PYTHON TRAINING

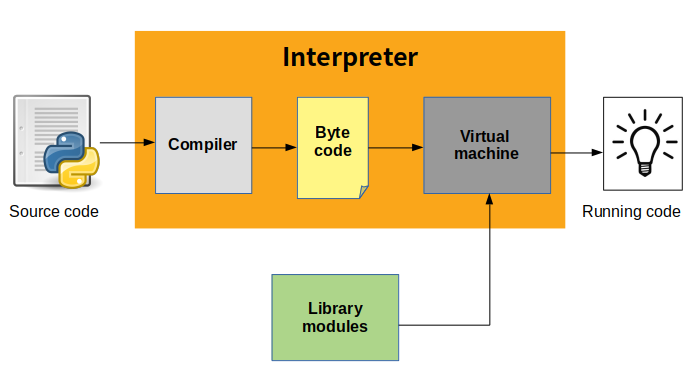
Name: KOPPULA GANESH

**Python:** Python is an object-oriented, high-level language, interpreted, interactive, dynamic and multipurpose programming language. It was authored by Guido Van Rossum in the year of 1991 later it was developed by the python organisation. It is a case sensitive in nature. Author wanted to name the programming language unique and short. Inspired by Monty Python’s Flying Circus, a BBC comedy series, he named it Python.



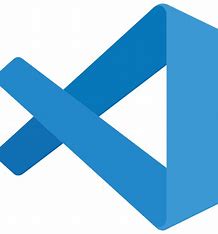
**For example:** print() is the right function and Print() is the wrong function to be used in the python programming.

**print(“Hello world!”)** here, we get an output>>> **Hello world!**

**Compiler:** A compiler is a software that converts the source code to the object code. In other words, we can say that it converts the high-level language to machine /binary language.

**Interpreter:** A program that can be analyse and execute a program line by line.

**IDE (Integrated Development Environment):** An IDE is a software for building applications that combines common developer tools into a single graphical user interface (GUI). Here I’m using Visual studio code.

**Visual studio code:** VS code is the source code editor which was developed by Microsoft for Windows, Linux and MacOS.

**Comments in python:** Comments are ignored by the compiler and interpreter. Comments are mainly used for the simple understanding of the complex code.

1. **Single line comment: # (hashtag)** is used for the single line comment.

#This is the single line comment..

1. **Multiline comment:** Triple quotes are used for the multiline comment.

'''This is the multiline comment,

which gonna be ignored by interpreter and compiler.'''

**Note:** To comment out multiple lines at once use ctrl + / (forward slash)

**Variable:** Variable is like a container which can store value/data or address of a value and later we can retrieve the data using the variable name. we can change the contents of a variable in a later statement.

**Example:** x = 5

In the above example x is **the** **variable name** and 5 is **the value**.

**Rules which to be followed for the variable:**

1. Don’t start the variable with digit or symbols.

Ex: 123 =1 or @123=2 (wrong)

Ex: abc=0 or abc12=3 (right)

1. Starting letters of the variable name should be alphabets or underscore.

Ex: \_abc=0

abc=0

print(abc)

#ouput>> 0

1. Python is a **case sensitive**.

Ex: Print() (wrong)

Ex: print() (right)

**Note:** for the print() function “p” should be used by lower case.

1. We cannot assign one value to multiple variables.

Ex: a,b,c=1 (wrong)

1. We can assign multiple values to multiple variables.

Ex: a,b,c=1,2,3 (right)

a,b,c=1,2,3,

print(a,b,c)

**Output:** 1 2 3

1. We can assign multiple values to single variable.

Ex: k=1,2,3

1. We can assign single value to multiple variables equally at once.

Ex: x=y=z=12

x=y=z=12

print(x)

print(y)

print(z)

**Output:**

12

12

12

**Note:** The interpreter uses keywords to recognize the structure of the program, and they cannot be used as variable names.

