1. **Variables**

* Python variables are the reserved memory locations used to store values with in a Python Program.
* This means that when you create a variable you reserve some space in the memory.
* Based on the data type of a variable, Python interpreter allocates memory and decides what can be stored in the reserved memory.
* Therefore, by assigning different data types to Python variables, you can store integers, decimals or characters in these variables.
* A Python variable is created automatically when you assign a value to it

1. **Python Variables - Naming Convention**

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

e.g var=”khan” or \_var=”khan”

* A variable name cannot start with a number or any special character like $, (, \* % etc.
* A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and \_ )
* Python variable names are case-sensitive which means Name and NAME are two different variables in Python.
* Python reserved keywords cannot be used naming the variable.

Why special characters are not allowed in a variable name?

* Fundamentally it is because they're mostly used as operators or separators, so it would introduce ambiguity

naming patterns

* **Camel case** − First letter is a lowercase, but first letter of each subsequent word is in uppercase. For example: kmPerHour, pricePerLitre
* **Pascal case** − First letter of each word is in uppercase. For example: KmPerHour, PricePerLitre
* **Snake case** − Use single underscore (\_) character to separate words. For example: km\_per\_hour, price\_per\_litre

Examples

#Variable

#when value is stored in memory fetching the data from its ID

str1="Shahriyar"

#print(str1)

print(id(str1))

#find the type of this varaible

print(type(str1))

#creates integer variable

a=12

#create string variable

b="Shahriyar Khan"

#create float variable

c=12.3

print(a,b,c)

#find

print(type(a),type(b),type(c))

#delete python variable

del a

#deleted show nameError

#print(a)

#multiple variable in one statement

var1=var2=var3=1

var4,var5,var6=5,14,10

#multiple values print in one statments

print(var1,var2,var3,var4,var5,var6)

#using naming convention

#1.start from letter or underscore

\_\_var1=12

print(\_\_var1)

#invalid

#%var1,1var,keywords if else etc

#using naming pattrens

varOne,VarOne,var\_one="CamelCase","PascalCase","Snackcase"

#operator is returns True if both the operands have same id() value.

var1 is var2 is var3

print(id(var1),id(var2),id)

Python Data Types

Python Data Types are used to define the type of a variable. It defines what type of data we are going to store in a variable. The data stored in memory can be of many types. For example, a person's age is stored as a numeric value and his or her address is stored as alphanumeric characters.

Python has various built-in data types which we will discuss with in this tutorial:

* Numeric - int, float, complex
* String - str
* Sequence - list, tuple, range
* Binary - bytes, bytearray, memoryview
* Mapping - dict
* Boolean - bool
* Set - set, frozenset
* None - NoneType

## Numeric Data Type

* int (signed integers)
* bool (subtype of integers.)
* float (floating point real values)
* complex (complex numbers)

#1 Numeric datatypes

var=1  #int(signed integers)

var2=2.3 #float(floating point real values)

var3=3+4j #complix(complex numbers)

var4=True #bool (subtype of integers)

## Sequence Data Type

* String Data Type
* List Data Type
* Tuple Data Type
* **Ranges Function**
* Python **range()** is an in-built function in Python which returns a sequence of numbers starting from 0 and increments to 1 until it reaches a specified number.
* We use **range()** function with for and while loop to generate a sequence of numbers
* range(start, stop, step)

# 2 Sequence datatype

# 2.1 string datatypes

#a string in python is in object of str class

str1="khan"

print(str1)#print complete

print(str1[1])#print second charactor

print(str1[0:3])#print from 0(first ch) to 3(3 exclude)

print(str1[1:])#from 1 to last

print(str1\*2)#prints string two time

print(str1+str(1))#concatinated two string

# 2.2 list datatypes...enclused [],elements and size can be changed,mutable

#collection of hetroginous data()

list=[1,"Shahriyar Khan",3.12,4+6j,1.23e-4]

#nested list

nestedList=[["one","two","three"],[1,2,3],[1.2,1.3,1.4]]

print(list)

print(nestedList)

# 2.3 tuple datatype ...enclused(),immutable(cannot be update),read only

tuple=(1,"Shahriyar Khan",3.12,4+6j,1.23e-4)

print(tuple)

