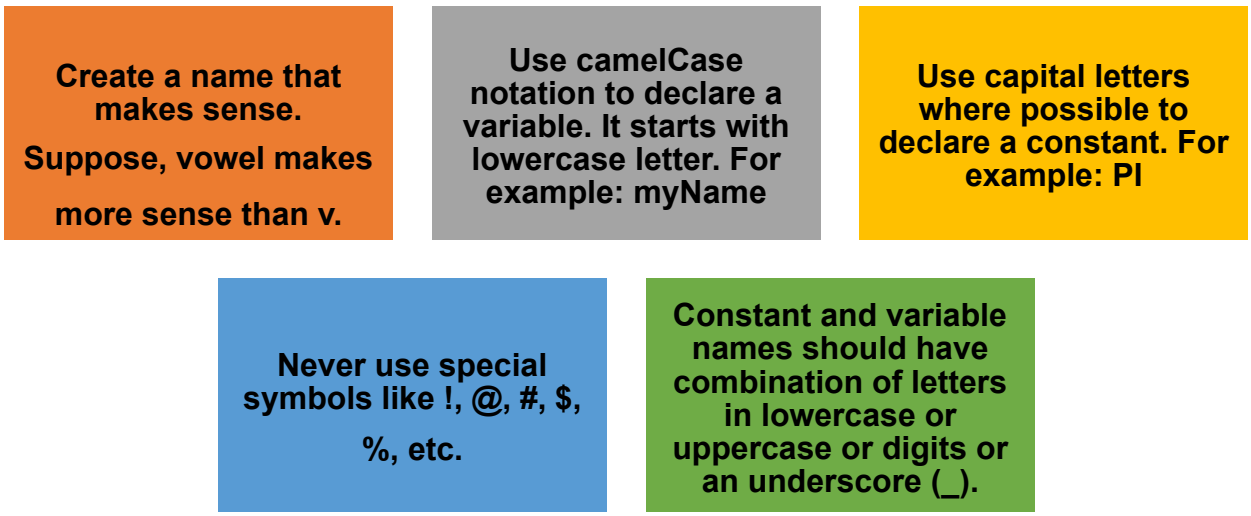
When you run the program the output will be,

Output

KRISH

9

Rules and Naming convention for variables and constants



In

Python Literals

Literals are representations of fixed values in a program. They can be numbers, characters, or strings, etc. For example, 'Hello, World!', 12, 23.0, 'C', etc.
Literals are often used to assign values to variables or constants. For example

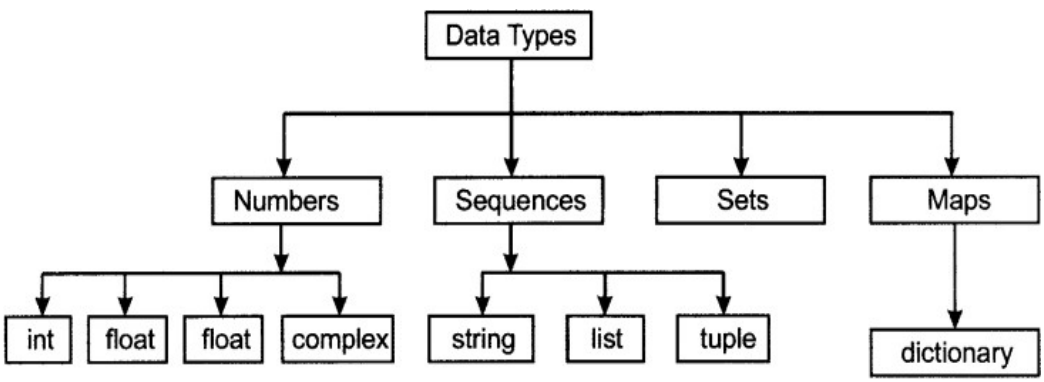
The basic difference between a variable, constant, and a literal in Python

Variables	Constants	Literals
A variable is a named location used to store data in the memory.	A constant is a type of variable whose value cannot be changed.	Literals are defined as raw value or data given in a variable or constant.
Mutable	Immutable	Mutable and Immutable both (depends on the type of literal)
For example: a=10 Here, a is the variable.	For example: PI=3.14 Here, PI is the constant.	For example: b="Hello"

Datatypes

Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

There are various data types in Python. Some of the important types are mentioned below in the image



Python Data types

In

Python Numbers

Number data type stores Numerical Values. These are of three different types:

a) Integer & Long
b) Float / floating point

Integer & Long Integer

Range of an integer in Python can be from -2147483648 to 2147483647, and long integer has unlimited range subject to available memory.

