**Python**

**Python Introduction:**

🡪Python is a popular programming language. It was created by Guido Van Rossum in the year 1991.

🡪Python is a high-level, interpreted, and general-purpose programming language.

🡪Python emphasizes code readability and simplicity, which makes it an excellent language for beginners and experienced developers.

🡪It is used for:

* Web development
* Software development
* Mathematics,
* System scripting

**What can Python do?**

**🡪**Python can be used to create web applications.

🡪Python can connect to databases. It can also read and modify the files.

🡪Python can be used to handle big data and perform complex mathematics.

**Why Python ?**

🡪Python works on different platforms like Windows, Linux and Mac.

🡪Python has simple syntax. It is easy to understand.

🡪Python has beginner friendly libraries like random, os, re etc.

🡪Python error messages are easy to understand and debug.

**Key features in Python:**

* **Easy to Learn and Use:**Python’s simple and readable syntax makes it beginner-friendly.
* **Cross-Platform Compatibility:**Python runs seamlessly on Windows, macOS, and Linux.
* **Extensive Libraries:** Includes robust libraries for tasks like web development, data analysis, and machine learning.
* **Dynamic Typing:**Variable types are determined automatically at runtime, simplifying code writing.
* **Versatile:** Supports multiple programming paradigms, including object-oriented, functional, and procedural programming.
* **Open Source:** Python is free to use, distribute, and modify.

**Program in Python:**

print (‘Hello World!’) #Output: Hello World!

**Python Comments:**

🡪Comments can be used to explain the code.

🡪Comments can be used to make the code more reliable.

🡪Comments can be used to prevent execution when testing the code.

**Creating a comment:**

🡪Comments starts with a **#,** and python will ignore them.

🡪For the multiple line comments we use triple quotes **(””” “””)**.

**Python Variables:**

🡪Variables are containers for storing data values.

🡪Python has no command to create a variable.

🡪A variable is created the moment when you assign a value to it.

🡪Variable do not need to be declared with any particular type, and can even be changes after they have been set.

**Example:**

x=5

y=’Hi’

print(x) #result---5

print(y) #result---Hi

**Casting:**

If you want to specify the data type of a variable, this can be done with casting.

**Example:**

x = str(3)    # x will be '3'  
y = int(3)    # y will be 3  
z = float(3)  # z will be 3.0

**Get the type:**

You can get the data type of a variable with the **type()** function.

**Example:**

x = 5  
y = "John"  
print(type(x)) # int  
print(type(y)) # str

**Variable Names:**

🡪Variable names are case-sensitive.

🡪A variable name can be a short name or a descriptive name.

🡪Rules for Python variables:

* A variable name must start with a letter or the underscore character
* A variable name cannot start with a number
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_)
* Variable names are case-sensitive (age, Age and AGE are three different variables)
* A variable name cannot be any of the [Python keywords](https://www.w3schools.com/python/python_ref_keywords.asp).

**Assign Multiple Values:**

🡪Python allows you to assign multiple values to multiple variables in one line:

**Example:**

x, y, z = "Orange", "Banana", "Cherry"  
print(x) # ‘Orange’  
print(y) # ‘Banana’  
print(z) #’Cherry’

🡪Make sure the number of variables matches the number of values, or else you will get an error.

**Output Variables:**

🡪The Python **print( )** function is often used to output variables.

**Example:**

x = "Python is awesome"  
print(x) # Python is awesome

**Local Variables**

Local variables are declared inside a function and can only be accessed within that function.

**Example:**

def my\_function():

local\_var = "I am local"

print(local\_var)

my\_function() # I am local

**Global Variables:**

🡪Variables that are created outside the function are known as global variables.

🡪Global variables are used by everyone, both inside of functions and outside.

**Example-1:**

Create a variable outside the function, and use it inside the function.

x = "awesome"  
def myfunc():  
  print("Python is " + x)  
myfunc() # Python is awesome

**Example-2:**

Create a variable inside a function, with the same name as the global variable.

x = "awesome"  
def myfunc():  
  x = "fantastic"  
  print("Python is " + x) # Python is fantastic  
myfunc()  
print("Python is " + x) # Python is awesome

**The global keyword:**

🡪To create a global variable inside a function, you can use the **global** keyword.