#tuple[0]=2 invalid in tuple

#list[0]=2 valid in list

#2.4 range function

for i in range(5):#start from 0 to 5(5 exclusive)

    print(i)

#print  from 1 to 5(5 not include)

for i in range(1,5):

    print(i)

#print start from 1 with gap 2 to 10

for i in range(1,10,2):

    print(i)

Mapping - dict

# dictionary datatype

#create empty dict

dictEmpty={}

#addding element in dict with key

dictEmpty["one"]="this is one"

dictEmpty[2]="this is two"

print(dictEmpty)#print full dict

print(dictEmpty['one'])

print(dictEmpty[2])

#create dict with key

dict1={1:"mangos",2:"oranges",3:"auro",4:"Keela"}

print(dict)

print(dict1.keys)#print only keys

print(dict1.values)#print only values

## Set Data Type

Set is a Python implementation of set as defined in Mathematics. A set in Python is a collection, but is not an indexed or ordered collection as string, list or tuple

set collection can be of different data types.

A set can store only **immutable** objects such as number (int, float, complex or bool), string or tuple. If you try to put a list or a dictionary in the set collection, Python raises a **TypeError**.

#set datatype

#create set

set1={"Monday","Tuesday","wednesday","thursday","friday","saturday","Sunday"}

print(set1)

print(type(set1))

**Hashing** is a mechanism in computer science which enables quicker searching of objects in computer's memory. **Only immutable objects are hashable**.

## Boolean Data Types

#Boolean Data Types

a=2

b=3

print(bool(a==b))#return false

print(a==b)#same

a=None

print(bool(a))#return false

a=()

print(bool(a))#return false

a1=0.0

print(bool(a1))#return false

x=b"khan"

print(type(x))

## Python Type Casting

## Python Type Casting is a process in which we convert a literal of one data type to another data type. Python supports two types of casting − implicit and explicit.

## Implicit Casting

When any language compiler/interpreter automatically converts object of one type into other, it is called automatic or **implicit casting**. Python is a strongly typed language. It doesn't allow automatic type conversion between unrelated data types

## Implicit int to float casting takes place when any arithmetic operation on int and float operands is done.

a=10#int change to 10.0

b=10.5#float

c=a+b#float

print(c)

#add boolean

d=True#this is equal to 1

c=a+b+d

print(c)

## Python Explicit Casting

Although automatic or implicit casting is limited to **int** to **float** conversion, you can use Python's built-in functions int(), float() and str() to perform the explicit conversions such as string to integer.

a=10#int change to 10.0

b=10.5#float

c=a+b#float

print(c)

#add boolean

d=True#this is equal to 1

c=a+b+d

print(c)

#using int function

a1=10.4

print(int(a1))#this convert float to int

#using float function

a2=20

print(float(a2))

#string to integer

str1="10"

print(int(str1))

#binary string to intger

#binary string 1 and 0 only with base 2

a=int("101101",2)

print(a)

#octal to integer

a=int("20",8)

print(a)

#hexa decimal to integer

a=int("2A4",16)

print(a)

#using float function

a=float(1.00E4)

print(a)

a=float(1.00E-4)

print(a)

#integer to string

a=str(20)

print(a)

#float to string

a=str(30.45)

print(type(a))

a=str(2.34E-4)

print(a)

#using list() function string to list

c=("Hello")

list1=list(c)

print(list1)

#String to tuple using tuple function

tuple1=tuple(c)

print(tuple1)

# Python - Unicode System

#string to bytes or encode

string="Shahriyar khan"

tobyte=string.encode("utf-8")

print(tobyte)

string1="khan"

tobyte1=string1.encode('utf-8')

print(tobyte1)

print(tobyte1.decode('utf-8'))

# Python – Literals

Python literals or constants are the notation for representing a fixed value in source code. In contrast to variables, literals (123, 4.3, "Hello") are static values or you can say constants which do not change throughout the operation of the program or application

## Types of Operators

Python language supports the following types of operators −

* Arithmetic Operators
* Comparison (Relational) Operators
* Assignment Operators
* Logical Operators
* Bitwise Operators
* Membership Operators
* Identity Operators

## Operator Precedence in Python

 the traditional **BODMAS** rule is also employed by Python interpreter, where the **brackets** are evaluated first, the **division** and **multiplication** operators next, followed by **addition** and **subtraction** operators. Hence, a will become 17 in the above expression.

