Introduction to Python

- A programming language is a formal language that specifies a set of instructions that can be used to produce various kinds of output.
- In simple Words, a programming language is a vocabulary and set of grammatical rules for instructing a computer to perform specific tasks.
- What is a program?
- 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.

Why Python

1

Easy to learn, read and maintain

Python has few keywords, simple structure, and a clearly defined syntax. A program written in Python is fairly easy-to-maintain.

2

A Broad Standard library

Python has a huge bunch of libraries with plenty of built in functions to solve variety of problems.

3

Interactive Mode

Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

4

Portability and Compatibility

Python can run on a wide variety of operating systems and hardware platforms, has the same interface on all platforms.

5

Extendable

We can add low-level modules to the Python interpreter. These modules enable programmers to customize their tools to be more efficient.

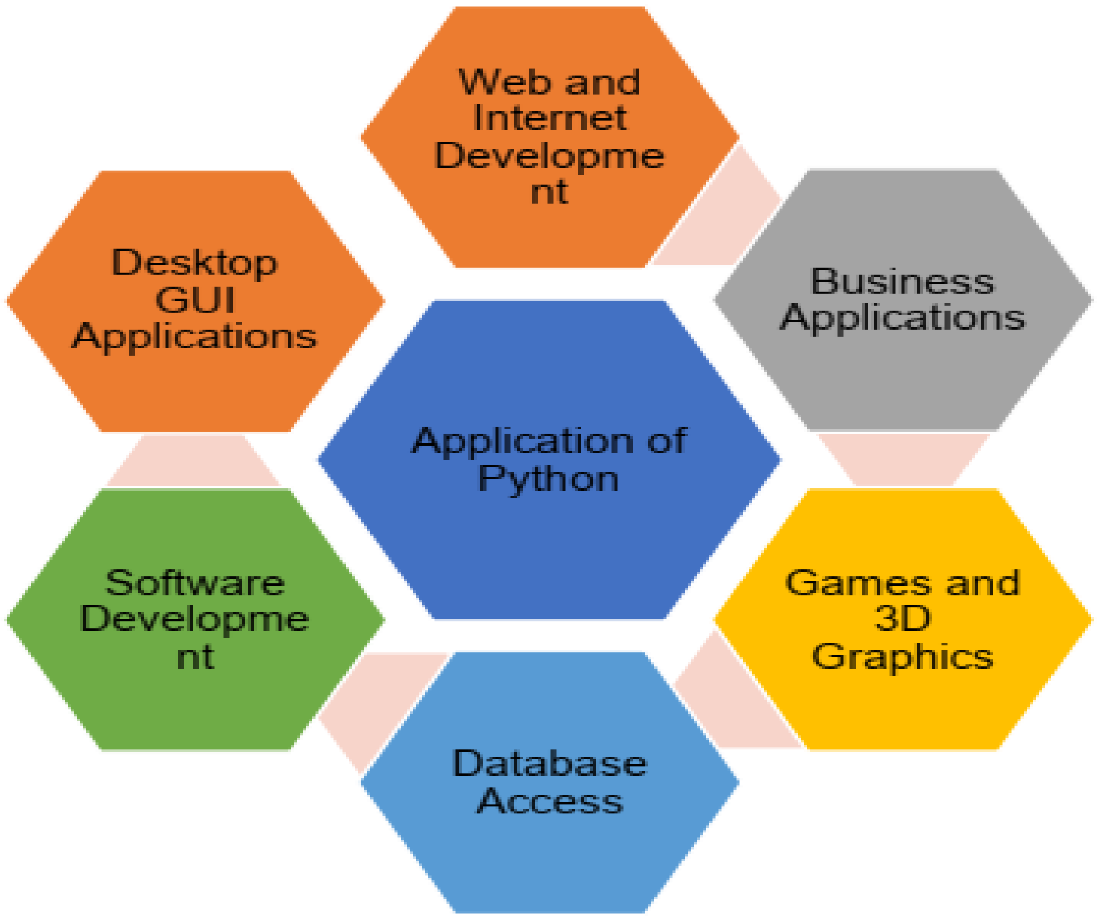
6

Databases And Scalable

Python provides interfaces to all major open source and commercial databases along with a better structure and support for large programs than shell scripting.

Applications of Python

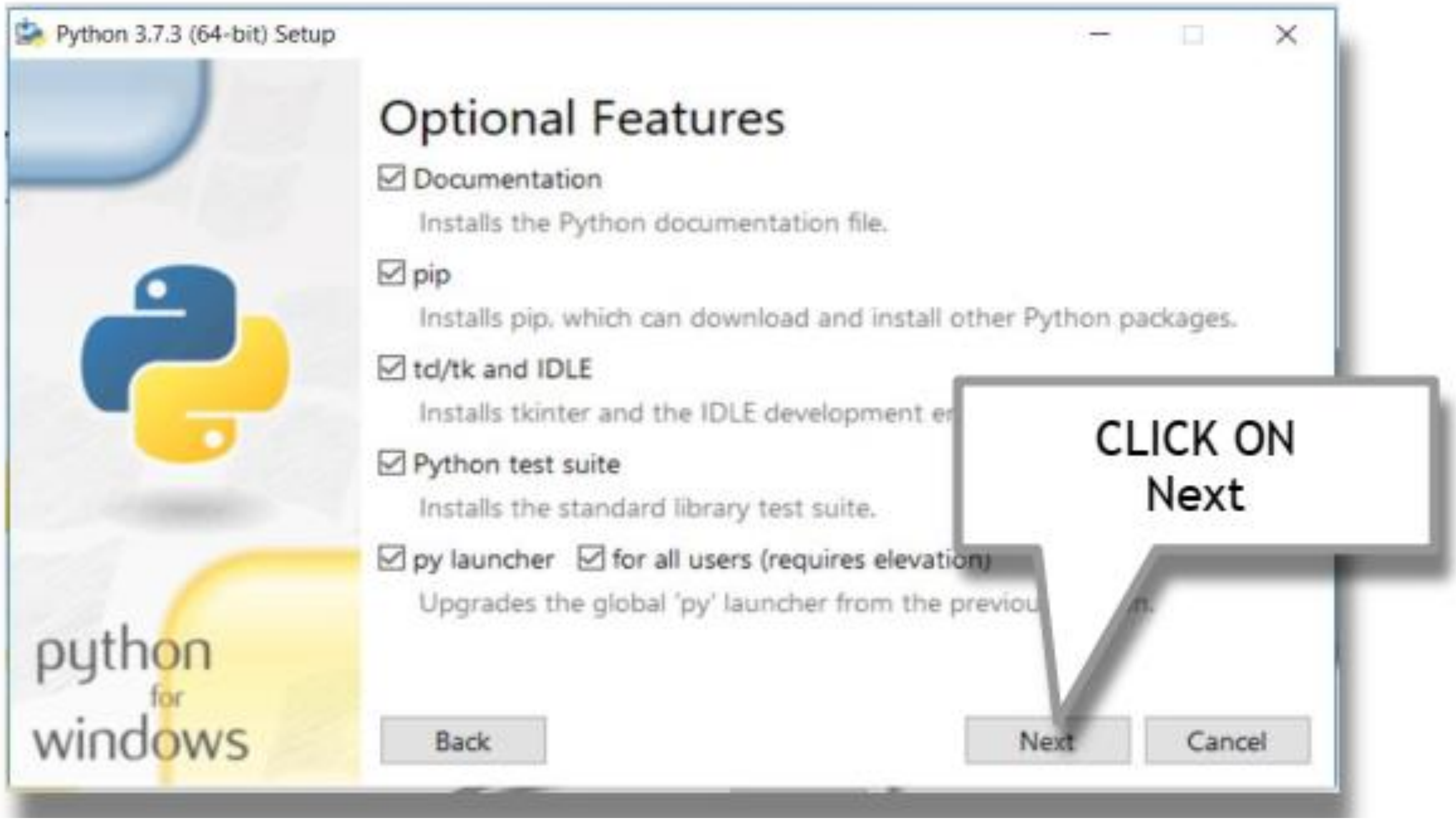
➢ Python is used for a large number of applications. Some of them are mentioned below:



Python

- ➢ Python is a cross-platform programming language, meaning, it runs on multiple platforms like Windows, MacOS, Linux.
- ➢ 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]

Python IDLE installation

➢ 

Python 3.7.3 (64-bit) Setup

Optional Features

☑ Documentation
Installs the Python documentation file.

☑ pip
Installs pip, which can download and install other Python packages.

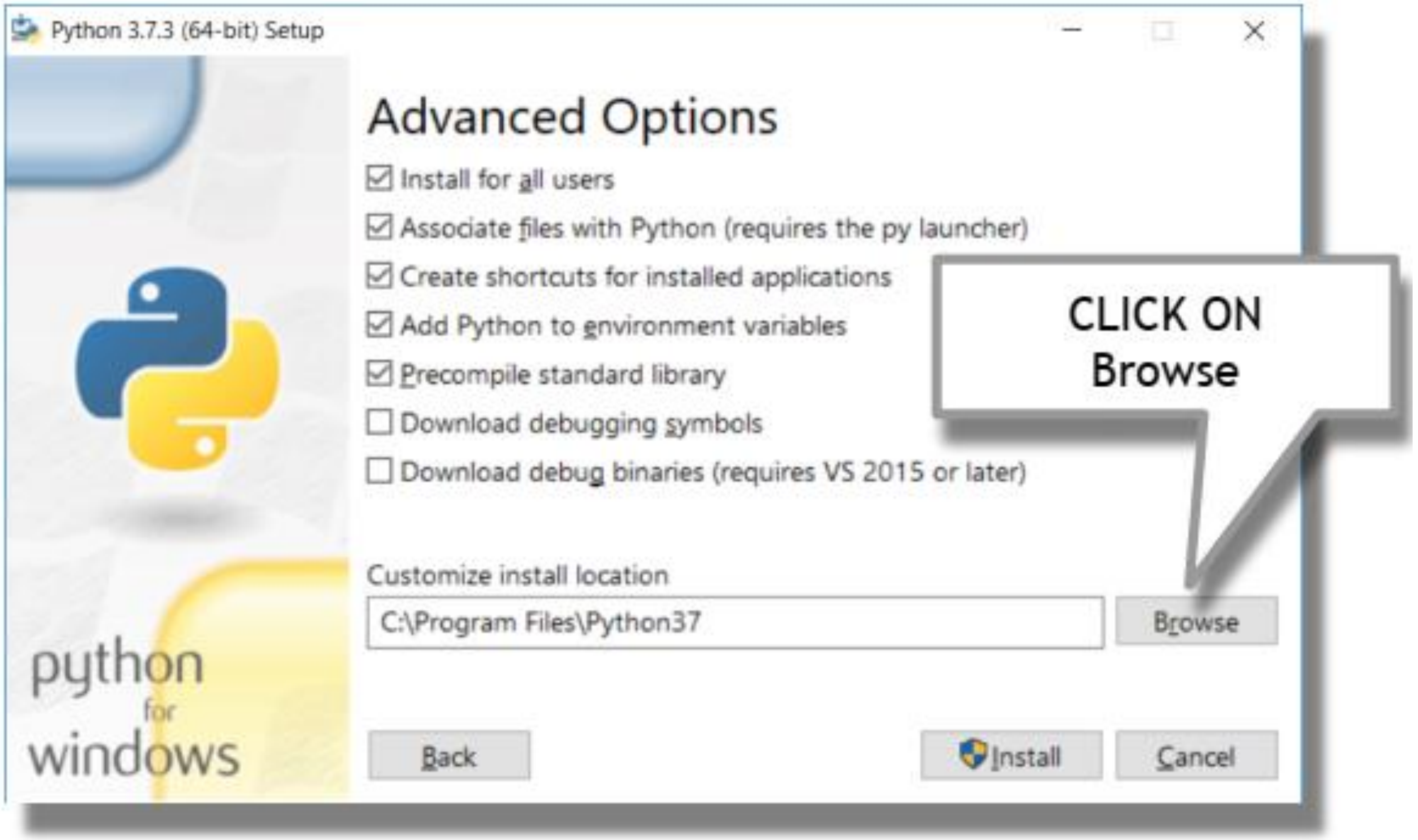
☑ tcl/tk and IDLE
Installs tkinter and the IDLE development e

☑ Python test suite
Installs the standard library test suite.

☑ py launcher ☑ for all users (requires elevation)
Upgrades the global 'py' launcher from the previou n.

Python IDLE installation

➢



Python 3.7.3 (64-bit) Setup

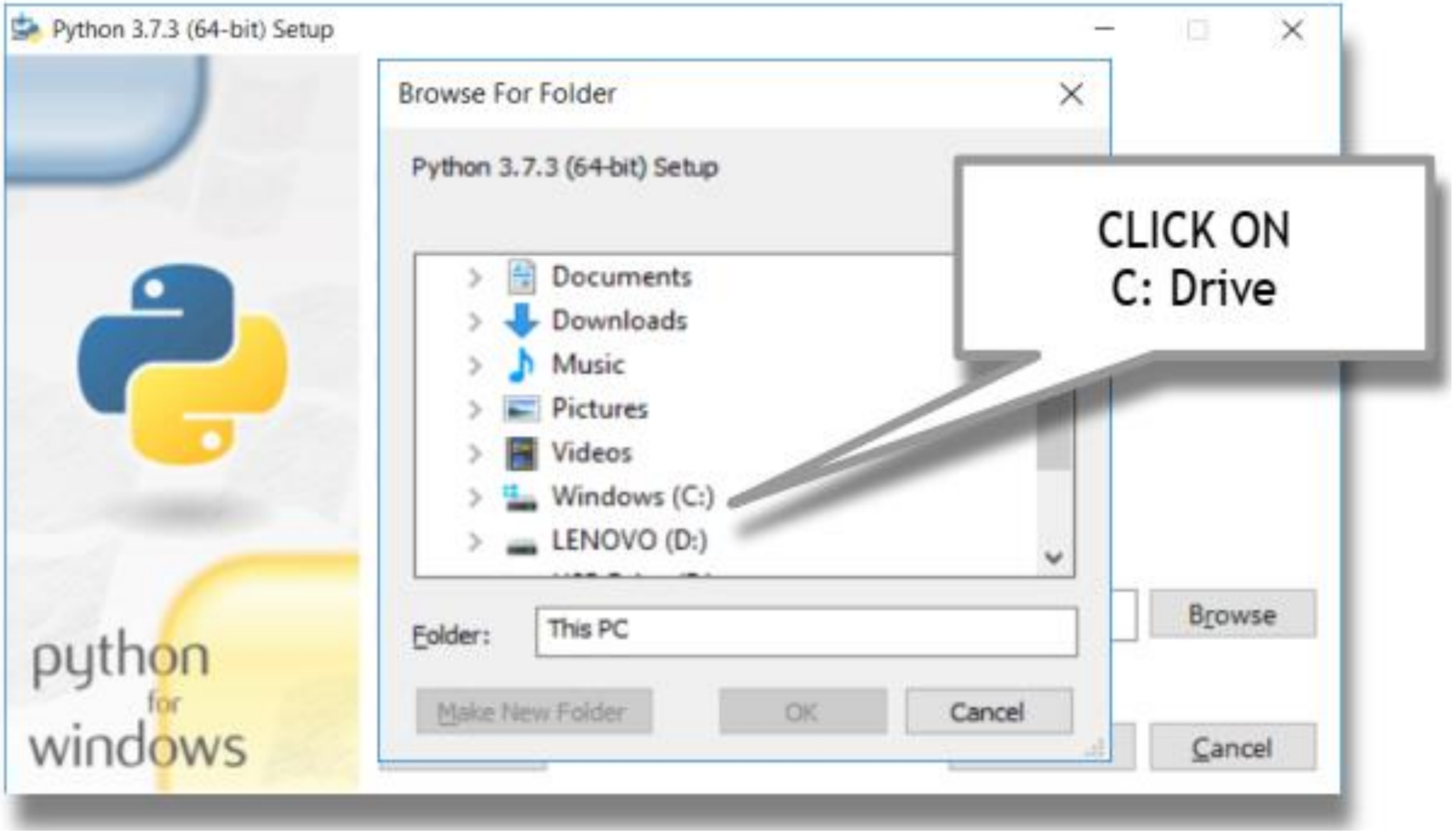
Advanced Options

- ☑ Install for all users
- ☑ Associate files with Python (requires the py launcher)
- ☑ Create shortcuts for installed applications
- ☑ Add Python to environment variables
- ☑ Precompile standard library
- ☐ Download debugging symbols
- ☐ Download debug binaries (requires VS 2015 or later)

