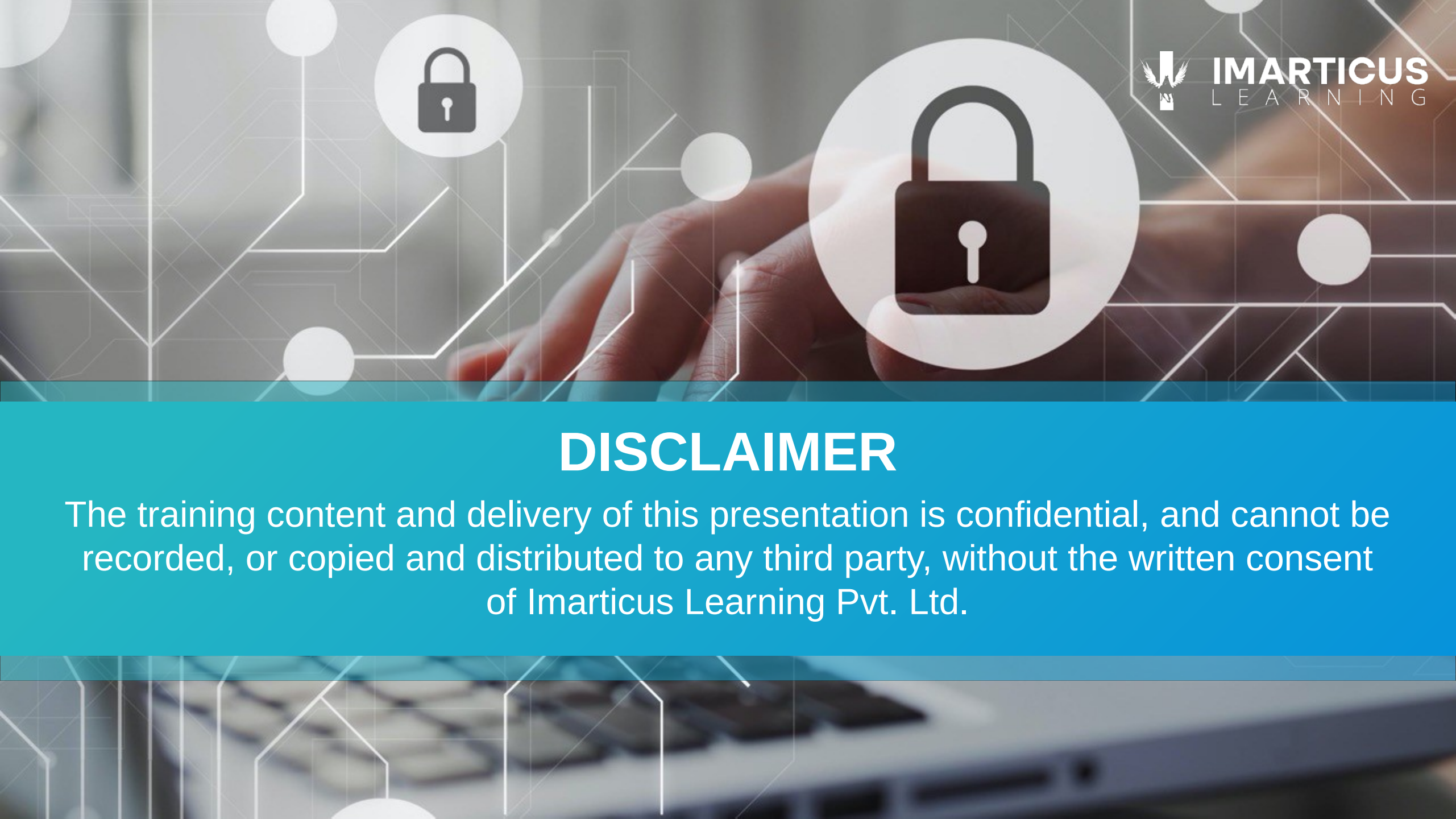
IMARTICUS LEARNING

Python Programming

Introduction to Python



IMARTICUS LEARNING

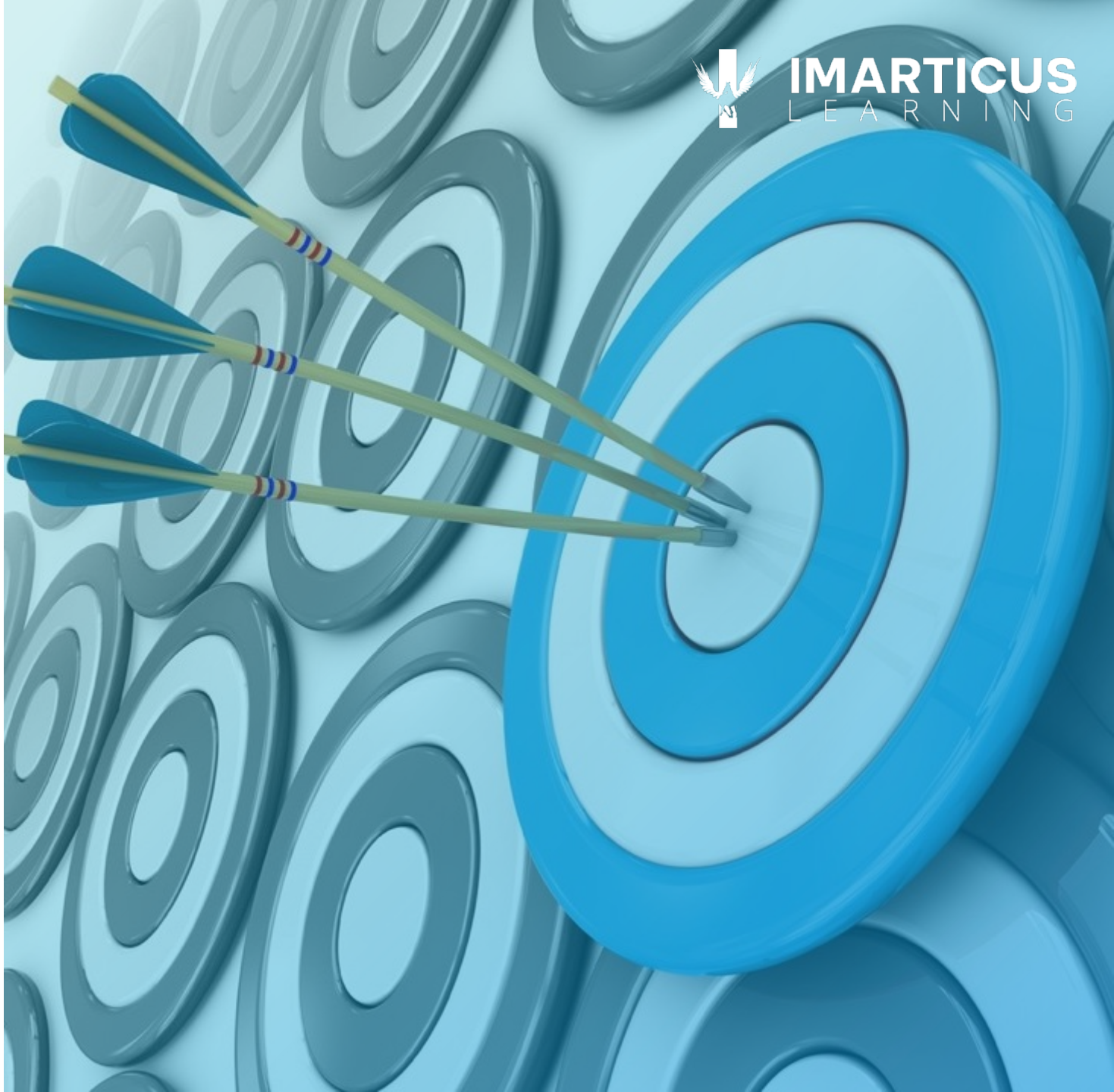
DISCLAIMER

The training content and delivery of this presentation is confidential, and cannot be recorded, or copied and distributed to any third party, without the written consent of Imarticus Learning Pvt. Ltd.

LEARNING OBJECTIVES

At the end of this session, you will learn:

- About History of Python
- Applications of Python Programming
- To create variables in Python
- To write Functions
- To use Python Operators
- Implement Python Flow Controls
- Implement Conditional Statements
- Implement Loops in Python



IMARTICUS LEARNING

Introduction to Python

HISTORY OF PYTHON

IMARTICUS LEARNING

1. Invented in December 1989
2. First public release in 1991
3. Open source from beginning
4. Managed by Python Software Foundation

Guido Van Rossum

PYTHON

IMARTICUS LEARNING

Python is powerful... and fast;
plays well with others;
runs everywhere;
is friendly & easy to learn;
is Open.

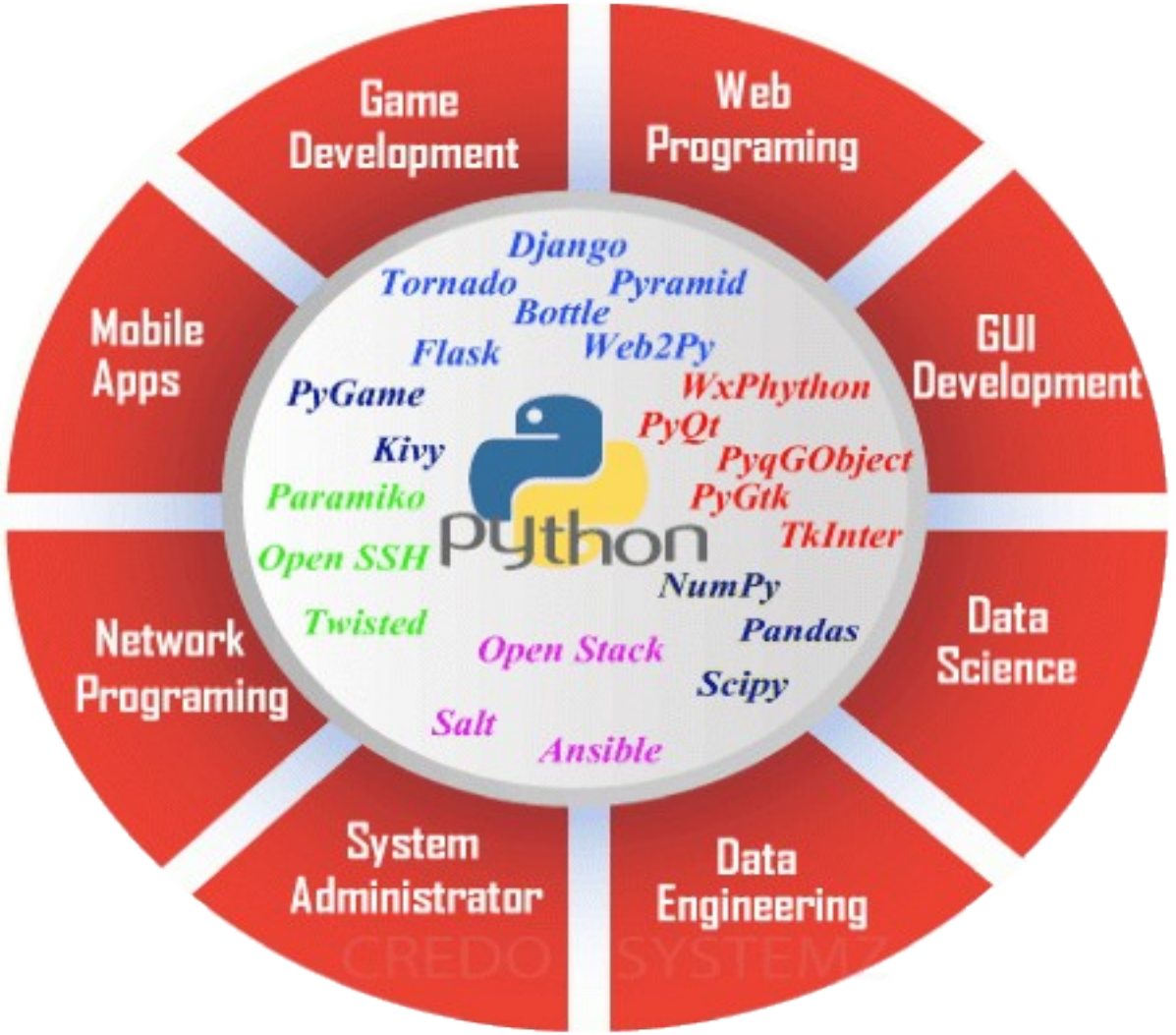
Python is an interpreted language, do not need to be compiled to run.

Python is a high-level language, which means a programmer can focus on what to do instead of how to do it.

Writing programs in Python takes less time than in another language.

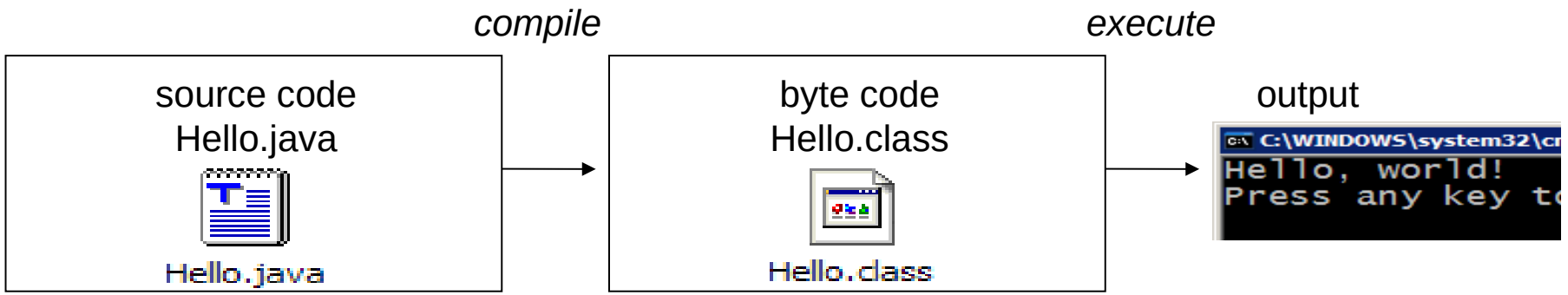
Python drew inspiration from other programming languages:

Python is used for…

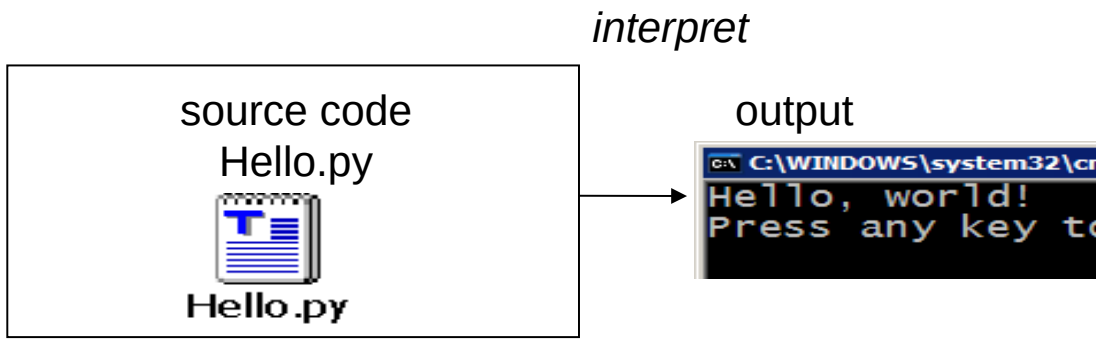


COMPILING AND INTERPRETING

Many languages are required to compile the program into a form (ByteCode) that the machine understands.