a,b,c,d,e=5,6,16,8,0

e=(a+b)\*c/d # e=11\*16/8=11\*2=22

print("the result of (a+b)\*c/d=",e)

e=((a+b)\*c)/d# e=(11\*16)/8=176/8=22

print("the result of ((a+b)\*c)/d=",e)

e=(a+b)\*(c/d)# e=(11)\*(2)=22

print("the result of (a+b)\*c/d=",e)

e=a+(b\*c)/d# e=5+96/8=5+12=17

print("the result of (a+b)\*c/d=",e)

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Example** |
| + | Addition | a + b = 30 |
| - | Subtraction | a – b = -10 |
| \* | Multiplication | a \* b = 200 |
| / | Division | b / a = 2 |
| % | Modulus | b % a = 0 |
| \*\* | Exponent | a\*\*b =10\*\*20 |
| // | Floor Division | 9//2 = 4 |

# using + op for addition

a=4

b=6

c=a+b

#1st method using format function

print("The addition of {0} and {1} is = {2}".format(a,b,c))

#2nd method

print(f"Result:{a}+{b}={c}")

#3rd method

print(f"Result:{a+b}")

#addition of integer and complix numbers

a1=4+5j

b1=4

c=a1+b1

print("The addition of {0} and {1} is = {2}".format(a1,b1,c))

#using - for subtraction

c=a-b

print("The subtraction of {0} and {1} is = {2}".format(a,b,c))

#using \* for mult

c=a\*b

print("The multiplication of {0} and {1} is = {2}".format(a,b,c))

#using / for divid but result in float

c=a/b

print(f"Division:{a}/{b}={c} ")

#using double // for divid but result in int

c=a//b

print("Division of {0} and {1} is = {2}".format(a,b,c))

#using % modulas for remainder

a2=13

b2=5

c=a2%b2

print("The remainder of {0} and {1} is = {2}".format(a2,b2,c))

#using exponent op \*\* or power

a3=4+5j

b3=2

c=a3\*\*b3

print("{0} Power {1} is = {2}".format(a3,b3,c))

print("""

      (4+5j)^2=16+40j+25j^2

      Remember that j^2 is equal to -1

      so the expression simplifies to

      16+40j−25=−9+40j

      """)

# Augmented Assignment Operators

#Augumneted OP(+=)

a=4

b=5

a+=b

print("Addition: ",a)

#using (-=)

a-=b

print("subtraction: ",a)

#using (\*=)

a\*=b

print("multi: ",a)

#using (/=)

a/=b

print("divid: ",a)

# Comparison Operators

#using less then op(<) and grater then(>)

a=5

b=5

c=6

result=a>b

print("{0} > {1}: {2} ".format(a,b,result))

result1=a<b

print("{0} < {1}: {2} ".format(a,b,result1))

result2=c>a

print("{0} > {1}: {2} ".format(c,a,result2))

#using less then or equal to(<=)

result2=a<=b

print("{0} <= {1}: {2} ".format(a,b,result2))

#using less then or equal to(>=)

result2=a>=b

result3=c>=a

#using less then or equal to(<=)

result2=a<=b

print("{0} <= {1}: {2} ".format(a,b,result2))

print("{0} >= {1}: {2} ".format(c,a,result3))

#using equal(==)

result4=a==b

result5=a==c

print("{0} == {1}: {2} ".format(a,b,result4))

print("{0} == {1}: {2} ".format(a,c,result5))

#using != is not equal

result6=a!=c

result7=a!=b

print("{0} != {1}: {2} ".format(a,c,result6))

print("{0} != {1}: {2} ".format(a,b,result7))

# Logical Operators

x=10

y=20

#using and logical op both condition true then true

print("x>0 and x<10 = ",x>0 and x<10)#T and F=F

print("x>0 and y>10 = ",x>0 and y>10)# T and T=T

print("x%2==0 and y%2==0 = ",x%2==0 and y%2==0)#T and T=T

#using or logic at least one true then true

print("\nx>0 or x<10 = ",x>0 or x<10)# F or T=T

print("x>0 or y>10 = ",x>0 or y>10)#T or T=T

print("x<0 or x<10 = ",x<0 or x<10)#F or F=F

# Bitwise Operators

#using and(&) operator

"""