**Example-1:**

If you use the **global** keyword, the variable belongs to the global scope:

def myfunc():  
  global x  
  x = "fantastic"  
myfunc()  
print("Python is " + x) # Python is fantastic

**Example-2:**

To change the value of a global variable inside a function, refer to the variable by using the global keyword:

x = "awesome"  
def myfunc():  
  global x  
  x = "fantastic"  
myfunc()  
print("Python is " + x) # Python is fantastic

**Data Types in Python**

Python provides various built-in data types to handle different types of data. These are categorized as follows:

**1. Numeric Types**

Used to store numerical values. There are 3 numeric types in python:

* **int**: Integer values (e.g., 10, -5)
* Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

**Example:**

x=1

print(type(x)) # int

* **float**: Floating-point numbers (e.g., 10.5, -3.14)
* Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

**Example:**

x=10.5

print(type(x)) # float

* **complex**: Complex numbers (e.g., 3+4j, -2-5j)
* Complex numbers are written with a "j" as the imaginary part:

**Example:**

x=3+5j

Print(type(x)) # complex

**2. Sequence Types**

Used to store ordered collections:

* **str**: String of characters (e.g., 'hello', "Python")
* Strings are represented in single quotes or double quotes.

**Example:**

Print(‘Hello’) # Hello

* **list**: It is a ordered collection (e.g., [1, 2, 3])
* List is Mutable. It means after creation of the list, we can perform any kind of operation on it.
* List is represented in square brackets [ ].

**Example:**

x=[1,2,3]

print(x) # [1,2,3]

* **tuple**: It is a ordered collection (e.g., (4, 5, 6))
* Tuple is Immutable. It means after creation of the list, we cannot perform any kind of operation on it.
* Tuple is represented in parenthesis ( ).

**Example:**

x=(1,2,3)

print(x) # (1,2,3)

**3. Mapping Type**

Used to store key-value pairs:

* **dict**: Dictionary (e.g., {'name': 'John', 'age': 30})
* It is a collection of key-value pairs.
* It is represented in curly braces { }.

**Example:**

x={'name': 'John', 'age': 30}

Print(x) # {'name': 'John', 'age': 30}

**4. Set Types**

Used to store unordered collections of unique elements:

* **set**:It is an unordered collection of elements(e.g., {1, 2, 3})
* It is represented in curly braces { }.
* Set does not allow duplicate elements.
* Set is Mutable, but the elements in the set are Immutable.

**Example:**

thisset = {"apple", "banana", "cherry"}  
print(thisset) # {‘apple’, ’banana’, ’cherry’)

* **frozenset**: Immutable version of a set (e.g., frozenset([1, 2, 3]))
* It is an unordered and unindexed collection of unique elements.
* It also does not allow duplicate elements.

**Example:**

mylist = ['apple', 'banana', 'cherry']  
x = frozenset(mylist) #frozenset({‘apple’, ’banana’, ’cherry’})

**5. Boolean Type**

Represents truth values:

* **bool**: Can be True or False.
* You can evaluate any expression in Python, and get one of two answers, True or False.
* When you compare two values, the expression is evaluated and Python returns the Boolean answer:

**Example:**

print(10 > 9) # True

print(10 == 9) # False

print(10 < 9) # True

**6. Binary Types**

Used to store binary data:

* **bytes**: Immutable sequence of bytes (e.g., b'hello')

**Example**:

x=b’Hello’

print(x) # b’Hello’

* **bytearray**: Mutable sequence of bytes (e.g., bytearray ([65, 66, 67]))

**Example**:

x=bytearray(5)

print(x) # bytearray(b'\x00\x00\x00\x00\x00')

* **memoryview**: Provides a view of binary data (e.g., memoryview(b'hello'))

**Example**:

x = memoryview(bytes(5))

print(x) # <memory at 0x00B08FA0>

**7. None Type**

Represents the absence of a value:

* **NoneType**: Only one value, None

These data types help in managing and processing data efficiently in Python.

