**Python Programming Language**

**Section: Python Basic**

1. Python Introduction
2. Python Installation & IDE & Python Syntax
3. Comments & Indentation
4. Variables & Casting
5. Python Input / Output
6. Python Data Types
7. Python Strings
8. Python Operators
9. Python if else
10. While Loops
11. For Loops
12. Python Continue and Break
13. Python List
14. Python Tuples
15. Python Sets
16. Python Dictionaries
17. Functions
18. Lambda Function
19. Variables Scope
20. Python Modules

**Section: Python Advanced**

1. Python Classes and Objects
2. Python Inheritance
3. Access Modifiers in Python
4. Operator Overloading in Python
5. Magic Methods in Python
6. \_\_main\_\_ and \_\_name\_\_ in Python
7. Python Exception Handling
8. File Handling in Python
9. Python Mysql

**Python Introduction**

**What is Python?**

Python is a popular programming language. It is being used in

* Machine Learning Applications
* Scientific Applications
* Software Development
* Web development
* Desktop Applications
* Web scraping
* Game Development

The most recent major version of Python is Python 3.

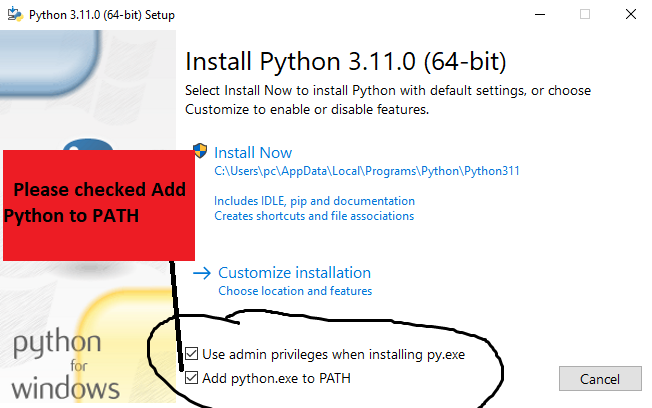
**Organizations using Python:**

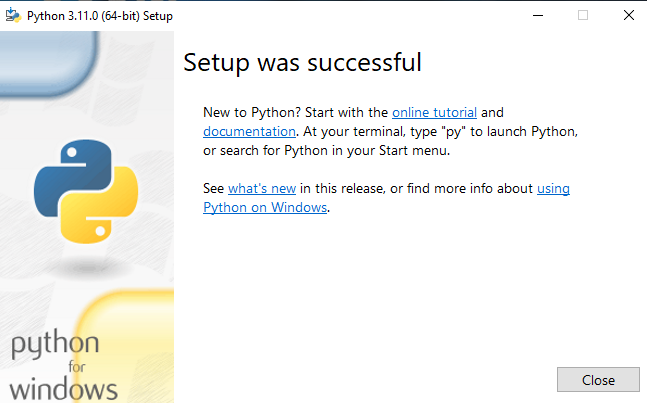
* Google( Components of Google spider and Search Engine )
* Yahoo(Maps)
* YouTube
* Mozilla
* Dropbox
* Microsoft
* Cisco
* Spotify
* Quora

**Python installation & IDE & Python Syntax**

**Download Python**

Download and install from the Python website <https://www.python.org/>





Below is the command to check whether python is installed or not

**python --version**

**IDE :-**

There are many Python IDE’s but we prefer Visual Studio.

**Python Syntax**

print(“Hello World”)

**The Python Command Line**

Python can be executed from the command line

Below is the command to run Python from the command line:

**C:\Users\User Name>python**

**Python Comments & Indentation**

**Comments**

Comments are used to explain Python code.

There are 2 types of comments in Python

1. # Commnet Text
2. ””” Multiple Line Comment Example Multiple Line Comment Example
   1. Multiple Line Comment Example Multiple Line Comment Example
   2. Multiple Line Comment Example Multiple Line Comment Example
   3. Multiple Line Comment Example Multiple Line Comment Example
   4. Multiple Line Comment Example Multiple Line Comment Example
   5. ”””

**Python Indentation**

## 

**Python Variables**

**x = 10**

**age = 23**

**name = "Mike"**

**Variable Name**

A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )

A variable name must start with a letter or the underscore character

A variable name cannot start with a number

Variable names are case-sensitive ( address, Address and Address are three different variables)

**In case of multiple words, we can use either technique for better readability.**

**Camel Case**

myVariableName = "Mike"

studentAge = "22"

**Pascal Case**

MyVariableName = "Rohan"

StudentAge = "25"

**Snake Case**

my\_variable\_name = "John"

student\_age = "26"

**Assign One Value to Multiple Variables**

x = y = z = 10

**Assign Many Values to Multiple Variables**

name, qty, price = “Banana”, 12, 2.50

**Case-Sensitive**

Variable names are case-sensitive. I.e age , Age, AGE are all different variables

**Variable Type**

**x = 50**

**y = "Mike"**

**print(type(x))**

**print(type(y))**

**Casting**

x = “12”

x = float(x)

x = int(x)

**Can not add int and string**

**x = 50**

**y = "20"**

**y = x + int(y)**

**print(z)**

**Variable Deletion**

**x = 50**

**print(x)**

**del x**

**print(x)**

**Python Input / Output**

**Output**

**print(“This is an example of print”)**

**print(“This is an example of print”, “This is an another example of print”)**

**name = “Mike”**

**print(name)**

**#In case of string, we can concatenate**

**print(“My name is ”+ name )**

**#in case of int, the above statement will not work**

**age = ”10”**

**Print ( f “ Your age is {age}”)**

**Print ( f “ Your age is {age}” , end = ’’)**

**Print ( f “ Your age is {age}”)**

**Python input**

**x = input ( “Enter your name ”)**

**age = int(input ( “Enter your age”))**

**price = float(input ( “Enter the price”))**

**Exercise**

Write a program to swap the two numbers.