Python is instead directly interpreted into machine instructions.

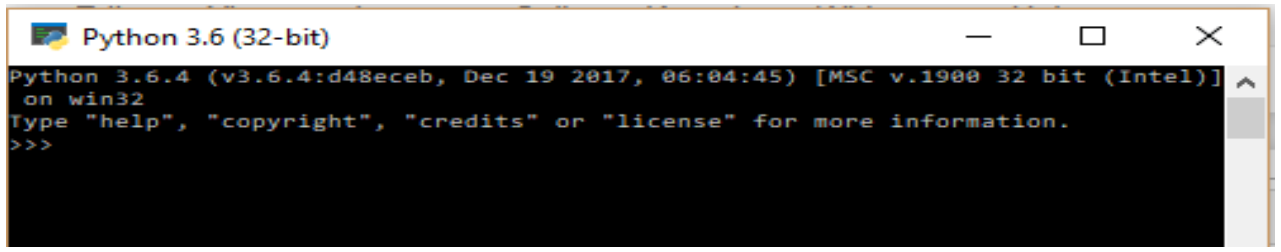


DEVELOPMENT ENVIRONMENT

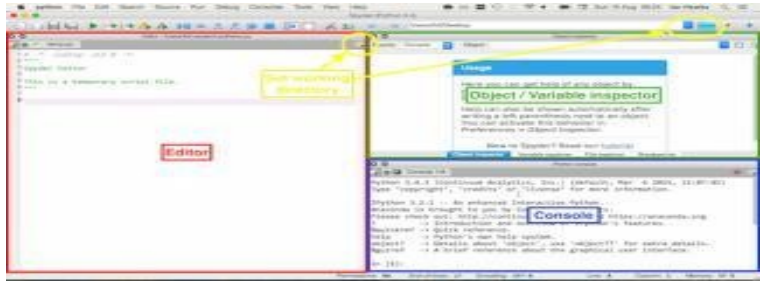
IMARTICUS LEARNING

Three most common options:

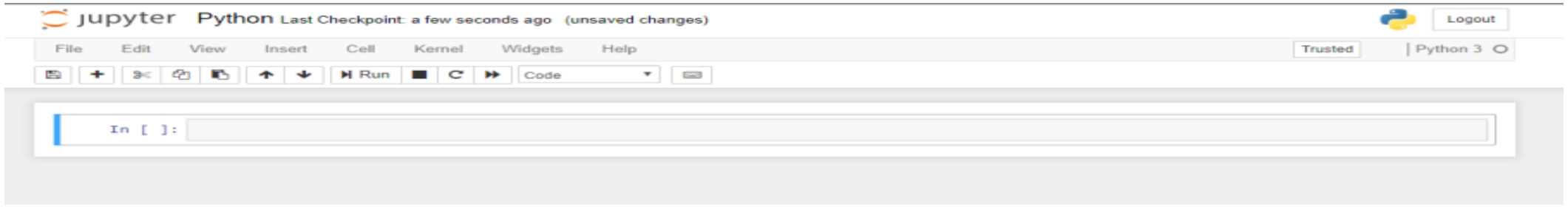
Terminal / Shell based



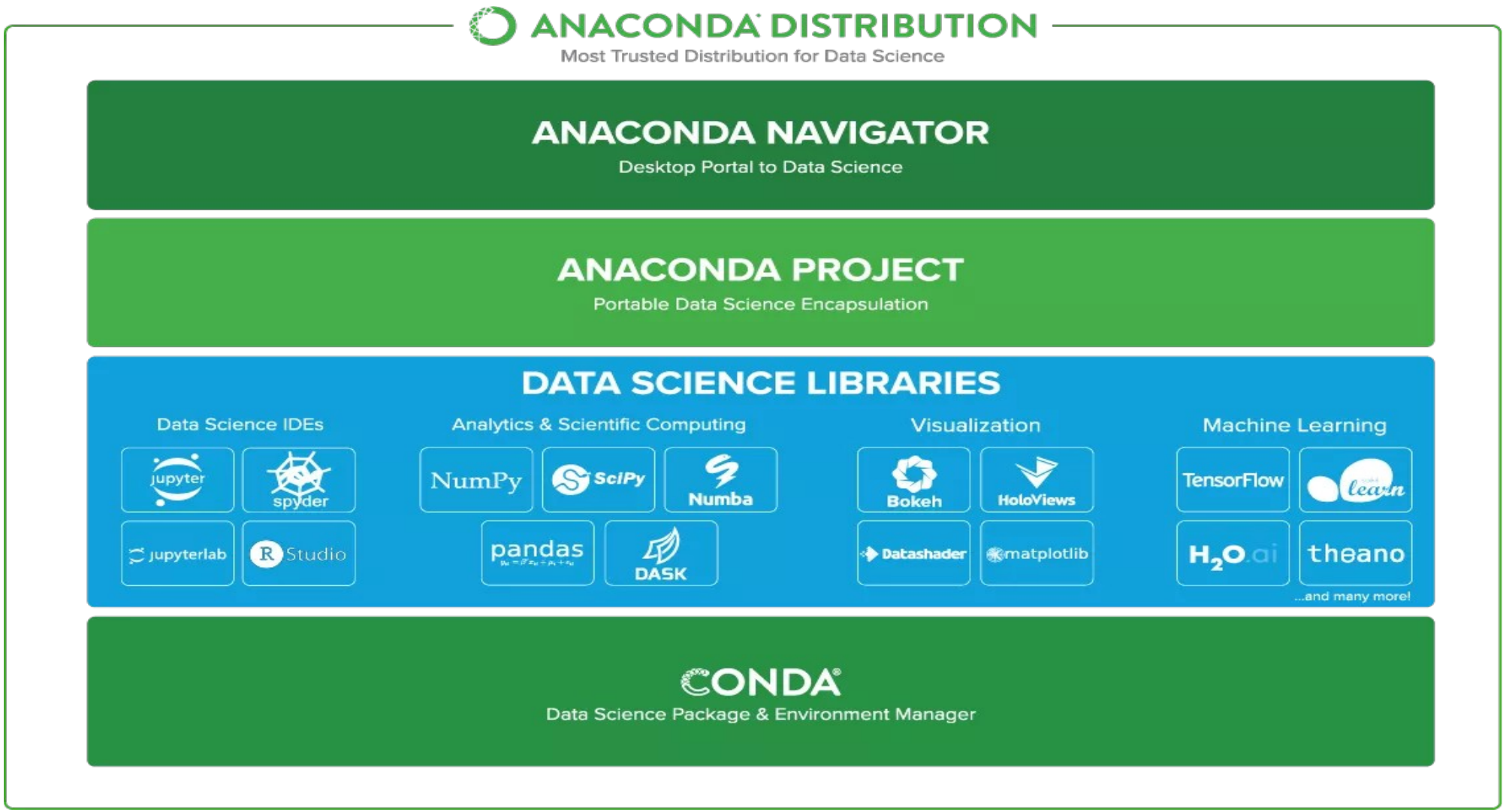
IDLE (Spyder IDE)



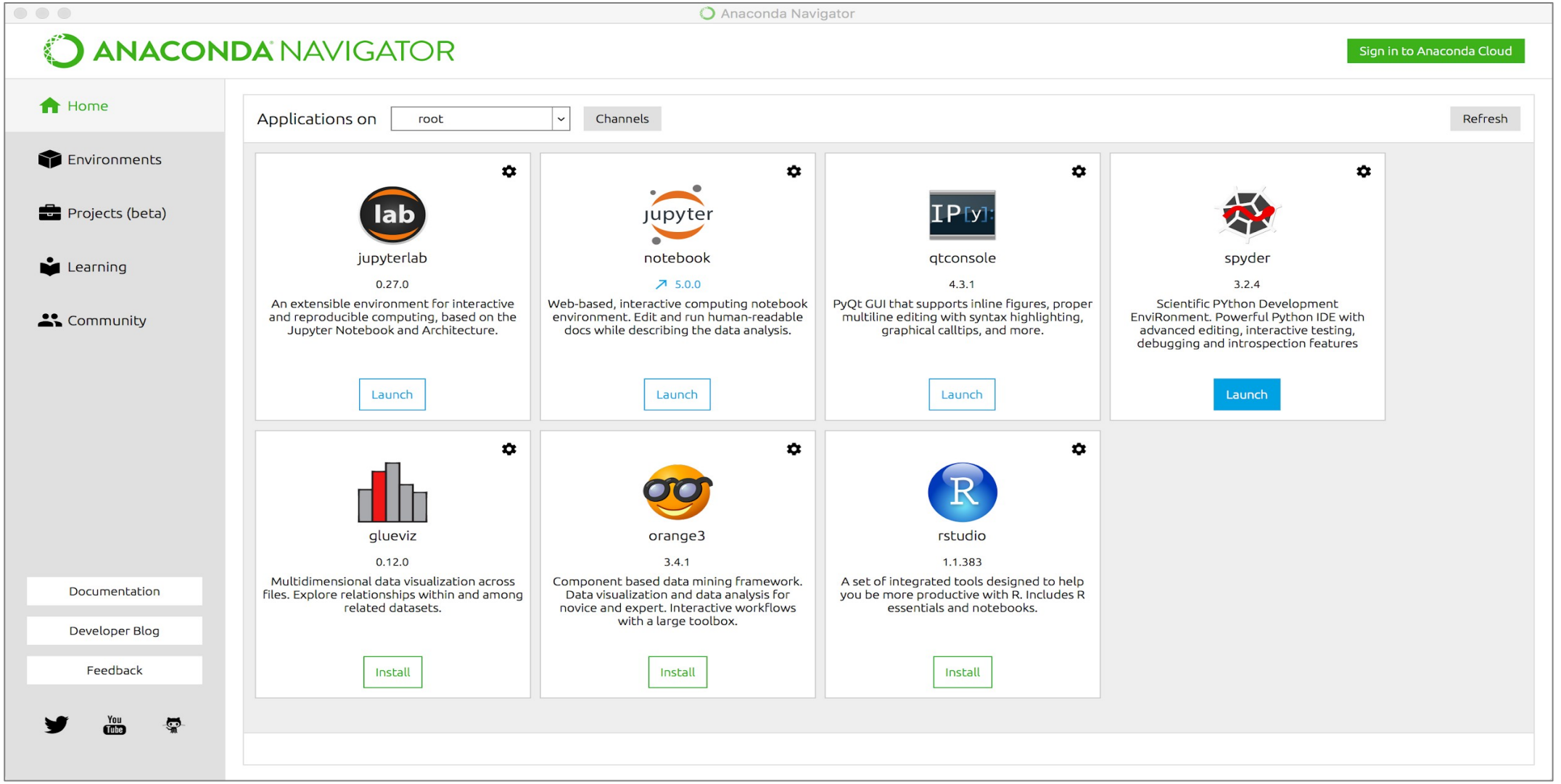
iPython notebook



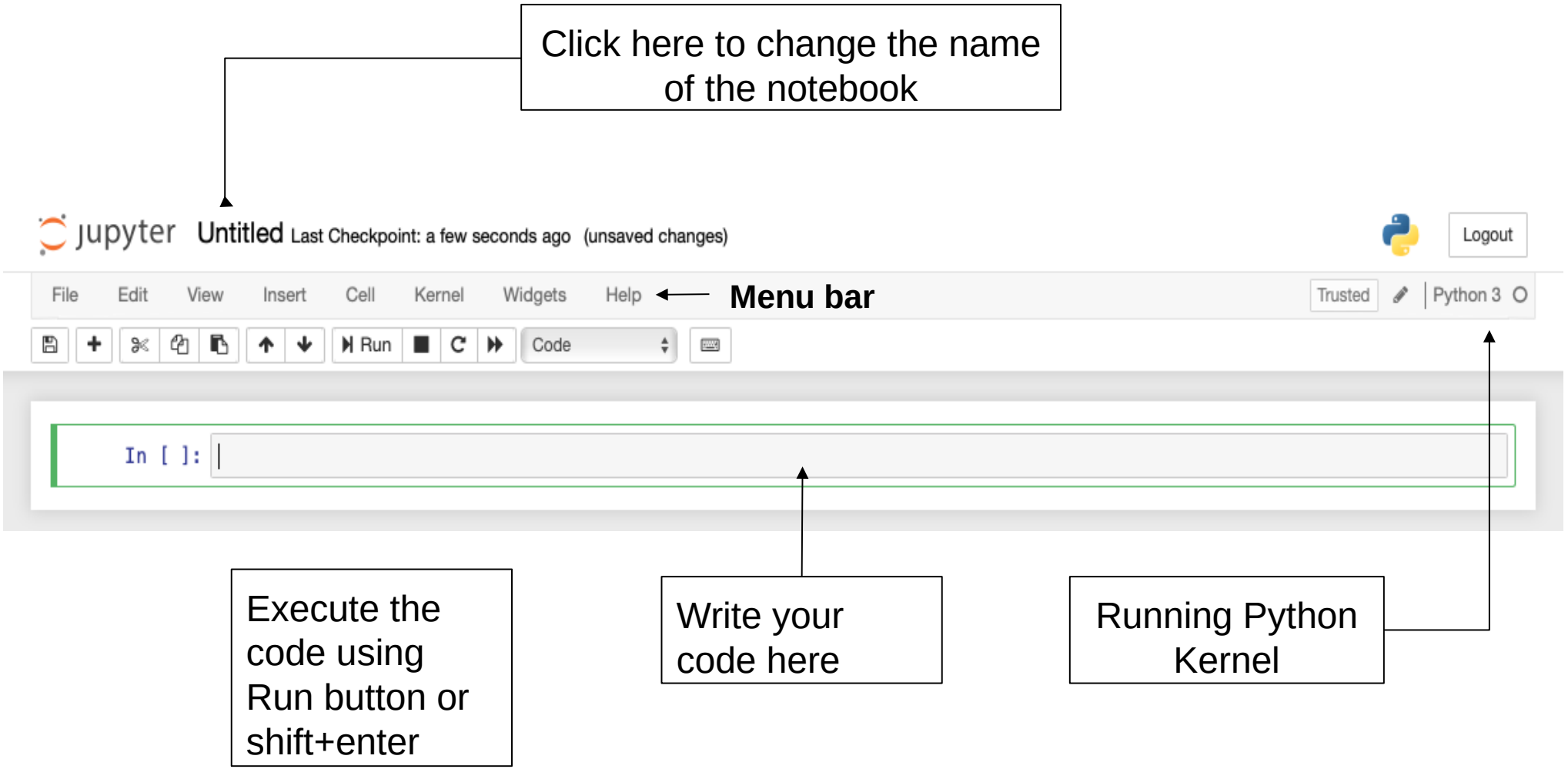
ANACONDA


ANACONDA DISTRIBUTION
Most Trusted Distribution for Data Science
ANACONDA NAVIGATOR
Desktop Portal to Data Science
ANACONDA PROJECT
Portable Data Science Encapsulation
DATA SCIENCE LIBRARIES
Data Science IDEs
Analytics & Scientific Computing
Visualization
Machine Learning
jupyter
spyder
jupyterlab
RStudio
NumPy
SciPy
Numba
pandas
DASK
Bokeh
HoloViews
Datashader
matplotlib
TensorFlow
learn
H2O.ai
theano
...and many more!
CONDA
Data Science Package & Environment Manager