**Operators in Python**

Python provides a rich set of operators to perform various operations on variables and values. These operators are categorized as follows:

**1. Arithmetic Operators**

These are used to perform mathematical operations:

* + (Addition): a + b
* - (Subtraction): a - b
* \* (Multiplication): a \* b
* / (Division): a / b
* // (Floor Division): a // b
* % (Modulus): a % b
* \*\* (Exponentiation): a \*\* b

**2. Comparison Operators**

Used to compare two values:

* == (Equal to): a == b
* != (Not equal to): a != b
* > (Greater than): a > b
* < (Less than): a < b
* >= (Greater than or equal to): a >= b
* <= (Less than or equal to): a <= b

**3. Logical Operators**

Used to perform logical operations:

* and: Returns True if both conditions are true.
* or: Returns True if at least one condition is true.
* not: Negates the truth value.

**4. Bitwise Operators**

Operate on binary numbers:

* & (AND): a & b
* | (OR): a | b
* ^ (XOR): a ^ b
* ~ (NOT): ~a
* << (Left Shift): a << b
* >> (Right Shift): a >> b

**5. Assignment Operators**

Used to assign values to variables:

* =: a = b
* +=: a += b
* -=: a -= b
* \*=: a \*= b
* /=: a /= b
* //=: a //= b
* %=: a %= b
* \*\*=: a \*\*= b

**6. Identity Operators**

Check if two objects are the same:

* is: a is b
* is not: a is not b

**7. Membership Operators**

Check for membership in sequences:

* in: a in b
* not in: a not in b

These operators help in performing various operations efficiently in Python.

**Control Statements and Loops in Python**

Python provides control statements and loops to manage the flow of execution in a program.

**1. Conditional Statements**

Conditional statements are used to execute different code blocks based on conditions.

**if Statement:**

Executes a block of code if the condition is True.

**Example:**

x = 10

if x > 5:

print("x is greater than 5") # x is greater than 5

**if-else Statement:**

Executes one block if the condition is True, otherwise executes another block.

**Example:**

x = 10

if x > 15:

print("x is greater than 15")

else:

print("x is not greater than 15") # x is not greater than 15

**if-elif-else Statement:**

Checks multiple conditions.

**Example:**

x = 10

if x > 20:

print("x is greater than 20")

elif x > 5:

print("x is greater than 5 but not more than 20")

else:

print("x is 5 or less") # x is greater than 5 but not more than 20

**Nested if Statement:**

**Example:**

An if statement inside another if statement.

x = 10

if x > 5:

if x < 15:

print("x is between 5 and 15") # x is between 5 and 15

**2. Loops**

Loops are used to execute a block of code multiple times.

**for Loop:**

Iterates over a sequence (list, tuple, string, etc.).

**Example:**

for i in range(5):

print(i) # 0 1 2 3 4

**while Loop:**

Repeats execution as long as the condition is True.

**Example:**

x = 0

while x < 5:

print(x)

x += 1 # 0 1 2 3 4

**3. Control Statements**

Control statements alter the flow of loops.

**break Statement:**

Terminates the loop completely.

**Example:**

for i in range(5):

if i == 3:

break

print(i) # 0 1 2

**continue Statement:**

Skips the current iteration and moves to the next one.

**Example:**

for i in range(5):

if i == 3:

continue

print(i) # 0 1 2 4

These statements and loops help in efficient flow control of Python programs.

**Dictionaries**

🡪Dictionaries are used to store the data values in key-value pairs.

🡪A dictionary is a ordered collection of elements, it does not allow duplicates. It is Mutable.

🡪Dictionaries are written with curly braces--{ }.

