PYTHON

What is python

* Python is a highlevel language
* Case sensitive
* Easy to learn like syntax
* Interpreted language
* It is object overrented and functional programming
* Dynamic typing
* Python is independent flat form

**Syntax:**

* syntax can be executed by writing directly in the Command Line

**Variable:**

* Variable is used to stored the value in the memory

1. Variable should start with only alphabetis and ‘\_’
2. It should not contain white spaces
3. It should not start with Special characters and numerics

Ex: var123 = "Priya"

my\_var = "priya"

{123var = "priya", my-var = "priya", my var = "priya"} --- when will print the output it’s shown syntax error

**Collection Data Types:**

* **Numeric Type: Integer, float**
* **Integer**: **int**is the whole number including negative numbers but not fractions. In Python, there is no limit to how long an integer value.

Ex: x =10

print(x) 🡺 Output: (10)

type(x) 🡺 Output: int

* **Float**: Float is used to store floating-point numbers, which are numbers with a decimal point. These numbers can represent both positive and negative values.

Ex: x =1.5

print(x) 🡺 Output: 1.5

type(x) 🡺 Output : float

* **Sequence Type: string, list, Tuple**

**String**: string is a sequence of characters enclosed in either single quotes ('') or double quotes (“”).

Ex: s = "priya"

print(s) 🡺 Output: Priya

type(s) 🡺 Output: Str

**String Methods:**

len(): to know the number of characters in a string

Ex: var = “programming language”

len(var) 🡺 Output: 20

upper(): To make all the letters in the string capitals

Ex: var = “programming language”

var.upper() 🡺 Output: 'PROGRAMMING LANGUAGE'

lower(): To make all the latters in a string to lower case

Ex: var = “Programming Language”

var.lower() 🡺 Output: 'programming language'

Capitalize(): To make the starting letter in the entire string capital

Ex: var = “proGRamMing laNgUAge”

var.capitalize() 🡺 Output: Programming language'

title(): If we want to make the starting letter fo every word in a string to be in caps

Ex: var = "programming language"

var.title() 🡺 Output: 'Programming Language'

count(): To count the number of occurences of a sub-string in the main string

Ex: var = "programming language"

var.count(“g”) 🡺 Output: 4

endswith() : To check whether a string is ending in the given sub-string

Ex: var = "programming language"

var.endswith(“ge”) 🡺 Output: True

Startswith(): To check whether a string starts with a given sequence

Ex: var = "programming language"

var.startwith(“program”) 🡺 Output: True

find(): it returns the index position of the staring character of the sub-string

Ex: var = "programming language"

var.find(“language”) 🡺 Output: 13

index(): This method works similar to the find() method

Ex: var = "programming language"

var.find(“language”) 🡺 Output: 13

isdigit(): To check if the string contains numerics

Ex: var = "1234"

var.isdigit() 🡺 Output: True

Ex: var = "1234p"

var.isdigit() 🡺 Output: False

replace(): this method replaces all occurrences of a specified substring in a string and returns a new string without modifying the original string.

Ex: var = "programming language"

var.replace(“g”, “@”) 🡺 Output: pro@rammin@ lan@ua@e

Strip(): to remove any leading and trailing white spaces and new line characters

Ex: var = " programming language "

var.strip() 🡺 Output: ‘programming language’

Ex: var = "#### programming language ####"

var.strip(“#”) 🡺 Output: ‘ programming language ’

Split(): splitting a string into sub-string based on default delimter(space)

Ex: text = "python is a high level programming language"

text(text.split()) 🡺 Output: [‘python', ‘is’, ‘a’, ‘high’, ‘level’, ‘programming’, ‘language’]

join(): The join() method is used to concatenate elements of an inerrable into a single string.

Ex: var = [“Allow”, “indexing”, “and”, “slicing”]

sent = “ ”.join(var)

sent 🡺 Output: ‘Allow indexing and slicing’

**List:**

* list one of the collection data type
* List is a mutable datatype(modify, add, delete)
* Store different data type elements
* It allows duplicates
* Allow indexing and slicing

Methods:

* Append(): it is use to add the element at the end of the list

Ex: lst = [1, 2, 3, 5]

Lst.append(10)

Print(lst) 🡺 Output: [1, 2, 3, 5, 10]

* Extend(): it is use to add more than one element

Ex: l1 = [1, 2, 3, 5]

l2 = [6, 7]

l1.extend(l2)

Print(l1) 🡺 Output: [1, 2, 3, 5, 6, 7]

* Remove(): remove the element in the list

Ex: lst = [1, 2, 3, 4, 5]

lst.remove(3)

print(lst) 🡺 Output: [1, 2, 4, 5]

* Pop(): use to remove element at the end (if we pass index remove the element base on index).