Ex:

class="India"

Here in the above example **class** is one of **the keywords in python**. So, it is illegal to use it as variable name. we get a syntax error over here **SyntaxError: invalid syntax**

**id Method:** id Method is used to find the memory location address of an object.

Ex: abc123=3

abc123=3

print(id(abc123))

Output: 140731005330280

**type() :** Type function is used to find the type of data types.

Ex:

a="Ganesh"

print(type(a))

**Output:** <class 'str'>

**len()** : Length function or method is used to determine number of characters in an object.

Ex1:

a="virat kohli"

print(len(a))

output>>> 11

In the above example we have only 10 letters in the string but we got an output as 11 why because “space” is also included in the length.

Ex2:

k=["virat","Dhoni"]

print(len(k))

output>>> 2

Here we get the output as **2** because these are two strings in the list.

**Data Types in python:**

1. **Integer (int):** Int, or integer, is a whole number, positive or negative, without decimals, of unlimited length.

Ex:

x = 1

y = 35656222554887711

z = -3255522

print(type(x))

print(type(y))

print(type(z))

**Output:**

<class 'int'>

<class 'int'>

<class 'int'>

1. **Float:** Float, or "floating point number" is a number, positive or negative, containing one or more decimals.

Ex:

x = 1.10

y = 1.0

z = -35.59

print(type(x))

print(type(y))

print(type(z))

**Output:**

<class 'float'>

<class 'float'>

<class 'float'>

1. **Complex:** Complex numbers are written with a "j" as the imaginary part.

Ex:

x = 3+5j

y = 5j

z = -5j

print(type(x))

print(type(y))

print(type(z))

**Output:**

<class 'complex'>

<class 'complex'>

<class 'complex'>

**Type Conversion:** We can convert from one data type to another with the int(), float() and complex() methods.

x=1

print(float(x))

Here, we get an **output** as **1.0** which is nothing but float.

Similarly, we can convert float into integer.

y=2.8

print(int(y))

Here, we get an output as **2** which is nothing but integer.

Similarly, we can convert integer to complex.

z=1

print(complex(z))

Here we get an output as **(1+0j)** which is complex.

**Implicit And Explicit Type Conversion:** Implicit and explicit datatype conversions in Python are two methods of converting one data type to another. Implicit conversion occurs automatically when Python needs to determine the appropriate data type for an operation or result. For instance, if you add an integer and a float, Python will implicitly convert the integer to a float and return a float as the result.

**Note:** Implicit type conversion has no data loss.

On the other hand, explicit datatype conversion happens when the programmer explicitly uses a function or a method to change the data type of a variable or a value. For example, if you want to convert a string to an integer, you can use the int() function.

**Note:** Explicit type conversion has data loss.

1. **String (str):** 
   * String is a collection of characters.
   * We can access the characters of string using square brackets [ ]
   * Each string has the length=1
   * String in python is immutable (we cannot modify a string)

**Types of declaring string:**

* Single quotes
  + - ex: ‘python’
* Double quotes
  + - ex: “python”
* Triple quotes
  + - ex: ‘‘‘python’’’

**String Methods:**

* **lower() :** By using lower() method we can convert the upper case string to lower case string.

ex:

Virat="CAPTAIN"

print(Virat.lower())

**Output:** captain

* **upper():** By using upper() method we can convert the lower case string to upper case string.

Ex:

jgkoppula="gmail"

print(jgkoppula.upper())

**Output:** GMAIL

* **endswith():** This method is used to check if a string ends with a specified suffix.

python="programming language"

print(python.endswith("language"))

**output:** True

* **startswith():** This method is used to check if a string starts with a specified prefix.