ANACONDA NAVIGATOR



JUPYTER IDE


Click here to change the name of the notebook
jupyter
Untitled
Last Checkpoint: a few seconds ago (unsaved changes)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Menu bar
Trusted
Python 3
Run
Code
In [ ]:
Execute the code using Run button or shift+enter
Write your code here
Running Python Kernel

JUPYTER IDE

jupyter Vectorization in Python Last Checkpoint: 11/27/2017 (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3

Code

In [15]:
```
# Lets create an array
a = np.array([1,2,3,4,5,6,7,8,9,10])
print(a)
```

```
[ 1  2  3  4  5  6  7  8  9 10]
```

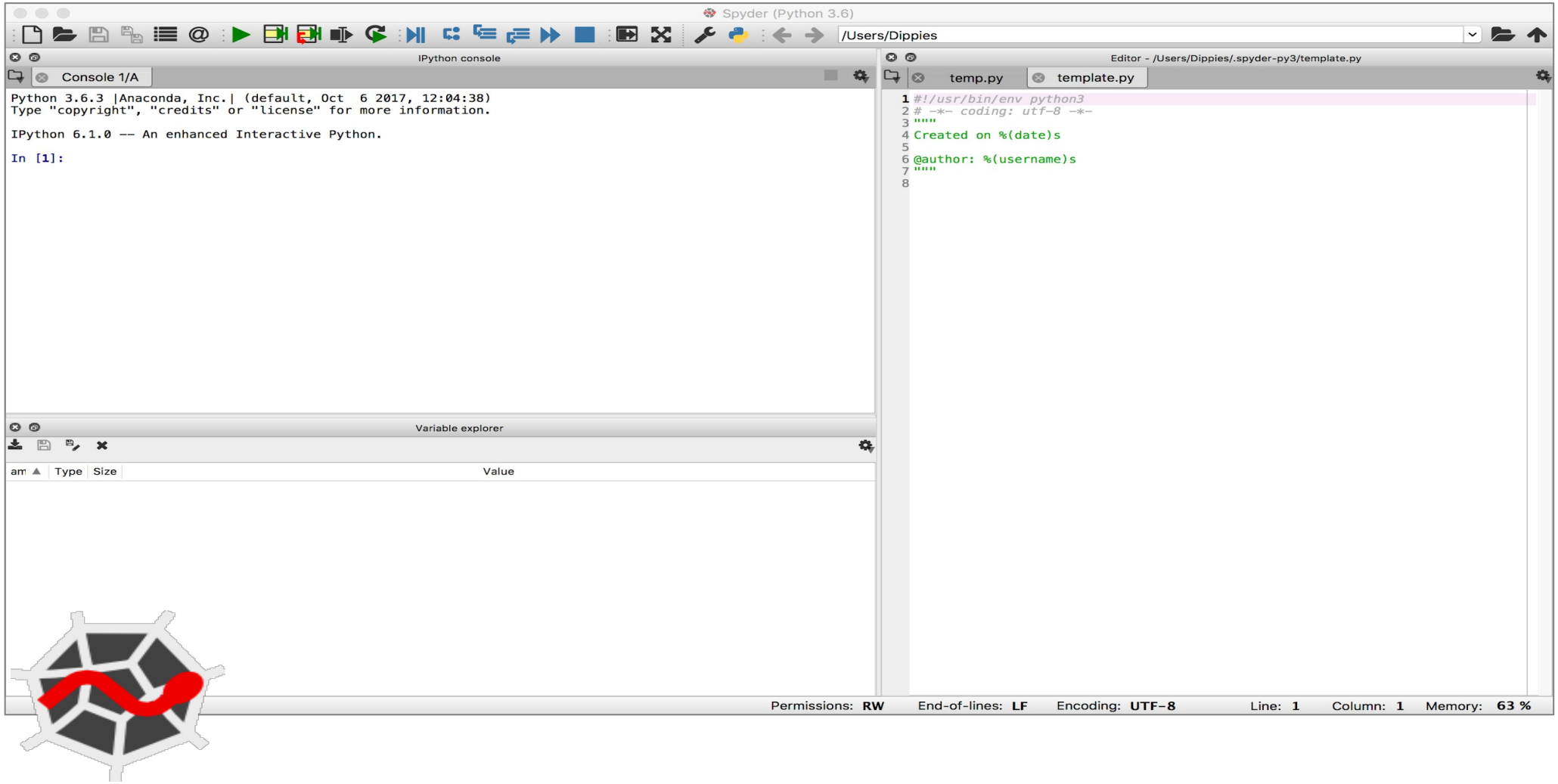
In [32]:
```
# Lets create random numbers
w = np.random.rand(1000000)
print(w)

x = np.random.rand(1000000)
print(x)

b = np.random.rand(1000000)
print(b)
```

```
[ 0.42500649  0.36084519  0.22833186 ...,  0.15959156  0.13275826
  0.9195299 ]
[ 0.23093721  0.17204381  0.93574301 ...,  0.681174    0.7920806
  0.43319432]
[ 0.17392007  0.52772832  0.35041441 ...,  0.72361219  0.73143161
  0.79200728]
```

In [41]:
```
# vectorized version
import time

starttime = time.time()
z = np.dot(w,x)
endtime = time.time()
print(z)

print("Vectorized Time taken: " + str(1000*(endtime - starttime)) + "milliseconds")
```

```
250144.722279
Vectorized Time taken: 2.3641586303710938milliseconds
```

SPYDER IDE


Spyder (Python 3.6)
/Users/Dippies
IPython console
Console 1/A
Python 3.6.3 |Anaconda, Inc.| (default, Oct 6 2017, 12:04:38)
Type "copyright", "credits" or "license" for more information.
IPython 6.1.0 -- An enhanced Interactive Python.
In [1]:
Editor - /Users/Dippies/.spyder-py3/template.py
temp.py
template.py
#!/usr/bin/env python3
-*- coding: utf-8 -*-
"""
Created on %(date)s
@author: %(username)s
"""
Variable explorer
Type
Size
Value
Permissions: RW
End-of-lines: LF
Encoding: UTF-8
Line: 1
Column: 1
Memory: 63 %

A PYTHON CODE SAMPLE

IMARTICUS LEARNING

```
In [1]: x = 34 - 23 # A comment.
        y = "Hello" # Another one.
        z = 3.45

        if z == 3.45 or y == "Hello":
         x = x + 1
         y = y +  " World" # String concat.
        print (x)
        print (y)

        12
        Hello World
```

- Assignment uses **=** and comparison uses **==**
- For numbers **+-*/%** are as expected
- Logical operators are words (**and, or, not**)
- The basic printing command is **print**
- Start comments with **#:** the rest of line is ignored

IMARTICUS LEARNING

Variables

VARIABLES

IMARTICUS LEARNING

Variables are reserved memory that store data

Unlike languages like C, C++ and Java, Python has no command for declaring variable

= operator is used to assign a value to a variable

```
name = "Gary"
age = 23
salary = 25750.5
marital_status = "Married"

print(name)
print(age)
print(salary)
print(marital_status)
```

```
Gary
23
25750.5
Married
```

REASSIGNING VARIABLES

```
name = "Garry"
name = "Roger"
print(name)
```

```
Roger
```

We would only receive the second assigned value as the output since that was the most recent assignment

REASSIGNING VARIABLES

```
friend_name = name
print(friend_name)
```

```
Gary
```

You can connect a different value with a previously assigned variable very easily through simple reassignment

MULTIPLE VARIABLE ASSIGNMENT

IMARTICUS LEARNING

```
gary_age = roger_age = tom_age = 24
print("Gary's age =", gary_age)
print("Roger's age =", roger_age)
print("Tom's age =", tom_age)
```

```
Gary's age = 24
Roger's age = 24
Tom's age = 24
```

Same value

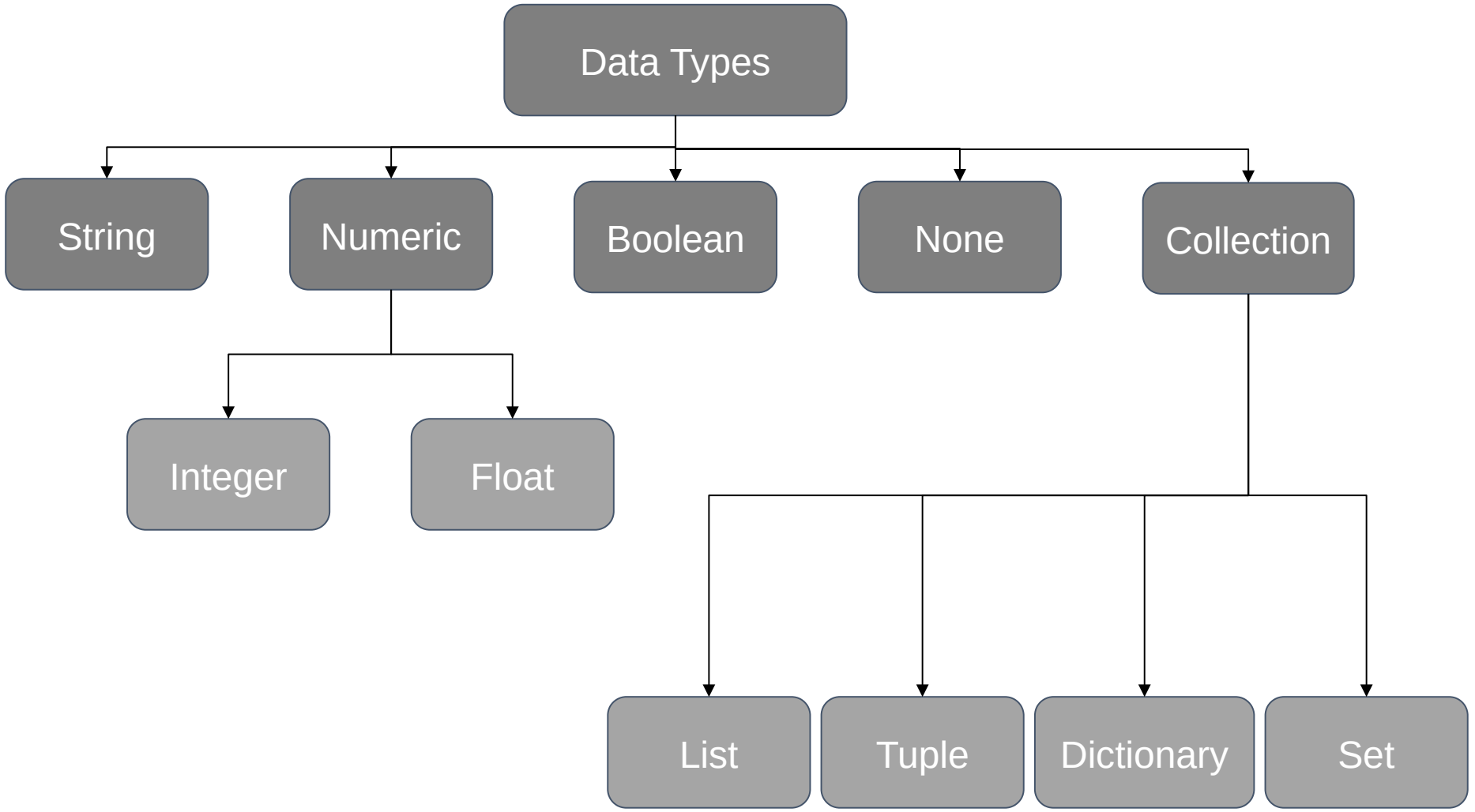
MULTIPLE VARIABLE ASSIGNMENT

```
roger_age, roger_income, roger_experience = 24, 10000, 7
print("Roger's age =", roger_age)
print("Roger's income =", roger_income)
print("Roger's experience =", roger_experience)
```

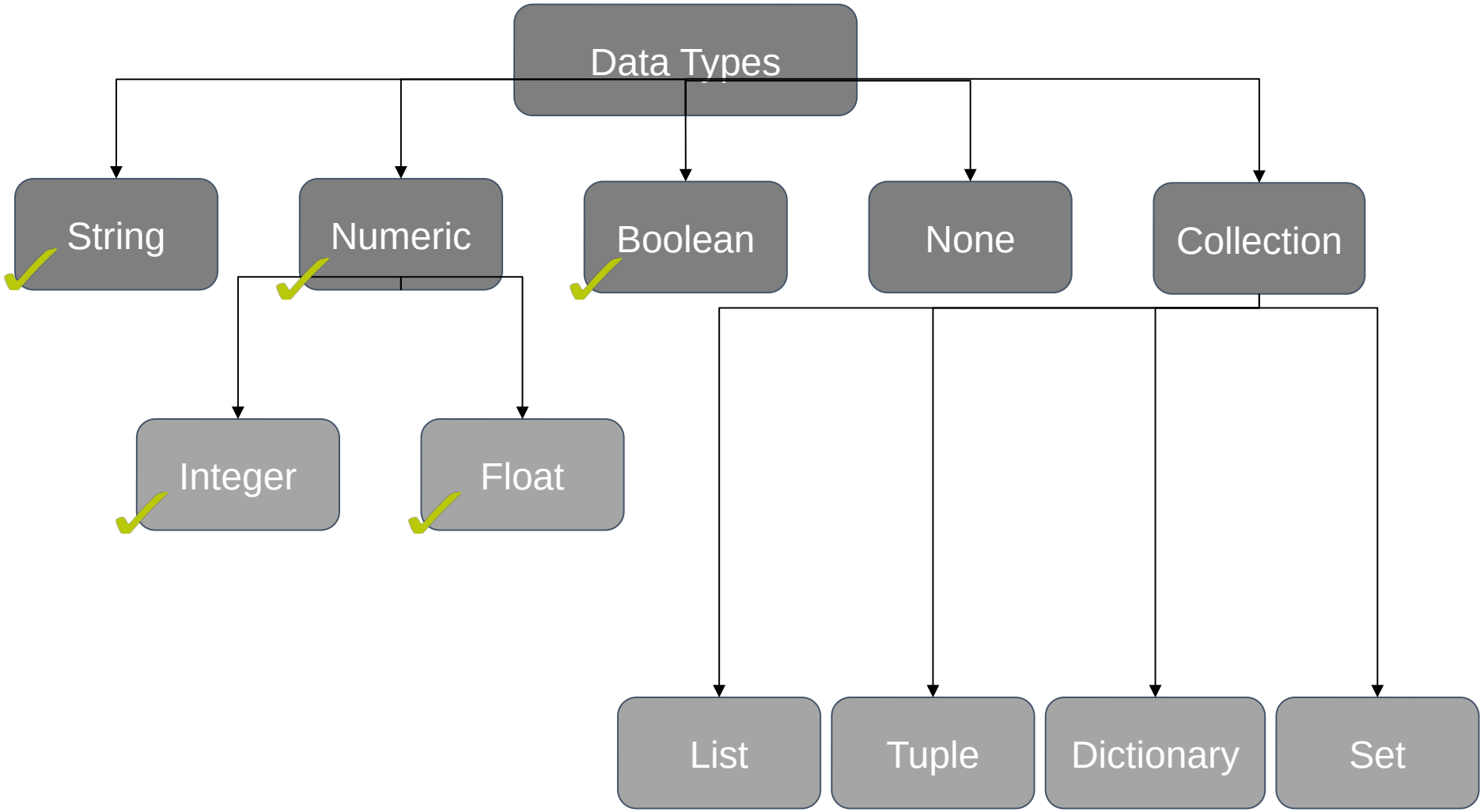
```
Roger's age = 24
Roger's income = 10000
Roger's experience = 7
```

With different value

DATA TYPES IN PYTHON



WE WILL LEARN THE FOLLOWING DATA TYPES TODAY



CREATING STRING VARIABLES

String variables are variables that hold zero or more characters such as letters, numbers, spaces, commas and many more.