**Python Data Types**

**Python has the following standard data types −**

1. Numbers
2. Boolean Type
3. String
4. List
5. Tuple
6. Set
7. Dictionary

**1. Numbers**

**There are three numeric types in Python:**

1. int
2. float
3. Complex

x = 12 # int

y = 4.52 # float

z = 5j # complex or z = 1+5j

a = -100 # int

Number can be converted in each other:

a = float(x)

b = int(y)

**2. Booleans**

**Boolean has two values: True or False.**

x = True

y = Flase

print( 9 > 7)

a = bool(12)

**Python Strings**

A sting is a set of characters and can be created by using single quotation marks, or double quotation.

x = “This is an example of string”

Name =”Mike”

**print(x)**

**Multiline Strings**

**x = """Lorem ipsum dolor sit amet,**

**consectetur adipiscing elit,**

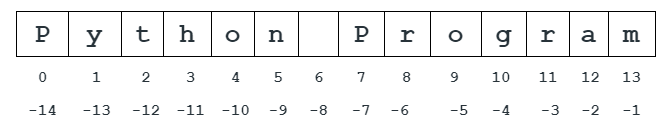
**sed do eiusmod tempor incididunt**

**ut labore et dolore magna aliqua."""**

**print(x)**

**Strings are Arrays**

**x = “Python Program”**



**print(x[5])**

**print(x[-2])**

**print(x[10])**

**print(x[-8])**

**String Length**

**a = "Hello, World!"**

**print(len(a))**

**String “in” and “not in”**

**x = "The world is a awesome place"**

**print(“world” in x )**

**print(“bad” in x )**

**x = "Python is an easy language"**

**print(“hard” not in x )**

**print(“easy” not in x )**

**Slicing Strings**

Return the sub string by using slice syntax

Get the characters from position 2 to position 5 (not included)

**x = “Python Program”**

**print(x[2:5])**

**Slice From the Start**

**x = “Python Program”**

**print(x[:5])**

**Slice to the end**

**x = “Python Program”**

**print(x[3:])**

**Negative Indexing**

Get the characters from position -7 to position -3 (not included)

**x = “Python Program”**

**print(x[-7:-3])**

**Slice From the Start**

**x = “Python Program”**

**print(x[:-2])**

**Slice to the end**

**x = “Python Program”**

**print(x[-9:])**

**String Concatenation**

**x = “Hello”**

**y = “World”**

**z = x + y**

**print(z)**

**z = x +’ ’+ y**

**print(z)**

**String Format**

we can combine strings and numbers by using the format() method!

quantity = 5

item = “Apple”

price = 21.50

str= "I want {} kg {} for {} dollars."

print(str.format(quantity, item , price))

quantity = 5

item = “Apple”

price = 21.50

str= "I want {1} kg {2} for {0} dollars."

print(str.format(price, quantity, item))

**Escape Sequencing**

If we want to put single or double quotes in any string, we can’t as string already contains Single and Double quote.

#Python is a “great” language

txt = "Python is a \“great\” language"

print(txt)

#I’m a programmer

txt = ‘I\’m a programmer’

print(txt)

**String Methods:**

There are many string manipulation functions in Python, you can find complete ref here on Python site <https://docs.python.org/2.5/lib/string-methods.html>

Some important string function are listed below:

**upper()**

Return a copy of the string converted to uppercase.

**x = "Hello, World!"**

**print(x.upper()) # returns HELLO WORLD**

**lower()**

Return a copy of the string converted to lowercase.

**x = "Hello, World!"**

**print(x.lower()) # returns hello, world!**

**capitalize()**

Return a copy of the string with only its first character capitalized.

**x = "hello, world!"**

**print(x.capitalize()) # returns Hello, world!**

**strip()**

Return a copy of the string with the leading and trailing characters removed.

**x = " Hello, World! "**

**print(x.strip()) # returns Hello, World!**

**lstrip()**

Return a copy of the string with leading characters removed.

**x = " Hello, World! "**

**print(x.lstrip()) # returns Hello, World!**

**rstrip()**

Return a copy of the string with trailing characters removed.

**x = "Hello, World! "**

**print(x.rstrip()) # returns Hello, World!**

**replace(old, new)**

Return a copy of the string with all occurrences of substring old replaced by new.

**x = "Hello, World!"**

**print(x.replace("Hello", "Hey")) # returns Hey, World!**

**split(sep)**

Return a list of the words in the string, using sep as the delimiter string.

**a = "Hello, World!"**

**print(a.split(",")) # returns ['Hello', ' World!']**

**islower()**

Return true if all cased characters in the string are lowercase

**x = "hello, world!"**

**print(x.islower()) #returns true**

**title()**

Return a titlecased version of the string:

**x = "hello world!"**

**print(x.title()) #returns Hello World**

**Python Operators**

Operators are used to perform operations on variables and values.

In the example below, + is the operator used to add two variables x and y

x = 10

y = 15

z = x + y

**Below are the different types of Python operators:**

* Arithmetic operators
* Comparison operators
* Logical operators
* Bitwise operators
* Assignment operators
* Identity operators
* Membership operators

**Arithmetic operators**

Arithmetic operators are used to performing mathematical operations like addition, subtraction, multiplication, and division.

| **Operator** | **Name** | **Syntax** |
| --- | --- | --- |
| + | Addition | x + y |
| - | Subtraction | x - y |
| \* | Multiplication | x \* y |
| / | Division | x / y |
| % | Modulus | x % y |
| \*\* | Exponentiation | x \*\* y |
| // | Floor division | x // y |

**Comparison Operators**

Comparison operators are used to compare two values.It either returns True or False according to the condition.

| **Operator** | **Name** | **Syntax** |
| --- | --- | --- |
| == | Equal | x == y |
| != | Not equal | x != y |
| > | Greater than | x > y |
| < | Less than | x < y |
| >= | Greater than or equal to | x >= y |
| <= | Less than or equal to | x <= y |

**Logical Operators**

Logical operators are used to combine conditional statements.