Ex: lst = [1, 2, 3, 4, 5]

lst.pop()

print(lst) 🡺 Output: [1, 2, 3, 4]

* Index(): it is used to know the index value of the element

Ex: lst = ["Mango", "Apple", "Banana", "Kiwi"]

print(lst.index("Apple")) 🡺 Output: 1

Ex: lst = ["Mango", "Apple", "Banana", "Kiwi"]

print(lst[3]) 🡺 Output : Kiwi

* Count(): it is use to count the occurrence of elements in the list

Ex: lst = [1,2,3,2,4,5,2,6,7,2]

lst.count(2) 🡺 Output: 4

* Insert(): This method inserts the specified value at the specified position.

Ex: lst = ["Mango", "Apple", "Banana", "Kiwi"]

lst.insert(2, "Orange")

print(lst) 🡺Output: ['Mango', 'Apple', 'Orange', 'Banana', 'Kiwi']

* copy(): This method used to copy a list.

Ex: lst = ["Mango", "Apple", "Banana", "Kiwi"]

a = lst.copy()

print(a) 🡺Output: ['Mango', 'Apple', 'Banana', 'Kiwi']

* clear(): Remove all the elements from the list

Ex: lst = ["Mango", "Apple", "Banana", "Kiwi"]

lst.clear()

print(lst) 🡺Output: []

* reverse(): This method reverses the sorting order of the elements.

Ex: lst = ["Mango", "Apple", "Banana", "Kiwi"]

lst.reverse()

print(lst) 🡺Output: ['Kiwi', 'Banana', 'Apple', 'Mango']

* sort(): This method sorts the list ascending by default.

Ex: lst = ["Mango", "Apple", "Banana", "Kiwi"]

lst.sort()

print(lst) 🡺 Output: ['Apple', 'Banana', 'Kiwi', 'Mango']

**Tuple:**

* Tuple items are ordered, unchangeable, and allow duplicate values.
* Allow different type of elements
* Allow index And Slicing
* It is Immutable
* No method we can use built-in

**Tuple Operators:**

* Concatenation
* Interaction
* Membership operator
* Identify operator
* Repletion

**Tuple Methods:**

count(): This method returns the number of times a specified value appears in the tuple.

Ex: tup = (1,5,4,3,7,5,5,9)

x = tup.count(5)

print(x) 🡺Output: 3

Index(): This method returns the first occurrence of the given element from the tuple.

Ex: tup = (1,5,4,3,7,5,5,9)

x = tup.count(7)

print(x) 🡺Output: 4

* **Mapping Type: dict**

**Dictionary:**

* Dictionaries are used to store data values in key:value pairs.
* A dictionary is a collection which is ordered, changeable and do not allow duplicates.

**Methods for Dictionary:**

Pop(): pop method removes and returns an element from a dictionary having the given key.

Ex: employee = {"name":"john", "Id":"0085", "Blood Group":"AB+", "Age":23}

employee.pop("Blood Group")

employee 🡺 Output: {'name': 'john', 'Id': '0085', 'Age': 23}

Popitem(): this method removes and returns the element last inserted key-value pair from the dictionary.

Ex: employee = {"name":"john", "Id":"0085", "Blood Group":"AB+", "Age":23}

removed\_item = employee.popitem()

print(removed\_item)

employee 🡺Output: {'name': 'john', 'Id': '0085', 'Blood Group': 'AB+}

clear(): This method removes all the elements from a dictionary.

Ex: employee = {"name":"john",

"Id":"0085",

"Blood Group":"AB+",

"Age":23}

employee.clear()

employee 🡺{}

copy(): This method returns a copy of the specified dictionary.

fromkeys(): This method returns a dictionary with the specified keys and the specified value

Ex: keys = ["a", "b", "c"]

newkeys = dict.fromkeys(keys, 1)

print(newkeys) 🡺 Output: {'a': 1, 'b': 1, 'c': 1}

get(): This method returns the value of the item with the specified key.

Ex: employee = {"name":"john",

"Id":"0085",

"Blood Group":"AB+",

"Age":23}

name = employee.get("name")

print(name) 🡺 Output: john

update(): This method inserts the specified items to the dictionary.

Ex: dict1 = {"a":300, "b":350}

dict1.update(b=400, c=450)

print(dict1) 🡺 Output: {'a': 300, 'b': 400, 'c': 450}

del(): he del keyword removes the item with the specified key name

Ex: employee = {"name":"john",

"Id":"0085",

"Blood Group":"AB+",

"Age":23}

del employee['name']

employee 🡺 {'Id': '0085', 'Blood Group': 'AB+', 'Age': 23}

key(): Returns a view object that displays a list of all the keys in the dictionary.