Use type(variable_name) to check the data type of variable declared

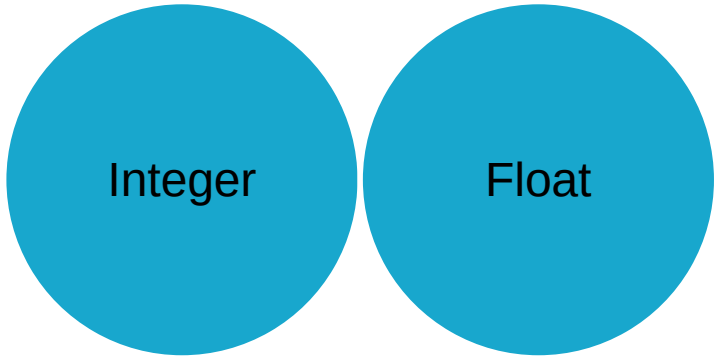
```
name = "Charles"
print(name)
type(name)
```

```
Charles

str
```

CREATING NUMERIC VARIABLES

A numeric variable is one that may take on any value within a finite or infinite interval

As seen earlier, there are two types of numeric data types:




```
sales_price_per_unit = 4.5
num_of_units_purchased = 5

print(sales_price_per_unit)
print(num_of_units_purchased)
```

```
4.5
5
```

```
type(num_of_units_purchased)
```

```
int
```

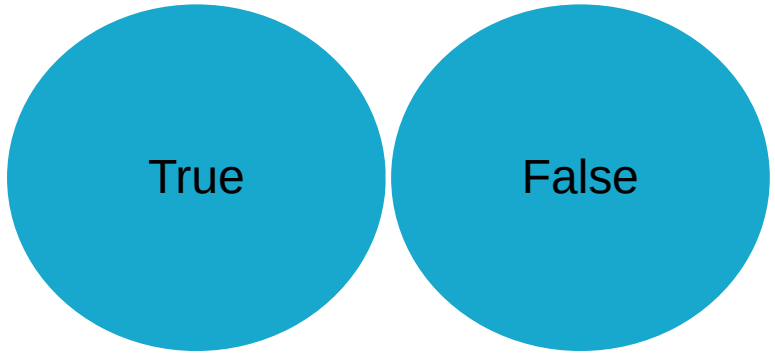
```
type(sales_price_per_unit)
```

```
float
```

CREATING BOOLEAN VARIABLES

IMARTICUS LEARNING

Boolean variables are variables that can have only two possible values:




```
isStudent = True
has_paid_fees = False
print(isStudent)
print(has_paid_fees)
```

```
True
False
```

```
type(isStudent)
```

```
bool
```

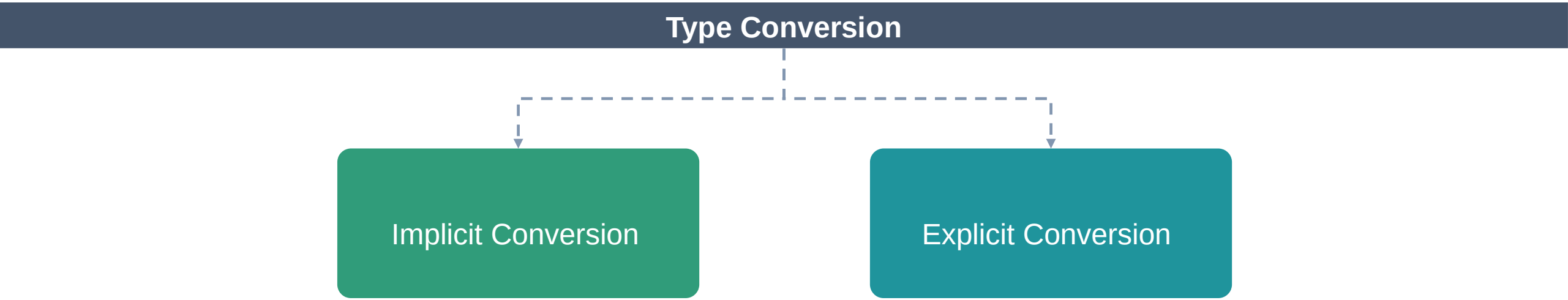
```
type(has_paid_fees)
```

```
bool
```

DATA TYPE CONVERSION

IMARTICUS LEARNING

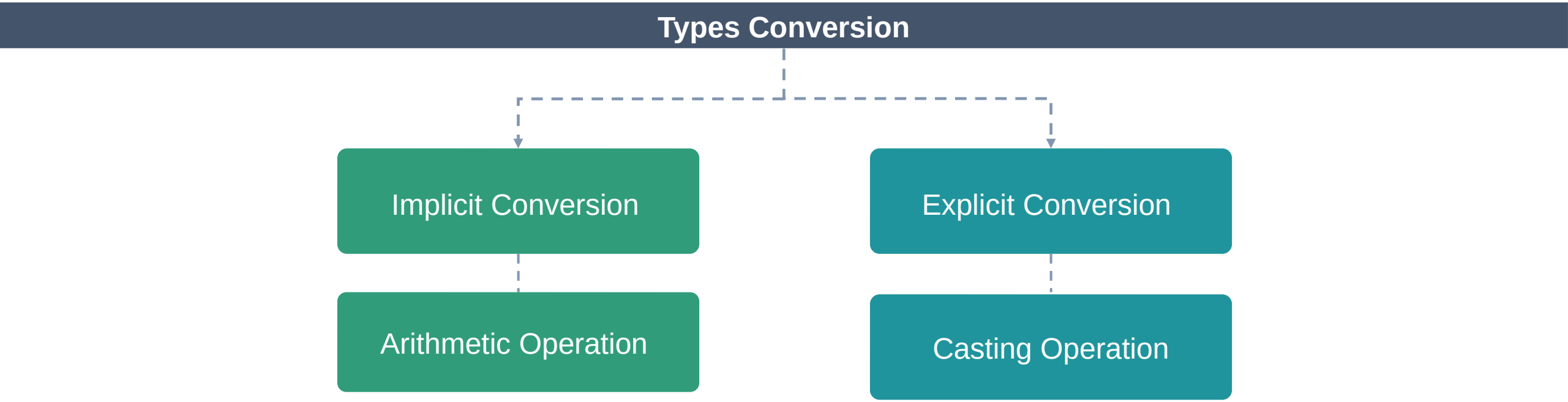
Python supports conversion of one data type to another



DATA TYPE CONVERSION

Implicit Conversion: Conversion done by Python interpreter without programmer's intervention

Explicit Conversion: Conversion that is user-defined that forces an expression to be of specific data type



IMPLICIT CONVERSION

IMARTICUS LEARNING

In Implicit Conversion, the conversion is done by Python interpreter without programmer's intervention

```
# float data type
sales_price_per_unit = 4.5

# integer data type
num_of_units_purchased = 5

# initialize resultant variable as integer data type
total_price = 0

# upon multiplication, the resultant variable converts to float
total_price  = sales_price_per_unit * num_of_units_purchased

print(total_price)

# check type
type(total_price)
```

```
22.5

float
```

EXPLICIT CONVERSION

IMARTICUS LEARNING

In Explicit Conversion, conversion that is user-defined forces an expression to be of specific data type

Example

```
# integer data type
price = 250

# string data type
tax = "2.5"

# we get error if we add both values
total_price = price + tax
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-21-94bdb13a5917> in <module>()
      6
      7 # we get error if we add both values
----> 8 total_price = price + tax

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

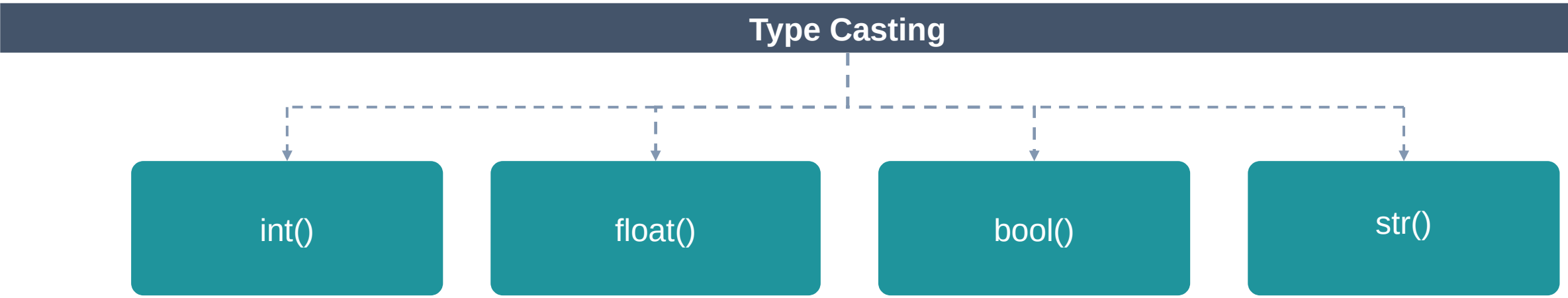
EXPLICIT CONVERSION

```
# integer data type
price = 250

# string data type
tax = "2.5"

# We convert the str data type to float and add both values to get the result
total_price = price + float(tax)
print(total_price)
```

```
252.5
```

Explicit data type conversion using float() function

TYPE CASTING



TYPE CASTING

IMARTICUS LEARNING

```
# declare a float variable
unit_price = 52.8

# type cast the declared float value into string
str(unit_price)
```

```
'52.8'
```

```
hasSpeed, isLivingThing = 1,0
bool(hasSpeed)
```

```
True
```

```
bool(isLivingThing)
```

```
False
```

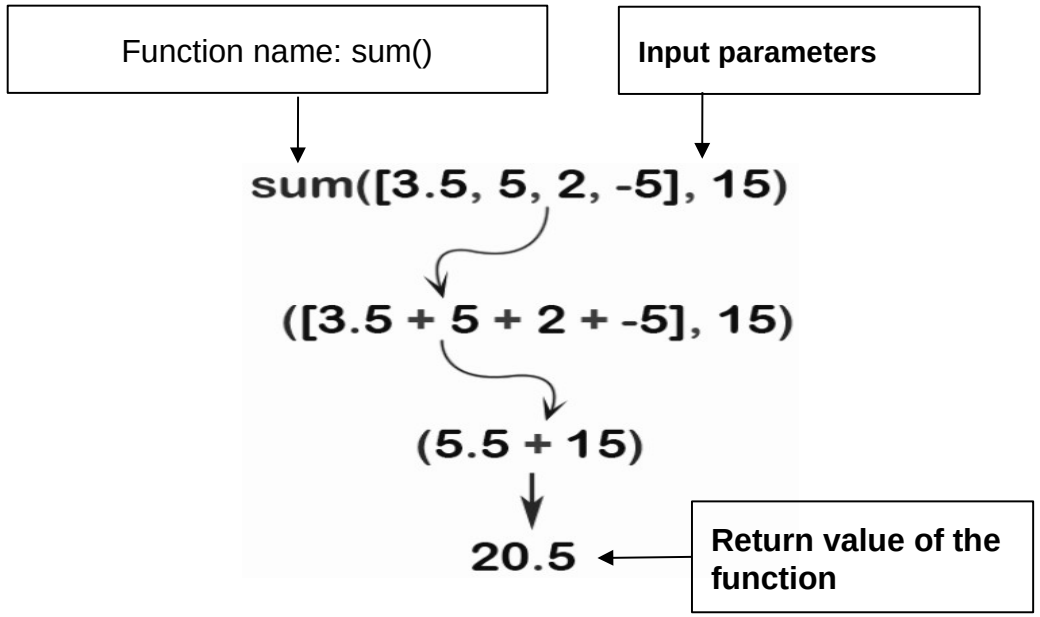
IMARTICUS LEARNING

Functions

WHAT ARE FUNCTIONS?

IMARTICUS LEARNING

A function is a block of reusable code that runs when called



- A function has a name. You call the function by its name
- A function can take input(s), known as input parameters
- A function can return data as a result

WHAT ARE FUNCTIONS?

IMARTICUS LEARNING

There are three types of functions:

- Built-in Functions
- User-defined Functions
- Lambda Functions

Some of the built-in functions are:

print() - to print the output

input() - to take input from user

the type casting functions like int(), float(), str(), bool(), etc.

We will study the user-defined functions and lambda functions in detail in our upcoming sessions

THE PRINT() IN PYTHON

The print() function prints the specified message to the screen

The message can be a string, or any other object

The object will be converted into a string before written to the screen

```
print("Hello World")
```

```
Hello World
```

```
print("Hello", "Mr. Thomas")
```

```
Hello Mr. Thomas
```

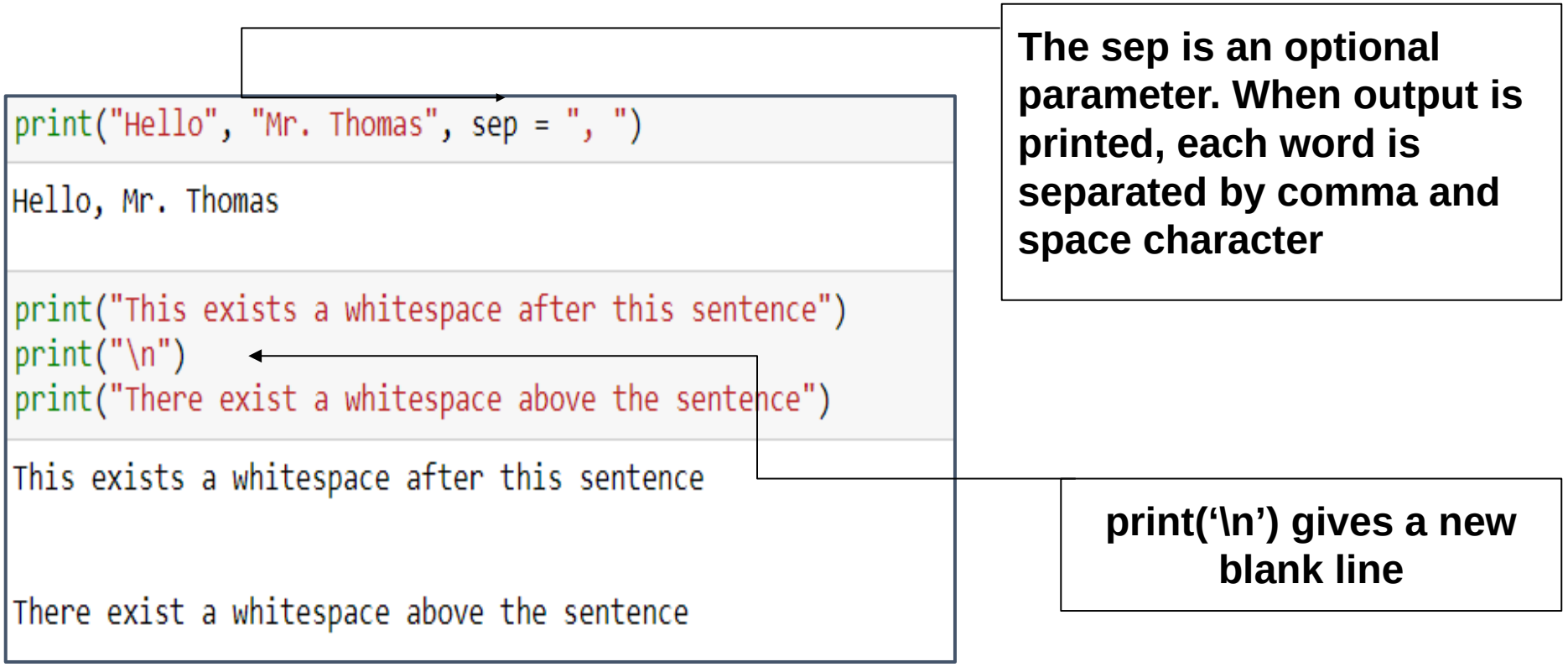
```
computer_accessories = ("CPU", "Mouse", "Keyboard")
print(computer_accessories)
```