| **Operator** | **Description** | **Syntax** |
| --- | --- | --- |
| and | Returns True if both statements are true | x > y and a < b |
| or | Returns True if one of the statements is true | x > y or a < b |
| not | Reverse the result, returns False if the result is true | not ( x > y or a < b ) |

**Bitwise Operators**

Bitwise operators are used to compare (binary) numbers.

| **Operator** | **Name** | **Description** | **Syntax** |
| --- | --- | --- | --- |
| & | Bitwise AND | Sets each bit to 1 if both bits are 1 | x & y |
| | | Bitwise OR | Sets each bit to 1 if one of two bits is 1 | x | y |
| ~ | Bitwise NOT | Inverts all the bits( 0 to 1 and 1 to 0 ) | ~x |
| ^ | Bitwise XOR | Sets each bit to 1 if only one of two bits is 1 | x ^ y |
| << | Bitwise left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off | x << 2 |
| >> | Bitwise right shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off | x >> 3 |

**Assignment Operators**

Assignment operators are used to assign values to variables.

| **Operator** | **Syntax** | **Same as** |
| --- | --- | --- |
| = | x =10 | x =10 |
| += | x +=y | x = x + y |
| -= | x -=y | x = x - y |
| \*= | x \*=y | x = x \* y |
| /= | x /=y | x = x / y |
| %= | x %=y | x = x % y |
| //= | x //=y | x = x // y |
| \*\*= | x \*\*=y | x = x \*\* y |
| &= | x &=y | x = x & y |
| |= | x |=y | x = x | y |
| ^= | x ^=y | x = x ^ y |
| >>= | x >>=y | x = x >> y |
| <<= | x <<=y | x = x << y |

**Identity operators**

is and is not are the identity operators both are used to check if two values are located on the same part of the memory.

| **Operator** | **Description** | **Syntax** |
| --- | --- | --- |
| is | Returns True if both variables are the same object | x is y |
| is not | Returns True if both variables are not the same object | x is not y |

**Membership Operators**

in and not in are the membership operators; used to test whether a sequence is presented in an object or not.

| **Operator** | **Description** | **Syntax** |
| --- | --- | --- |
| in | Returns True if a sequence with the specified value is present in the object | x in y |
| is not | Returns True if a sequence with the specified value is not present in the object | x in not y |

**Python if else**

if *condition*:

# Statements to execute if

# condition is true

x = 100

y = 20

if x > y:

print("x is greater than y")

**else**

x = 100

y = 20

if x > y:

print("x is greater than y")

else:

print("x is not greater than y")

**If -elif - else:**

**Short Hand If:**

**if a > b: print("a is greater than b")**

**Short Hand If ... Else (Also known as Ternary Operators )**

**a = 2**

**b = 330**

**print(a) if a > b else print(b)**

**Nested If**

**The pass Statement**

**If x > y**

**pass**

**While Loop**

**There are two types of loop in Python:**

while loops

for loops

**While Loop**

**while expression:**

**Statement1**

**statement2**

**Example:**

**i = 1**

**while i < 11:**

**print(i)**

**i += 1**

**While loop with else**

**i = 1**

**while i < 11:**

**print(i)**

**i += 1**

**else:**

**print(“Loop is ended”)**

The else clause is only executed when while condition becomes false

**Exercise:**

**1.** Write a program to sum all the numbers between 1 to 100 using a while loop.

**2.** Write a program to print all the even numbers between 1 to 100 using a while loop.

**3.** Write a program to check whether a number is prime or not.

**4.** Write a program to prints all the characters except vowels (a, e, i, o, u) in a string given by the user.

**5.** Write a program to find the sum of all the odd numbers between 1 to 100 using a while loop.

**For Loop**

**A for loop is used for iterating over a sequence**

**for var in iterable:**

**# statements**

**Looping Through a String**

**name = “Python”**

**for x in name:**

**print(x)**

**The range(start, end, step ) Function**

It’s a built-in function that is used when a user needs to perform an action a specific number of times

**for x in range(10):**

**print(x)**

**for x in range(1, 11):**

**print(x)**

**for x in range(1, 11, 2):**

**print(x)**

**For loop with else**

Executed when the for loop is finished

**for x in range(1, 11):**

**print(x)**

**else:**

**print("Finally finished!")**

**Pass Statement**

for i in range(1, 11)

pass

**Exercise:**

**1.** Write a program to sum all the numbers between 1 to 100 using a for loop.

**2.** Write a program to print all the even numbers between 1 to 100 using a for loop.

**3.** Write a program to check whether a number is prime or not using for loop.

**4.** Write a program to prints all the characters except vowels (a, e, i, o, u) in a string given by the user.

**5.** Write a program to find the sum of all the odd numbers between 1 to 100 using a for loop.

**Python Continue and Break**

**Continue Statement**

Continue Statement returns the control to the beginning of the loop in both while and for loop

# Prints all letters except 'e', and 'o'

str = "Hello World!"

for letter in str:

if letter == 'e' or letter == 'o':

continue

print(‘Letter :', letter)

**Break Statement**

The break statement break the loop and brings control out of the loop.

#break the loop as soon 'e' or ‘o’ comes

str = ‘Hello World!’

for letter in str:

if letter == 'e' or letter == 'o':

break

print(‘Letter :', letter)

In case of break, “loop else” statement will not be executed as it executes after finishing of the loop.

**Python List**

**There are 4 types of built-in data types used to store collections of data.**

1. List
2. Tuple
3. Set
4. Dictionary

**List**

Lists are used to store multiple items in a single variable. Lists are created using square brackets:

fruit\_list = ["apple", "orange", "banana", "cherry"]

print( fruit\_list )

List item can be any type of data type like int, float, string, boolean, list tuple, set etc

List items are indexed, the first item has index [0], the second item has index [1] etc.

List items are **ordered**, **changeable**, and allow **duplicate** values.

## **Ordered**

Items in the list have a defined order, and that order will not change. If you add new items to a list, the new items will be placed at the end of the list

**Changeable**

The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.