The startswith() method returns “True” if the string starts with the specified prefix and “False” otherwise.

python="programming language"

print(python.startswith("programming"))

**output:** True

* **replace():** This method is used to replace a specified substring with another substring in a given string. The replace() method takes two arguments: the substring to be replaced and the substring to replace it with.

sachin="God of Cricket"

print(sachin.replace("God","Lord"))

**Output:** Lord of Cricket

**Note:** The strings in python are immutable, so the replace() method doesn’t modify the original string but instead creates a new string with the replacements

* **split():** This method is used to split into a list of substrings.

fruits="apple grapes mango"

print(fruits.split())

**Output:** ['apple', 'grapes', 'mango']

* **count():** This method is used to count the number of occurrences of a specified element in a string.

BRS="Baratiya Rashtra Samithi"

print(BRS.count("a"))

#o/p: 6 ( 'a' repeated for 6 times)

* **Rstrip():** This method is used to remove the spaces characters from the right end of a string.

name3="Koppula Rajamani      "

print(len(name3))

output>>> 22

print(name3.rstrip())

output>>> Koppula Rajamani

j=name3.rstrip()

print(len(j))

output>>> 16

* **Lstrip():** This method is used to remove the leading spaces from a string. Its doesn’t modify the original string but returns a new string with the leading spaces removed.

name2="       Mettu Ramulu"

print(len(name2))

print(name2.lstrip())

#output>> “Mettu Ramulu”

p=name2.lstrip()

print(len(p))

#output>> 12

* **strip():** This method is used to remove the leading and trailing spaces from a string. It doesn’t modify the original string but returns a new string with the spaces removed.

name="     Koppula Ganesh   "

print(len(name)) #o/p 22

print(name.strip()) #o/p:Koppula Ganesh

v=name.strip()

print(len(v)) #o/p 14

* **removeprefix():**  This method is used to remove a specified prefix from a string.

python="van russom"

print(python.removeprefix("van"))

#we get an output as “russom”

* **removesuffix():** This method is used to remove a specified suffix from a string.

leaf="Ranapala Aku"

print(leaf.removesuffix("Aku"))

#we get an o/p as “Ranapala”

* **Index():** This method is used to find the index of the first occurrence of a specified element in a string.

python="programming language"

print(python.index("programming"))

#we get output as "0"

print(python.index("software"))

#we get output as an error

* **Find():** This method is used to locate the index of a substring within a string. It returns the lowest index of the substring if it is found, and -1 if the substring is not present.

python="programming language"

print(python.find("programming"))

#we get o/p as "zero"

print(python.find("ganesh"))

#we get o/p as "-1"

**Mutable &Immutable datatypes:**

* Mutable datatypes in which we can be modify or change the elements in it. Immutable are cannot be changed.
* Mutable include **list [], dictionary {}, sets {}**
* Immutable include **number, string, tuple ()**

**Control statements:** Control statements are used to control the flow of the program. These statements are also known as decision making statements.

1. **Conditional statements:** Conditional statements in Python are used to make decisions in the code based on certain conditions. They allow us to execute different blocks of code depending on whether a specified condition evaluates to **True or False.**

**Ex: if-elif-else statements**

age=17

if age>18:

print("YOU CAN VOTE SIR")

elif age==18:

print("YOU CAN VOTE MY FRIEND")

else:

print("PLEASE WAIT")

output>> PLEASE WAIT

**Note: “else”** do not have any condition.

1. **Looping statements:** Looping (iteration) statements in Python are used to repeatedly execute a block of code. They allow you to automate **repetitive tasks** by instructing the computer to perform a set of instructions multiple times. In simple terms, loops are like a recipe that tells the computer to do something over and over again until a specific condition is met.

There are two main types of loops in Python: **for loops** and **while loops**.

* **For loop:** For loop is for **element loop.**

#for:- for loop is for element loop mentioned.

a=[20,21,22,23] #"[]" represents "list".

for i in a:

print(i)

Output>>>

20

21

22

23

#range:-

for i in range (0,5):

#In range of zero to five

    print(i)

Output>>>

0

1

2

3

4

for i in range (0,10,2): #'2' represents even numbers.

    print(i)

Output>>>

0

2

4

6

8

#In the o/p we get only even numbers in the range of zero to ten.