```
('CPU', 'Mouse', 'Keyboard')
```

strings separated by a comma within a print() function get concatenated.

THE PRINT() IN PYTHON

IMARTICUS LEARNING



The backslash "\" is a special character that represents whitespaces.
For example '\t' is a tab and '\n' is a new line.

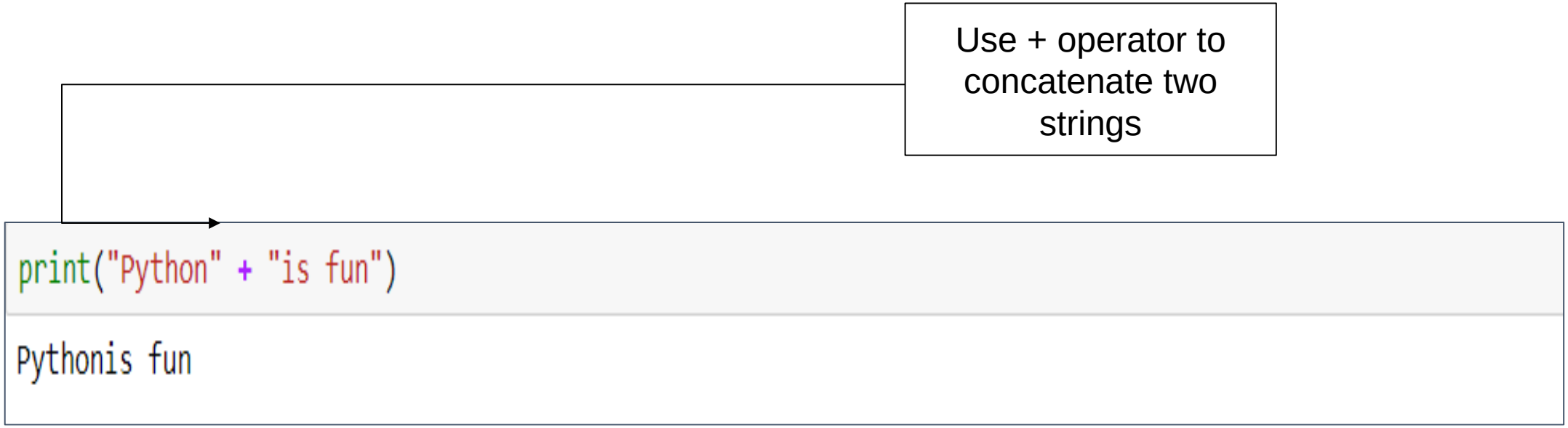
IMARTICUS LEARNING

Invalid use of opening and closing quotes is not allowed.

```
print("Python is fun')
```

```
  File "<ipython-input-44-eca1d9a41081>", line 1
    print("Python is fun')
                          ^
SyntaxError: EOL while scanning string literal
```

CONCATENATION USING PRINT()



IMARTICUS LEARNING

You cannot concatenate string & number.

```
print("Sam's age is " + 25)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-48-9218b2366aa0> in <module>()
----> 1 print("Sam's age is " + 25)

TypeError: must be str, not int
```

Adding numbers to strings does not make any sense. Please consider explicit conversion to convert the number to string first, in order to join them together.

CONCATENATION WITH TYPE CASTING

IMARTICUS LEARNING

```
print("Sam's age is " + str(25))
```

```
Sam's age is 25
```

Explicit conversion of a number to string with str() function.

We learn more such type conversions in our upcoming sessions

IMARTICUS LEARNING

Python Operators

PYTHON OPERATORS

IMARTICUS LEARNING

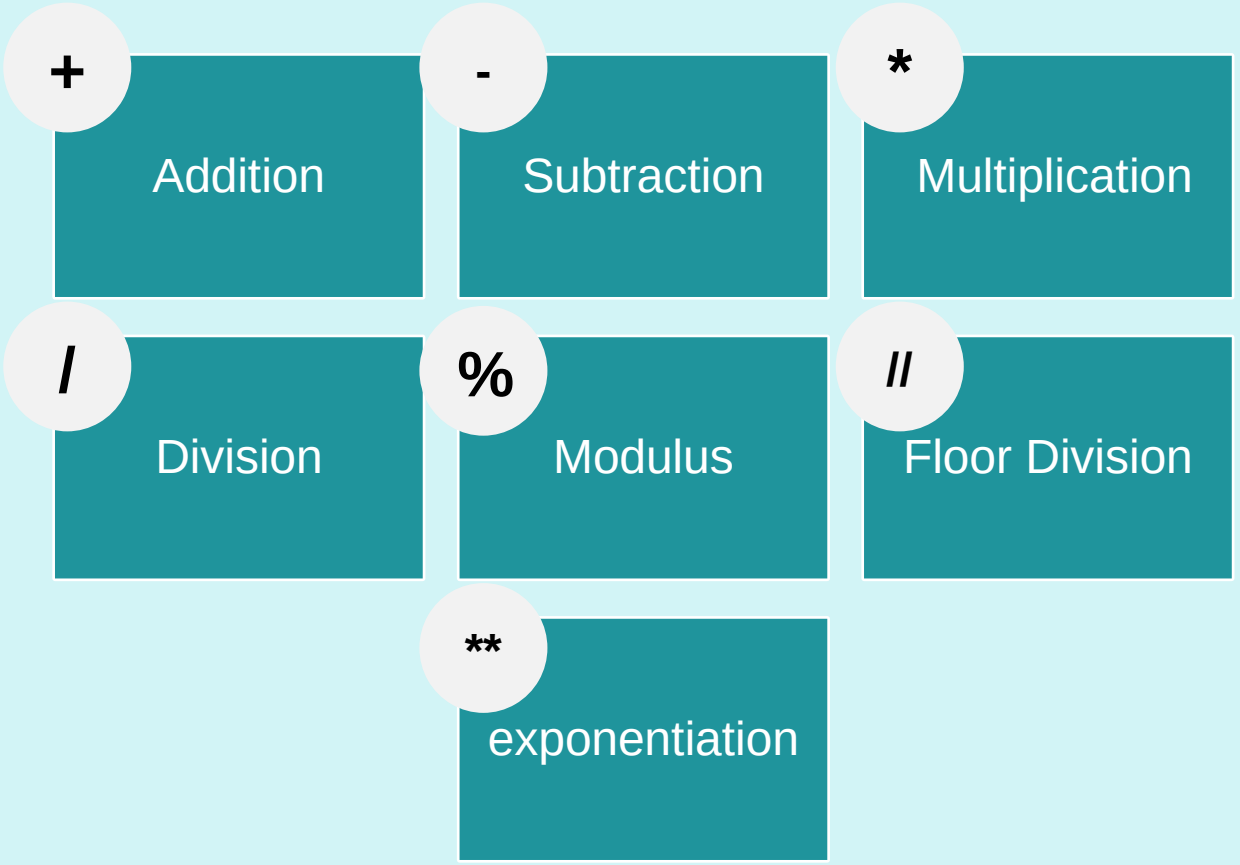
The Python Operators are:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Membership Operators
- Bitwise Operators
- Assignment Operators
- Identity Operators

ARITHMETIC OPERATORS

IMARTICUS LEARNING

The most common Arithmetic Operators are:



ARITHMETIC OPERATORS

IMARTICUS LEARNING

Addition

```
# adding two integers
income = 100000
incentives = 10000
total_income = income + incentives
print(total_income)
```

```
110000
```

```
# adding one float and one integer values
income = 100000
incentives = 12550.50
total_income = income + incentives
print(total_income)
```

```
112550.5
```

ARITHMETIC OPERATORS

IMARTICUS LEARNING

Addition

```
1 # Adding two strings
2 first_name = "Mike"
3 second_name = "Anderson"
4 full_name = first_name+ " " + second_name
5 print(full_name)
```

```
Mike Anderson
```

```
1 # Adding two strings
2 text = "Age is "
3 age = "22"
4 combine = text + age
5 print(combine)
```

```
Age is 22
```

Note that 22 is a string

Here two strings are getting concatenated

ARITHMETIC OPERATORS

IMARTICUS LEARNING

Subtraction

```
# subtracting two integers
income = 100000
tax = 2500
total_income = income - tax
print(total_income)
```

```
97500
```

```
# subtracting one float and one integer value
income = 100000
tax = 2550.75
total_income = income - tax
print(total_income)
```

```
97449.25
```

ARITHMETIC OPERATORS

IMARTICUS LEARNING

Subtraction

```
1 # Subtracting two strings
2 first_name = "Mike"
3 second_name = "Anderson"
4 full_name = first_name - second_name
5 print(full_name)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-997d527bdf02> in <module>()
      2 first_name = "Mike"
      3 second_name = "Anderson"
----> 4 full_name = first_name - second_name
      5 print(full_name)

TypeError: unsupported operand type(s) for -: 'str' and 'str'
```

When two strings are added, the + (addition) operator basically concatenates the two strings. Subtracting two strings does not make any sense.

ARITHMETIC OPERATORS

IMARTICUS LEARNING

Multiplication

```
# multiplication of two integer values
unit_price = 20
total_units_purchased = 150
total_amount = unit_price * total_units_purchased
print(total_amount)
```

```
3000
```

```
# multiplication of one integer and one float value
unit_price = 12.2
total_units_purchased = 150
total_amount = unit_price * total_units_purchased
print(total_amount)
```

```
1830.0
```

ARITHMETIC OPERATORS

IMARTICUS LEARNING

Multiplication

```
# multiplication of string with integer
value = "Singapore "
count = 3
result = value * count
print(result)
```

```
Singapore Singapore Singapore
```

ARITHMETIC OPERATORS

IMARTICUS LEARNING

Multiplication

```
# multiplication of string with a string
value = "Singapore "
count = "3"
result = value * count
print(result)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-65-a6364a59b5a7> in <module>()
      2 value = "Singapore "
      3 count = "3"
----> 4 result = value * count
      5 print(result)

TypeError: can't multiply sequence by non-int of type 'str'
```

ARITHMETIC OPERATORS

IMARTICUS LEARNING

Division

```
# Division using 2 integer variables
total_cost = 2000
quantity = 100
price_per_unit = total_cost / quantity
print(price_per_unit)
```

```
20.0
```

```
# Division using 1 integer & 1 float variable
total_cost = 1560.75
quantity = 100
price_per_unit = total_cost / quantity
print(price_per_unit)
```

```
15.6075
```

ARITHMETIC OPERATORS

IMARTICUS LEARNING

Division

```
# Cannot divide a string by a number
service = "airlines"
value = 3
result = service / value
print(result)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-81-7c4127a42622> in <module>
      2 service = "airlines"
      3 value = 3
----> 4 result = service / value
      5 print(result)

TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

ARITHMETIC OPERATORS

IMARTICUS LEARNING

Modulus

```
# Using modulus operator to find the remainder
total_cost = 2700
quantity = 23
price_per_unit = total_cost % quantity
print(price_per_unit)
```

```
9
```

```
# Using modulus operator to find the remainder
total_cost = 2700.50
quantity = 23
price_per_unit = total_cost % quantity
print(price_per_unit)
```

```
9.5
```

IMARTICUS LEARNING

Floor Division

```
# division operation to get quotient
# it is known as floor division
total_cost = 13000
quantity = 23
price_per_unit = total_cost // quantity
print(price_per_unit)
```

```
565
```

```
# division operation to get quotient
# it is known as floor division
total_cost = 13000.50
quantity = 23
price_per_unit = total_cost // quantity
print(price_per_unit)
```

```
565.0
```

ARITHMETIC OPERATORS

IMARTICUS LEARNING

Exponentiation

```
# Squaring values
square_side = 20
area_of_square = square_side**2
print(area_of_square)
```

```
400
```

```
# Cubing values
cube_side = 20
volume_of_cube = cube_side**3
print(volume_of_cube)
```

```
8000
```

RUNTIME VARIABLE ASSIGNMENT

IMARTICUS LEARNING

```
# runtime variable assignment
name = input("Enter your name: ")
print("Welcome", name)
```

```
Enter your name: Bill
Welcome Bill
```

```
type(name)
```

```
str
```

RUNTIME VARIABLE ASSIGNMENT

```
price = input("Enter price:")
quantity = input("Enter quantity")
total_cost = price * quantity
print(total_cost)
```

```
Enter price:34
Enter quantity34

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-130-c4a9ee6d0895> in <module>
      1 price = input("Enter price:")
      2 quantity = input("Enter quantity")
----> 3 total_cost = price * quantity
      4 print(total_cost)

TypeError: can't multiply sequence by non-int of type 'str'
```

RUNTIME VARIABLE ASSIGNMENT

IMARTICUS LEARNING

```
price = int(input("Enter price: "))
quantity = int(input("Enter quantity:"))
total_cost = price * quantity
print(total_cost)
```

```
Enter price: 34
Enter quantity:34
1156
```

RELATIONAL OPERATORS

Relational operators are used to compare values and take certain decisions based on the outcome

Following are some of the relational operators:

Operators	Meaning
<	Is less than
<=	Is less than & equal to
>	Is greater than
>=	Is greater than & equal to
==	Is equal to
!=	Is not equal to

RELATIONAL OPERATORS

```
# declare two variables
ketty_age = 28
kenny_age = 25
```

```
ketty_age == kenny_age
```

```
False
```

```
ketty_age <= kenny_age
```

```
False
```

```
ketty_age >= kenny_age
```

```
True
```

```
ketty_age < kenny_age
```

```
False
```

```
ketty_age > kenny_age
```