**Allow Duplicates**

Duplicate items are allowed in any list

## **Access Items**

fruit\_list = ["apple", "orange", "banana", "cherry", "kiwi", "melon", "mango"]

print( fruit\_list[2] )

fruit\_list = ["apple", "orange", "banana", "cherry", "kiwi", "melon", "mango"]

print( fruit\_list[-3] )

**Slicing of a List**

fruit\_list = ["apple", "orange", "banana", "cherry", "kiwi", "melon", "mango"]

print( fruit\_list[2:6] )

print( fruit\_list[2:] )

print( fruit\_list[:6] )

print( fruit\_list[-6:-2] )

print( fruit\_list[-6:] )

print( fruit\_list[:-2] )

**len() function**

Returns the length

**type() function**

Returns the type

fruit\_list = ["apple", "orange", "banana", "cherry", "kiwi", "melon", "mango"]

print( len( fruit\_list ) )

print( type( fruit\_list ) )

**The list() Constructor**

It is also possible to use the list() constructor to make a list.

fruit\_list = list(("apple", "orange", "banana", "cherry", "kiwi", "melon", "mango"))

print( fruit\_list )

**Check if Item Exists**

if ( "mango" in fruit\_list ):

print("Yes mango is in the fruits list")

**Update, Add, Remove item from a list**

**Update**

fruit\_list = ["apple", "orange", "banana", "cherry", "kiwi", "melon", "mango"]

fruit\_list[0] = "green apple"

print( fruit\_list )

**Add**

The append() method add an item to the end of the list

fruit\_list = ["apple", "orange", "banana"]

print( fruit\_list )

fruit\_list.append("mango")

print( fruit\_list )

The insert() method add an item at a specified index

fruit\_list.insert(1, "cherry")

print( fruit\_list )

**Remove**

The remove() method removes the specified item.

fruit\_list.remove("orange")

print( fruit\_list )

The pop() method removes the specified index. By default it removes last item

fruit\_list.pop(1)

print( fruit\_list )

fruit\_list.pop(1)

print( fruit\_list )

The del keyword also removes the specified index:

del fruit\_list [0]

print( fruit\_list )

The del keyword can also delete the list completely.

del fruit\_list

print( fruit\_list )

**Looping Through a List**

fruit\_list = ["apple", "orange", "banana", "cherry", "kiwi", "melon", "mango"]

for x in fruit\_list:

print( x )

#using len() function

numbers = [20, 50, 68, 89, 100]

sum = 0

for x in numbers:

sum += x

print(sum )

#using range() and len() function

numbers = [20, 50, 68, 89, 100]

sum = 0

for x in range( len( numbers ) ):

sum += numbers[x]

print(sum )

#using while loop

numbers = [20, 50, 68, 89, 100]

sum = 0

i = 0

while i < len( numbers ):

sum += numbers[i]

i += 1

print(sum )

**List Comprehension**

If you have list of number and want to create new list only containing even number, below is the code to achieve this.

numbers = [20, 50, 68, 89, 100, 119, 34, 8, 19]

even\_list =[]

for x in numbers:

if 0 == x%2:

even\_list.append(x)

print( even\_list )

But by using list comprehension we can achieve this with only one line of code in python

numbers = [20, 50, 68, 89, 100, 119, 34, 8, 19]

even\_list = [ x for x in numbers if 0 == x%2 ]

print( even\_list )

**The syntax:**

**newlist = [ expression for item in iterable if condition == True]**

**Multiply List**

#Multiply list by 2

two\_fruit\_list = fruit\_list \* 2

print( two\_fruit\_list )

**List Functions**

**append()**

**clear()**

**copy()**

**count()**

**extend()**

**index()**

**insert()**

**pop()**

**remove()**

**reverse()**

**sort() & sort(reverse = True)**

**Python Tuples**

Tuples are used to store multiple items in a single variable. Tuples are written with round brackets:

weekdays = ( "Mon", "Tue", "Wed", "Thu", "Fri" )

print( weekdays )

Tupple item can be any type of data type like int, float, string, boolean, list tuple, set etc

Tupple items are indexed, the first item has index [0], the second item has index [1] etc.

Tupple items are **ordered**, **unchangeable**, and allow **duplicate** values.

## **Ordered**

Items in the tuple have a defined order, and that order will not change. If you add new items to a list, the new items will be placed at the end of the list

**Unchangeable**

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

**Allow Duplicates**

Duplicate items are allowed in any tuple

## **Access Items**

weekdays = ( "Mon", "Tue", "Wed", "Thu", "Fri" )

print( weekdays [2] )

print( weekdays [-3] )

**Create tuple with one item**

If there is only one item in the tuple, add comma

weekdays = ("Mon",)

#not adding command like fruit\_list = ("apple") will be invalid

print( weekdays )

**Slicing of a tuple**

weekdays = ( "Mon", "Tue", "Wed", "Thu", "Fri" )

print( weekdays [1:3] )

print( weekdays [1:] )

print( weekdays [:4] )

print( weekdays [-4:-2] )

print( weekdays [-4:] )

print( weekdays [:-2] )

**len() function**

Returns the length

**type() function**

Returns the type

weekdays = ( "Mon", "Tue", "Wed", "Thu", "Fri" )

print( len( weekdays ) )

print( type( weekdays ) )

**The tuple() Constructor**

It is also possible to use the tuple() constructor to make a tuple.

weekdays = tuple(( "Mon", "Tue", "Wed", "Thu", "Fri" ))

print( weekdays )

**Update, Add, Remove item from a tuple**

Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created. But there are some ways to do that.