Customize install location

C:\Program Files\Python37

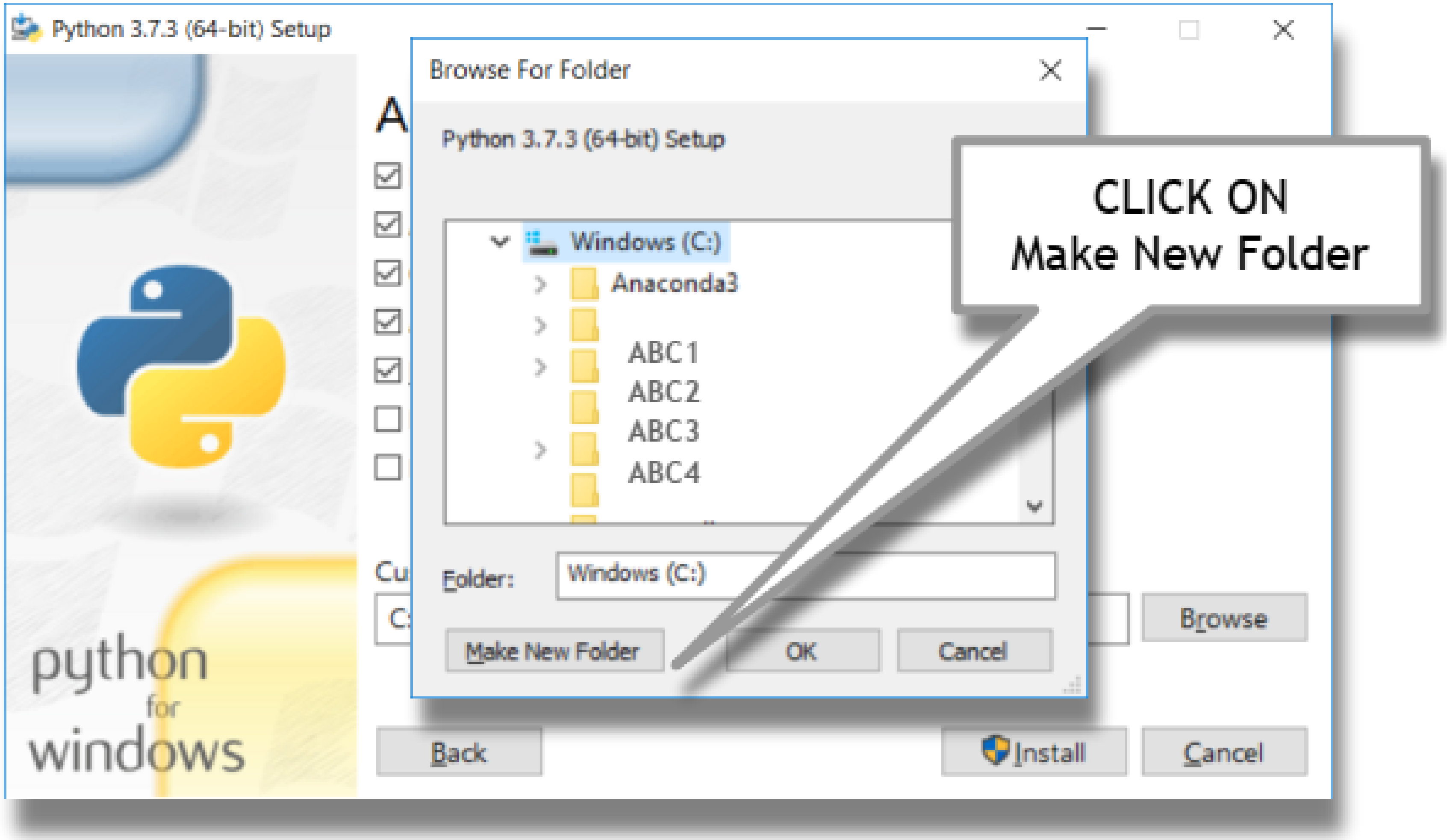
Python IDLE installation

➢ 

CLICK ON
C: Drive

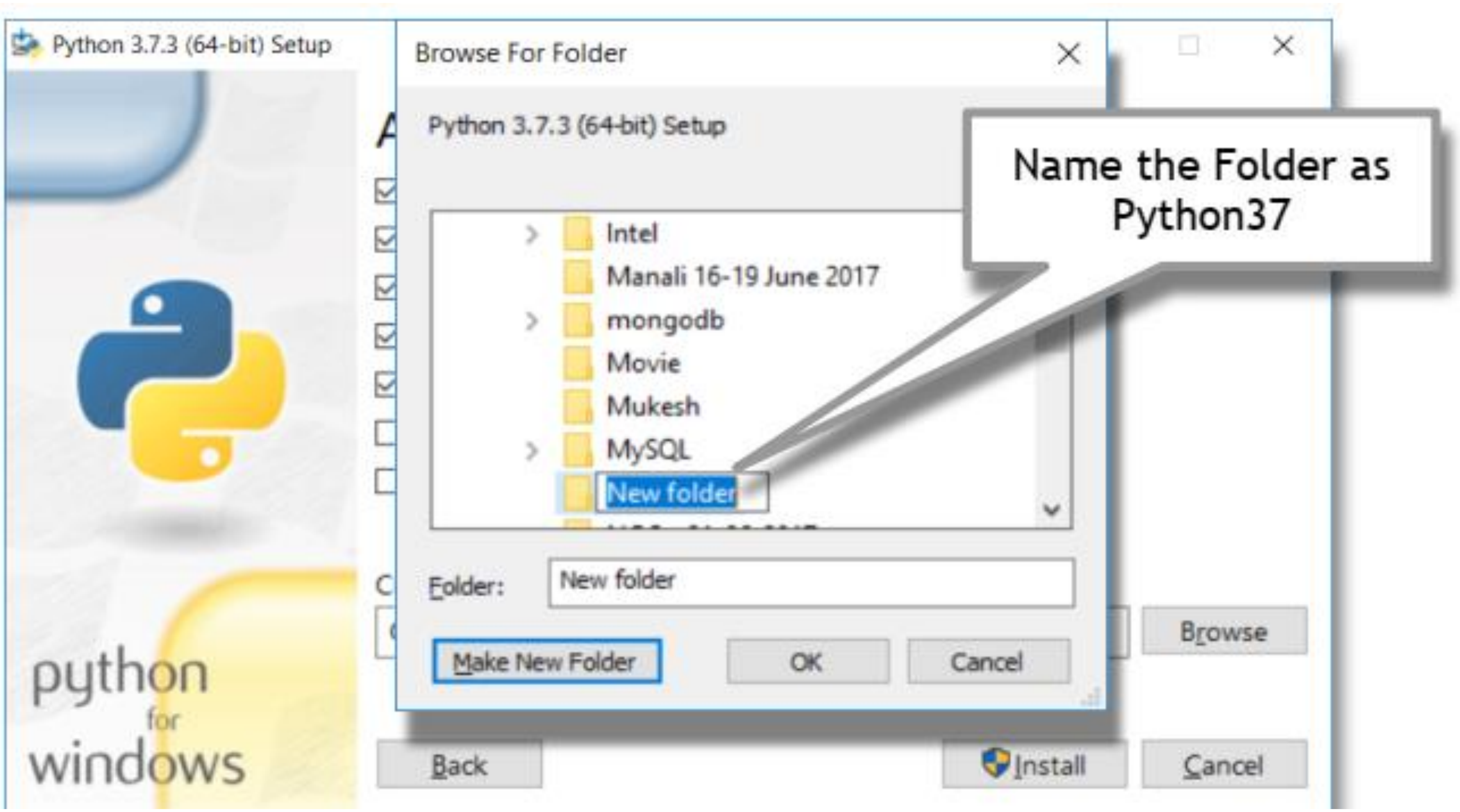
Python IDLE installation

➤



CLICK ON
Make New Folder

Python IDLE installation

➤ 

Python 3.7.3 (64-bit) Setup

Browse For Folder

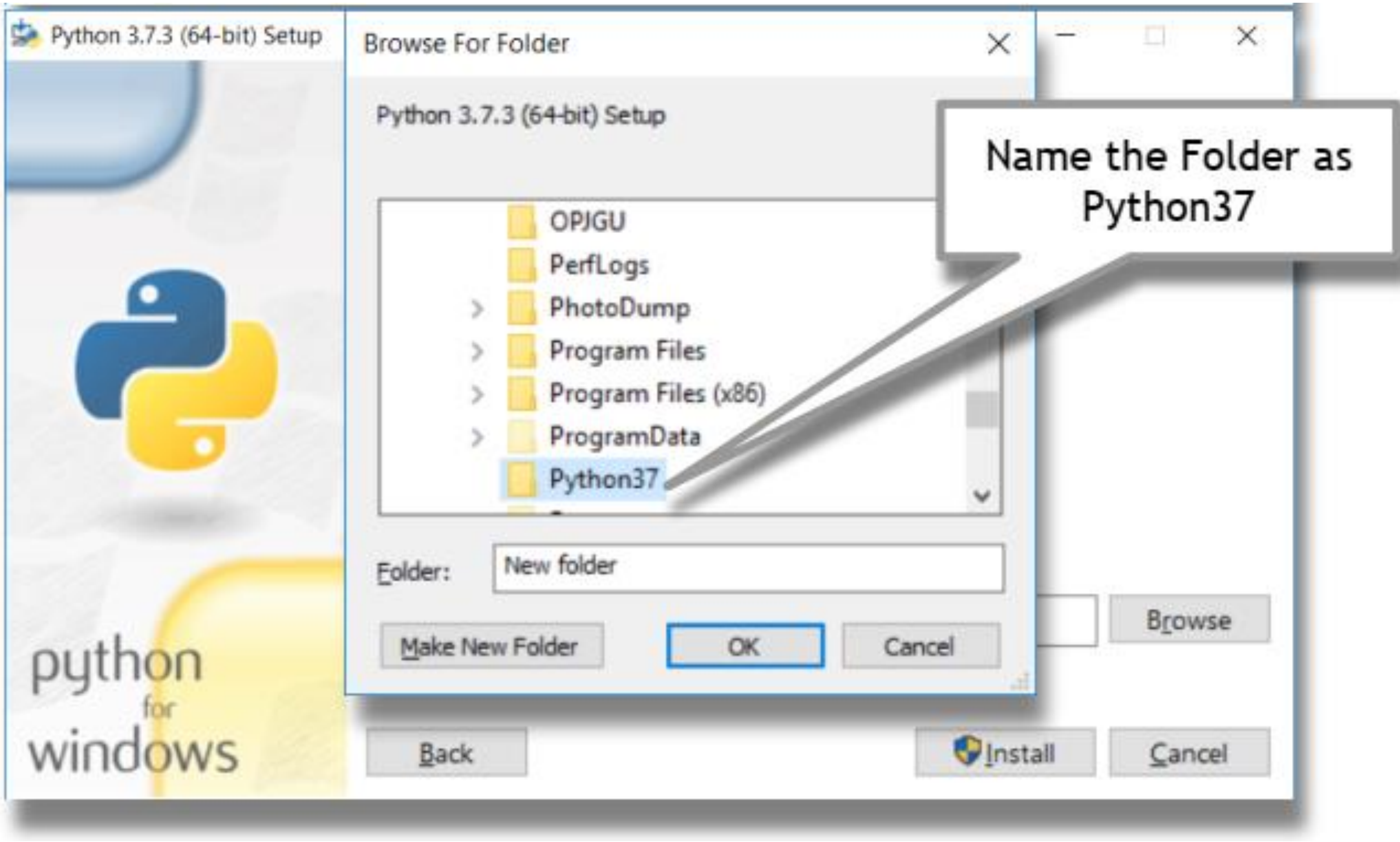
Python 3.7.3 (64-bit) Setup

Intel
Manali 16-19 June 2017
mongodb
Movie
Mukesh
MySQL
New folder

Folder: New folder

Python IDLE installation

➢



Python 3.7.3 (64-bit) Setup

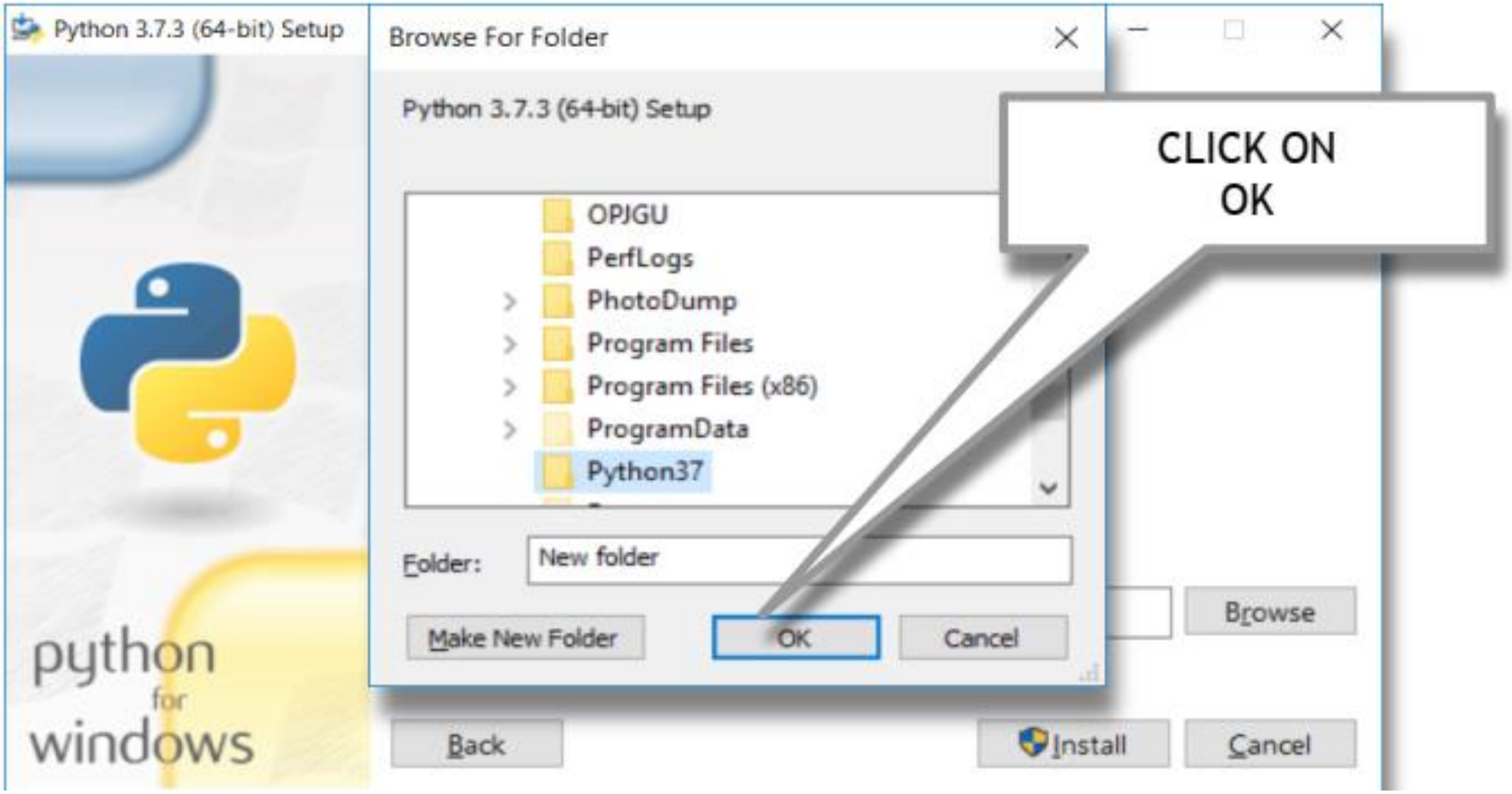
Browse For Folder

Python 3.7.3 (64-bit) Setup

Name the Folder as Python37

Python IDLE installation

➢



Python 3.7.3 (64-bit) Setup

Browse For Folder

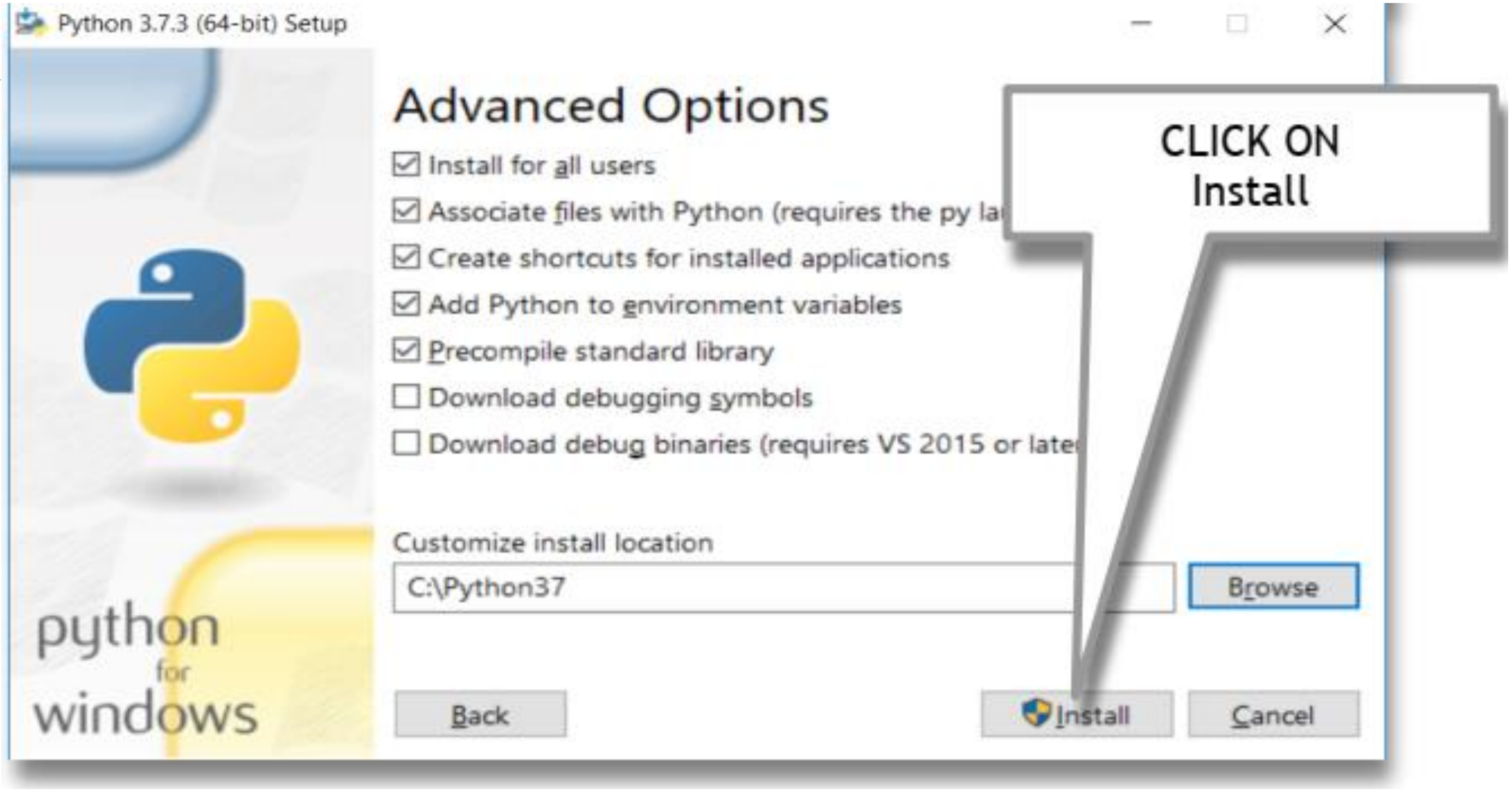
Python 3.7.3 (64-bit) Setup

OPJGU
PerfLogs
PhotoDump
Program Files
Program Files (x86)
ProgramData
Python37

Folder: New folder

Python IDLE installation

➢



Python 3.7.3 (64-bit) Setup

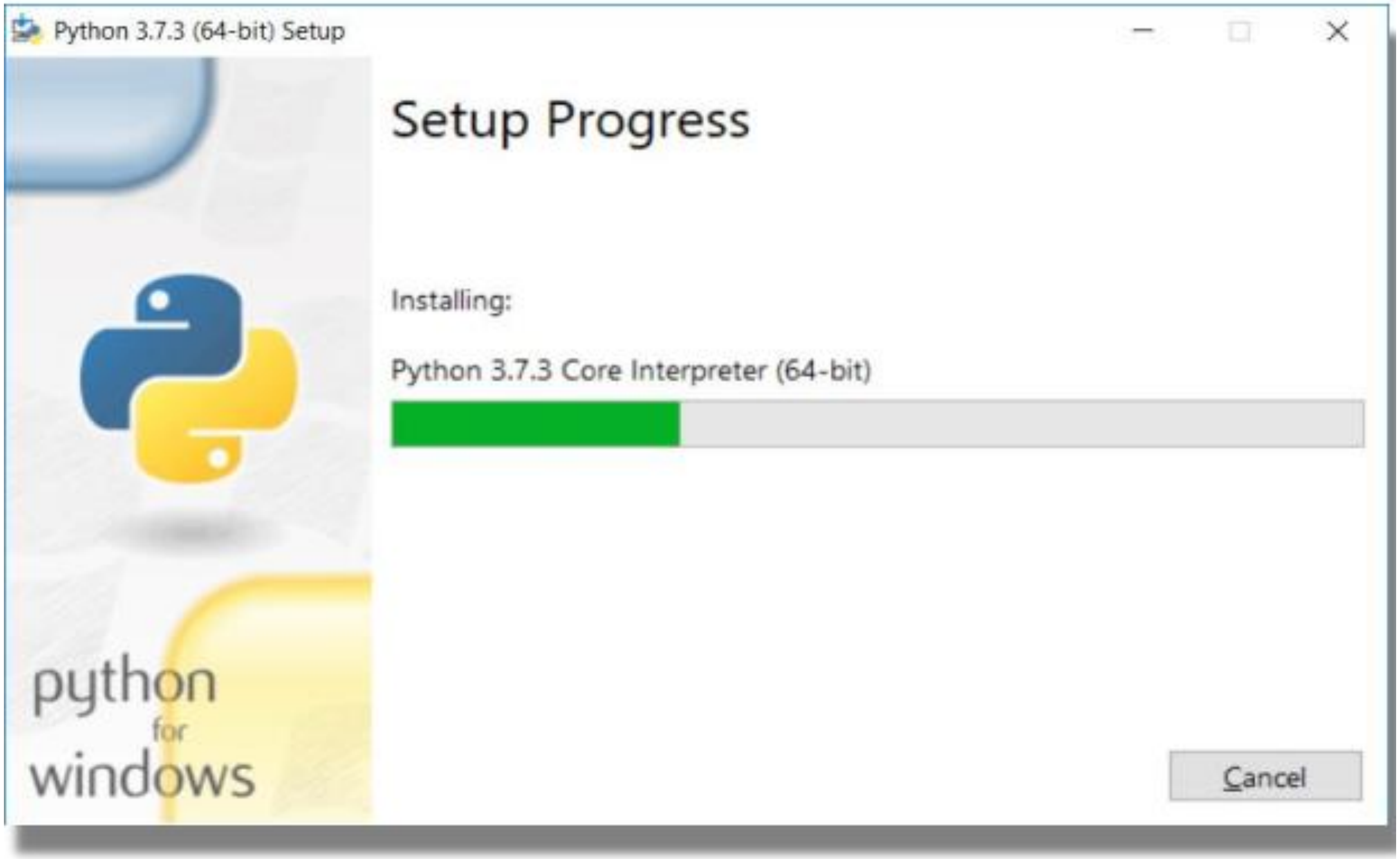
Advanced Options

- ☑ 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

Python IDLE installation

➢ 

Python 3.7.3 (64-bit) Setup

Setup Progress

Installing:

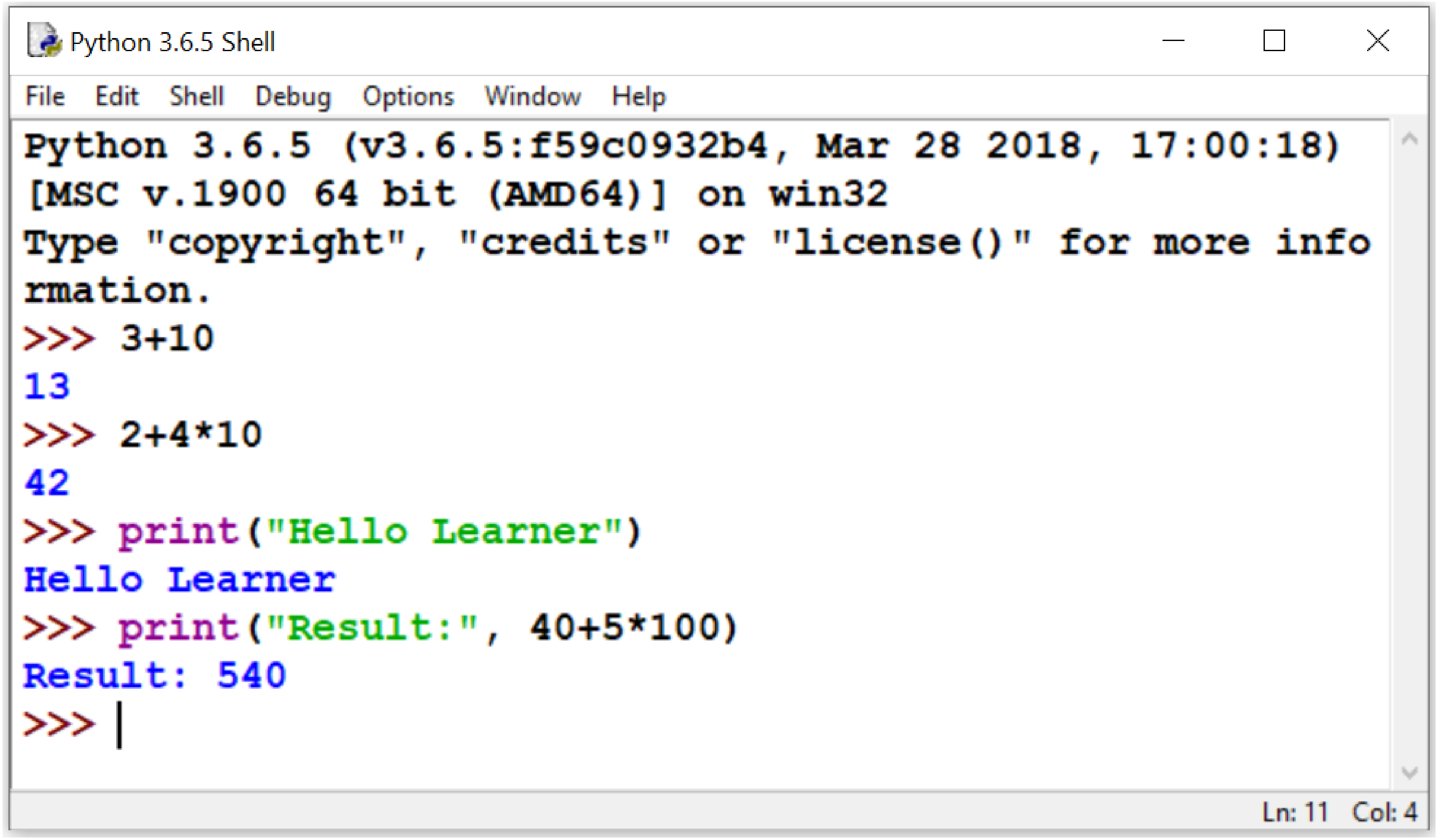
Python 3.7.3 Core Interpreter (64-bit)

Run in the Integrated Development Environment (IDE)

- When we install Python, an IDE named **IDLE** is also installed. We can use it to run Python on our computer.
- IDLE (GUI integrated) is the standard, most popular Python development environment. IDLE is an acronym of Integrated Development Environment .
- Python shell can be used in two ways, viz., interactive mode and script mode.

Interactive Mode

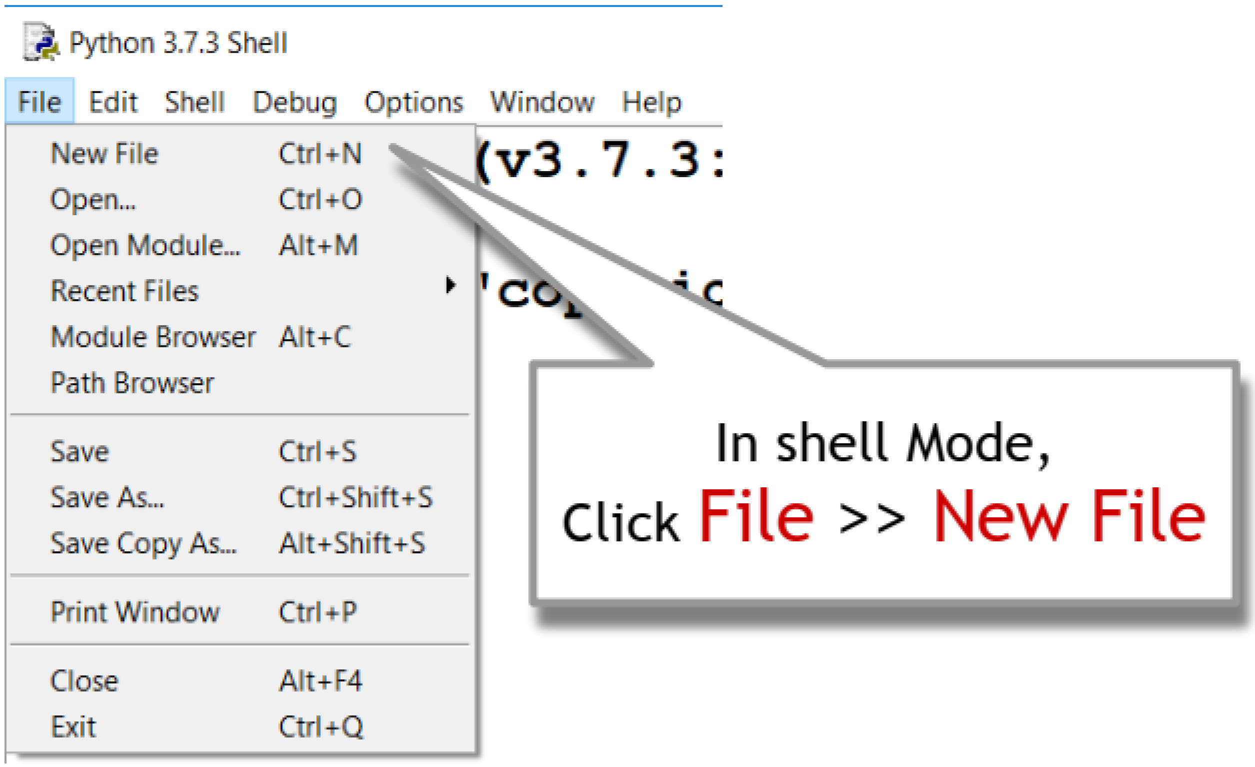
➤