#Note: In for loop only we use "range"

* **While loop:** While loop is the infinity loop of an element.

#while\_loop:- 'while loop' is an infinite loop of the object.

#ex:-

K=10

while 21<20:

    print("Hello")

#There is no output because K=21 value is not less than 20

#Note: In 'while loop' we don’t use "range"

T=20

while T<40:

    print("Hi")

    T+=1 #'assignment operator' used to stop the infinite loop.

* **Nested loop:** Nested loop nothing but loop within loop.

#nested loop (for loop):- loop within loop

for i in range(0,10):

    for j in range(0,20):

        print(i+j)

1. **Jumping statements:** In Python, "jumping statements" generally refer to control flow statements that affect the flow of a program, allowing it to jump to a different section of code or skip certain parts. There are three main jumping statements in Python: break, continue, and pass.

* **Pass:** The pass statement is a no-operation statement.

if True:

    pass #pass

* **Break:** The break statement is used to exit a loop prematurely.

a="pythonworld"

for i in a:

    if i=="w":

        break  #break

    print(i)

output>>>

p

y

t

h

o

n

* **Continue:** The continue statement is used to skip the rest of the code inside a loop for the current iteration and move to the next iteration. It does not terminate the entire loop but rather skips to the next iteration.

b="Ganesh koppula"

for i in b:

    if i==" ":

        continue #continue

    print(i)

output>>>

G

a

n

e

s

h

k

o

p

p

u

l

a

**Operators and Operands:** Operators are particular symbols which operate on some values and produce an output. The values are known as Operands.

Example: **print(4 + 5)**

Here 4 and 5 are Operands and (+), (=) signs are the operators. **We get the output 9**

**Python supports the following operators:**

* **Arithmetic Operators:**

|  |  |
| --- | --- |
| **Operators** | **Description** |
| **//** | Perform Floor division (gives integer value after division) |
| + | To perform addition |
| \* | To perform multiplication |
| - | To perform subtraction |
| / | To perform division |
| % | To return remainder after division (Modulus) |
| \*\* | To perform exponent (raise to the power) |

print(4+5) #addition

o/p>> 9

print(2\*\*3) #exponent

o/p>> 8

print(4/2) #division

o/p>> 2.0

print(5//2) #floor division

o/p>> 2

print(5%2) #modulus

o/p>> 1

print(6-2) #subtraction

o/p>> 4

* **Relational operators:**

|  |  |
| --- | --- |
| **Operators** | **Description** |
| < | Less than |
| > | Greater than |
| <= | Less than or equal to |
| >= | Greater than or equal to |
| == | Equal to |
| != | Not equal to |

print(2>1) o/p>> True

print(2<1) o/p>> False

print(2>=1) o/p>> True

print(2<=1) o/p>> False

print(2==2) o/p>> True

print(2!=3) o/p>> True

* **Logical operators:**

|  |  |
| --- | --- |
| **Operators** | **Description** |
| and | Logical AND (When both conditions are true output will be true) |
| or | Logical OR (If any one condition is true output will be true |
| not | Logical NOT (Compliment the condition i.e., reverse) |

a=5>4 and 3>2

print(a)

o/p>> True

b=5>4 or 3<2

print(b)

o/p>> True

c=not(5>4)

print(c)

o/p>> False

* **Membership operators:**

|  |  |
| --- | --- |
| **Operators** | **Description** |
| in | Returns true if a variable is in sequence of another variable, else false |
| not in | Returns true if a variable is not in sequence of another variable, else false |

a="Ganesh"

b= "Gaddar"

names=["Ganesh","Ramu","seetha","Raju"]

if (a in names):

    print("Yeah we found Ganesh")

else:

    print("Sorry we not found Ganesh")

if (b not in names):

    print("Gaddar not found")

else:

    print("We dont care..!")