First change tuple in a list using list constructor and perform any add/update/remove operation and then convert the list into a tuple again using tuple constructor.

weekdays = ( "Mon", "Tue", "Wed", "Thu", "Fri" )

weekdays\_list = list(weekdays )

#once list has been created, we can perform any of list #operation on it

weekdays\_list.append("Sat")

weekdays = tuple( weekdays\_list)

print( weekdays )

**Add tuple to a tuple**

weekdays\_sun = ("Sun",)

weekdays = weekdays + weekdays\_sun

print( weekdays )

The del keyword can also delete the tuple completely.

del weekdays

print( weekdays )

## **Unpacking a Tuple**

in Python, we can extract the tuple values into variables.

weekdays = ( "Mon", "Tue", "Wed", "Thu", "Fri" )

(day1, day2, day3, day4, day5 ) = weekdays

print( day1 )

print( day2 )

print( day3 )

print( day4 )

print( day5 )

**Looping Through a Tuple**

weekdays = ( "Mon", "Tue", "Wed", "Thu", "Fri" )

for day in weekdays :

print( day )

for day in range( len( weekdays ) ):

print( weekdays[day] )

#using while loop

i = 0

while i < len( weekdays ):

print( weekdays[day])

i += 1

**Multiply Tuples**

#Multiply tuple by 2

weekdays = ( "Mon", "Tue", "Wed", "Thu", "Fri" )

two\_weeks = weekdays \* 2

print( two\_weeks )

**List Functions**

**count()**

**index()**

**Python Sets**

**Sets**

Sets are used to store multiple items in a single variable. Sets are created using curly brackets:

+

countries = {"Pakistan", "USA", "Japan", "Canada"}

print( countries )

List items are unindexed.

Set items are **unordered**, **unindexed**, **unchangeable**, and **duplicate** values are not allowed.

## **Unordered**

Set do not have a defined order.

**Unchangeable**

Set items are unchangeable but you can remove items and add new items

**Duplicates Not Allowed**

Duplicate items are allowed in any set

countries = {"Pakistan", "USA", "Japan", "Canada", "Japan"}

print( countries )

## 

## 

## 

## 

## **Access Items**

As sets are not index, item can not be accessible by index but we can loop through the items using a for loop,

countries = {"Pakistan", "USA", "Japan", "Canada", "Japan"}

for name in countries:

print( name )

**len() function**

Returns the length

**type() function**

Returns the type

print( len( countries ) )

print( type( countries ) )

**The set() Constructor**

It is also possible to use the list() constructor to make a list.

countries = set(("Pakistan", "USA", "Japan", "Canada", "Japan"))

print( fruit\_list )

**Check if Item Exists**

if ( "USA" in countries ):

print("Yes USAis in the country list")

**Update, Add, Remove item from a set**

**Update**

Once a set is created, we cannot change its items, but we can add new items or remove any item.

**Add**

The add() method add an item to the set

countries = set(("Pakistan", "USA", "Japan", "Canada"))

countries.add("UK")

print( countries)

**Remove**

The remove() and discard() are the two methods to removes the specified item from the set..

countries.remove("USA")

print( countries )

The del keyword can also delete the set completely.

del countries

print( countries )

**Set Functions**

**add()**

**clear()**

**copy()**

**difference()**

**difference\_update()**

**discard()**

**intersection()**

**intersection\_update()**

**isdisjoint()**

**issubset()**

**issuperset()**

**pop()**

**remove()**

**symmetric\_difference()**

**ymmetric\_difference\_update()**

**union()**

**update()**

**Python Dictionaries**

**Dictionary**

Dictionaries are used to store data values in key:value pairs. We can create a Dictionary using curly brackets, and have keys and values:

person = {

"name": "Edward",

"country": "USA",

"birth\_year": 1975

}

Dictionary item can contain any type of data type like int, float, string, boolean, list tuple, set, dictionary etc

We can’t refer to an dictionary item by using an index.

List items are **ordered**, **changeable**, and **duplicate** values are not allowed.

## **Ordered**

Dictionary’s Items have a defined order, and that order will not change.

**Changeable**

Dictionaries are changeable, meaning that we can change, add or remove items after the it has been created.

**Duplicates Not Allowed**

Duplicate items are not allowed in any Dictionary

## **Access Items**

We can access the items of a dictionary by referring to its key name

person = {

"name": "Edward",

"country": "USA",

"birth\_year": 1975

}

print( person ["country"] )

#There is also a get method to access the items

print( person.get("country") )

**len() function**

Returns the length

**type() function**

Returns the type

print( len( person ) )

print( type( person ) )

**The dict() Constructor**

It is also possible to use the list() constructor to make a list.

c

print( person )

**Check if Item Exists**

if ( "country" in person ):

print("Yes country is in the dictionary")

**Get all Keys**

The keys() method will return a list of all the keys in the dictionary.

x = person.keys()

**Get all Values**

The keys() method will return a list of all the keys in the dictionary.

x = person.values()

The list of the values is a view of the dictionary, meaning that any changes done to the dictionary will be reflected in the values list.

x = person.values()

print(x) #before the change

person["country"] = "Pakistan"

print(x) #after the change

**Get Items**

The items() method will return each item in a dictionary, as tuples in a list.

x = thisdict.items()

**Update, Add, Remove item from a dic**

**Update**

person = {

"name": "Edward",

"country": "USA",

"birth\_year": 1975

}

person["name"] = "Jack"

print( person )

**Add**

person = {

"name": "Edward",

"country": "USA",

"birth\_year": 1975

}

person["industry"] = "Software"

print( person )

**Remove**

There are several methods to remove items from a dictionary

person.pop("country")

print( person )

del person["name"]

print( person)

The del keyword can also delete the list completely.

del person

print( person)

**Looping Through a List**

#x is the key

for x in person:

print( x )

#print all values

for x in person:

print( person[x] )

#x is the value

for x in person.values():

print( x )

#x is the key

for x in person.keys():

print( x )

#x is the key and y is the value

for x,y in person.items():

print( x )

print( y )

**Dictionary Functions**

**clear()**

Removes all the elements from the dictionary

person.clear()

**copy()**

another\_person = person.copy()

print( another\_person )

#We can also use dict() function to copy

third\_person = dict(person)

print( third\_person )

**update()**

The update() method will update the dictionary with the items from a given argument. If the item does not exist, the item will be added.