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18)
[MSC v.1900 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more info
rmation.
>>> 3+10
13
>>> 2+4*10
42
>>> print("Hello Learner")
Hello Learner
>>> print("Result:", 40+5*100)
Result: 540
>>> |
```

Script Mode

- In script mode, we type Python program in a file and then use the interpreter to execute the content from the file.
- Working in interactive mode is convenient for beginners and for testing small pieces of code, as we can test them immediately. But for coding more than few lines, we should always save our code so that we may modify and reuse the code.



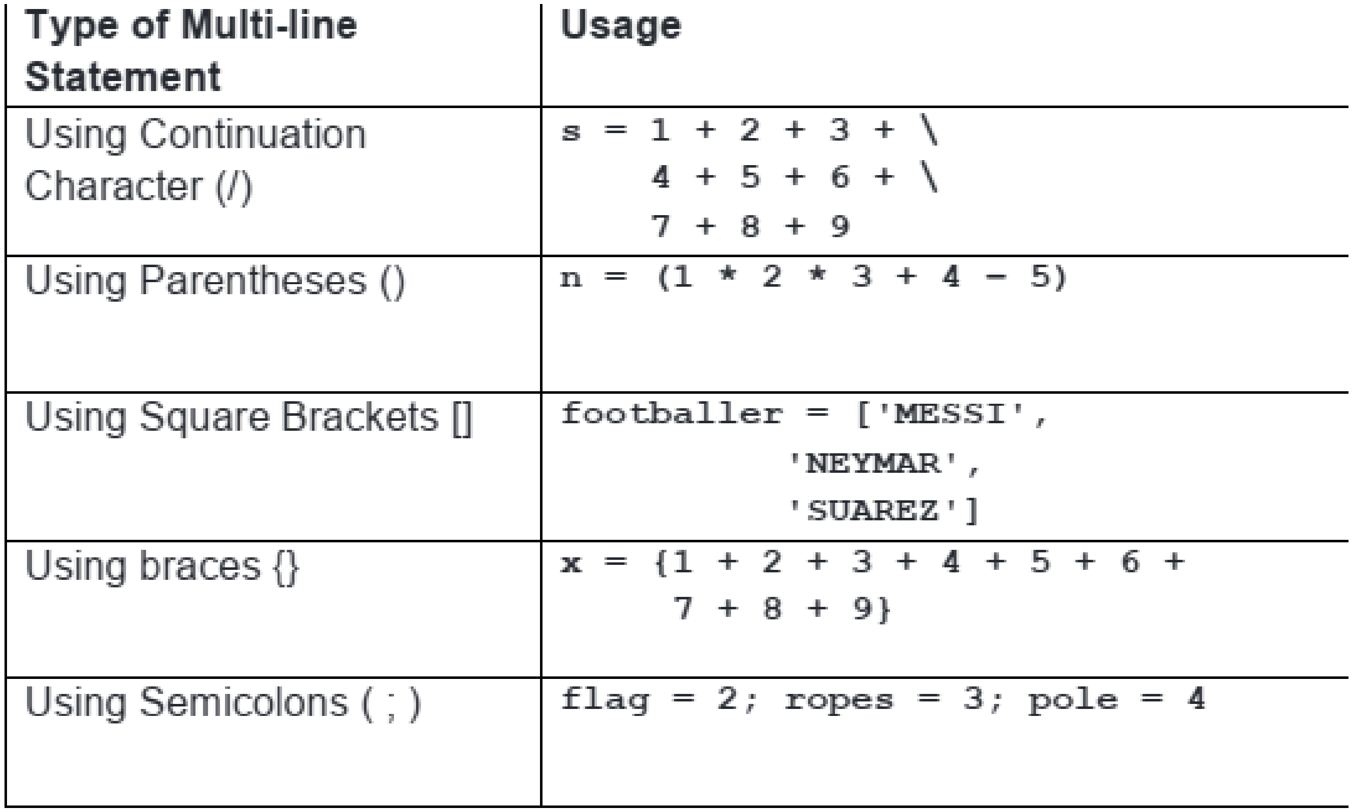
To write a Python script/program, we need to open a new file - **File >> New File**, type a sequence of Python statements for solving a problem, save it with a meaningful name – **File >> Save**, and finally **Run** the program to view the output of the program.

Python Statement and Comments

- 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

- Statements in Python can be extended to one or more lines using parentheses (), braces {}, square brackets [], semi-colon (;). When we need to do long calculations and cannot fit these statements into one line, we can make use of these characters.
- 

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)`

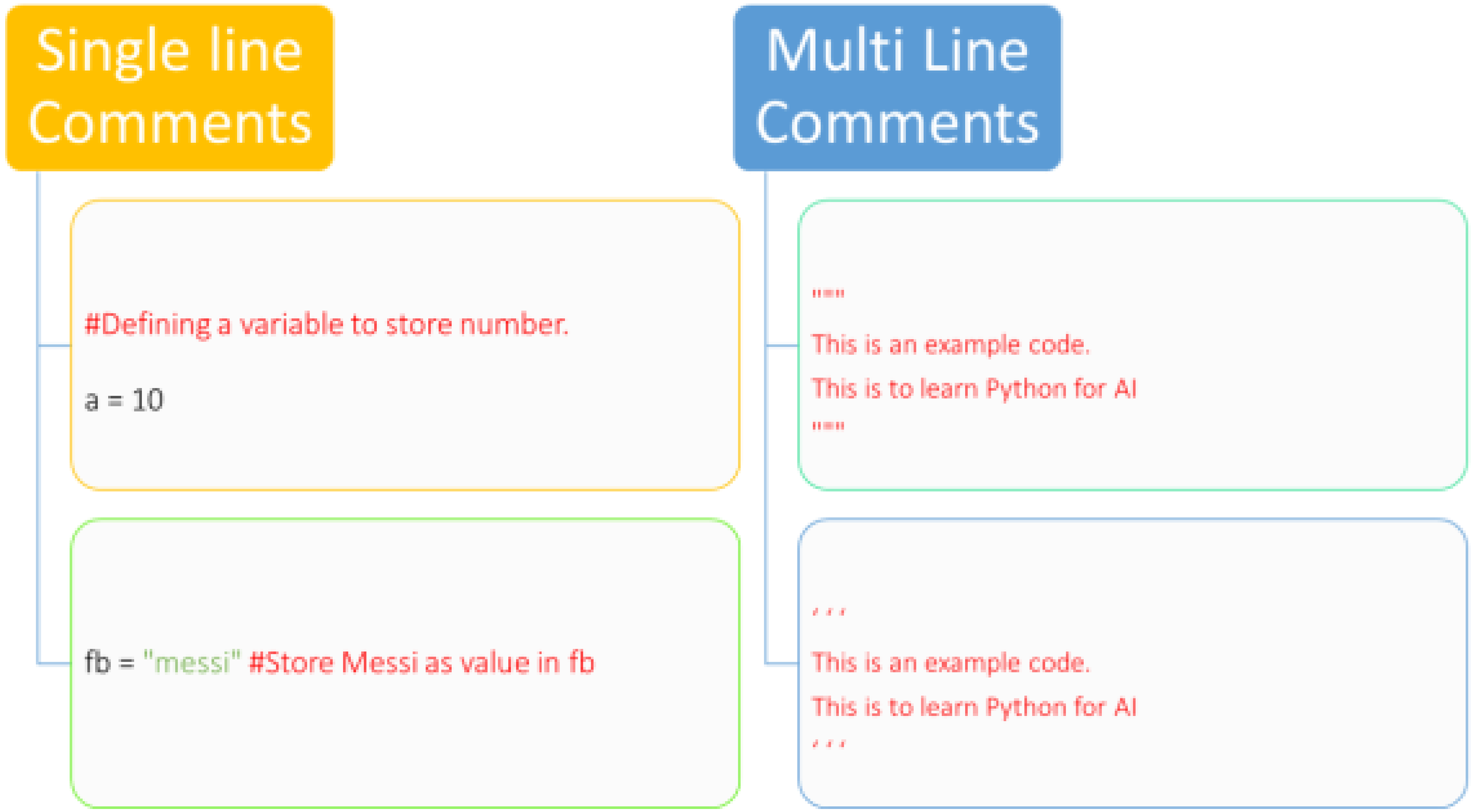
Using Square Brackets []	`footballer = ['MESSI',`
	`          'NEYMAR',`
	`          'SUAREZ']`

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 piece of text to let someone know what you have done in a program or what is being done in a block of code. In Python, we use the hash (#) symbol to start writing a comment.