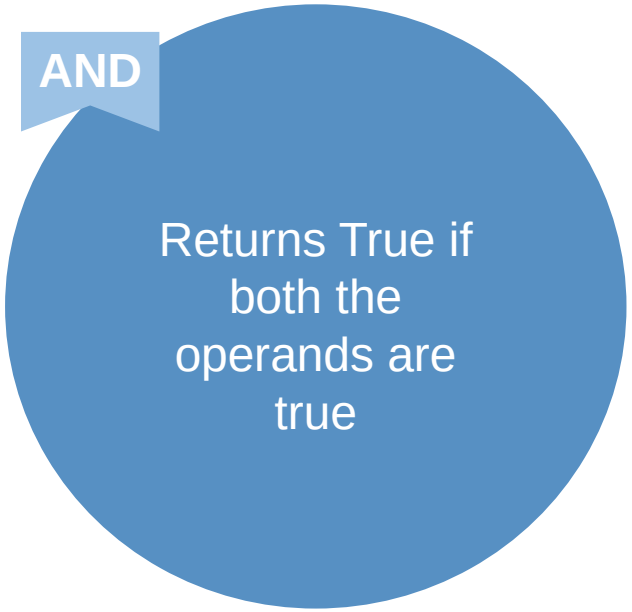
```
True
```

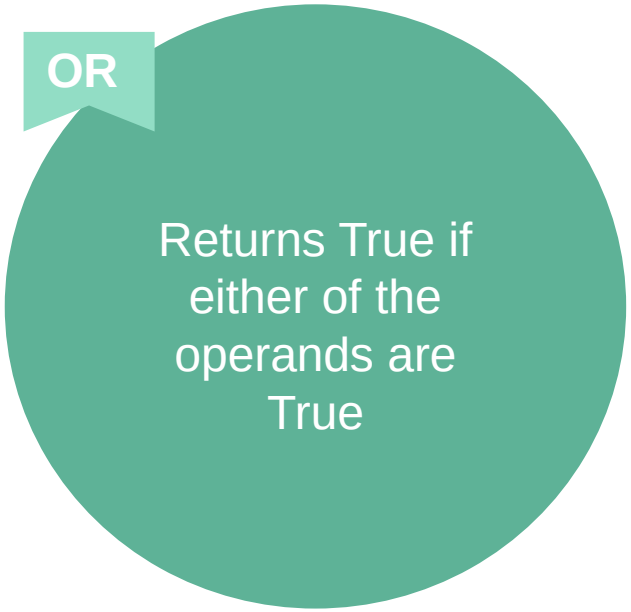
LOGICAL OPERATORS

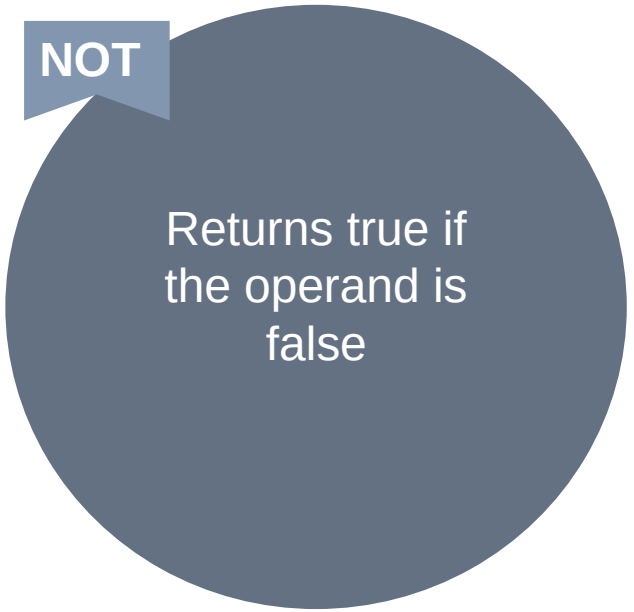
IMARTICUS LEARNING

Logical operators in Python allow a program to make a decision based on multiple conditions

Each operand is considered a condition that can be evaluated to a true or false value





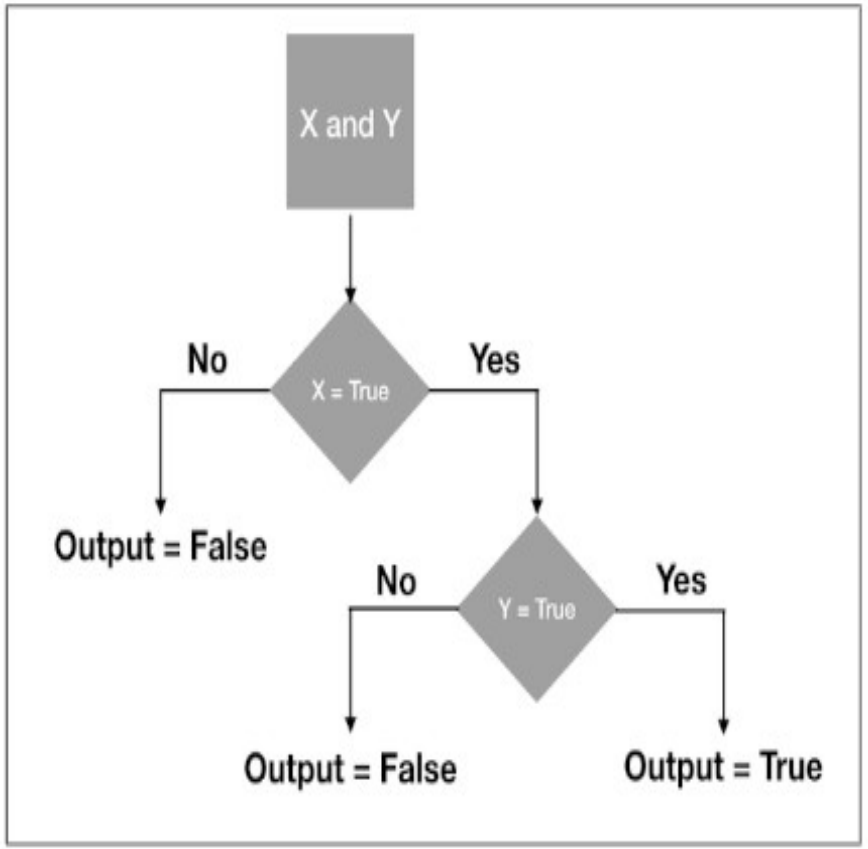


LOGICAL OPERATORS

IMARTICUS LEARNING

AND

If both the operands are true then it returns true




```
is_student = True
in_college = True
in_school = False
```

```
is_student and in_college
```

```
True
```

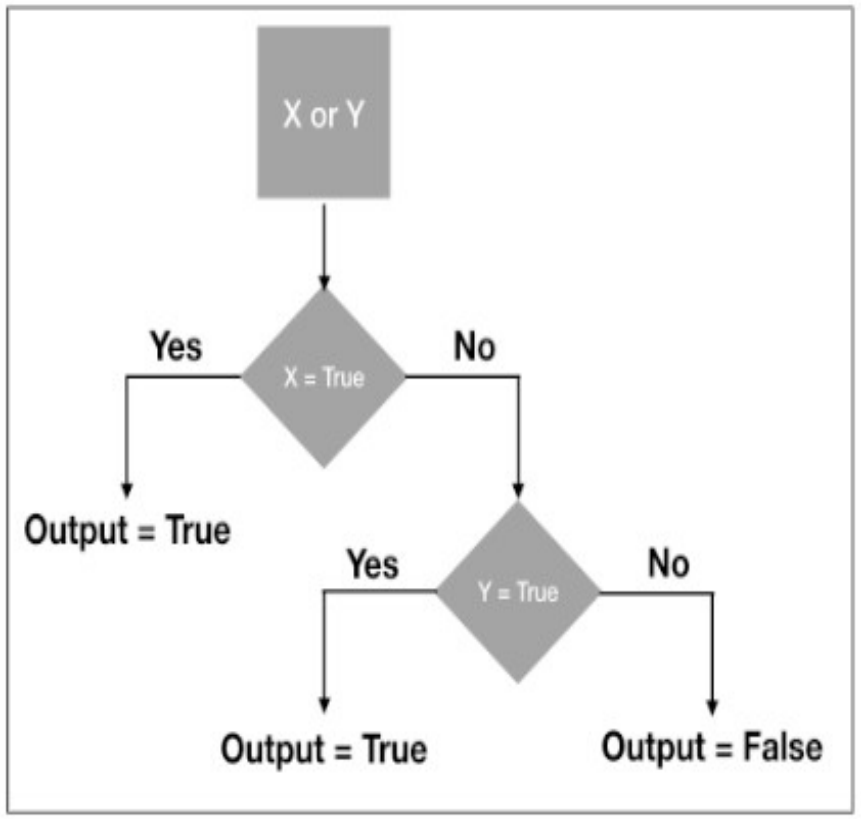
```
is_student and in_school
```

```
False
```

LOGICAL OPERATORS

OR

If one of the operands are true then it returns true




```
is_student = True
in_college = True
in_school = False
```

```
is_student or in_college
```

```
True
```

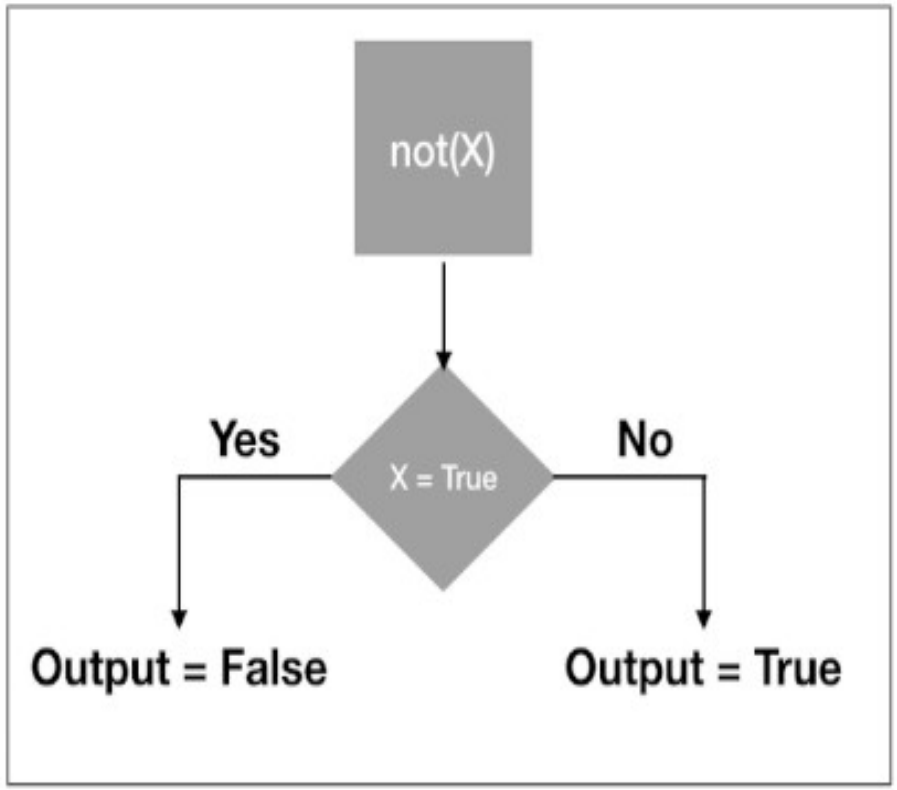
```
is_student or in_school
```

```
True
```

LOGICAL OPERATORS

NOT

Reverses the result




```
is_student = True
in_college = True
in_school = False
```

```
not(is_student)
```

```
False
```

IDENTITY OPERATORS

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location

is	Is not
• Returns True if both variables are stored in same memory location • Ex:- x is y	• Returns True if both variables are stored in Different memory location • Ex:- x is not y

IDENTITY OPERATORS

IMARTICUS LEARNING

is

```
# Declare a list of numbers to a variable
list1 = [10,20,30]
list2 = [10,20,40]
list3 = list1
```

```
# Lets check the Memory location of the variable
print('Memory Location for list1 =',id(list1))
print('Memory Location for list2 =',id(list2))
print('Memory Location for list3 =',id(list3))
print('list1 is list2 =',list1 is list2)
print('list1 is list3 =',list1 is list3)

if id(list1) == id(list2):
    print('list1 and list2 store in same memory location')
elif id(list2) == id(list3):
    print('list2 and list3 store in same memory location')
elif id(list1) == id(list3):
    print('list1 and list3 store in same memory location')
```

```
Memory Location for list1 = 1735244909376
Memory Location for list2 = 1735244960192
Memory Location for list3 = 1735244909376
list1 is list2 = False
list1 is list3 = True
list1 and list3 store in same memory location
```

IDENTITY OPERATORS

IMARTICUS LEARNING

Is not

```
x = ["Jeevan", "Navya", "Kavya"]
y = ["Jeevan", "Navya", "Kavya"]
z = x

# Returns False because z is the same object as x
print(x is not z)

# Returns True because x is not the same object as y,
# even if they have the same content
print(x is not y)

# To demonstrate the difference betweeen "is not" and "!=":
# This comparison returns False because x is equal to y
print(x != y)
```

```
False
True
False
```

MEMBERSHIP OPERATORS

IMARTICUS LEARNING

Membership operators checks whether a value is a member of a sequence. The sequence may be a list, a string, a tuple, or a dictionary

in	not in
• The ‘in’ operator is used to check if a value exists in any sequence object or not	• A ‘not in’ works in an opposite way to an ‘in’ operator. A ‘not in’ evaluates to True if a value is not found in the specified sequence object. Else it returns a False.

MEMBERSHIP OPERATORS

IMARTICUS LEARNING

in

```
# declare a vowel string variable
vowels = "aeiou"

# check if the character 'u' is present in vowels
print("u" in vowels)
```

```
True
```

```
# declare a vowel string variable
vowels = "aeiou"

# check if the character 'x' is present in vowels
print("x" in vowels)
```

```
False
```

MEMBERSHIP OPERATORS

IMARTICUS LEARNING

not in

```
# declare a vowel string variable
vowels = "aeiou"

# check if the character 'u' is not present in vowels
print("u" not in vowels)
```

```
False
```

```
# declare a vowel string variable
vowels = "aeiou"

# check if the character 'x' is not present in vowels
print("x" not in vowels)
```

```
True
```

SOME BITWISE OPERATORS

IMARTICUS LEARNING

Bitwise operators are used to performing bitwise calculations on integers. The integers are first converted into binary and then operations are performed on bit by bit, hence the name bitwise operators. Then the result is returned in decimal format.

OPERATOR	DESCRIPTION	SYNTAX
&	Bitwise AND	x & y
|	Bitwise OR	x | y
~	Bitwise NOT	~x
^	Bitwise XOR	x^y
>>	Bitwise Right Shift	x>>
<<	Bitwise Left Shift	x<<

TRUTH TABLES

IMARTICUS LEARNING

& (Bitwise and)

Input		Output
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

| (Bitwise or)

Input		Output
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

^(Bitwise XOR)

Input		Output
TRUE	TRUE	FALSE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

SOME BITWISE OPERATORS

Example

& operator

```
# & operator example
number = 10
(number % 2 == 0) & (number % 3 == 0)
```

```
False
```

(10 % 2 == 0) & (10 % 3 == 0)

Implies True & False. This results in False.

| operator

```
# | operator example
number = 10
(number % 2 == 0) | (number % 3 == 0)
```

```
True
```

(10 % 2 == 0) | (10 % 3 == 0)

Implies True | False. This results in True.