#below code will update the key country to Pakistan as key exists #already.

person.update({"country": "Pakistan"})

print(person)

below code will add the new key city as key doesn’t exist.

person.update({"city": "New Delhi"})

print(person)

**Dictionary Functions**

**clear()**

**copy()**

**fromkeys()**

**get()**

**items()**

**keys()**

**pop()**

**popitem()**

**setdefault()**

**update()**

**values()**

**Python Functions**

A function is a block of statements which we have to define.

Only defining function will do nothing, we have to call it to perform any action.

We may pass arguments when we call the function.

A function may or may not return the value

def fun():

print("Hello")

fun()

**Function with Arguments**

def fun( name ):

print("Hello ", name )

fun( "David" )

fun( "Rohan" )

fun( "Mike" )

**Numbers of arguments**

We can any number of arguments in the function definition.

def fun( name, age ):

print( f" Hello {name}, you age is {age}" )

fun( "David", 25 )

fun( "Rohan", 20 )

fun( "Mike", 40 )

def sum( a, b ):

s = a + b

print( f" Sum = {s}" )

sum( 10, 25 )

Functions should be called with the same numbers of arguments defined in the function defination.

Below code will give an error as sum function requires 2 arguments in order to call:

sum( 10 )

**Default Parameter Value**

We can set default value of any argument in the function definition and can call that function without argument.

def sum( a, b = 10 ):

s = a + b

print( f" Sum = {s}" )

sum( 20 )

sum( 10, 25 )

**Passing any data type as an Argument**

**def display\_fruits( food ):**

**for x in food:**

**print(x)**

**fruit\_list = [Apple", "Orange", "Banana"]**

**display\_fruits( fruit\_list )**

**Return Values**

Function may have a return value

def sum( a, b = 10 ):

s = a + b

return s

sum = sum( 20, 25 )

print(sum)

**Pass Statement**

def test():

pass

**Keyword Arguments**

You can also call the function with the *key* = *value* syntax.

def print\_numbers( a, b, c ):

print(a)

print(b)

print(c)

print\_numbers( c = 30, b = 20, a = 10 )

**Arbitrary Arguments, \*args**

If we dont know how many arguments, we need to pass at the time of function calling, we can add the \* before the parameter name in the function definition.

In this way, function will receive tuple of arguments and can access accordingly.

def test\_function( \*numbers ):

print(numbers)

my\_function(10, 20, 30, 40)

my\_function( 30, 40 )

**Arbitrary Keyword Arguments, \*\*kwargs**

If we dont know how many keyword arguments, we need to pass at the time of function calling, we can add the \*\* before the parameter name in the function definition.

In this way, function will receive dictionary of arguments and can access accordingly.

def test\_function( \*\*numbers ):

print(numbers)

my\_function( d = 10, a = 20, a = 30, b = 40)

my\_function( x = 30, y = 40 )

**Passing a any type of data as an Argument**

We can pass any type of data in the function calling and it will be treated as the same data type inside the function

**Function Recursion**

Python also support recusion in the function, recursion means that a function calls itself. One of the example of recursion in the solving factorial problem.

In the blow function, If we want to print “Hello World” 5 times.

def print\_rec( i ):

if i <= 5 :

print("Hello World!")

print\_rec( i+1 )

return

else:

return

print\_rec( 1 )

**Lambda Function**

It can take any number of arguments, but can only have one expression. It is also known as an anonymous function.

lambda arguments : expression

fun1 = lambda a, b : a \* b

print(fun1(5, 6))

Best use of lambda function is to use it inside another function how?

def lambda\_multiple ( n )

return lambda a : a \* n

x = lambda\_multiple ( 10 )

print( x (2) )

**Variable Scope**

Scope is the region where we can create and access any variable.

There are two types of scope in Python.

1. Local Scope
2. Global Scope

**Local Scope**

A variable created inside any function can be used inside that function.

def func\_scope():

x = 10

print(x)

func\_scope()

print(x) # Will print the error

**Global Scope**

A variable created outside any function can be used anywhere in the program.

x = 10

def func\_scope():

print(x)

func\_scope()

**Global Keyword**

If we try to change any global variable inside any function like in the below code, value will be changed only inside the function and will be remain the same outside the function.

x = 10

def func\_scope():

x = 20

print(x)

func\_scope()

print(x)

use the global keyword if you want to make any change to a global variable inside any function.

x = 10

def func\_scope():

global x

x = 20

print(x)

func\_scope()

print(x)

**Python Modules**

There are two types of modules in Python

1. Built-in Modules
2. User Defined Modules

**Built-in Modules**

There are sever al built-in modules in Python, To use any moudle, just need to iimport the module.

**The import Statement**

import math

x = math.sqrt(25)

print(x**)**

p = math.pi

print(p**)**

a = math.pow(5, 3)

print(a**)**

**The from...import Statement**

Instead of loading the full module, you can choose to import only parts from a module, by using the from keyword

#from math import sqrt

from math import sqrt, pow

x = sqrt(25)

print(x**)**

a = pow(5, 3)

print(a**)**

**Random Module**

import random

print(random.randint(10, 50))

**Import as alias**

By using the “as” keyword, You can create an alias when you import a module,

import math as m

x = m.sqrt(25)

print(x**)**

**dir() Function**

The dir() is the function which returns the list all the function and variable names in a module

import math

print(dir(math)**)**

**User-Defined Modules**

User-defined module is nothing but it’s a python file that may contain variables and functions.

Just create a file and save with “.py” extension

**my\_module.py**

def my\_func():

print("This is from my module")

my\_variable = "100"

Importing user-defined module is the same as import in built module.

**The reload() Function**

reload() function fromimportlib is used to import a previously imported module again

import math

import importlib

importlib.reload(math**)**

**Python Package**

A package is a library that contains modules and sub-packages.

**Package**

module1.py

module2.py

\_\_init\_\_.py

Need to import modules in \_\_init\_\_.py file

import module2

import module2

**What is PIP?**

PIP is a package manager

pip --version

pip install pip

**NumPy Package**

NumPy is a Python library used for working with arrays.