```
#Defining a variable to store number.
a = 10
```

```
fb = "messi" #Store Messi as value in fb
```

```
"""
This is an example code.
This is to learn Python for AI
"""
```

```
'''
This is an example code.
This is to learn Python for AI
'''
```

Python keywords and identifiers

keywords (reserved words in Python) and identifiers (names given to variables, functions, etc.).

Hey, the other day I wrote a simple code " finally = 1 ", but it gave me an error "SyntaxError: invalid syntax" . I dont Know what is the problem
Oh, I know what the problem is - "Finally" is a keyword and you cannot use it as a variable. Let me tell you more about it.

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

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`	

An identifier is a name given to entities like class, functions, variables, etc. It helps to differentiate one entity from another.

Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore _.

An identifier cannot start with a digit. 1variable is invalid, but variable1 is perfectly fine.

Keywords cannot be used as identifiers.

We cannot use special symbols like !, @, #, $, % etc. in our identifier.

Identifier can be of any length.

Note:

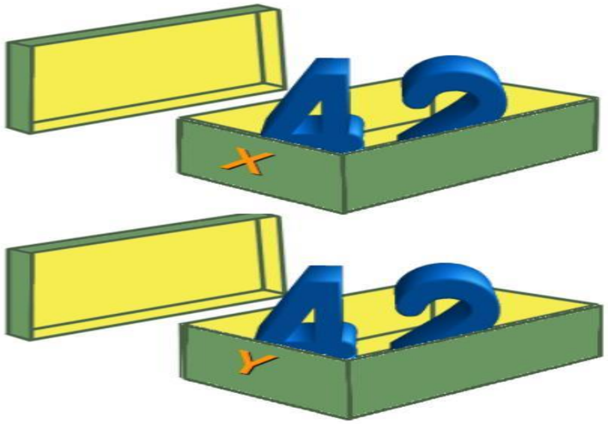
Python is a case-sensitive language. This means, `Variable` and `variable` are not the same. Always name identifiers that make sense.

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

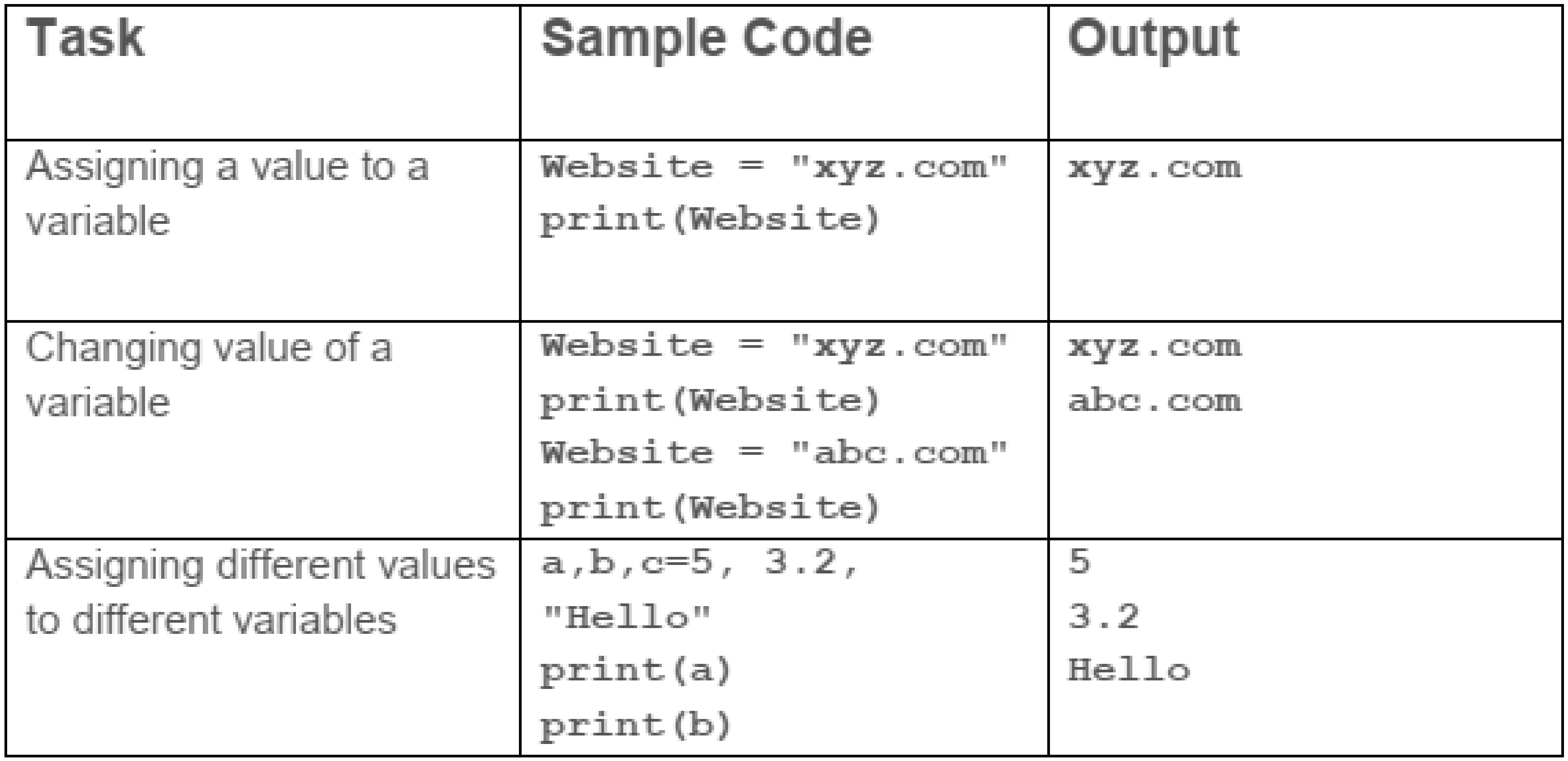
Variables and Datatypes

- **Variables**
  A variable is a named location used to store data in the memory. It is helpful to think of variables as a container that holds data which can be changed later throughout programming.
  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. It's like the two shoeboxes, which you can see in the picture. These shoeboxes are labelled with x and y and the corresponding values are stored in the shoeboxes.

➢ Examples on Variables:



Task	Sample Code	Output

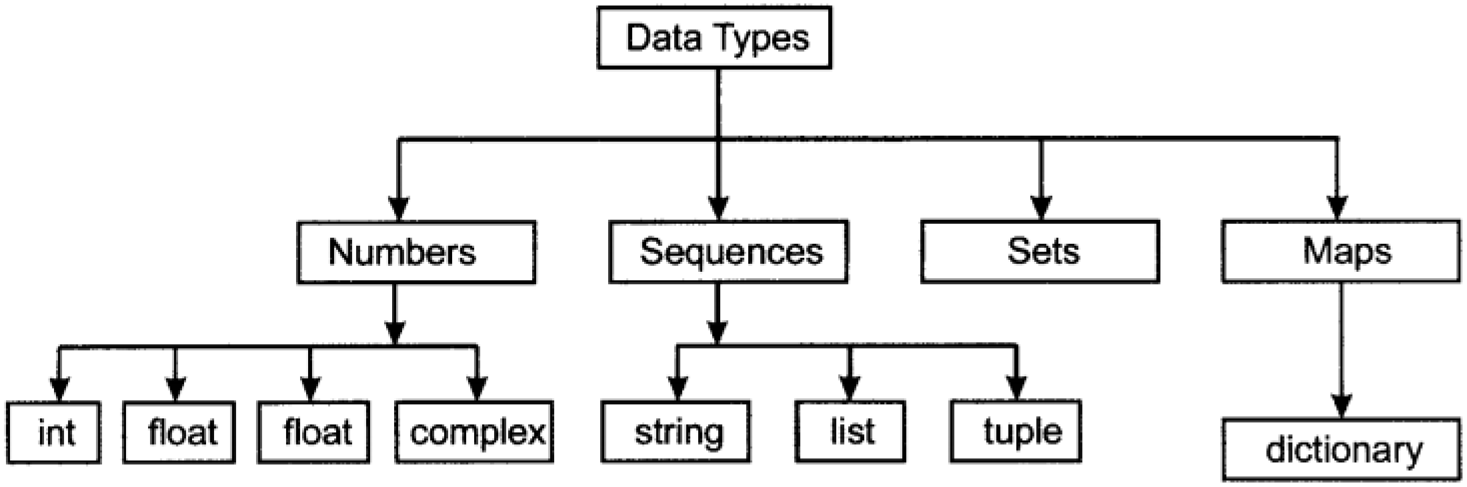
Assigning a value to a variable	`Website = "xyz.com"`
	`print(Website)`
		`xyz.com`

Changing value of a variable	`Website = "xyz.com"`
	`print(Website)`
	`Website = "abc.com"`
	`print(Website)`
		`xyz.com`
		`abc.com`

Assigning different values to different variables	`a,b,c=5, 3.2,`
	`"Hello"`
	`print(a)`
	`print(b)`
		`5`
		`3.2`
		`Hello`

Datatypes

- Every value in Python has a datatype. There are various data types in Python. Some of the important types are mentioned below in the image .
-



Python Data types

- **1) Python Numbers**
  Number data type stores Numerical Values.
  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,

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.

- **Sequence** :
- A sequence is an ordered collection of items, indexed by positive integers
  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.
  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.

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

One of the most important differences between list and tuple is that list is mutable, whereas a tuple is immutable. This means that lists can be changed, and tuples cannot be changed.

- **Sets**
  Set is an unordered collection of values, of any type, with no duplicate entry.
  Example:

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

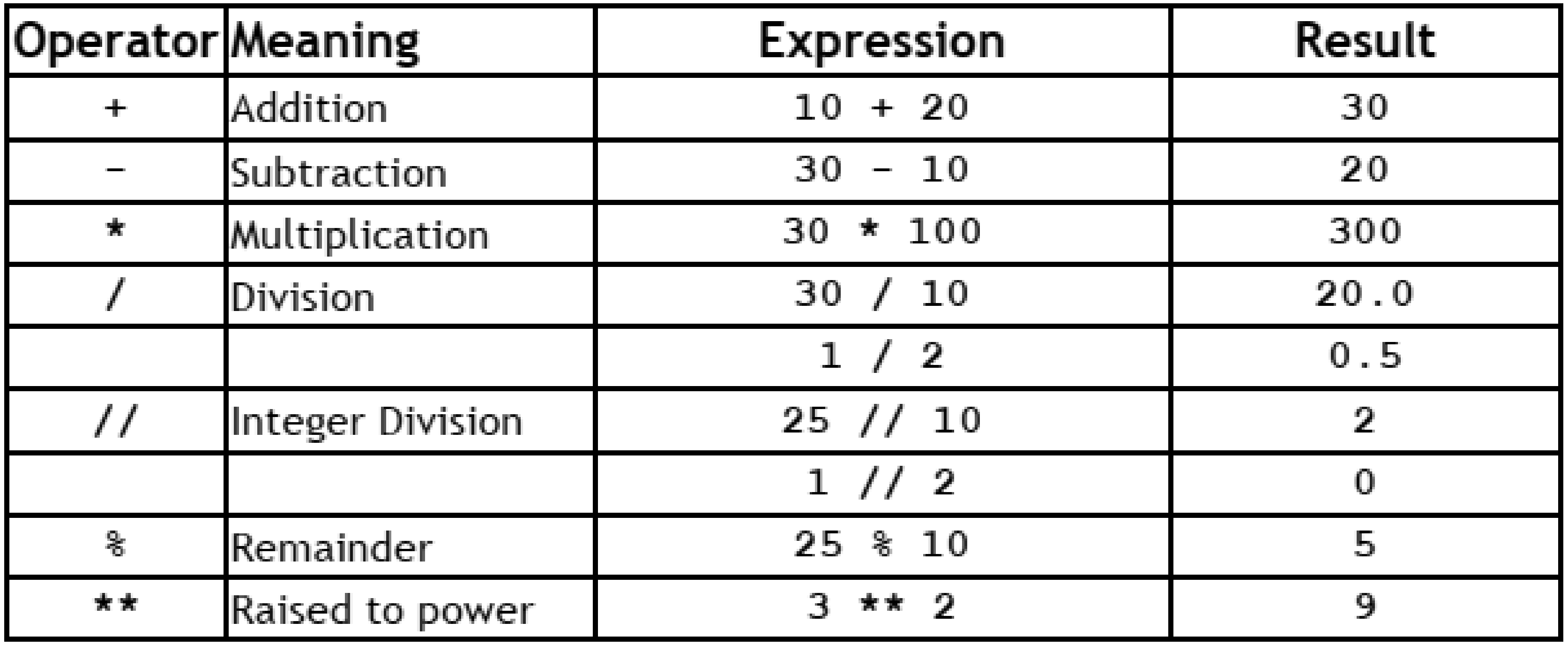
- **Mapping**
  This data type is unordered. Dictionaries fall under Mappings.
  Dictionaries
  Dictionary is an unordered collection of key-value pairs. It is generally used when we have a
  huge amount of data.
  In Python, dictionaries are defined within braces {} with each item being a pair in the
  form key: value. Key and value can be of any type.
  Example

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

Python Operators

Operators are special symbols which represent computation. They are applied on operand(s), which can be values or variables. Same operators can behave differently on different data types. Operators when applied on operands form an expression. Operators are categorized as Arithmetic, Relational, Logical and Assignment. Value and variables when used with operator are known as operands.

Arithmetic Operators



Operator	Meaning	Expression	Result
+	Addition	10 + 20	30
-	Subtraction	30 - 10	20
*	Multiplication	30 * 100	300
/	Division	30 / 10	20.0
		1 / 2	0.5
//	Integer Division	25 // 10	2
		1 // 2	0
%	Remainder	25 % 10	5
**	Raised to power	3 ** 2	9

Example Code	Sample Output

`a = 20`
`b = 10`
`print(a + b)`
	`30`

`print(15 + 35)`	`50`
`print("My name is Kabir")`	`My name is Kabir`

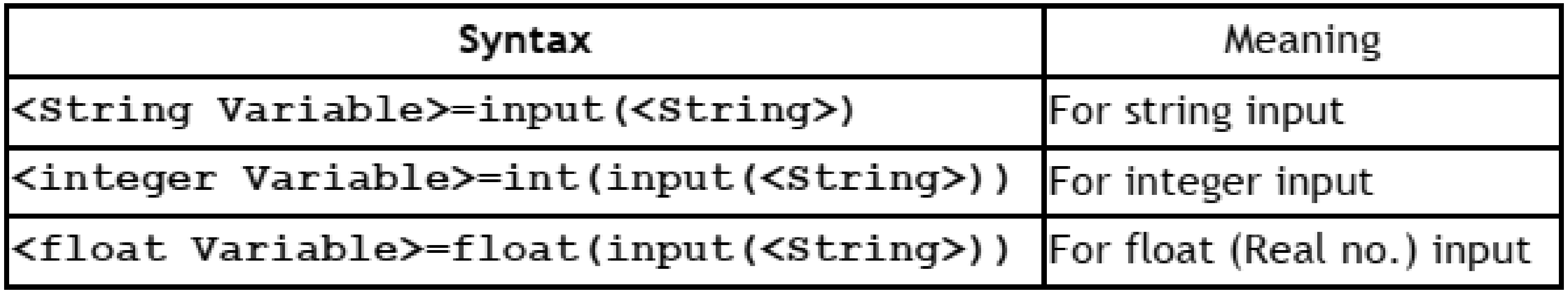
`a = "tarun"`
`print("My name is :",a)`
	`My name is : tarun`

`x = 1.3`
`print("x = /n", x)`
	`x =`
	`1.3`

`m = 6`
`print(" I have %d apples",m)`
	`I have 6 apples`

➢ 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 .



Syntax	Meaning
`<String Variable>=input(<String>)`	For string input
`<integer Variable>=int(input(<String>))`	For integer input
`<float Variable>=float(input(<String>))`	For float (Real no.) input

- Type Conversion
  The process of converting the value of one data type (integer, string, float, etc.) to another
  data type is called type conversion. Python has two types of type conversion.
  1. Implicit Type Conversion
  2. Explicit Type Conversion
  Implicit Type Conversion:
  In Implicit type conversion, Python automatically converts one data type to another data type. This process doesn't need any user involvement.

➢ Example:

```
# Code to calculate the Simple Interest

principle amount = 2000
roi = 4.5
time = 10

simple interest = (principle amount * roi * time)/100

print("datatype of principle amount : ", type(principle_amount))
print("datatype of rate of interest : ", type(roi))


print("value of simple interest : ", simple interest)
print("datatype of simple interest : ", type(simple_interest))
```

```
datatype of principle amount : <class 'int'>
datatype of rate of interest : <class 'float'>

value of simple interest : 900
datatype of simple interest : <class 'float'>
```

In the above program,

• We calculate the simple interest by using the variable priniciple_amount and roi with time divide by 100

• We will look at the data type of all the objects respectively.

• In the output we can see the datatype of principle_amount is an integer, datatype of roi is a float.

• Also, we can see the simple_interest has float data type because Python always converts smaller data type to larger data type to avoid the loss of data.

- Explicit Type Conversion
  In Explicit Type Conversion, users convert the data type of an object to required data type.
  We use the predefined functions like int(), float(), str(), etc to perform explicit type conversion. This type of conversion is also called typecasting because the user casts (changes) the data type of the objects.

  Syntax:

  (required_datatype)(expression)

```
Birth_day = 10
Birth_month = "July"

print("data type of Birth day before type casting :", type(Birth day))
print("data type of Birth month : ", type(Birth month))

Birth_day = str(Birth_day)
print("data type of Birth_day after type casting :",type(Birth_day))

Birth_date = Birth_day + Birth_month

print("birth date of the student : ", Birth_day)
print("data type of Birth date : ", type(Birth date))
```

When we run the above program, the output will be

```
data type of Birth day before type casting : <class 'int'>
data type of Birth month : <class 'str'>

data type of Birth day after type casting : <class 'str'>

birth date of the student : ' 10 July '
data type of Birth date : <class 'str'>
```

➢

Python Operators II

Comparison operators

Operator	Meaning	Expression	Result
>	Greater Than	`20 > 10`	`True`
		`15 > 25`	`False`
<	Less Than	`20 < 45`	`True`
		`20 < 10`	`False`
==	Equal To	`5 == 5`	`True`
		`5 == 6`	`False`
!=	Not Equal to	`67 != 45`	`True`
		`35 != 35`	`False`
>=	Greater than or Equal to	`45 >= 45`	`True`
		`23 >= 34`	`False`
<=	Less than or equal to	`13 <= 24`	`True`
		`13 <= 12`	`False`

➢

Logical operators

Operator	Meaning	Expression	Result
And	And operator	True and True	True
		True and False	False
Or	Or operator	True or False	True
		False or False	False
Not	Not Operator	not False	True
		not True	False

Assignment operators

Assignment operators are used in Python to assign values to variables.

Operator	Expression	Equivalent to
=	x=5	x = 5
+=	x +=5	x = x + 5
-=	x -= 5	x = x - 5
*=	x *= 5	x = x * 5
/=	x /= 5	x = x / 5

Python Operators

➢

Identity Operators :

➢ Identity operators are used to determine whether the value of a variable is of a certain type or not. There are two identity operators.

Operator	Description	Example (Try in Lab)

is	Evaluates True if the variables on either side of the operator point towards the same memory location and False otherwise. var1 is var2 results to True if id(var1) is equal to id(var2)	>>> num1 = 5
		>>> type(num1) is int
		True
		>>> num2 = num1
		>>> id(num1)
		1433920576
		>>> id(num2)
		1433920576
		>>> num1 is num2
		True

is not	Evaluates to False if the variables on either side of the operator point to the same memory location and True otherwise. Var1 is not var2 results to True if id(var1) is not equal to id(var2)	>>> num1 is not num2 False

Python Operators

➢

Membership Operators :

➢ Membership operators are used to check if a value is a member of the given sequence or not.

Operator	**Description**	**Example (Try in Lab)**

in	Returns True if the variable/value is found in the
	the
	specified sequence and False otherwise
		>>> a = [1,2,3]
		>>> 2 in a
		True
		>>> '1' in a
		False

not in	Returns True if the variable/value is not found in
	in
	the specified sequence and False otherwise
		>>> a = [1,2,3]
		>>> 10 not in a
		True
		>>> 1 not in a
		False

Python Operators

➢

Precedence of Operators :

- Evaluation of the expression is based on precedence of operators. When an expression contains different kinds of operators, precedence determines which operator should be applied first.
- Higher precedence operator is evaluated before the lower precedence operator. Most of the operators studied till now are binary operators. Binary operators are operators with two operands.

.

Python Operators

Order of Precedence	Operators	Description
1	**	Exponentiation (raise to the power)
2	~ ,+, -	Complement, unary plus and unary minus
3	* ,/, %, //	Multiply, divide, modulo and floor division
4	+, -	Addition and subtraction
5	<= , < , > , >=, == , !=	Relational and Comparison operators
6	=, %=, /=, //=, -=, +=, *=, **=	Assignment operators
7	is, is not	Identity operators
8	in, not in	Membership operators
9	not	Logical operators
10	and	
11	or	

Note:

a) Parenthesis can be used to override the precedence of operators. The expression within () is evaluated first.

b) For operators with equal precedence, the expression is evaluated from left to right.

How will Python evaluate the following expression?

➢ 20 + 30 * 40

= 20 + (30 * 40)	#Step 1
#precedence of * is more than that of + = 20 + 1200	#Step 2
= 1220	#Step 3

- How will Python evaluate the following expression?
  20 - 30 + 40
- The two operators (–) and (+) have equal precedence. Thus, the first operator, i.e., subtraction is applied before the second operator, i.e., addition (left to right).

-
  ```
  = (20 – 30) + 40        #Step 1
  = -10 + 40              #Step 2
  = 30                    #Step 3
  ```
- How will Python evaluate the following expression?
  (20 + 30) * 40
- How will the following expression be evaluated in Python?
- 15.0 / 4 + (8 + 3.0)

- Program of explicit type conversion from int to float.

-
  ```
  num1 = 10
  num2 = 20
  num3 = num1 + num2
  print(num3)
  print(type(num3))
  num4 = float(num1 + num2)
  print(num4)
  print(type(num4))
  ```

- o/p :

-
  ```
  30
  <class 'int'>
  30.0
  <class 'float'>
  ```

Program of explicit type conversion from float to int.

```
num1 = 10.2
num2 = 20.6
num3 = (num1 + num2)
print(num3)
print(type(num3))
num4 = int(num1 + num2)
print(num4)
print(type(num4) )
```

.

- Type conversion between numbers and strings.

-
  ```
  priceIcecream = 25
  priceBrownie = 45
  totalPrice = priceIcecream + priceBrownie
  print("The total is Rs." + totalPrice )
  ```

```
priceIcecream = 25
priceBrownie = 45
totalPrice = priceIcecream + priceBrownie
print("The total in Rs." + str(totalPrice))
```

➢ Which of the following identifier names are invalid and why?

i	Serial_no.	v	Total_Marks
ii	1st_Room	vi	total-Marks
iii	Hundred$	vii	_Percentage
iv	Total Marks	viii	True

Question 5. Write the output of the following:

```
a) num1 = 4
num2 = num1 + 1
num1 = 2
print (num1, num2)
b) num1, num2 = 2, 6
num1, num2 = num2, num1 + 2
print (num1, num2)
c) num1, num2 = 2, 3
num3, num2 = num1, num3 + 1
print (num1, num2, num3)
```

.

Answer:

a) num1 = 4

num2 = num1 + 1

num1 = 2

Output: 2,5

.

➢

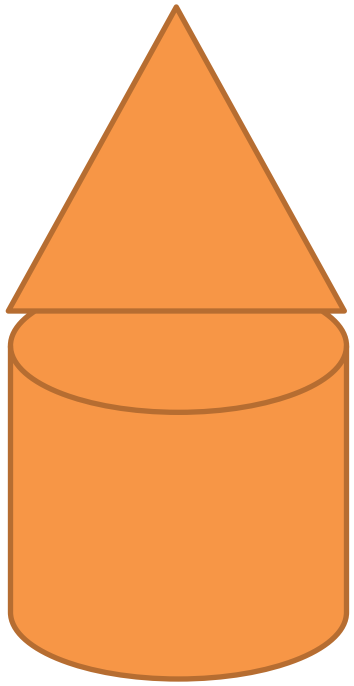
Which data type will be used to represent the following data values and why?
a) Number of months in a year
b) Resident of Delhi or not
c) Mobile number
d) Pocket money
e) Volume of a sphere
f) Perimeter of a square
g) Name of the student
h) Address of the student

.

DATA VALUES	DATA TYPES	REASON
Number of months in a year	Integer	Number of months contain only number values (integer).
Resident of Delhi or not	Boolean	It gives the answer in the form of true and false
Mobile number	Integer	Mobile number only contain integer number value
Pocket money	Float	Money can be count as 100rs 50 paisa(100.50)
Volume of a sphere	Float	The volume can be calculated in the decimal point
Perimeter of a square	Float	The perimeter can be calculated in the decimal point
Name of the student	String	Name is a set of character, hence data type of name of student is string
Address of the student	String	Address is a set of character, hence data type of address of student is string

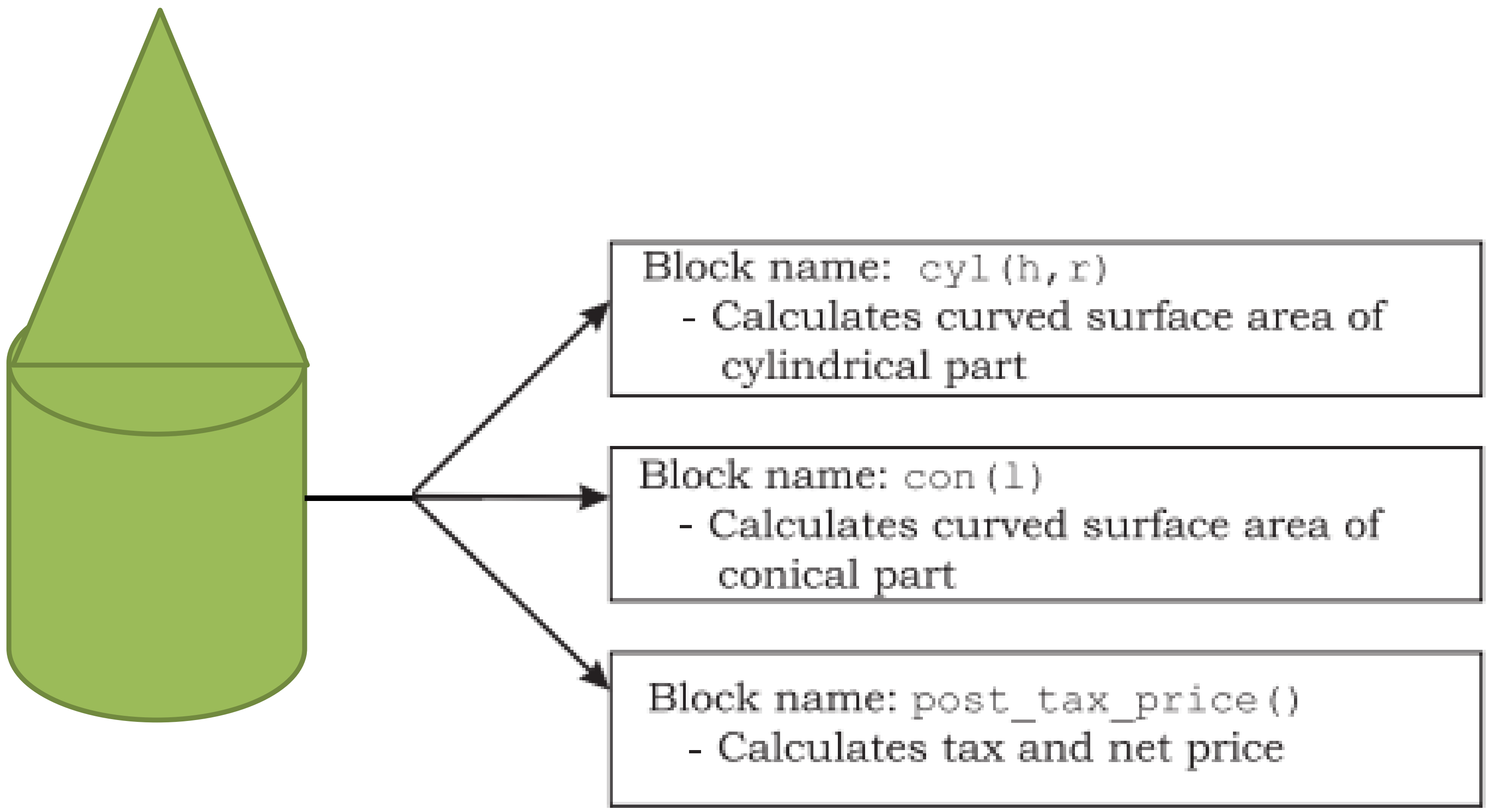
Functions

- Till now we have written some programs and might have realised that as the problem gets complex, the number of lines in a program increase, which makes the program look bulky and difficult to manage.
- The company performs the following tasks to fix the selling price of each tent.
  1. Accept user requirements for the tent, such as
  a) height
  b) radius
  c) slant height of the conical part
- 2. Calculate the area of the canvas used
- 3. Calculate the cost of the canvas used for making the tent.
- 4. Calculate the net payable amount by the customer that is inclusive of the 18% tax



➢
```
print( "Enter values for the cylindrical part of the tent in meters\n")
h = float(input("Enter height of the cylindrical part: "))
r = float(input("Enter radius: "))
l = float(input("Enter the slant height of the conical part in meters: "))
csa_conical = 3.14*r*l              #Area of conical part
csa_cylindrical = 2*3.14*r*h     #Area of cylindrical part
# Calculate area of the canvas used for making the tent
canvas_area = csa_conical + csa_cylindrical
print("The area of the canvas is",canvas_area,"m^2")
#Calculate cost of the canvas
unit_price = float(input("Enter the cost of 1 m^2 canvas: "))
total_cost= unit_price * canvas_area
print("The total cost of canvas = ",total_cost)
#Add tax to the total cost to calculate net amount payable by the
#customer
tax = 0.18 * total_cost;
net_price = total_cost + tax
print("Net amount payable = ",net_price)
```

- Another approach to solve the above problem is to divide the program into different blocks of code as shown in Figure.


Block name: cyl(h,r)
- Calculates curved surface area of cylindrical part
Block name: con(l)
- Calculates curved surface area of conical part
Block name: post_tax_price()
- Calculates tax and net price

The process of dividing a computer program into separate independent blocks of code or separate sub-problems with different names and specific functionalities is known as modular programming.

Functlons

- In programming, the use of function is one of the means to achieve **modularity** and **reusability**. Function can be defined as a named group of instructions that accomplish a specific task when it is invoked. Once defined, a function can be **called repeatedly** from different places of the program without writing all the codes of that function every time, or it can be called from inside another function, by simply writing the name of the function and passing the required parameters

Functlons

➢
```
#function definition
def cyl(h,r):
area_cyl = 2*3.14*r*h #Area of cylindrical part
return(area_cyl)
#function definition
def con(l,r):
area_con = 3.14*r*l #Area of conical part
return(area_con)
#function definition
def post_tax_price(cost): #compute payable amount for the tent
tax = 0.18 * cost;
net_price = cost + tax
return(net_price)
print("Enter values of cylindrical part of the tent in meters:")
h = float(input("Height: "))
r = float(input("Radius: "))
csa_cyl = cyl(h,r) #function call
l = float(input("Enter slant height of the conical area in meters: "))
csa_con = con(l,r) #function call
#Calculate area of the canvas used for making the tent
canvas_area = csa_cyl + csa_con
print("Area of canvas = ",canvas_area," m^2")
#Calculate cost of canvas
unit_price = float(input("Enter cost of 1 m^2 canvas in rupees: "))
total_cost = unit_price * canvas_area
print("Total cost of canvas before tax = ",total_cost)
print("Net amount payable (including tax) = ",post_tax_price(total_cost))
```

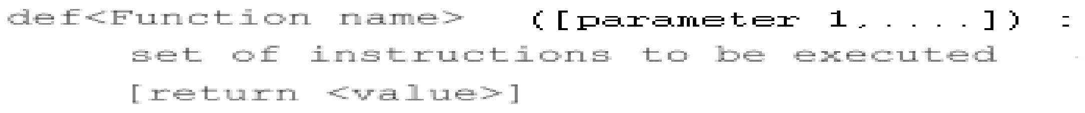
The Advantages of Function

- Suppose in further the company decides to design another type of tent whose base is rectangular, while the upper part remains the same. In such a scenario, some part of the existing code can be reused by calling the function con(l,r).
- If the company develops other products or provides services, and where 18% tax rate is to be applied, the programmer can use the function post_tax_price(cost)directly.
-

- ➢ Thus, following are the advantages of using functions in a program:
  • **Increases readability**, particularly for longer code as by using functions, the program is better organised and easy to understand.
- ➢
  • **Reduces code length** as same code is not required to be written at multiple places in a program. This also makes debugging easier.
- ➢
  • **Increases reusability**, as function can be called from another function or another program. Thus, we can reuse or build upon already defined functions and avoid repetitions of writing the same piece of code.
- ➢
  • Work can be easily divided among team members and completed in parallel.

User defined functions

- ➢ A function definition begins with def (short for define). The syntax for creating a user defined function is as follows:



```
def<Function name>   ([parameter 1,....]) :
     set of instructions to be executed
     [return <value>]
```

  The items enclosed in "[ ]" are called parameters and they are optional. Hence, a function may or may not have parameters. Also, a function may or may not return a value.
- ➢ Function header always ends with a colon (:).
- ➢ Function name should be unique. Rules for naming identifiers also applies for function naming.
  • The statements outside the function indentation are not considered as part of the function

Functions

- Write a user defined function to add 2 numbers and display their sum.
- #The requirements are listed below:
  #1. We need to accept 2 numbers from the user.
  #2. Calculate their sum
  #3. Display the sum.