**Creating the dictionary:**

mydict={‘Name’ : ’Alice’, ‘Age’ : 30}

Print(mydict) # {‘Name’ : ’Alice’, ‘Age’ : 30}

**Ordered:** It means the items have a defined order, and that order will not change.

**Unordered:** It means the items do not have a defined order, you cannot refer to an item by using an index.

**Dictionary Length:**

* **len()** is used to determine how many items are present in the dictionary.
* The values in the dictionary items can be of any datatype such as String, int, and Boolean.
* **dict()** constructor is also used to make the dictionary.

thisdict=dict(‘name’ : ’Adam’, ‘Gender’ : ‘Male’, ‘Age’ : 60)

print(thisdict) # {‘name’ : ’Adam’, ‘Gender’ : ‘Male’, ‘Age’ : 60}

**Accessing Items:**

* You can access the items of a dictionary by referring to its key name, inside square brackets. It raises error, if the key is not present in the dictionary.
* **get()—**It also returns same output. But, if the item is not present the it does not give any output.
* **keys()—**It returns all the keys in the dictionary
* **values()—**It returns all the values in the dictionary.
* **items()—**It returns all the keys and values from the dictionary.
* We use **in** keyword to determine if a specified key is present in a dictionary or not.

**Changes values :**

* We can change the value of a specific item by referring to its key name.

**Example:**

thisdict = {  
   "brand": "Ford",  
   "model": "Mustang",  
   "year": 1964  
}  
thisdict[‘year’]=2018

Output:

# {‘brand’:’Ford’, ’model’:’Mustang’, ’year’:2018}

* **update()—**It will update the dictionary with items from the given argument.

**Example:**

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.update({"year": 2020})

Output:

# {‘brand’:’Ford’, ’model’:’Mustang’, ’year’:2020}

**Adding Items:**

* Adding an item to the dictionary is done by using a new index key and assigning a value to it:

**Example:**

thisdict = {

“brand” : “Ford”

“model” : “Mustang”

“year” : 1964

}

thisdict[“color”]=“red”

print(“thisdict”)

Output:

{‘brand’:’Ford’, ‘model’:’Mustang’, ’year’:1964, ’color’:’red’}

**Removing Items:**

* **pop()—**It removes the item with specified key name.
* **popitem()—**It removes the last inserted item.
* **del–** It removes the item with specified key name. It can also delete whole dictionary.
* **clear()—**This method empties the dictionary.

**Example :**

thisdict = {

“brand” : “Ford”

“model” : “Mustang”

“year” : 1964

}

thisdict.pop(“model”)

print(thisdict) # {'brand': 'Ford', 'year': 1964}

**Looping the dictionary :**We can loop the dictionary using **for** loop

**Copy Dictionaries :**

* You cannot copy a dictionary simply by typing dict1=dict2, because : dict2 will only be a reference to the dict1 and changes made in dict1 will automatically also be made in dict2.
* There are **2** ways to make a copy, one way is to use the built-in dictionary method **copy()**.----mydict = thisdict.copy()
* The another way to make a copy is to use the built-in function **dict()**.---mydict = dict(thisdict)

**Nested Dictionaries:**

* A dictionary can contain the dictionaries, this is called nested dictionaries.

**Dictionary Methods :**

* **clear()—**Removes all the elements from the dictionary.
* **copy()—**Returns a copy of the dictionary.
* **fromkeys()–** Returns a dictionary with the specified keys and value.
* **get()–** Returns the value of the specified key.
* **items()--** Returns a list containing a tuple for each key value pair.
* **keys()--** Returns a list containing the dictionary's keys.
* **values()--** Returns a list of all the values in the dictionary.
* **setdefault()--** Returns the value of the specified key. If the key does not exist: insert the key, with the specified value.

**Sets**

* Sets are used to store multiple items in a single variable.
* A set is an unordered, unchangeable collection of elements. It is Mutable, but the elements in the set are immutable.
* Set items are unchangeable, but we can remove items and add new items.
* Sets are written within curly brackets.

**Creating a Set:**

thisset = {"apple", "banana", "cherry"}  
print(thisset) # {‘apple’, ’banana’, ’cherry’}

* Set does not allow duplicate elements.

**Access Set items:**

* You cannot access items in a set by referring to an index or a key.
* But you can loop through the set items using a **for** loop,  or ask if a specified value is present in a set, by using the **in** keyword.

**Change Items:**

* Once a set is created, you cannot change the items, but you can add the new items.