It stores the value of the same data type.

pip install numpy

In Python we have lists that serve the purpose of arrays, but they are slow to process.

pip

import numpy

arr = numpy.array([10, 20, 30, 40, 50])

print(arr)

print(arr[1])

print(arr[1:4])

print(arr.sum())

**Python Classes/Objects**

Python is an object-oriented programming language (OOPs)

**Create a Class**

class Person:

name= "Jon"

p1 = Person()

print( p1.name )

**Create a Class using Constructors**

class Person:

def \_\_init\_\_(self, age ):

self.age = age

p1 = Person(25)

print( p1.age )

p2 = Person(30)

print( p2.age )

**The \_\_init\_\_() Function**

The \_\_init\_\_() function is always executed when the class is being initiated and this is the function which assign values to object properties.

class Person:

def \_\_init\_\_(self, age ):

self.age = age

def display\_age(self):

print( " Age is :", self.age )

p1 = Person(25)

p1.display\_age()

p2 = Person(30)

p2.display\_age()

Here age is the class property and display\_age is the class method. A class might have any number of properties and methods.

**The pass Statement**

class Person:

pass

**Python Inheritance**

In Python, You can inherit all the methods and properties from a class into another class. This is called inheritance.

The parent class is the class being inherited from, also called the base class.

The child class is the class that inherits from another class, also called derived class.

class A:

def \_\_init\_\_(self, a):

self.a = a

def display\_a(self):

print( " a =", self.a )

class B (A):

def \_\_init\_\_(self, a, b):

super().\_\_init\_\_(a)

self.b = b

def display\_b(self):

print( " a =", self.a, " b =",self.b )

o1 = B(10,20)

print(o1.a)

print(o1.b)

o1.display\_a()

o1.display\_b()

**Types of Inheritecne**

B is inherited by class .

**1. Single inheritance**

**2. Multiple inheritance**

A class inherits from more than one class

**3. Multilevel inheritance**

Class C is derived from class B and class B itself is derived from class A

**4. Hierarchical inheritance**

Many classes are derived from a single class

**5. Hybrid inheritance**

Class B and Class C are derived from a single class A and class D is derived from class B and C.

**Difference between Public, Protected and Private**

**Public Members in Python**

Attributes are always public and can be accessed using the dot (.) within the class and outside the class using its object

class A:

def \_\_init\_\_(self, a):

self.a = a

def display(self):

print( " a =", self.a )

o1 = A(10)

print(o1.a)

o1.display()

**Protected Members in Python**

Protected method are created by using prefix \_ (single underscore)

class A:

def \_\_init\_\_(self, a):

self.\_a = a

def display(self):

print( " a =", self.\_a )

o1 = A(10)

print(o1.\_a)

o1.display()

**Private members in Python**

Private members are created by using prefix \_\_ (double underscore) and cannot be accessed directly via its object.

class A:

def \_\_init\_\_(self, a):

self.\_\_a = a

def display(self):

print( " a =", self.\_\_a )

class B(A):

def \_\_init\_\_(self, a):

super().\_\_init\_\_( a )

def display(self):

print( " a =", self.\_A\_\_a )

o1 = A(10)

print(o1.\_A\_\_a)

o1.display()

**Operator Overloading**

As we have already seen that we can use operators like +, -, <, > etc with data type like int, loat string etc. you can also use these operators with the class objects.

class A():

def \_\_init\_\_(self, n):

self.a = n

def \_\_add\_\_(self, x):

return self.a + x.a

a1 = A(10)

a2 = A(100)

print(a1+a2)

+ \_\_add\_\_(self, other)

– \_\_sub\_\_(self, other)

\* \_\_mul\_\_(self, other)

& \_\_and\_\_(self, other)

< \_\_lt\_\_(self, other)

> \_\_gt\_\_(self, other)

<= \_\_le\_\_(self, other)

>= \_\_ge\_\_(self, other)

== \_\_eq\_\_(self, other)

!= \_\_ne\_\_(self, other)

**Magic Methods**

In Python are the special methods that start and end with the double underscores. They are called automatically on a certain action.

\_\_init\_\_ method is invoked when an instance of a class is created.

\_\_str\_\_() method is invoked when we print any object of a class

class Test():

def \_\_init\_\_(self, n):

self.a = n

def \_\_str\_\_(self, x):

return self.a

t1= Test(10)

print(t1)

**\_\_main\_\_ and \_\_name\_\_ in Python**

\_\_name\_\_ is the special variable which value will be \_\_main\_\_ in the top level scope ( in the main file ), If it is import as a module, them \_\_name\_\_ value will not be \_\_main\_\_ but it will be the name of the file.

test.py

def my\_fun():

print("This is a call from function")

print("This is the debugging code")

my\_fun()

a.py

import test

print( "In a.py", \_\_name\_\_)

You can put condition to check name

test.py

def my\_fun():

print("This is a call from function")

print ( \_\_name\_\_ )

if \_\_name\_\_ == '\_\_main\_\_':

print("This is the debugging code")

my\_fun()

**Exceptions Handling**

Exceptions Handling is very important feature to handle the run time error.

Whenever code in try block generate an error, the except block will be executed.

try:

x = 10

Y = 0

z = x / y ssdd

print( z )

except:

print("An exception occurred")

try:

x = 10

Y = 0

z = x / y

print( z )

except Exception as e:

print("An exception occurred", e)

**try**

The try block lets you test a block of code for errors.

**except**

The except block lets you handle the error.