```
#function definition
def addnum():
fnum = int(input("Enter first number: "))
snum = int(input("Enter second number: "))
sum = fnum + snum
print("The sum of ",fnum,"and ",snum,"is ",sum)
#function call
addnum()
```

Arguments and Parameters

- In the above example, the numbers were accepted from the user within the function itself, but it is also possible for a user defined function to receive values at the time of being called.
- An argument is a value passed to the function during the function call which is received in corresponding parameter defined in function header.

Write a program using a user defined function that displays sum of first n natural numbers, where n is passed as an argument.

- #The requirements are:
  #1. n be passed as an argument
  #2. Calculate sum of first n natural numbers
  #3. Display the sum

Arguments and Parameters

-
  ```
  #function header
  def sumSquares(n): #n is the parameter
      sum = 0
      for i in range(1,n+1):
      sum = sum + i
      print("The sum of first",n,"natural numbers is: ",sum)
  ```

```
num = int(input("Enter the value for n: "))
#num is an argument referring to the value input by the user
sumSquares(num) #function call
```

Functions Returning Value

- ➢ A function may or may not return a value when called. The return statement returns the values from the function.
- ➢ If They do not return any value, Such functions are called void functions. But a situation may arise, wherein we need to send value(s) from the function to its calling function. This is done using return statement. The return statement does the following:
  - • returns the control to the calling function.
  - • return value(s) or None.
- ➢ Write a program using user defined function calcPow() that accepts base and exponent as arguments and returns the value Baseexponent where Base and exponent are integers

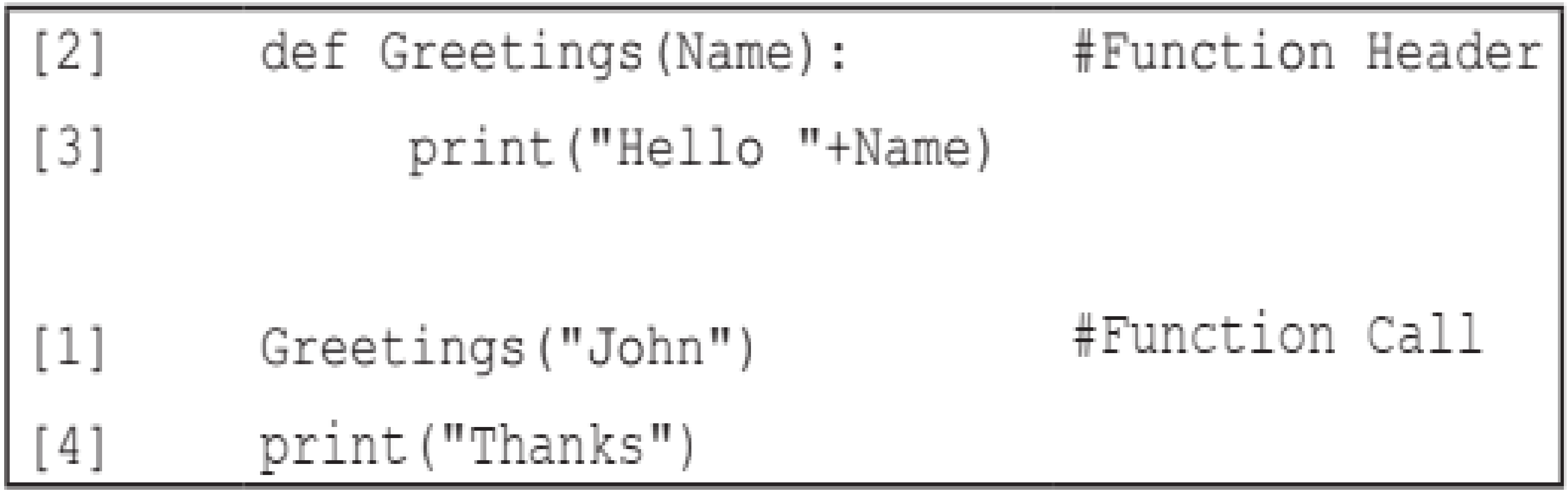
➢

```
#The requirements are listed below:
#1. Base and exponent are to be accepted as arguments.
#2. Calculate Baseexponent
#3. Return the result (use return statement )
#4. Display the returned value.
def calcpow(number, power): #function definition
        result = 1
        for i in range(1,power+1):
        result = result * number
        return result
base = int(input("Enter the value for the Base: "))
expo = int(input("Enter the value for the Exponent: "))
answer = calcpow(base,expo) #function call
print(base,"raised to the power",expo,"is",answer)
```

Flow of Execution

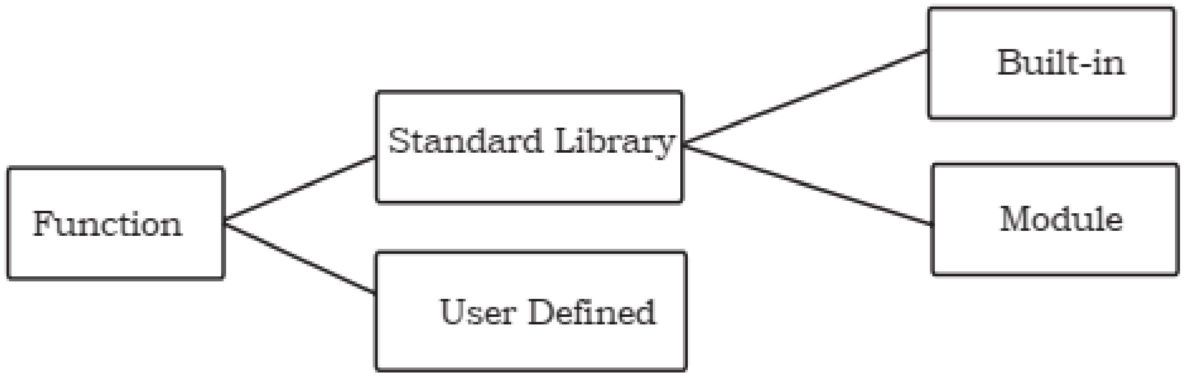
- Flow of execution can be defined as the order in which the statements in a program are executed.
- The Python interpreter starts executing the instructions in a program from the first statement.
- The statements are executed one by one, in the order of appearance from top to bottom.
- When the interpreter encounters a function definition, the statements inside the function are not executed until the function is called.
- when the interpreter encounters a function call, there is a little deviation in the flow of execution. In that case, instead of going to the next statement, the control jumps to the called function and executes the statement of that function .
- After that, the control comes back the point of function call so that the remaining statements in the program can be executed.
- It is also important to note that a function must be defined before its call within a program.

Flow of Execution



```
[2]     def Greetings(Name):            #Function Header
[3]          print("Hello "+Name)


[1]     Greetings("John")               #Function Call
[4]     print("Thanks")
```

Python standard library

➢ 

Python has a very extensive standard library. It is a collection of many built in functions that can be called in the program as and when required, thus saving programmer's time of creating those commonly used functions every time.

➢

Built-in functions

- Built-in functions are the ready-made functions in Python that are frequently used in programs.
- #Program to calculate square of a number

```
a = int(input("Enter a number: ")
b = a * a
print(" The square of ",a ,"is", b)
```

- In the above program **input(), int()** and **print()** are the built-in functions. The set of instructions to be executed for these built-in functions are already defined in the python interpreter.

Following is a categorised list of some of the frequently used built-in functions in Python:

Built-in Functions

Input or Output	Datatype Conversion	Mathematical Functions
input()	bool()	abs()
print()	chr()	divmod()
	dict()	max()
	float()	min()
	int()	pow()
	list()	sum()
	ord()	
	set()	
	str()	
	tuple()	

Module

- Other than the built-in functions, the Python standard library also consists of a number of modules.
- A module is a file containing Python definitions (i.e. functions) and statements. Standard library of Python is extended as module(s) to a programmer.
- Definitions from the module can be used within the code of a program.
- To use a module, we need to import the module. Once we import a module, we can directly use all the functions of that module. The syntax of import statement is as follows:

```
import modulename1 [,modulename2, …]
```

This gives us access to all the functions in the module(s). To call a function of a module, the function name should be preceded with the name of the module with a dot(.) as a separator. The syntax is as shown below:

```
modulename.functionname()
```

Module name : math

- **import math**
-

Function Syntax	**Arguments**	**Returns**	**Example Output**

math.ceil(x)	x may be an integer or floating point number	ceiling value of x	>>> math.ceil(-9.7)
			-9
			>>> math.ceil (9.7)
			10
			>>> math.ceil(9)
			9

math.floor(x)	x may be an integer or floating point number	floor value of x	>>> math.floor(-4.5)
			-5
			>>> math.floor(4.5)
			4
			>>> math.floor(4)
			4

- import statement can be written anywhere in the program
  - Module must be imported only once
  - In order to get a list of modules available in Python, we can use the following statement:

```
>>> help("module")
```

  - To view the content of a module say math, type the following:

```
>>> help("math")
```

From Statement

➢

Instead of loading all the functions into memory by importing a module, from statement can be used to access only the required functions from a module. It loads only the specified function(s) instead of all the functions in a module.
Its syntax is
>>> from modulename import functionname [, functionname ,……]

Create a user defined module basic_math that contains the following user defined functions:

➢

1. To add two numbers and return their sum.
2. To subtract two numbers and return their difference.
3. To multiply two numbers and return their product.
4. To divide two numbers and return their quotient and print “Divisionby Zero” error if the denominator is zero.
5. Also add a docstring to describe the module. After creating module, import and execute functions.

#The requirement is:
#1. Write a docstring describing the module.
#2. Write user defined functions as per the specification.
#3. Save the file.
#4. Import at shell prompt and execute the functions.

"""Docstrings""" is also called Python documentation strings. It is a multiline comment that is added to describe the modules, functions, etc. They are typically added as the first line, using 3 double quotes.

```
"""
basic_math Module
*******************
➢ This module contains basic arithmetic operations
that can be carried out on numbers
"""
#Beginning of module
def addnum(x,y):
return(x + y)
def subnum(x,y):
return(x - y)
def multnum(x,y):
return(x * y)
def divnum(x,y):
if y == 0:
print ("Division by Zero Error")
else:
return (x/y) #End of module
```

Expressions in Python

- An expression is a combination of operators and operands that is interpreted to produce some other value.
- In any programming language, an expression is evaluated as per the precedence of its operators.
- So that if there is more than one operator in an expression, their precedence decides which operation will be performed first.

- **Constant Expressions:** These are the expressions that have constant values only.

  # Constant Expressions
- x = 15 + 1.3
- print(x)
- **Output**16.3

- **Arithmetic Expressions:** An arithmetic expression is a combination of numeric values, operators, and sometimes parenthesis. The result of this type of expression is also a numeric value. The operators used in these expressions are arithmetic operators like addition, subtraction, etc.
- # Arithmetic Expressions
- x = 40
- y = 12

- add = x + y
- sub = x - y
- pro = x * y
- div = x / y

- print(add)
- print(sub)
- print(pro)
- print(div)
- **Output** 52 ,   28 ,   480 ,   3.3333333333333335

- **Integral Expressions:** These are the kind of expressions that produce only integer results after all computations .
- # Integral Expressions
- a = 13
- b = 12.0
- c = a + int(b)
- print(c)
- o/p : 25

- **Floating Expressions:** These are the kind of expressions which produce floating point numbers as result after all computations .
- # Floating Expressions
- a = 13
- b = 5

- c = a / b
- print(c)
- **Output**2.6

- **Relational Expressions:** In these types of expressions, arithmetic expressions are written on both sides of relational operator (> , < , >= , <=). produce a boolean output in the end. These expressions are also called Boolean expressions.
- # Relational Expressions
- a = 21
- b = 13
- c = 40
- d = 37

- p = (a + b) >= (c - d)
- print(p)

- **Output**
- True

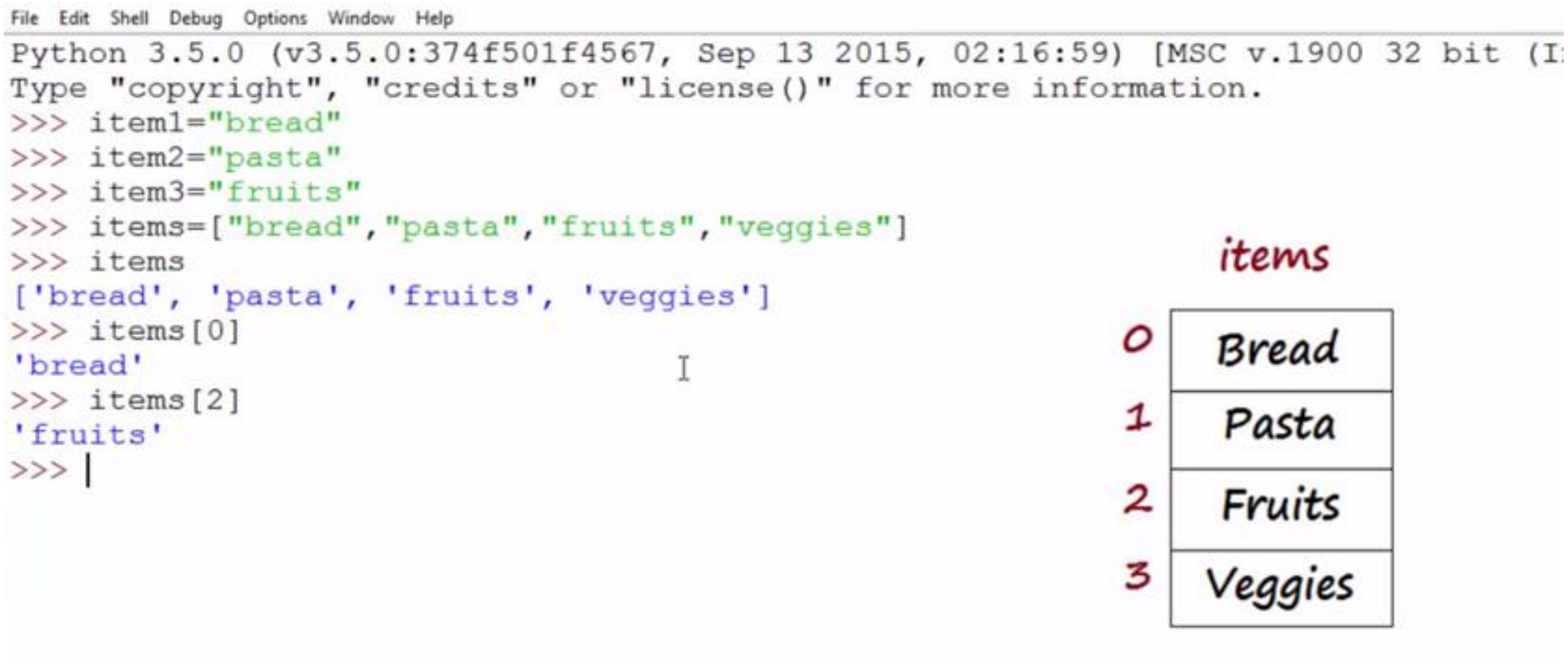
- **Logical Expressions:** These are kinds of expressions that result in either *True* or *False.* It basically specifies one or more conditions. For example, (10 == 9) is a condition if 10 is equal to 9. As we know it is not correct, so it will return False.

Operator	**Syntax**	**Functioning**
and	P and Q	It returns true if both P and Q are true otherwise returns false
or	P or Q	It returns true if at least one of P and Q is true
not	not P	It returns true if condition P is false

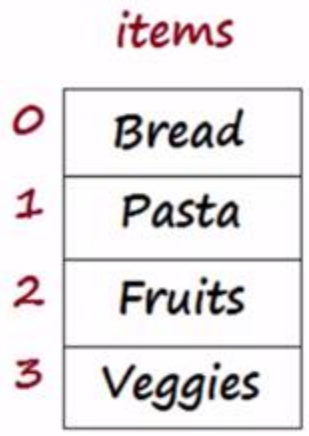
```
P = (10 == 9)
Q = (7 > 5)