Ex: employee = {"name":"john","Id":"0085","Blood Group":"AB+","Age":23}

keys = employee.keys()

print(keys) 🡺 dict\_keys(['name', 'Id', 'Blood Group', 'Age'])

value(): This method returns a view object, which means any changes made to the dictionary will be reflected in the view.

Ex: employee = {"name":"john","Id":"0085","Blood Group":"AB+","Age":23}

values = employee.values()

print(values) 🡺 Output: dict\_values(['john', '0085', 'AB+', 23])

items(): This method returns a view object. The view object contains the key-value pairs of the dictionary

Ex: employee = {"name":"john","Id":"0085","Blood Group":"AB+","Age":23}

items = employee.items()

print(items) 🡺 dict\_items([('name', 'john'), ('Id', '0085'), ('Blood Group', 'AB+'), ('Age', 23)])

* **Boolean: bool**
* Boolean means a result that can only have one of two possible values: true or false
* **Set Type: set**
* Do not allow duplicates
* No index and it is unordered
* It is immutable data type as a set elements

**Operators in Set:**

* Union
* Intersection
* Difference
* Issubset
* Issuperset

**Set Method:**

* Add(): Only one item to add in the list

Ex: my\_set = {40, -3, 9.0, 'Hi', True, 56}

my\_set.add(44)

my\_set 🡺 Output: {-3, 40, 44, 56, 9.0, 'Hi', True}

* Update(): we can add multiple elements using update

Ex: my\_set = {40, -3, 9.0, 'Hi', True, 56}

my\_set.update([67, -0.45, 9])

my\_set 🡺 Output: {-0.45, -3, 40, 56, 67, 9.0, 'Hi', True}

* Pop(): pop is used to delete element from the given list

Ex: my\_set = {40, -3, 9.0, 'Hi', True, 56}

my\_set.pop()

print(my\_set) 🡺 {'Hi', 40, 9.0, 56, -3}

* Remove(): deletes a specific element from a set.

Ex: my\_set = {40, -3, 9.0, 'Priyanka', True, 56}

my\_set.remove('Priyanka')

my\_set 🡺 Output: {-3, True, 9.0, 40, 56}

* Clear(): clear the all elements from the set, making it an empty set.

Ex: my\_set = {40, -3, 9.0, 'Hi', True, 56}

my\_set.clear()

print(my\_set) 🡺 Output: set()

* copy(): creates a new set with the same elements as the original.

Ex: my\_set = {40, -3, 9.0, 'Hi', True, 56}

new= my\_set.copy()

new 🡺 Output: {-3, 40, 56, 9.0, 'Hi', True}

* discord(): Discard the value from the set, if the value is not there in the set, discard will not throw any error.

Ex : my\_set = {40, -3, 9.0, 'Hi', True, 56}

my\_set.discard(88)

print(my\_set) 🡺 {True, 'Hi', 40, 9.0, 56, -3}

**Conditional Statements:**

* Conditional statements are used to execute certain blocks of code based on specific conditions.

**If Condition:**

* ifstatementis the most simple decision-making statement. If the condition evaluates to True, the block of code inside the if statement is executed.

Ex: a = 33

b = 20

if b < a:

print("b is less than a") 🡺 Output: b is less than a

**else Statement:**

* If the condition in the if statement is not true, the else block will be executed.

Ex: temperature = 25

if temperature > 30:

print("It's hot outside!")

else:

print("It's not too hot today.") 🡺 Output: It's not hot today.

[**Elif Statement**](https://www.codechef.com/blogs/if-else-conditions-in-python#the-elif-statement)**:**

* if the first condition is false, it moves on to the next “elif” statement to check if that condition is true.

Ex : x = 10

if x > 10:

Print(“x is greater than 10”)

elif x < 10:

Print(“x is less than 10”)

else:

print(“x is equal to 10”) 🡺Output: x is equal to 10

**Nested if statement:**

* if statements inside if statements, this is called nested if statements.

Ex: age = 35

if age >= 60:

print("You are a senior citizen.")

else:

if age >= 18:

print("You are an adult.")

else:

print("You are a teenager.") 🡺 Output: You are an adult.

**LOOPS:** Two primary types of loops: for loops and while loops.

**For Loop:**

## For loop executes the set of statements or block with repeatedly to every elements of the string/collation objects.

Ex: fruits = ["mango", "apple", "banana"]

for i in fruits:

print(i) 🡺Output: mango

apple

banana

**While Loop:**

* While loop executes the set of statements (or block repeatedly) until condition will become false

Ex: i = 1

while i < 6:

print(i)

i += 1

Output: 1

2

3

4

5

**Control Statements:**

**break**: Immediately terminates the loop.

Ex: i = 1

while i < 6:

print(i)

if i == 3:

break

i += 1

Output: 1

2

3

**continue**: Skips the rest of the current iteration and moves to the next one.