Integers are the whole numbers consisting of + or – sign with decimal digits like 100000, -99, 0, 17. While writing a large integer value, don't use commas to separate digits. Also, integers should not have leading zeros.

Floating Point:

Numbers with fractions or decimal point are called floating point numbers. A floating-point number will consist of sign (+,-) sequence of decimals digits and a dot such as 0.0, -21.9, 0.98333328, 15.2963. These numbers can also be used to represent a number in engineering/ scientific notation.

-2.0 x 105 will be
represented as -2.0e5
2.0X10-5 will be 2.0E-5

None

This is special data type with single value. It is used to signify the absence of value/false in a situation. It is represented by None.

1) **Sequence**

A sequence is an ordered collection of items, indexed by positive integers. It is a combination of mutable and non-mutable data types. Three types of sequence data

In

type available in Python are:

a) Strings
b) Lists
c) Tuples

String

String is an ordered sequence of letters/characters. They are enclosed in single quotes
(‘ ‘) or double (“ “). The quotes are not part of string. They only tell the computer where the string constant begins and ends. They can have any character or sign, including space in them.

Lists

List is also a sequence of values of any type. Values in the list are called elements / items. These are indexed/ordered. List is enclosed in square brackets.

Example:

```
dob = [19,"January",1990]
```

Tuples:

Tuples are a sequence of values of any type, and are indexed by integers. They are immutable. Tuples are enclosed in ().

Example:

```
t = (5,'program',2.5)
```

Sets

Set is an unordered collection of values, of any type,

with no duplicate entry. Example:

In

```
>>> a = {1,2,2,3,3,3}
>>> a
{1,2,3}
```

2)

Mapping

This data type is unordered. Dictionaries fall under Mappings.

Dictionaries

Dictionary is an unordered collection of key-value pairs. It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value.

In Python, dictionaries are defined within braces {} with each item being a pair in the form key: value. Key and value can be of any type.

Example

```
>>> d = {1:'Ajay','key':2}
>>> type(d)
<class 'dict'>
```

User input

In all the examples till now, we have been using the calculations on known values (constants). Now let us learn to take user's input in the program. In python, input() function is used for the same purpose.

By default, the **input()** function takes input as a string so if we need to enter the integer or float type input then the **input()** function must be type casted.

Syntax	Meaning
`<String Variable>=input(<String>)`	For string input
`<integer Variable>=int(input(<String>))`	For integer input

In

`<float Variable>=float(input(<String>))`	For float (Real no.) input

Lab Examples (For Practise):

Print Python Variables and Literals

Variables

```
num = 10
numlist = [1, 3, 5, 7, 9]
str = 'Hello World'
print(num)
print(numlist)
print(str)
```

Changing the value of a Variable

```
val = 50
print("Initial value:", val)
val = 100   # assigning new value
print("Updated value:", val)
```

Assign multiple values to multiple Variables

```
name, age, city = 'David', 27, 'New York'
print(name)
print(age)
print(city)
```

Assigning Values to Constants

```
# create a separate constant.py file
PI = 3.14
GRAVITY = 9.8
# main.py file
import constant as const


print('Value of PI:', cons.PI)
print('Value of Gravitational force:', cons.GRAVITY)
```

Numeric Literals

```
num1 = 32423 #Number literal
num2 = 5j #Complex literal
```

In [ ]:
```
num3 = 5.23 #float literal

print(num1, type(num1))
print(num2, type(num2))
print(num3, type(num3))
```

String Literals:

```
#in single quote
a='Christ'
#in double quotes
b="Christ"
#in triple quotes”
c="'Christ
        UNV'"
print(a)
print(b)
print(c)
```