# Logical Expressions
R = P and Q
S = P or Q
T = not P

print(R)
print(S)
print(T)
```

List

➢

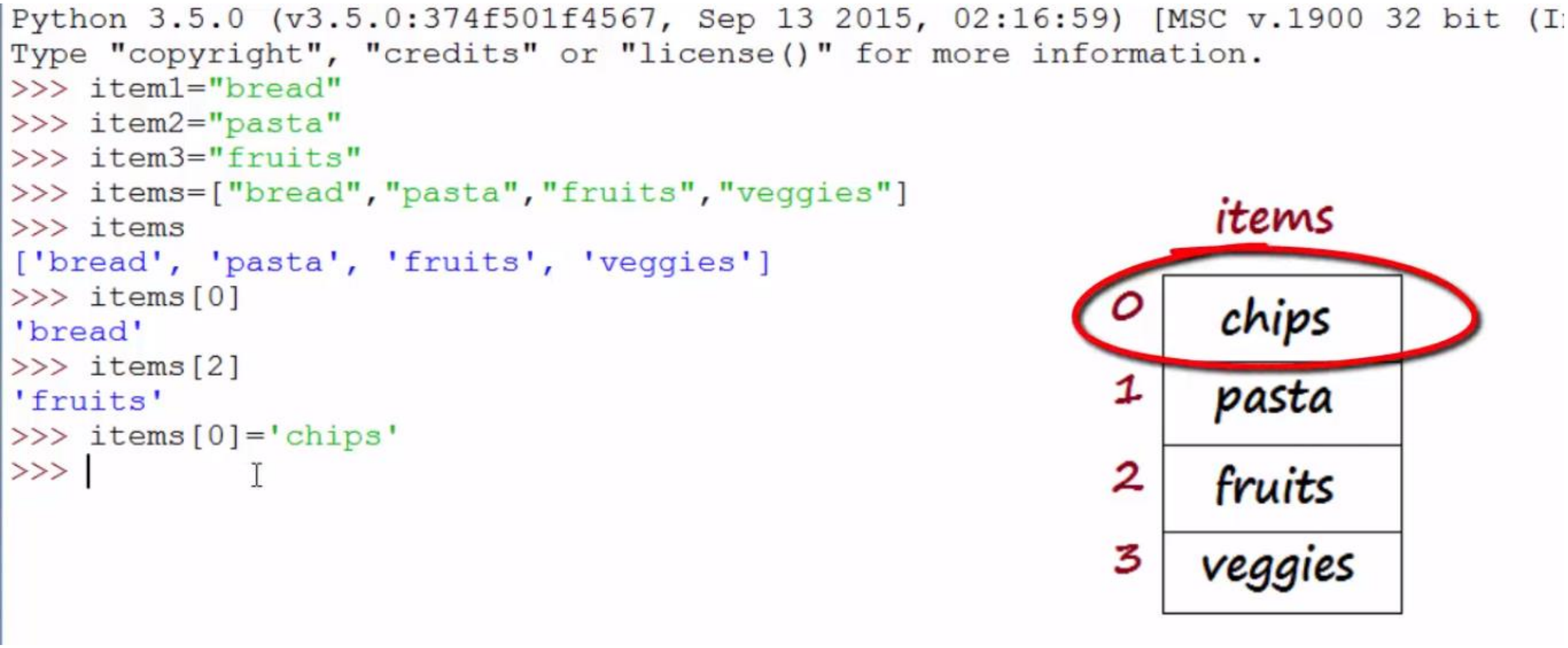


```
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:16:59) [MSC v.1900 32 bit (I
Type "copyright", "credits" or "license()" for more information.
>>> item1="bread"
>>> item2="pasta"
>>> item3="fruits"
>>> items=["bread","pasta","fruits","veggies"]
>>> items
['bread', 'pasta', 'fruits', 'veggies']
>>> items[0]
'bread'
>>> items[2]
'fruits'
>>> 
```

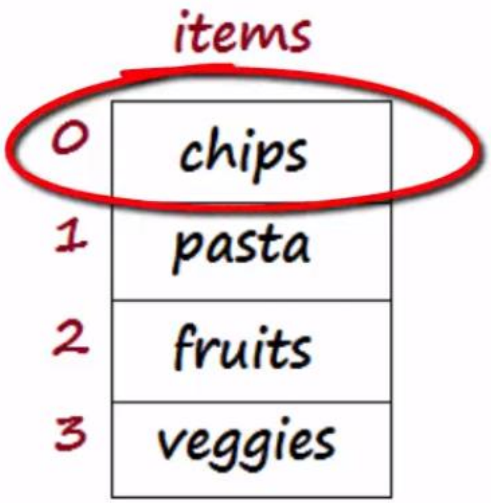


List

➢

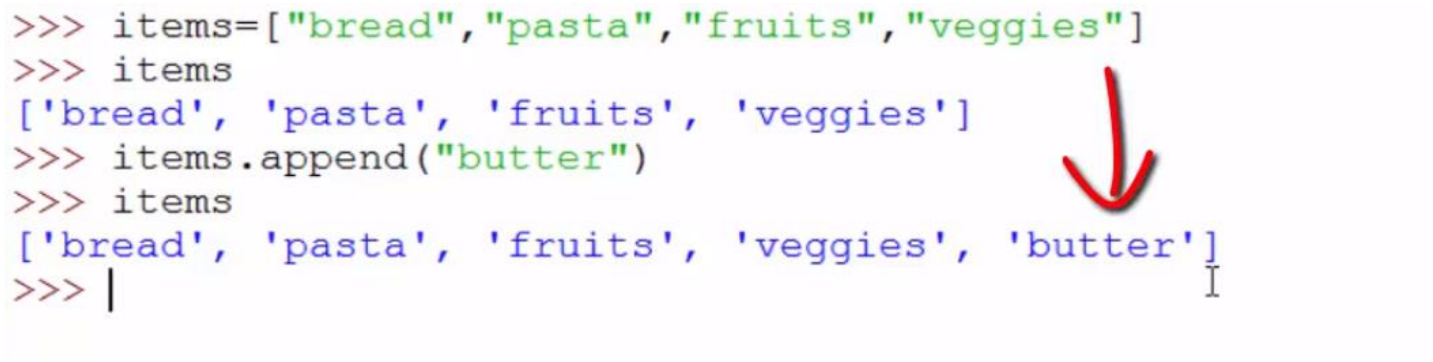


```
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13 2015, 02:16:59) [MSC v.1900 32 bit (I
Type "copyright", "credits" or "license()" for more information.
>>> item1="bread"
>>> item2="pasta"
>>> item3="fruits"
>>> items=["bread","pasta","fruits","veggies"]
>>> items
['bread', 'pasta', 'fruits', 'veggies']
>>> items[0]
'bread'
>>> items[2]
'fruits'
>>> items[0]='chips'
>>> 
```

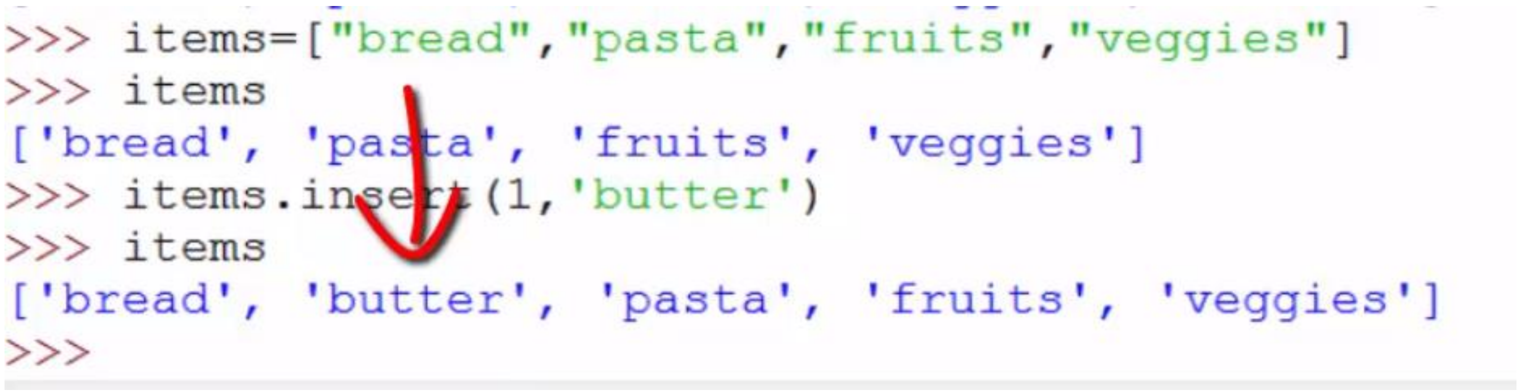


List

➢



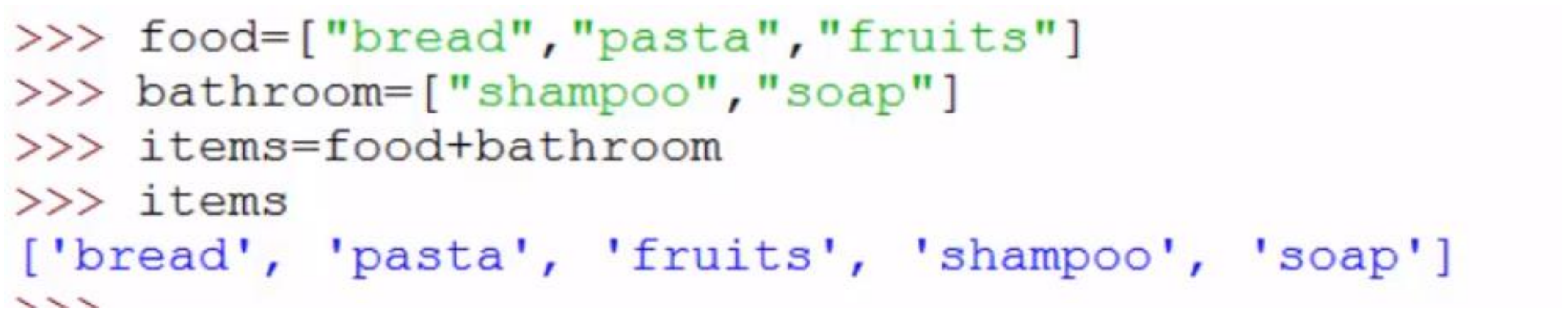
```
>>> items=["bread","pasta","fruits","veggies"]
>>> items
['bread', 'pasta', 'fruits', 'veggies']
>>> items.append("butter")
>>> items
['bread', 'pasta', 'fruits', 'veggies', 'butter']
>>> 
```



```
>>> items=["bread","pasta","fruits","veggies"]
>>> items
['bread', 'pasta', 'fruits', 'veggies']
>>> items.insert(1,'butter')
>>> items
['bread', 'butter', 'pasta', 'fruits', 'veggies']
>>> 
```

List

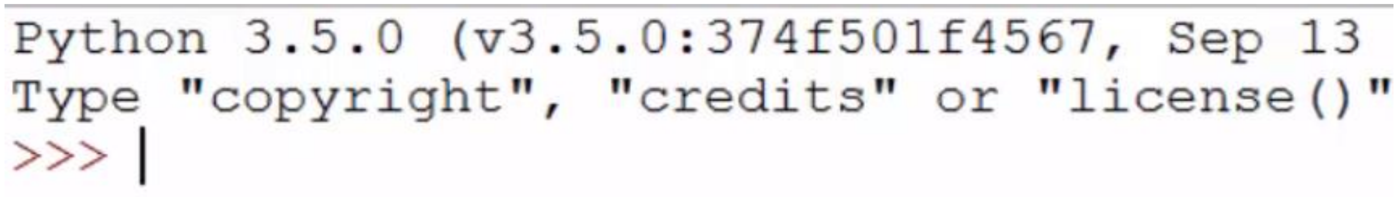
➢



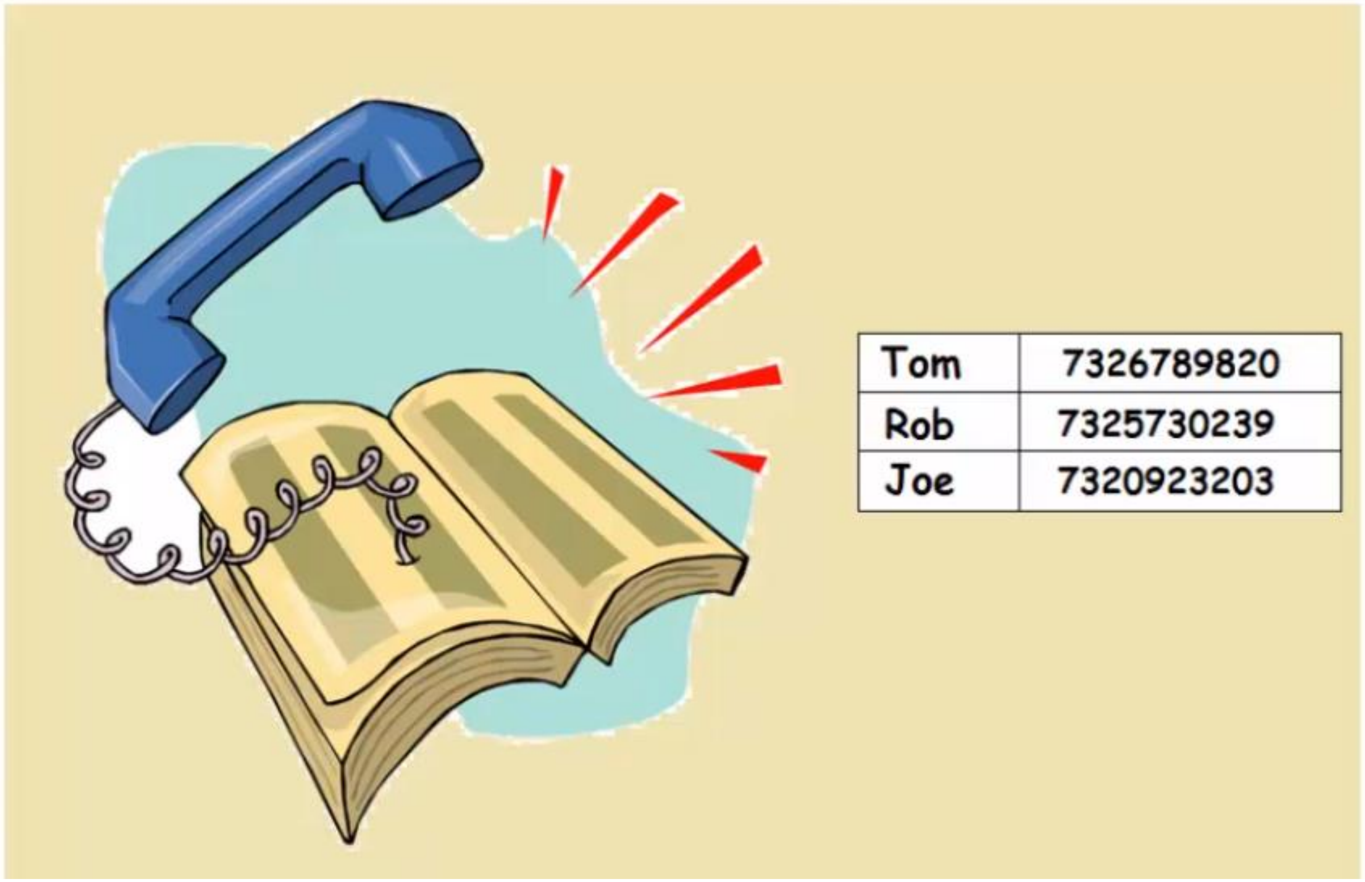
```
>>> food=["bread","pasta","fruits"]
>>> bathroom=["shampoo","soap"]
>>> items=food+bathroom
>>> items
['bread', 'pasta', 'fruits', 'shampoo', 'soap']
```

Dist

➢

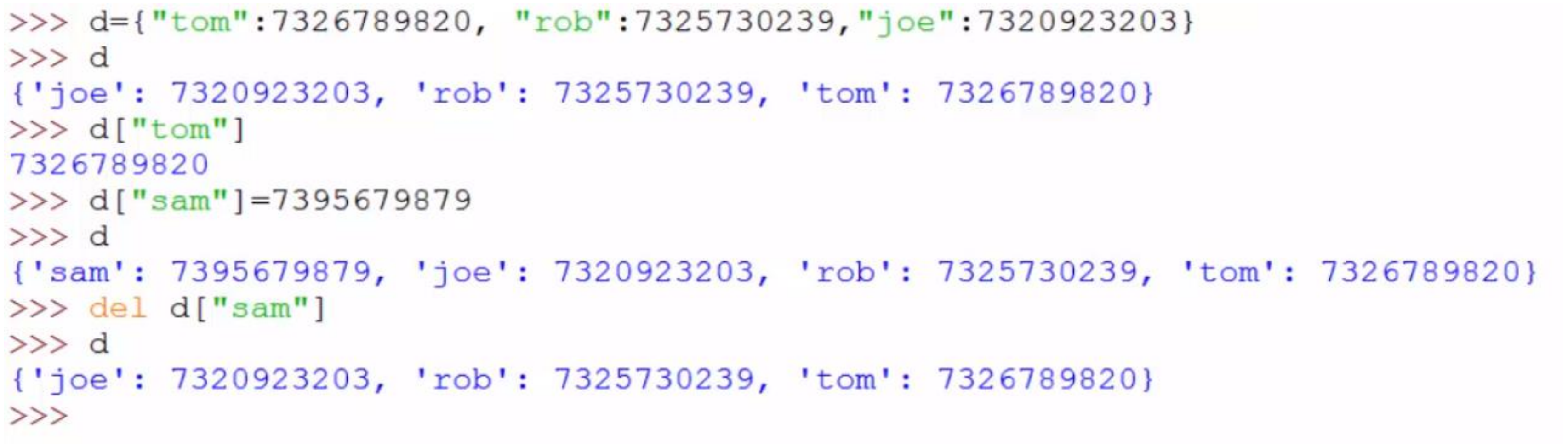