DIFFERENT TYPE OF BITWISE OPERATORS WITH EXAMPLE

IMARTICUS LEARNING

```
# To convert int to binary
print(bin(10))
print(bin(12))
# Printing Bitwise AND(&) operator
print('10 & 12 =',10 & 12)
# Printing Bitwise OR(|) operator
print('10 | 12 =',10 | 12)
# Printing Bitwise XOR(^) operator
print('10 ^ 12 =',10 ^ 12)
# Printing Bitwise Not(~) operator
# Not operator always fallow -(x+1) rulse
print('~10 =',~10)
# Printing Right Shift
print('10 >>2 =',10 >>2)
# Printing Left Shift
print('10 <<2 =',10 <<2)
```

```
0b1010
0b1100
10 & 12 = 8
10 | 12 = 14
10 ^ 12 = 6
~10 = -11
10 >>2 = 2
10 <<2 = 40
```

MORE ASSIGNMENT OPERATORS

IMARTICUS LEARNING

The assignment operators are used to store data into a variable

assignment operation	Equivalent
a += b	a = a + b
a *= b	a = a * b
a /= b	a = a / b
a %= b	a = a % b
a **= b	a = a ** b
a //= b	a = a // b

IMARTICUS LEARNING

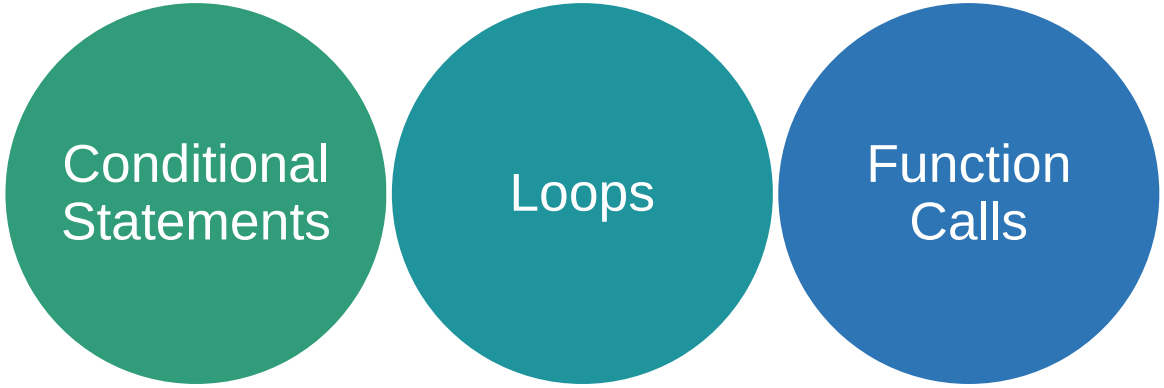
Python Flow Controls

PYTHON FLOW CONTROL

IMARTICUS LEARNING

Any program has a flow and the flow is the order in which the program's code executes

The control flow of a Python program is controlled by:



We cover the basics of conditional statements and loops in today's session

IMARTICUS LEARNING

Conditional Statements

THE IF-STATEMENT

- The *if* statement is used in Python for decision making
- It is written by using the *if* keyword
- The i*f* keyword is followed by condition later followed by indented block of code which will be executed only if the condition is true

Syntax:

if (condition):
statement(s)

THE IF-STATEMENT

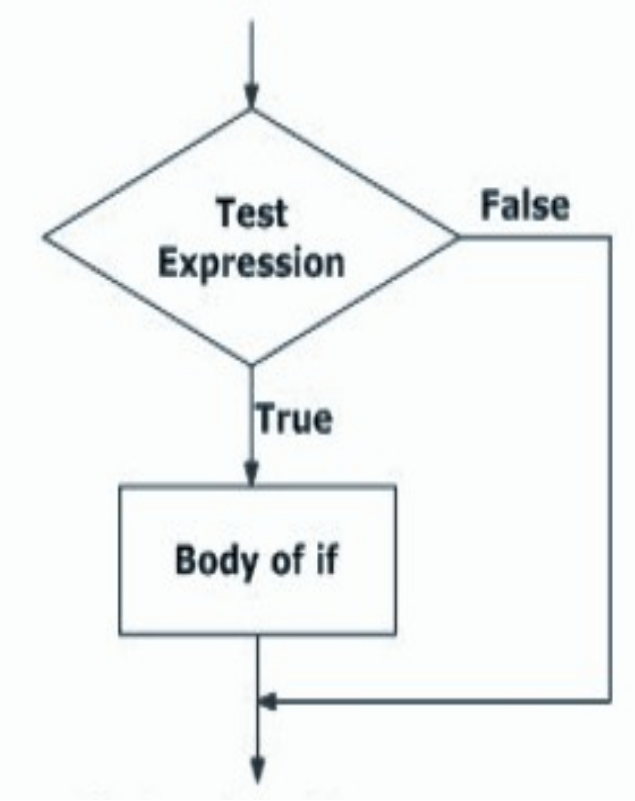
Example

```
# check if a number is greater than zero
num = 8
if(num > 0):
    print(num, "is greater than zero")
print("This is always printed")
```

```
8 is greater than zero
This is always printed
```

```
# check if a number is greater than zero
num = -1
if(num > 0):
    print(num,"is greater than zero")
print("This is always printed")
```

```
This is always printed
```



THE IF-ELSE STATEMENT

- The ‘if..else’ statement evaluates a *test expression* and will execute:
  - the indented *if* block of code if the condition is *True*
  - the indented *else* block of code if the condition in the if statement is *False*

Syntax:

```
if (condition):
    Body of if
else:
    Body of else
```

THE IF-ELSE STATEMENT

IMARTICUS LEARNING

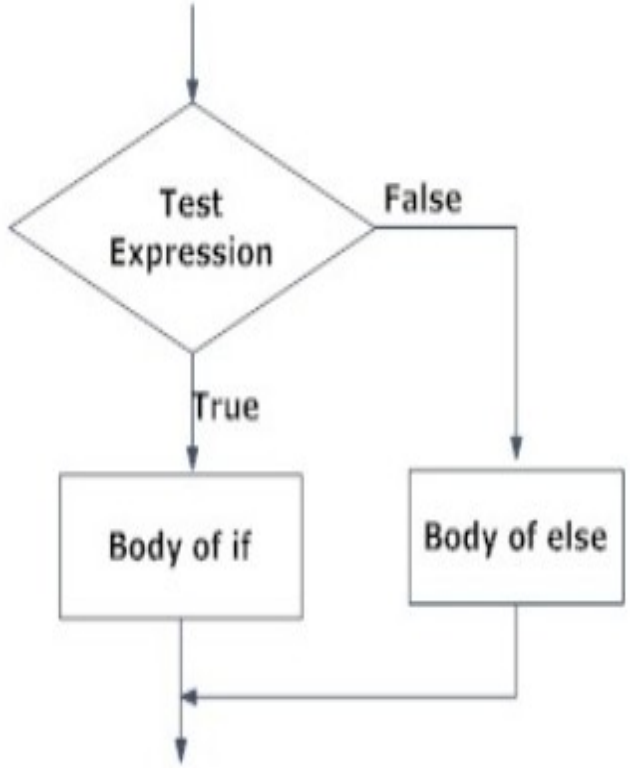
Example

```
# check if a number is greater than zero
num = -1
if(num > 0):
    print(num,"is greater than zero")
else:
    print(num, "is less than zero")
print("This is always printed")
```

```
-1 is less than zero
This is always printed
```

```
# check if a number is greater than zero
num = 8
if(num > 0):
    print(num,"is greater than zero")
else:
    print(num, "is less than zero")
print("This is always printed")
```

```
8 is greater than zero
This is always printed
```



THE IF ELIF ELSE STATEMENT

IMARTICUS LEARNING

Elif statement is used to control multiple conditions only if the given if condition is false

Syntax:

```
if (condition1):
    Body of if
elif (condition2):
    Body of elif
elif (condition3):
    Body of elif
.
.
.
else:
    Body of else
```

THE IF ELIF ELSE STATEMENT

IMARTICUS LEARNING

Example

```
# display bus fares as per age entered by user
age = int(input("Enter age: "))
if (0 < age < 3):
    print("No fares for age group 0-3")
elif(3 < age < 10):
    print("Bus fare is 20")
elif(10 < age < 20):
    print("Bus fare is 25")
else:
    print("Bus fare is 30")
```

```
Enter age: 17
Bus fare is 25
```

NESTED IF-ELSE STATEMENT

- Nesting means using an if statement within another if statement
- If the first ‘if’ condition is satisfied then the program executes the commands within that ‘if’

Syntax:

```
if (condition):
    statement(s)
    if (condition):
        statement(s)
    else:
        statement(s)
else:
        statement(s)
```

NESTED IF-ELSE STATEMENT

IMARTICUS LEARNING

Example

```
# find the largest number among the three numbers using nested if-else
num1 = 10
num2 = 34
num3 = 5
if (num1 > num2):
    print(num1, "is greater")
else:
    if (num3 > num2):
        print(num3, "is greater")
    else:
        print(num2, "is greater")
```

```
34 is greater
```

NESTED IF STATEMENT

IMARTICUS LEARNING

Example

```
# Program to check password for a used id
user_id = input("Enter user id: ")
password = input("password: ")
if (user_id == "ds_py"):
    if (password == "abc123"):
        print("Password Accepted. Logged in")
    else:
        print("Wrong Password. Please enter correct password")
```

```
Enter user id: ds_py
password: abc123
Password Accepted. Logged in
```

IMARTICUS LEARNING

Loops

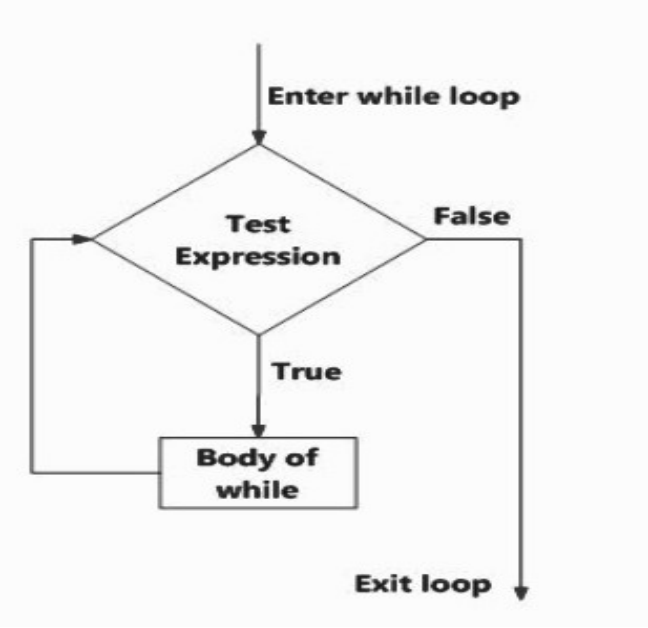
WHILE LOOP

- Loops are used to execute of a specific block of code repetitively
- The ‘while loop’ in Python is used to iterate over a block of code as long as the test expression holds true
- This loop is used when the number of times to iterate is not known to us beforehand

Syntax:

while (condition):

Body of while



WHILE LOOP

IMARTICUS LEARNING

Example

```
# program to find the summation of all natural numbers till 10
num = 10

# initialize addition and counter
addition = 0
i = 1

while(i <= num ):                # while loop with the termination condition
    addition = addition + i
    i = i + 1                    # incrementing the counter by 1 everytime the loop is executed

# print the sum
print("Addition is: ", addition)
```

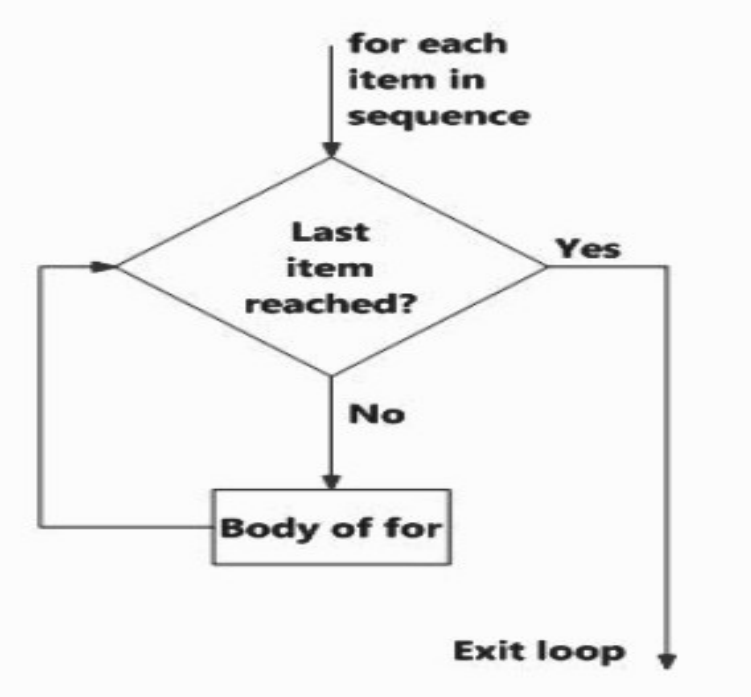
```
Addition is:  55
```

THE FOR LOOP

- The *'for loop'* in Python is used to iterate over the items of a sequence object like list, tuple, string and other iterable objects
- The iteration continues until we reach the last item in the sequence object

Syntax:

for i in sequence:
Body of for



THE FOR LOOP

IMARTICUS LEARNING

Example

```
# program to find the summation of all natural numbers till 10 using for loop
number = range(1,11)
addition = 0
for i in number:                    # i takes one value at a time from 1 till 10 after every single iteration
    addition = addition + i         # previous value of addition is added to the new value taken by i from the iterable

# print the sum
print("Addition is: ", addition)
```

```
Addition is:  55
```

Iteration is a general term for taking each item from a sequence one after another

GETTING OUT OF A LOOP

IMARTICUS LEARNING

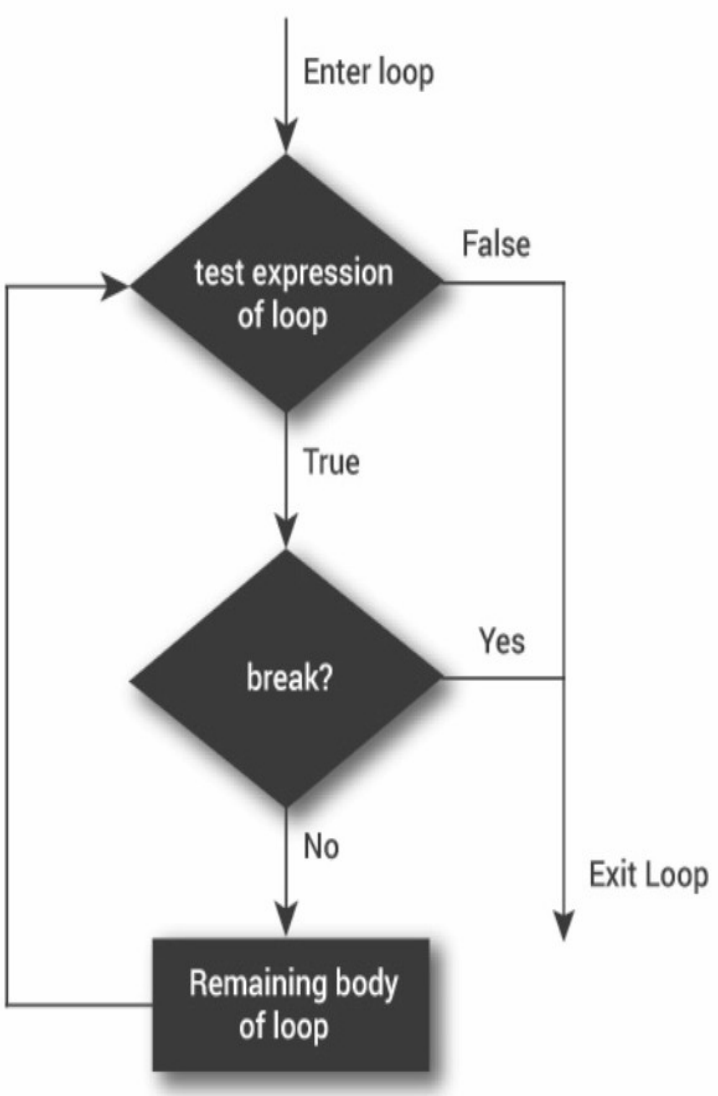
Loops in Python allows us to repeat tasks

But at times, you may want to:

- Exit a loop completely when a certain condition is triggered
- Skip part of a loop and start next execution

GETTING OUT WITH BREAK STATEMENT

- The ‘break’ statement ends the loop and resumes execution at the next statement
- The break statement can be used in both ‘while’ loop and ‘for’ loop
- It is always used with conditional statements