Character literal

```
a='b'
c="d"
print(a)
print(c)
```

Boolean Literals:

```
a=(0==False)
b=(0==True)
c=5-False
d=True+7
print(a)
```

```
print(b)
print(c)
print(d)
```

Print Concatenated Strings

Using '+' operator

```
s1="CSE"
s2="DEPT"
s3=s1+s2
s4=s1+" "+s2
print(s3)
print(s4)
```

Using % operator

```
var1 = "Hello"
var2 = "World"
# % Operator is used here to combine the string
print("% s % s" % (var1, var2))
```

String Concatenation using join() Method

```
var1 = "Hello"
var2 = "World"
# join() method is used to combine the strings
print("".join([var1, var2]))
# join() method is used here to combine
# the string with a separator Space(" ")
var3 = " ".join([var1, var2])
print(var3)
```

String Concatenation using format() function

In [ ]
```
var1 = "Hello"
var2 = "World"

# format function is used here to
# combine the string
print("{} {}".format(var1, var2))

# store the result in another variable
var3 = "{} {}".format(var1, var2)
print(var3)

#String Concatenation using (, comma)

var1 = "Hello"
var2 = "World"

# To combine data types with a single whitespace.
print(var1, var2)
```

Python User Input

```
# Python program showing a use of input()
val = input("Enter your value: ")
print(val)

# Read and Print Employee Information
name = input("Enter Employee Name: ")
salary = input("Enter salary: ")
company = input("Enter Company name: ")

# Display all values on screen
```

In [ ]:
```
print("\n")
print("Printing Employee Details")
print("Name", "Salary", "Company")
print(name, salary, company)
```

 # Convert in out into int or # program to calculate addition of two input integer numbers

```
first_number = int(input("Enter first number "))
second_number = int(input("Enter second number "))
print("\n")
print("First Number:", first_number)
print("Second Number:", second_number)
sum1 = first_number + second_number
print("Addition of two number is: ", sum1)
```

#Single execution of the input () function for three different variables.

```
name, age, marks = input ("Enter your Name, Age, Percentage separated by space “). split()
print("\n")
print("User Details: ", name, age, marks)
```

The data type of the user input is converted from string to integer.

 # Python program showing

 # a use of input()

```
name = input("Enter your name: ")  # String Input
age = int(input("Enter your age: ")) # Integer Input
```

In [ ]
```
marks = float(input("Enter your marks: ")) # Float Input
print("The name is:", name)
print("The age is:", age)
print("The marks is:", marks)
```

#Define the values of different data types and checking its type.

```
a=10
b="Hi Python"
c = 10.5
print(type(a))
print(type(b))
print(type(c))
```

#Use single, double, or triple quotes to define a string.

```
str = "string using double quotes"
print(str)
s = """A multiline
string'''
print(s)
```

#RANGE

Range: range() function returns an immutable sequence of numbers starting from 0, increments by 1 and ends at a specified number.

range(start,stop,step)

You must specify the required argument stop, which can be any positive integer.

If you don't specify the start index, the default start index of 0 is used.

In [ ]: If you don't specify the step value, the default step size of 1 is used.

EX:

```
start = 5
stop = 12
step = 4
  print(list(range(start, stop, step)))
```

Example 1:

```
print(list(range(0)))

# using the range(stop)
print(list(range(4)))

# using the range(start, stop)
print(list(range(1,7 )))
```

Example 2: String

```
s = 'Python'
len(s)
 for i in range(len(s)):
 print(i, s[i])
```

EX:3

range(start, stop) - 2 Parameter Form

```
list(range(5, 10))
```

[5, 6, 7, 8, 9]

In

```
list(range(5, 7))
```

[5, 6]

```
list(range(5, 6))
```

[5]

```
list(range(5, 5))  # start >= stop, no numbers
```

[]

```
list(range(0, 5))
```

[0, 1, 2, 3, 4]

```
list(range(0, 0))
```

[]

```
list(range(0, -1))
```

[]

range(start, stop, step) - 3 Parameter Form

```
list(range(0, 10, 2))
```

[0, 2, 4, 6, 8]

```
list(range(0, 10, 6))
```

[0, 6]

```
list(range(200, 800, 100))
```

[200, 300, 400, 500, 600, 700]