```
Python 3.5.0 (v3.5.0:374f501f4567, Sep 13
Type "copyright", "credits" or "license()"
>>> |
```



Tom	7326789820
Rob	7325730239
Joe	7320923203

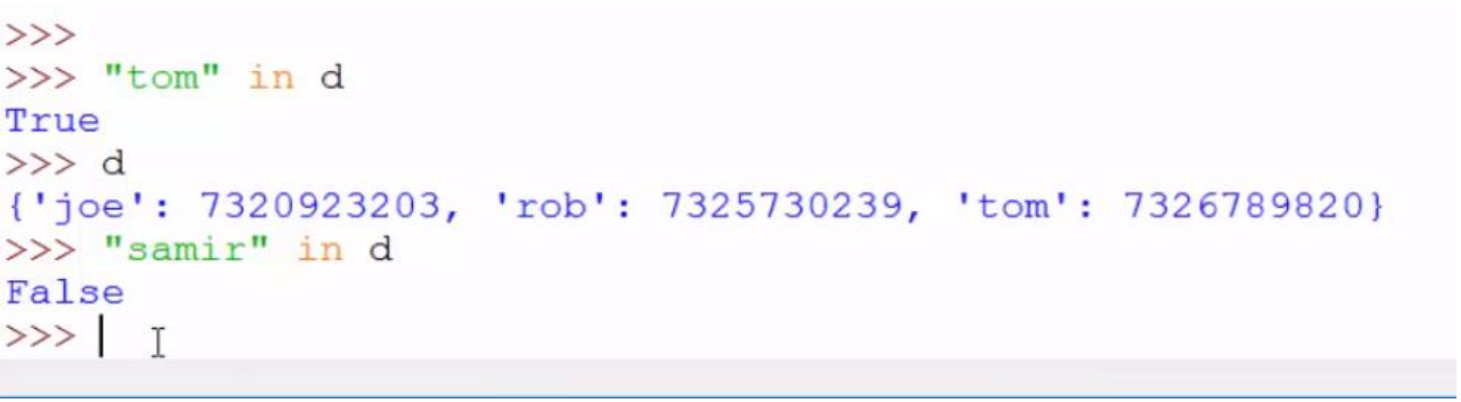
add into dist



```
>>> d={"tom":7326789820, "rob":7325730239,"joe":7320923203}
>>> d
{'joe': 7320923203, 'rob': 7325730239, 'tom': 7326789820}
>>> d["tom"]
7326789820
>>> d["sam"]=7395679879
>>> d
{'sam': 7395679879, 'joe': 7320923203, 'rob': 7325730239, 'tom': 7326789820}
>>> del d["sam"]
>>> d
{'joe': 7320923203, 'rob': 7325730239, 'tom': 7326789820}
>>>
```

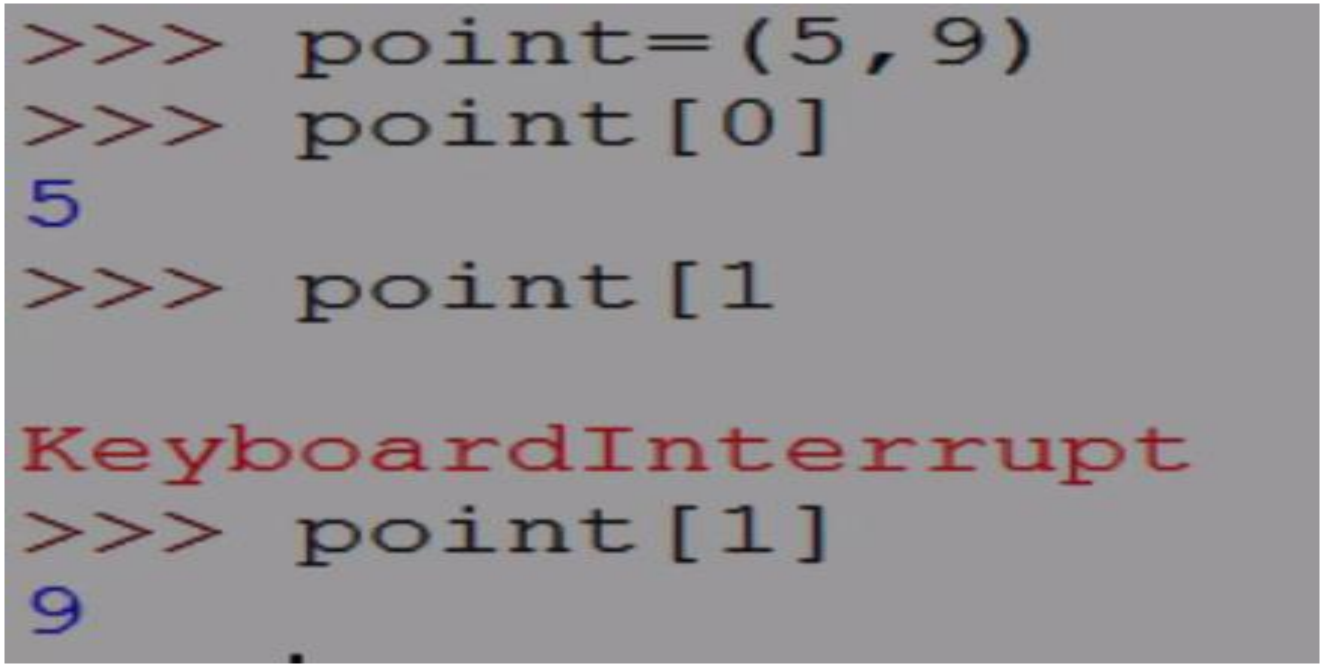
in dist

➢



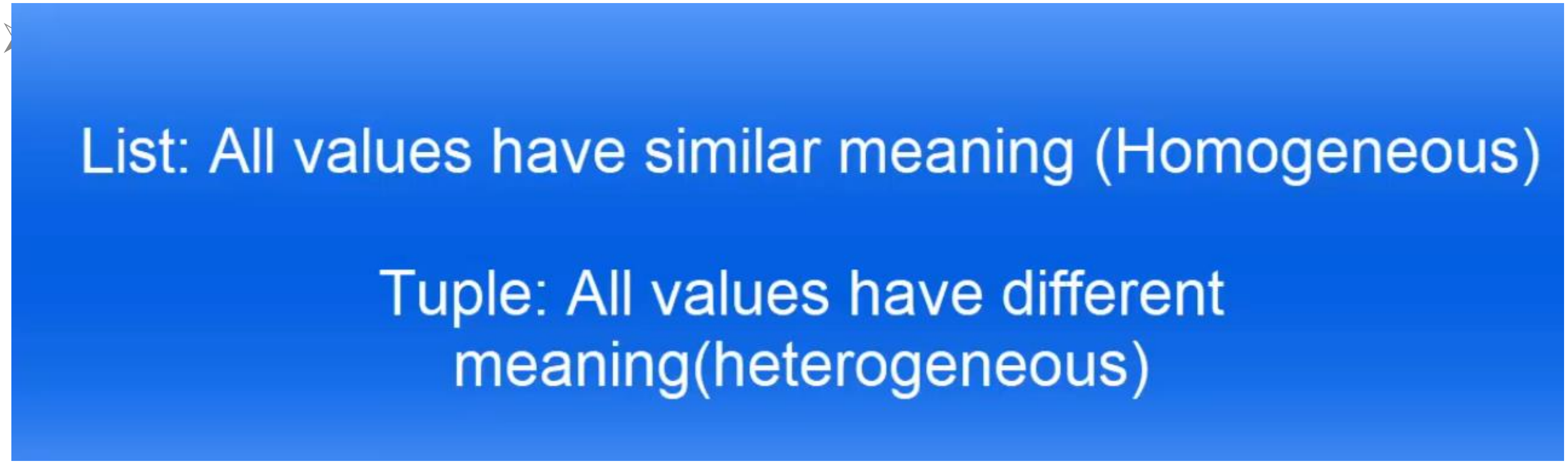
```
>>>
>>> "tom" in d
True
>>> d
{'joe': 7320923203, 'rob': 7325730239, 'tom': 7326789820}
>>> "samir" in d
False
>>>
```

tuple

- Tuple is a list of value s grouped together
- 

```
>>> point=(5,9)
>>> point[0]
5
>>> point[1

KeyboardInterrupt
>>> point[1]
9
```

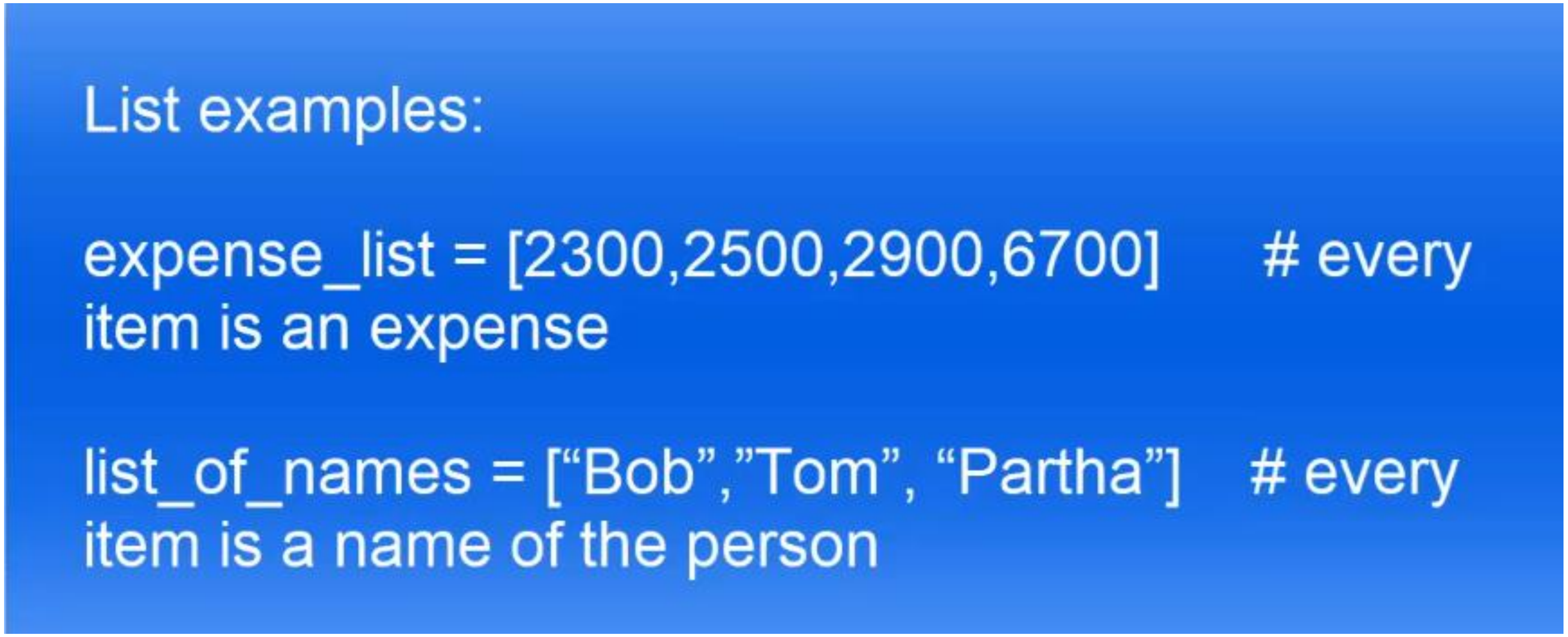
tuple



List: All values have similar meaning (Homogeneous)

Tuple: All values have different meaning(heterogeneous)

tuple



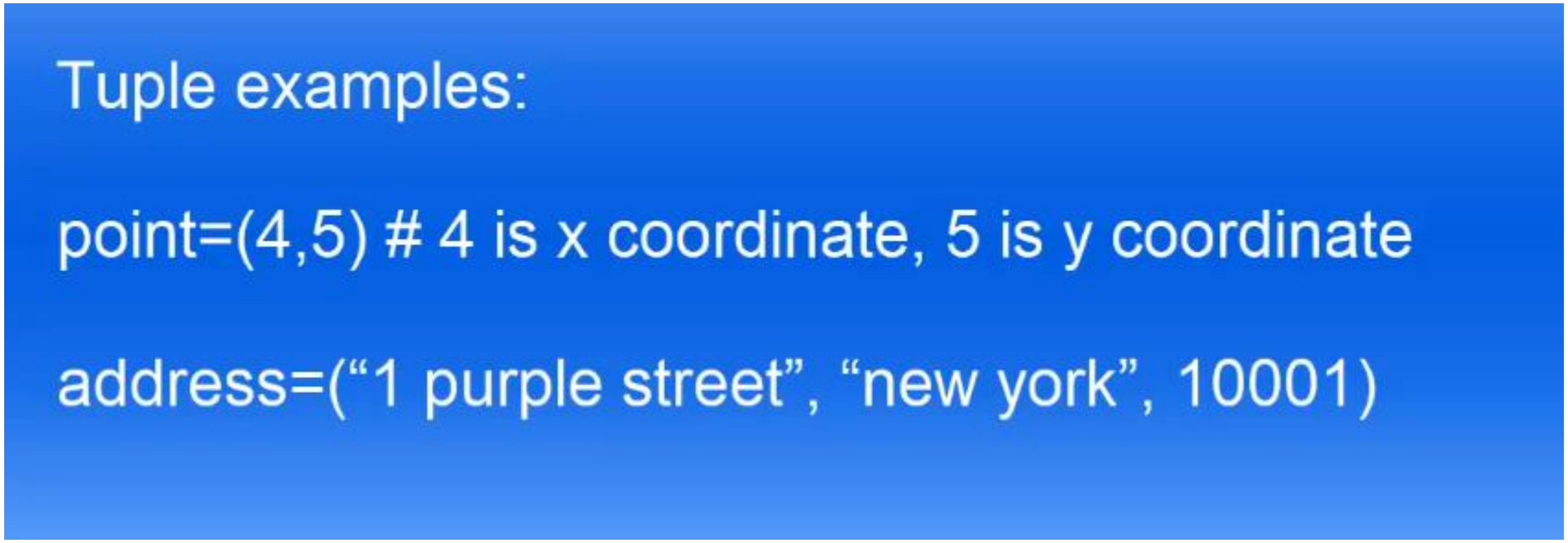
➢ List examples:

expense_list = [2300,2500,2900,6700]   # every item is an expense

list_of_names = [“Bob”,”Tom”, “Partha”]   # every item is a name of the person

tuple

➢

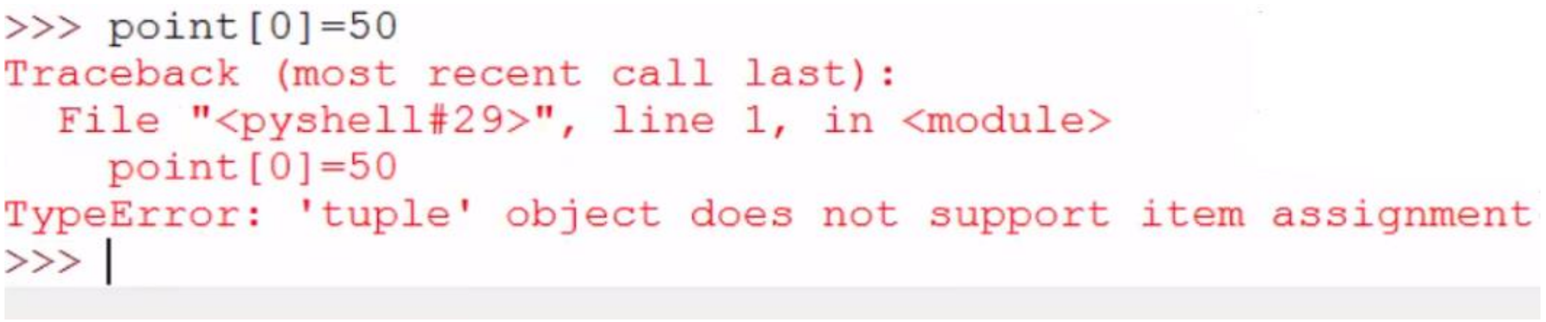


Tuple examples:

```
point=(4,5) # 4 is x coordinate, 5 is y coordinate

address=("1 purple street", "new york", 10001)
```

tuple

➢



```
>>> point[0]=50
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in <module>
    point[0]=50
TypeError: 'tuple' object does not support item assignment
>>> |
```