GETTING OUT WITH BREAK STATEMENT

IMARTICUS LEARNING

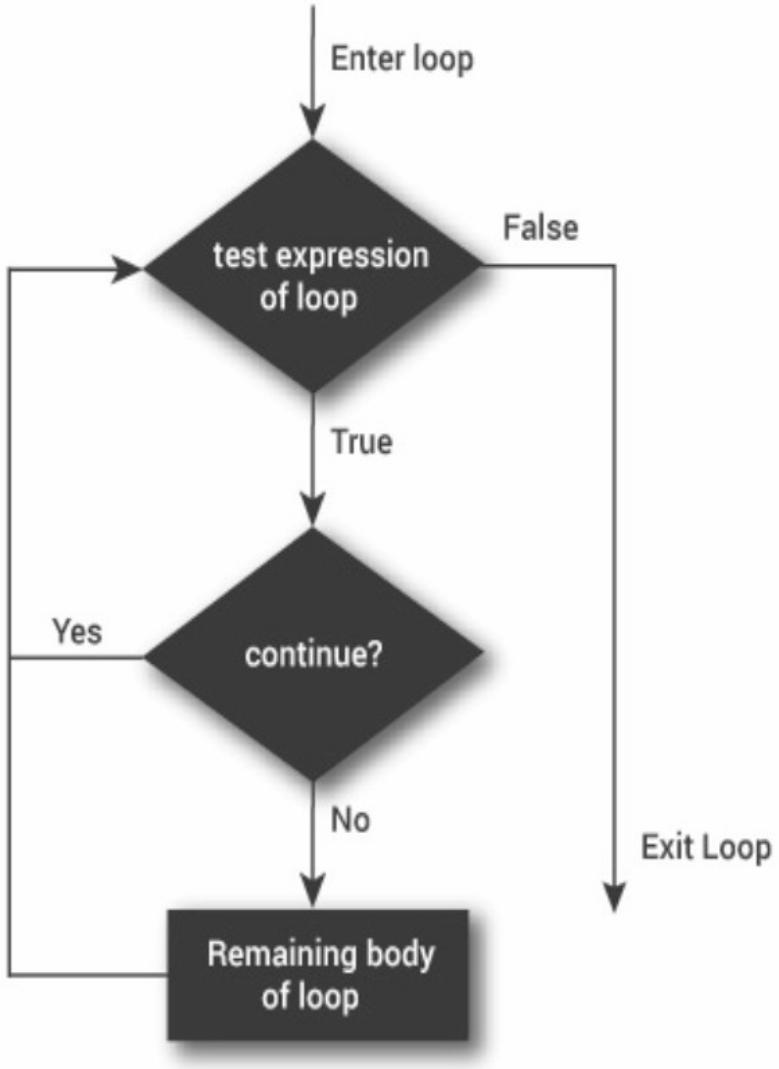
Example

```
# declare a string variable
text = "Knowledgehut"
for letter in text:     # letter take one letter at a time for every iteration over the string
    if letter == 'h':   # condition to stop the execution of the loop
        break
    print( letter)
print("The for loop stopped executing")
```

```
K
n
o
w
l
e
d
g
e
The for loop stopped executing
```

SKIP LOOP STEP WITH CONTINUE STATEMENT

- The ‘continue’ statement in Python ignores all the remaining statements in the iteration of the current loop and moves the control back to the beginning of the loop
- The continue statement can be used in both ‘while’ loop and the ‘for’ loop
- Like break, continue is also always used with conditional statements



SKIP LOOP STEP WITH CONTINUE STATEMENT

IMARTICUS LEARNING

Example

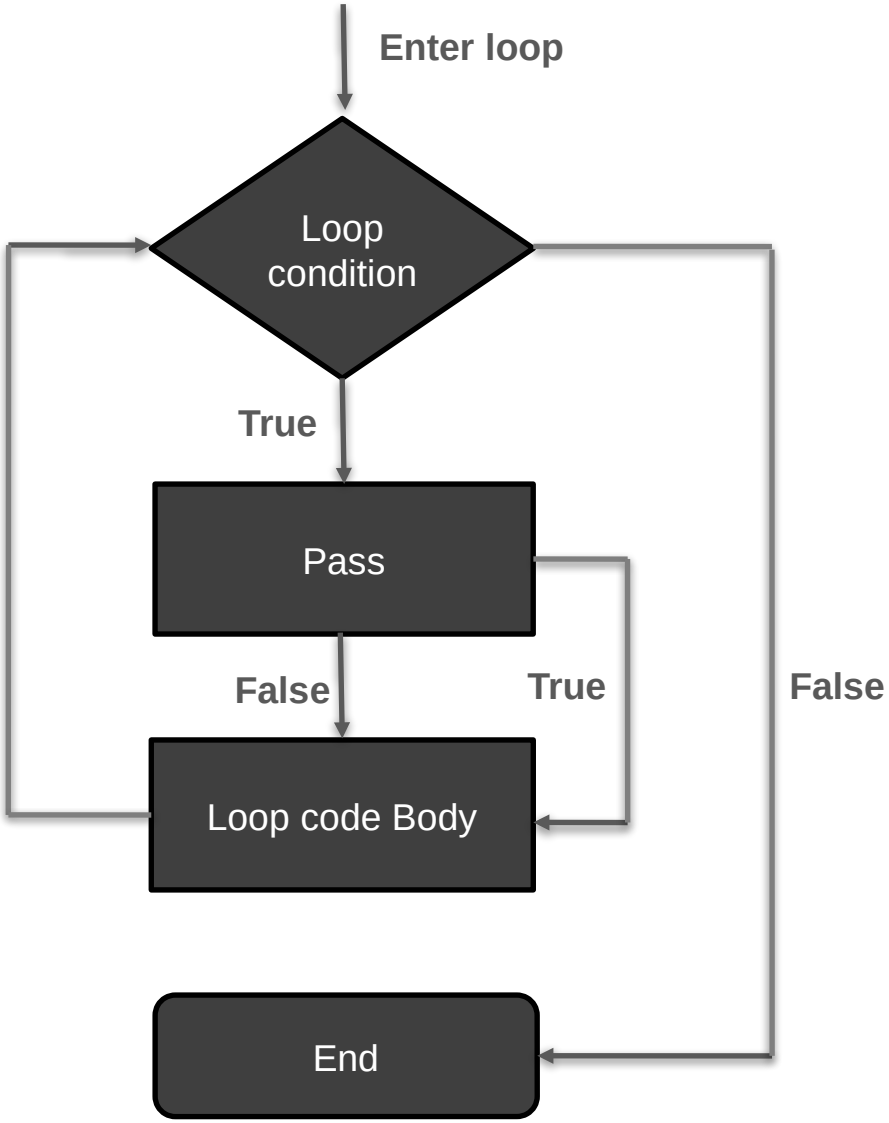
```
# declare a string variable
text = "Knowledge hut"
for letter in text:     # letter take one letter at a time for every iteration over the string
    if letter == ' ':   # condition to skip printing of letter
        print("The loop skipped a space here")
        continue
    print( letter)
```

```
K
n
o
w
l
e
d
g
e
The loop skipped a space here
h
u
t
```

NULL OPERATION WITH PASS STATEMENT

IMARTICUS LEARNING

- The Python Pass Statement is used as a placeholder within loops, functions, classes, and if-statements that will be implemented later.
- A Pass statement is a null statement that is used in cases where the loop, function, or class is to be ignored or written and executed in the future.
- As the name suggest Pass statement simply does nothing.



NULL OPERATION WITH PASS STATEMENT

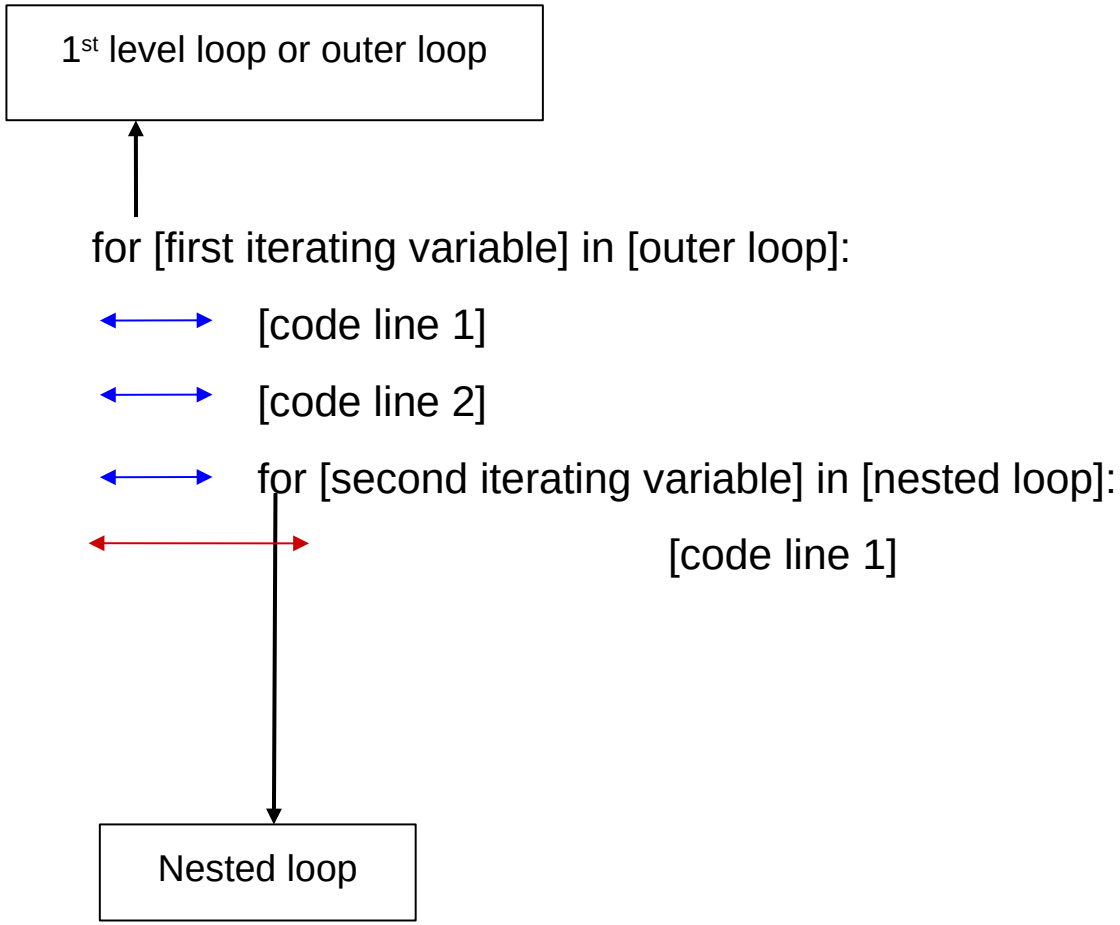
IMARTICUS LEARNING

Example

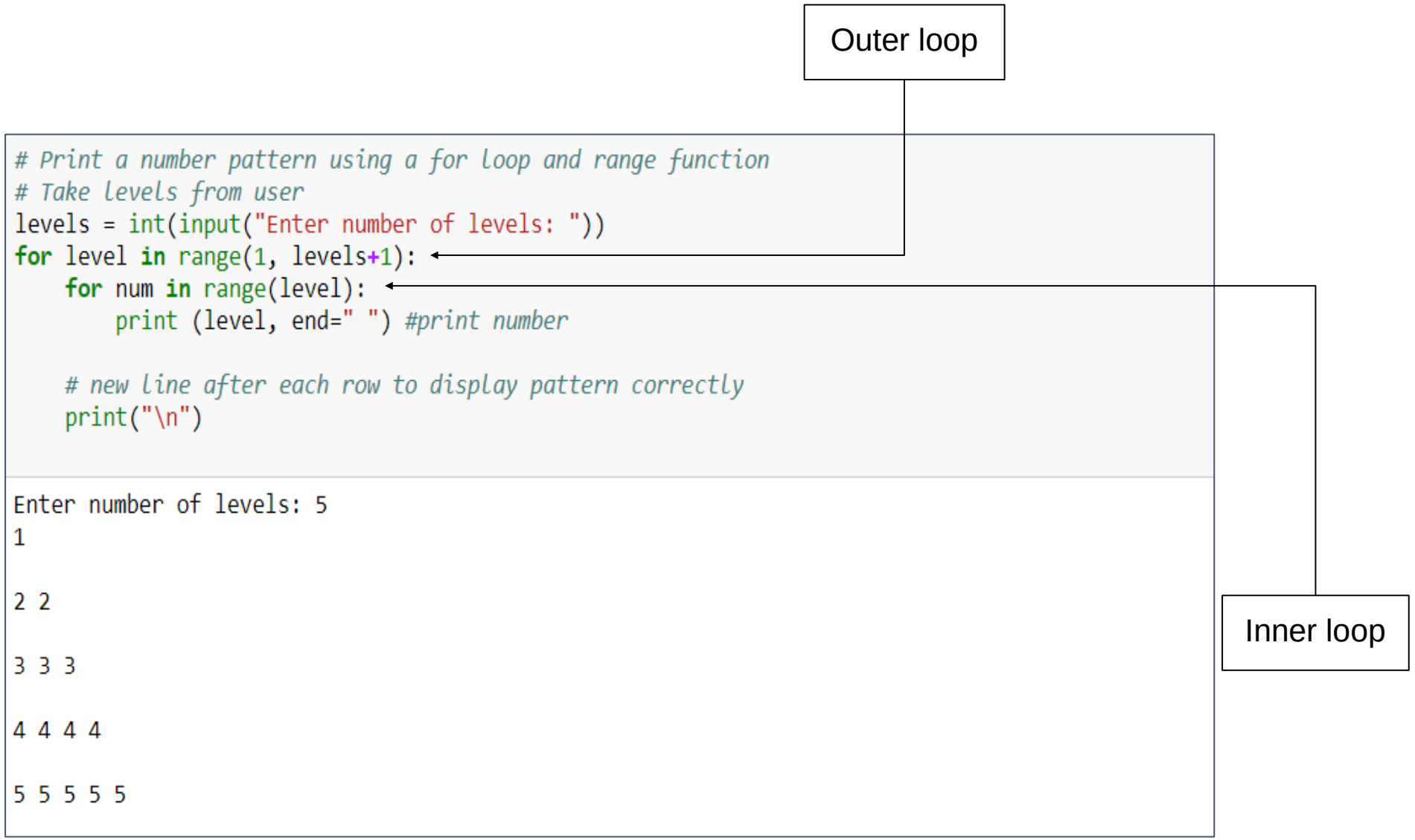
```
# Declare a list of number in a variable called num
num = [1, 3, 6, 33, 76, 29, 17, 60, 100, 47, 53, 88]

print('Odd numbers are: ')
for i in num:
    # check if the number is even
    if i % 2 == 0:
        # if even, then pass
        pass
    # print the odd numbers
    else:
        print (i)
```

```
Odd numbers are:
1
3
33
29
17
47
53
```

NESTED FOR LOOP



NESTED FOR LOOP



We're committed to empower you to be

#FutureReady

through powerful training solutions.

IMARTICUS LEARNING

We build the workforce of the future.

250+
Corporate Clients

30,000+
Learners Trained

25000+
Learners Placed

FREE WEBINARS

Sign up for free webinars with industry experts every fortnight!

Enroll Now!

Contact us today!

https://imarticus.org/