o/p>>

Yeah we found Ganesh

Gaddar not found

* **Identity operators:**

|  |  |
| --- | --- |
| **Operators** | **Description** |
| is | Returns true if identity of two operands are same, else false. |
| is not | Returns true if identity of two operands are not same, else false. |

a=20

b=20

if (a is b):

    print ("a, b has same identity")

else:

    print ("a, b is different")

b=10

if (a is not b):

    print ("a, b has different identity")

else:

    print ("a, b has same identity")

o/p>>

a, b has same identity

a, b has different identity

**Mini Calculator using python:**

first=input ("Enter first number: ")

operator= input("mathematical operator (+,-,\*,/,%,\*\*,%): ")

second=input("Enter second number: ")

first= int(first)

second=int(second)

if operator=="+":

    print(first + second)

elif operator=="-":

    print(first - second)

elif operator=="\*":

    print(first \* second)

elif operator=="/":

    print(first / second)

elif operator=="%":

    print(first % second)

elif operator=="\*\*":

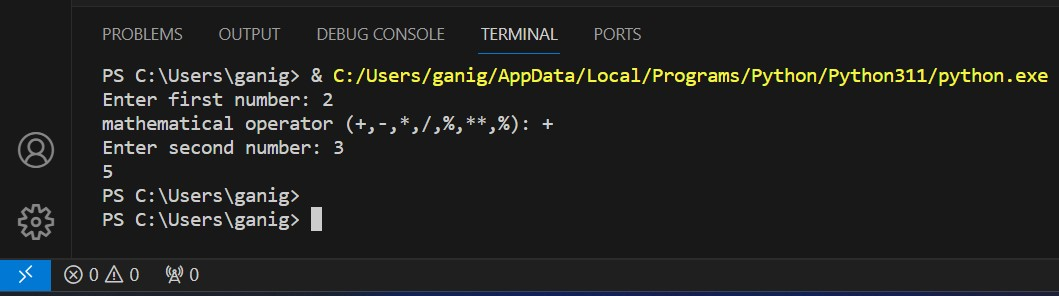
    print(first \*\* second)

elif operator=="%":

    print(first % second)

else:

    print("Invalid operator")

**output>> 2 + 3 = 5**

* **List:** List is a sequence of values. The values in a list are called elements or items. We use **square brackets “[ ]”** for list. It is a mutable data type where we can change the values later. We can store different types of elements in it. It also allows duplicates in the same list and indexing.

x=[]

print(type(x))

output>>> <class 'list'>

x=[1,2,3,4,5] #a list of 5 integers

y=["Ganesh","Ramesh","Suresh","Rajesh"]

#a list of strings

z=["Raju",2,2.76,[10,20]]

#a list within another list is called "nested"

p=[] #a list contains no elements is known as empty list.

Lists are **mutable** because you can change the order of items in a list or reassign an item in a list.

numbers=[2,4,6,8,10,12,14,16,18]

numbers[0]=3

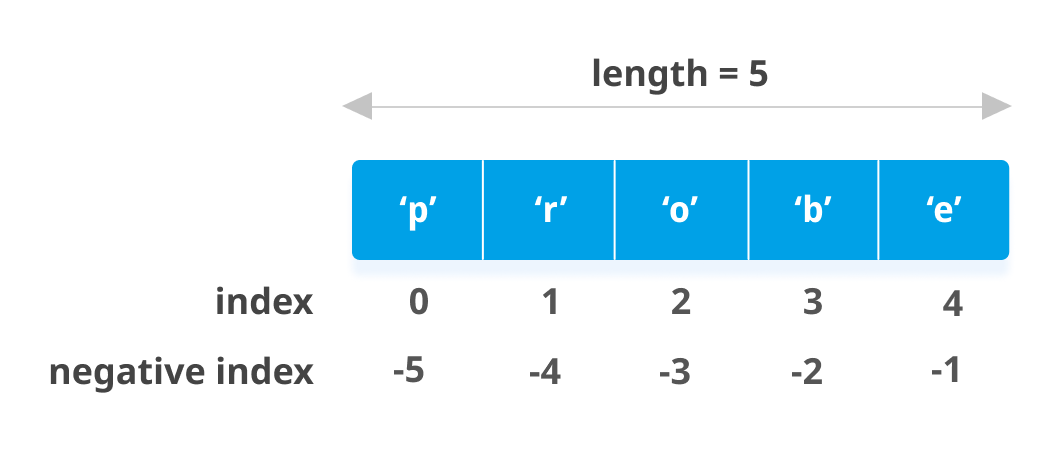
print(numbers)