In

```
list(range(10, 5, -1))
[10, 9, 8, 7, 6]

list(range(10, 5, -2))
[10, 8, 6]

list(range(6, 5, -2))
[6]

list(range(5, 5, -2))  # equal to stop is omitted
[]

list(range(4, 5, -2))  # beyond the stop is omitted
[]

 r =
 ran
 ge(
 10
 )
 pri
 nt(
 typ
 e(r
 ))
 pri
 nt(
 r)

 r = range (10)

r = range (10, -25) print(type(r))
```

```
In
for i in r:
print(r)
for i in r:

r = range (1,21,1)
for i in r:

r = range (1,21,3)

r = range (21,1,-)



or r in range(8, 2, -2):
    print(r)

for r in range (8, -2, -2):
    print(r)

print(list(range (10,-5,-1)))

#Demonstarte the string operations
str1 = 'hello javatpoint' #string str1
str2 = ' how are you' #string str2
print (str1[0:2]) #printing first two character using slice operator
print (str1[4]) #printing 4th character of the string
print (str1*2) #printing the string twice
print (str1 + str2) #printing the concatenation of str1 and str2

Operations on List
```

```
In [ ]: list1 = [1, "hi", "Python", 2]

        #Checking type of given list
        print(type(list1))

        #Printing the list1
        print (list1)

        # List slicing
        print (list1[3:])

        # List slicing
        print (list1[0:2])

        # List Concatenation using + operator
        print (list1 + list1)

        # List repetation using * operator
        print (list1 * 3)


        #Operations on Tuple

        tup  = ("hi", "Python", 2)

        # Checking type of tup
        print (type(tup))

        #Printing the tuple
```

```
In [ ]: print (tup)

        # Tuple slicing
        print (tup[1:])
        print (tup[0:1])

        # Tuple concatenation using + operator
        print (tup + tup)

        # Tuple repatation using * operator
        print (tup * 3)

        # Adding value to tup. It will throw an error.
        t[2] = "hi"

        #Dictionary

        d = {1:'Jimmy', 2:'Alex', 3:'john', 4:'mike'}

        # Printing dictionary
        print (d)

        # Accesing value using keys
        print("1st name is "+d[1])
        print("2nd name is "+ d[4])
          print (d.keys())
        print (d.values())

        Sets:
```

In [ ]

```
# Creating Empty set
set1 = set()
  set2 = {'James', 2, 3,'Python'}

#Printing Set value
print(set2)

# Adding element to the set

set2.add(10)
print(set2)

#Removing element from the set
set2.remove(2)
print(set2)

# Python program to demonstrate the application of iskeyword()
# importing keyword library which has lists
import keyword

# displaying the complete list using "kwlist()."
print("The set of keywords in this version is: ")
print( keyword.kwlist )
help("keywords")
```

True and False are those keywords that can be allocated to variables or parameters and are compared directly.

Code

In [ ]
```
print( 4 == 4 )
print( 6 > 9 )
print( True or False )
print( 9 <= 28 )
print( 6 > 9 )
print( True and False )
print( True == 3 )
print( False == 0 )
print( True + True + True)
```

Range:

- range() function is used to generate sequence of numbers.

Exercises to execute(Record)

1. Print the version of Python
2. Write a Python program to display your name using Interactive Mode
3. Write a Python program to display “CHRIST” using Script Mode
4. Demonstrate the Running instructions in Interactive interpreter and a Python Script?
5. Write a program to demonstrate different number datatypes in python (use type ())
6. Write a program to purposefully raise Indentation Error and Correct it
7. Write a program to find the area of a triangle.
8. Write a program to perform addition and read the values form command line arguments and printsits sum.
9. Write a program to demonstrate different number datatypes in python (use +)
10. Write a program to demonstrate working with i) tuples ii) List iii) set iv) Dictionary

In

11. Demonstarte the operations on Range function(Any Five)

Respected Sir/Madam,