**set() constructor:**

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

**Example:**

thisset = set(("apple", "banana", "cherry"))  
print(thisset) # {‘apple’, ’banana’, ’cherry’}

**Add Set Items:**

**Add Item:**

* To add one new item to a set use the **add()** method.

**Add Set:**

* To add items from another set into the current set, use the **update()** method.

**Add any iterable:**

* The object in the **update()** method does not have to be a set, it can be any iterable object (tuples, lists, dictionaries etc.).

**Example:**

thisset = {"apple", "banana", "cherry"}  
thisset.add("orange")  
print(thisset) # ‘apple’, ‘banana’, ‘cherry’, ‘orange’}

**Remove Set Items:**

**Remove Item :**

* To remove an item in a set, use the **remove()** , or the **discard()** method.
* If the item to remove does not exist, **remove()** will raise the error.
* If the item to remove does not exist, **discard()** will NOT raise the error.
* **pop()–** It is also used to remove an item, but this method will remove the random item, you cannot be sure what item gets removed.
* The value of the **pop()** method is the removed item.
* The **clear()** method empties the set.
* The **del** keyword will delete the set completely. It raises the error.

**Loop Items :**

* We can loop through the set items by using a **for** loop.

**Example:**

thisset = {"apple", "banana", "cherry"}  
for x in thisset:  
  print(x) # {"apple", "banana", "cherry"}

**Join Sets :**

* The **union()** and **update()** methods joins two or more sets in python.
* The **intersection()** method keeps only the duplicates.
* The **difference()** method keeps the items from the first set that are not in the other set(s).
* The **symmetric\_difference()** method keeps all items EXCEPT the duplicates.

**Union :**

* The **union()** method returns a new set with all items from both sets.

**Example :**

set1 = {"a", "b", "c"}  
set2 = {1, 2, 3}  
set3 = set1.union(set2)  
print(set3) **#** {‘a’, ‘b’, ‘c’, 1, 2 , 3}

**Intersection :**

* The **intersection()** method will return a new set, that only contains the items that are present in both sets.

**Example :**

set1 = {"apple", "banana", "cherry"}  
set2 = {"google", "microsoft", "apple"}  
set3 = set1.intersection(set2)  
print(set3) # {‘apple’}

**Difference :**

* The **difference()** method will return a new set that will contain only the items from the first set that are not present in the other set.

**Example :**

set1 = {"apple", "banana", "cherry"}  
set2 = {"google", "microsoft", "apple"}  
set3 = set1.difference(set2)  
print(set3) # {‘banana’, ‘cherry’}

**Symmetric\_differences :**

* The **symmetric\_difference()** method will keep only the elements that are NOT present in both sets.

**Example :**

set1 = {"apple", "banana", "cherry"}  
set2 = {"google", "microsoft", "apple"}  
set3 = set1.symmetric\_difference(set2)  
print(set3) # {‘banana’, ‘cherry’, ‘microsoft’, ‘google’}

**Set Methods:**

* **add()🡪**Adds an element to the set.
* **clear()🡪**Removes all the elements from the set.
* **copy()🡪**Returns a copy of set.
* **difference()🡪**Returns a set containing the difference between two or more sets.
* **discard()🡪**Remove the specified item.
* **intersection()🡪**Returns a set, that is the intersection of two other sets.
* **isdisjoint()🡪**Returns whether two sets have a intersection or not.
* **issubset()🡪**Returns whether another set contains this set or not.
* **pop()🡪**Removes an element from the set.
* **remove()🡪**Removes the specified element.
* **symmetric\_difference🡪**Returns a set with the symmetric differences of two sets.
* **union()🡪**Return a set containing the union of sets.
* **update()🡪**Update the set with the union of this set and others.

**frozenset :**

* The **frozenset** is an unordered and unindexed collection of unique elements.
* It is Immutable. It means we cannot perform any operation on the frozenset.
* As it is a set object, it does not have duplicate elements.
* Frozensets are represented by the built-in function which if **frozenset( )**.

**Example :**

mylist = ['apple', 'banana', 'cherry']  
x = frozenset(mylist)

Output:

frozenset({‘apple’, ’banana’, ’cherry’})