output>>> [3, 4, 6, 8, 10, 12, 14, 16, 18]

here we have changed the index “0” element

“2” to “3”

Accessing the index element. The indexing always starts with **“0”** in **“forward”** and starts with **“-1”** from **“backward”**



rollno=[20,30,40,50,60,70,80,90]

print(rollno[-1]) #backward or negative indexing

output>>> 90

rollno=[20,30,40,50,60,70,80,90]

print(rollno[0])  #forward or positive indexing

output>>> 20

**Note**: The **in** operator also works on list.

Actors=["Prabhas","Samantha","Trisha"]

print("Trisha" in Actors) #in operator

output>>> True #Boolean

print("Salman" in Actors)

output>>> False #Boolean

Although a list can contain another list, the nested list still counts as a single element.

z=["Raju",2,2.76,[10,20]]

print(len(z))

output>>> 4

Here, we have “4” elements/items in a list.

Some of the **list operations:**

* **“ + ” Operator concatenates (adding of list) list.**

p=[1,2,3,4,5]

q=[6,7,8,9,10]

print(p+q)

output>>> [1,2,3,4,5,6,7,8,9,10]

* **“ \* ” Operator repeats the list a given number of times.**

r=[1]

print(r\*8)

output>>> [1,1,1,1,1,1,1,1]

Here, the element in the list got repeated for 8 times.

* **List Slicing:** This operator used to **slice** a particular **range** of a list or a sequence. Slice is used to retrieve a subset of values.

**Syntax:** list1[start:stop:step]

n=[1,2,3,4,5,6,7,8,9,10]

print(n[0:4])

output>>> [1, 2, 3, 4]

print(n[:])

output>>> [1,2,3,4,5,6,7,8,9,10]

print(n[:9])

output>>> [1,2,3,4,5,6,7,8,9]

print(n[0:])

output>>> [1,2,3,4,5,6,7,8,9,10]

Here we can also change/modify the elements in the list using the slicing method.

k=[3,6,9,12,15,18,21,24,27]

k[0:5]=[1,2,3,4,5]

output>>> [1, 2, 3, 4, 5, 18, 21, 24, 27]

h=[1,2,3,4,5,6,7,8,9,10]

print(h[0:9:2])

output>>> [1,3,5,7,9]

* **List Methods:**
  1. **append( ):** Append adds a new element to the end of a list.

name=["ganesh","rajesh","suresh","ramesh","kamesh"]

name.append("mallesh")

print(name)

output>>> ['ganesh', 'rajesh', 'suresh', 'ramesh', 'kamesh', 'mallesh']

* 1. **extend( ):** extend() method add items of an iterable (list, tuple, dictionary, etc) at the end of list.