**else**

The else block lets you execute code when there is no error.

try:

x = 10

Y = 0

z = x / y

print( z )

except Exception as e:

print("An exception occurred", e)

else:

print("No Error in try block")

**finally**

The finally block lets you execute code, regardless of the result of the try- and except blocks.

try:

x = 10

Y = 0

z = x / y

print( z )

except Exception as e:

print("An exception occurred", e)

else:

print("No Error in try block")

finally:

print("This is the finally block")

**Type Of Exception**

ZeroDivisionError

NameError

**File Handling in Python**

You can perform several operation with file like creating, reading, updating, and deleting files.

f = open(<file-name>, <access-mode>, <buffering>)

**File Handling Access Mode  
"r" - Open an existing file for a read operation**

**"w" - Open an existing file for a write operation. If the file already contains some data then it will be overridden. creates the file if it does not exist**

**"a" - open an existing file for append operation. It won’t override existing data**

**"r+" - To read and write data into the file. The previous data in the file will not be deleted.**

**"w+" - To write and read data. It will override existing data.**

**"a+" - To append and read data from the file. It won’t override existing data, creates the file if it does not exist**

**Python File Open**

**f = open("testfile.txt", "r")**

**print(f.read())**

**print(f.read(5))**

**print(f.readline())**

**f.close()**

**The with statement**

**with open("testfile.txt", "r") as f:**

**print(f.read())**

**Write to a File**

**f = open("testfile2.txt", "a") #or f = open("testfile2.txt", "w")**

**f.write("This is the content to put in testfile2")**

**f.close()**

**Delete File**

import os

os.remove("testfile.txt")

**Python Mysql**

**Install Module**

**pip install mysql-connector-python**

**import mysql.connector**

**mydb = mysql.connector.connect(**

**host="localhost",**

**user="*yourusername*",**

**password="*yourpassword*",**

**database="mydatabase"**

**)**

**mycursor = mydb.cursor()**

**mycursor.execute("SELECT \* FROM customers")**

**myresult = mycursor.fetchall()**

**for x in myresult:**

**print(x)**

**sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"**

**val = ("John", "Highway 21")**

**mycursor.execute(sql, val)**

**mydb.commit()**

**NumPy Module**

It is a library used in scientific applications.

It is used for working with arrays ( numerical data )

NumPy is fast compared to List

NumPy requires less memory of data

It stores same type of data type

**How to install**

pip install numpy

**One dimensional array**

import numpy as nd

my\_array = nd.array([1,10, 20, 40, 50, 45, 34])

# Printthe array

print(my\_array)

# Find the type

print(type(my\_array))

#access elements

print(my\_array[1])

#find array data type

print(my\_array.dtype)

# Update array

my\_array[1] = 100

print(my\_array)

# Slicing

print(my\_array[1:4])

**Two dimensional array**

import numpy as nd

my\_array = nd.array([ [1,2,3,4,5] ,  [11,12, 13,14,15] , [21,22,23,24,25]])

# Printthe array

print(my\_array)

# Find the type

print(type(my\_array))

#access row

print(my\_array[1])

#access elements

print(my\_array[1,3])

#find array data type

print(my\_array.dtype)

print(my\_array.shape)

# Update array

my\_array[1,2] = 100

print(my\_array)

# Slicing

print(my\_array[1:3,2:4 ])

# Conditions in array

print(my\_array < 20)

print(my\_array < 20)

print(my\_array[my\_array < 20])

#reshape

print(my\_array.reshape(5,3))

#creating array with arange

arr2 = nd.arange(1, 100)

print(arr2)

print(arr2.reshape(11, 9))

#create array with ones

arr\_one = nd.ones( (3,5) )

print(arr\_one)

#create array with ones

arr\_zero = nd.zeros( (3,5) )

print(arr\_zero)

**Pandas**

Pandas is a Python library used for working with data sets

**Install Pandas**

pip install pandas

**Create Dataframe**

**#Create Dataframe**

**my\_file = pd.DataFrame( [[10, 20, 30],[100, 200, 300],[100, 200, 300]] )**

**print(my\_file)**

**#Create Dataframe with rows and column**

**my\_file = pd.DataFrame( [[10, 20, 30],[100, 200, 300],[100, 200, 300]],['Row1', 'Row2', 'Row3'], ['Col1','Col2', 'Col3']  )**

**print(my\_file)**

**#Create Dataframefrom Dictionary**

**d =  {**

**"person1" : {"name":"Qadir", "Salary":12000, "Profile": "SD1"},**

**"person2" : {"name":"Mahmood", "Salary":11000, "Profile": "SD2"},**

**"person3" : {"name":"Fazil", "Salary":14000, "Profile": "SD3"}**

**}**

**d\_file = pd.DataFrame( d )**

**print(d\_file)**

**#Create Dataframe using Numpy**

**dnum = pd.DataFrame( nd.arange(1, 51).reshape(10, 5), ['Row1', 'Row2', 'Row3', 'Row4', 'Row5', 'Row6', 'Row7', 'Row9', 'Row9', 'Row10'], ['Col1','Col2', 'Col3', 'Col4', 'Col5'] )**

**print(dnum)**

**#type**

**print( type(dnum))**

**#info**

**dnum.info()**

**#Top and bottom data**

**print(dnum.head())**

**print(dnum.tail())**

**#describe**

**print(dnum.describe())**

**#indexing**

**print(dnum['Col1']) #column**

**print(dnum[['Col1', 'Col2']]) #column #multiple column**

**print(dnum.loc['Row1']) #row**

**# Save Dataset as csv file**

**dnum.to\_csv('test.csv')**

#Pandas read CSS

import pandas as pd

df = pd.read\_csv('names.csv')

print(df)

print(df.describe())

**To read more about Pandas visit official website**

<https://pandas.pydata.org/>

**Common mistakes in python**

**Indentation Error**

**d = [1, 2, 3, 5, 6]**

**sum = 0**

**for x in d:**

**s = x\*x**

**sum += s**

**print(sum)**

**Naming Conflicts**

**len = 100**

**x = "Hello World!"**

**print(len(x))**

**#Same thing with module name like math etc**

**Mutable Default Args**

**def add\_names( name , name\_list = None):**

**if name\_list is None :**

**name\_list = []**

**name\_list.append(name)**

**print(name\_list)**

**names = ['Adnan', 'Tahir']**

**add\_names('Javed', names)**

**add\_names("Riyaz")**

**add\_names("Qasim")**

**Object Copy Problem**

**a = [10, 20, 30, 40, 50]**

**print(a)**

**print(b)**

**b[1]= 1000**

**print(b)**

**print(a)**