**Syntax:** t1.extend(t2)

o=["ganesh","ramesh"]

k=[1,2,3,4]

o.extend(k)

print(o)

output>>> ['ganesh', 'ramesh', 1, 2, 3, 4]

* 1. **sort( ):** sort is used to arrange the elements in order of (low to high).

p=[9,8,7,6,5]

p.sort()

print(p)

output>>> [5,6,7,8,9]

* 1. **pop( ):** pop is used to remove/delete the element in the list just by specifying the element’s index. If we don’t specify any index then the last element will be deleted by default.

j=[5,10,15,20,25]

j.pop() # we didn’t mentioned the index

print(j)

output>>> [5,10,15,20]

f=[7,14,21,28,35,42,49]

f.pop(-2) #here we did mentioned the index

print(f)

output>>> [7, 14, 21, 28, 35, 49]

* 1. **remove( ):** Using the remove( ) method we can remove the particular element in the list. We just need specify the element we wanted to remove. If we know the element we want to remove (but not the index), we can use remove( )

l=[4,8,12,16,20,24,28]

l.remove(20)

print(l)

output>>> [4, 8, 12, 16, 24, 28]

* 1. **insert( ):** insert( ) is used to insert the element into the list by specifying “index” and “element”

d=[6,12,18,24,30,36]

d.insert(0,42) #0 is the index and 42 is the element wanted to be inserted.

print(d)

output>>> [42, 6, 12, 18, 24, 30, 36]

* 1. **index ( ):** To find the index of the certain element in the list.

flowers=["jasmine","lotus","sun flower","rose"]

print(flowers.index("jasmine"))

output>>> 0

* 1. **count( ):** By using the count( ) method we can find how many times the particular element got repeated.

numbers=[1,2,3,2,1,1,1,2,2,3,]

numbers.count(1)

print(numbers.count(1))

output>>> 4 # “1” repeated for 4 times.

**List Functions:**

* + - * **maximum, minimum, length, sum**

r=[1,2,3,4,5,6,7,8,9,10]

print(max(r)) #we get output which is maximum element in the list.

output>>> 10

print(min(r)) #we get an output which is minimum element in the list.

output>>> 1

print(len(r)) #we get an output of number of elements in the list.

Output>>> 10

print(sum(r)) # we get an output of sum of elements in the list.

Output>>> 55

**Note: Sum( )** function works only when the elements in the list are **numbers**

**Converting string to list:**

* + - 1. **converting string to list using list( ) built-in function:**

x="Lenovo"

print(list(x))

output>>> ['L', 'e', 'n', 'o', 'v', 'o']

Here, the **list( )** function breaks the string into **individual letters**

* + - 1. **Converting string to list using split( ) method:** If we want to break the string into words we can use split( ) method.

b="My name is ganesh"

print(b.split())

output>>> ['My', 'name', 'is', 'ganesh']

c="class-class-class"

delimiter= "-"

print(c.split(delimiter)) #delimiter is used as a boundary to set for the words where they have to split.

output>>> ['class', 'class', 'class']

**Objects and values:**

We use **objects** and **values** interchangeably, but we can say that an object has a value.

a="doctor"

b="doctor"

a is b

print(a is b)

output>>> True

a="doctor"

b="doctor"

print(id(a))

print(id(b))

output>>>

1136835292400

1136835292400

Here, we can see that both the variables a and b have the same value stored in same memory location.

In this example, Python only created **one string object**, and **both a and b refer to it.**

But when we create two lists, we gonna get two objects:

a=[4,5,6,7,8]

b=[4,5,6,7,8]

print(id(a))

print(id(b))

output>>>

1136835294272

1136835430208

Here, we can see that both the variables a and b have the equivalent values but are not identical so they are seeming to be different objects and stored at different memory location.

a=[1,2,3,4]

b=[1,2,3,4]

a is b

print(a is b)

output>>> False

So, a and b has different objects but seems to be equivalent.

**Aliasing:** A circumstance where two or more variables refer to the same object.

a=[2,4,6,8,10]

b=a

print(a is b)

output>>> True

If aliased object is mutable then the changes made with one alias affect the other.

a[0]=22

print(b)

output>>> [22, 4, 6, 8, 10]

* + - * **Tuple:** A tuple is a sequence of values much like a list. The values stored in a tuple can be any type (integer, float, list, string, etc.) and they are indexed by integers. The important difference is that tuples are **immutable** (we cannot modify the elements in the tuple). A tuple is created by the placing all the elements inside **parentheses '()'**, separated by commas.Tuple is a comma-separated list of values. It can also allow duplicates.

a=(1,2,3,4,5.6,"ganeshkoppula",True)

print(type(a))

output>>> <class 'tuple'>

1. **Indexing:** We can use **the index operator []** to access an item in a tuple, where **the index starts from 0**. So, a tuple having 6 elements will have indices from 0 to 5. Trying to access an index outside of the tuple index range(6,7,... in this example) will raise an IndexError.

x=(1,2,3,4,5,6)

print(x[0]) #output: 1

print(x[1]) #output: 2

print(x[2]) #output: 3

print(x[3]) #output: 4

print(x[4]) #output: 5

print(x[5]) #output: 6

let’s try with the index number which is out of range. Which gives us IndexError.

print(x[6]) #output: IndexError: tuple index out of range

The index must be an integer, so we cannot use float or other types.This will result in TypeError.

print(x[1.0]) #output: TypeError: tuple indices must be integers or slices, not float

**Nested tuples** are accessed using nested indexing.

y=("ganesh",1,3.0,[123,23,4])

print(y[3][0]) #accessing the nested tuple.

#output: 123

1. **Negative Indexing:** The index of **-1 refers to the last item**, **-2 to the second last item and so on.**

z=(1,2,3,4,5,6)

print(z[-1]) #output: 6

print(z[-2]) #output: 5

print(z[-3]) #output: 4

print(z[-4]) #output: 3

print(z[-5]) #output: 2

print(z[-6]) #output: 1

1. **Slicing:** We can access the range of items from the tuple by using the **slicing operator colon “ : ”**

v=('p','q','r','s')

print(v[0:4]) #output: ('p', 'q', 'r', 's')

print(v[:]) #begining to end of elements

output : ('p', 'q', 'r', 's')

Although it is not necessary, it is common to enclose tuples in parentheses to help us quickly identify tuples when we look at Python code.

n=1,2,3,4,5 #no parentheses

print(type(n))

#output: <class 'tuple'>

To create a tuple with a single element, you have to include the **final comma.**

n=1,

print(type(n))

#output: <class ‘tuple’>

p=("a",) #with comma

print(type(p))

#output: <class ‘Tuple’>

p=("a") #without comma

print(type(p))

#output: <class ‘str’> #string

1. **Concatenation:** concatenation is **the act of joining**. We can join two tuples using the concatenation operator ‘+’.

#Concatenation: To add to tuples...

t1=(1,2,3,4,5)

t2=(6,7,8,9,0)

print(t1+t2)

#o/p:(1, 2, 3, 4, 5, 6, 7, 8, 9, 0)

1. **Membership:** We can apply the **‘in’** and **‘not in’** operator on the items. This tells us whether they belong to the tuple.

x=(1,2,3,4,5,6)

print(1 in x) #output: True

print(7 not in x) #output: True

print(0 in x) #output: False

1. **Tuple( ) method:**

#Using the tuple() method to make a tuple:

c=tuple(("ganesh","komal","suresh"))

# note the double round-brackets

print(c)

print(type(c))

1. **Identity operators:**

d1=(1,2,3,4,5,6,7)

d2=(23,34,56,78,90)

print(d1 is d2) #False

print(d1 is not d2) #True

1. **Repetition:** Python tuple repetition operator (\*) is used to the repeat a tuple, number of times which is given by the integer value and create a new tuple value.

#repetition:

a1=(1,2)

print(a1\*5)

#o/p: (1, 2, 1, 2, 1, 2, 1, 2, 1, 2)

1. **max( ), min( ), sum( ), len( ) functions:**

a=(1,2,3,4,5,6,7,8,9,0)

print(max(a)) #maximum element

output: 9

print(min(a)) #minimum element

output:0

print(sum(a)) #addition of elements

output:45

print(len(a))

#output: 10

#To determine how many items a tuple has, use the len() function