0 & 0 is 0

1 & 0 is 0

0 & 1 is 0

1 & 1 is 1

"""

a=60

b=13

c=a&b

print("a&b = {0} & {1} is ={2} ".format(a,b,c))

#understanding the above code

print("a:",bin(a))#00111100

print("b:", bin(b))#001101

"""

a: 0b111100

b: 0b1101

so

0011 1100 & 0000 1101=0000 1100=12

    """

print(int("1100",2))#output 12

#Betwise OR (|) operator

"""

0 | 0 is 0

0 | 1 is 1

1 | 0 is 1

1 | 1 is 1

"""

d=a|b

print("a|b(a or b) {0}|{1} is ={2}".format(a,b,d))

#understanding the working

"""

0011 1100 |(OR) 0000 1101= 0011 1101=61

    """

print("00111101=",int("00111101",2))#61

#Binary XOR Operator (^)

"""

0 ^ 0 is 0

0 ^ 1 is 1

1 ^ 0 is 1

1 ^ 1 is 0

    """

e=a^b

print("a^b(a XOR b) {0} ^ {1} is= {2}".format(a,b,e))

#understanding the working

"""

0011 1100 ^(XOR) 0000 1101= 00110001=49

    """

print("00110001=",int("00110001",2))#49

#Left Shift Operator (<<)

a=16

b=2

g=a<<b

print("a<<b {0} << {1} is={2}".format(a,b,g))

#understanding working

"""

print(bin(a))#10000

print(bin(b))#10

10000 << 2=1000000=64

print(int("1000000",2))

"""

#Right Shift Operator (>>)

a=16

b=2

h=a>>b

print("a>>b {0} >> {1} is={2}".format(a,b,h))

"""

print(bin(a))#10000

print(bin(b))#10

10000 >> 2=100=4

print(int("100",2))

"""

#Binary NOT Operator (~)

#first convert to bits then flips those bits 0 to 1 and 1 to 0

a1=8

a2=~a1

print("a1: {0} to ~{1} is ={2} ".format(a1,a1,a2))

#understanding working

"""

8=1000=~0000 0111=1111 0111

"""

print(int("11110111",2))

# Python - Membership Operators

The membership operators in Python help us determine whether an item is present in a given container type object, or in other words, whether an item is a member of the given container type object.

Python has two membership operators: **in** and **not in**. Both return a Boolean result. The result of **in** operator is opposite to that of **not in** operator.

#using two op in not in return boolean value

string="Shahriyar khan"

a="khan"

b="jan"

print(f"{a} in {string}=",a in string)

print(f"{b} in {string}=",b in string)

#using list

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

a=2

b=3

c=a in list

print("{0} in {1} ={2}".format(a,list,c))

print(f"{b} in {list} = {b in list}")

#using dictionary

#finding only keys not values

dict={1:"khan",2:40,3:10,3:4,4:6}

a=2

b=40

print(f"{a} in {dict} ={a in dict}")

print(f"{b} in {dict} ={b in dict}")

# Identity Operators

Python has two identity operators **is** and **is not**. Both return opposite Boolean values. The "in" operator evaluates to True if both the operand objects share the same memory location. The memory location of the object can be obtained by the "id()" function. If the id() of both variables is same, the "in" operator returns True (as a consequence, is not returns False).

Python User Input Functions

Python provides us with two built-in functions to read the input from the keyboard.

* The input () Function
* The raw\_input () Function

Python interpreter works in interactive and scripted mode. While the interactive mode is good for quick evaluations, it is less productive. For repeated execution of same set of instructions, scripted mode should be used.

#Example

name="Shahriyar"

city="karachi"

print("Hello my name is",name)

print("i am from ",city)

#using input function

name1=input("Enter the name: ")

city1=input("Enter the city name: ")

print("Hello my name is ",name1)

print("i am from ",city1)

#finding area of rectangle

width=int(input("Enter the width: "))

height=int(input("Enter the height: "))

area=width\*height

print("Area of rectangle",area)

#finding interest rate using float()

amount=float(input("Enter Amount: "))

rate=float(input("Enter rate of interest: "))

interest=amount\*rate/100

print(f"Amount of {amount} interest is :{interest}")

# Python - Numbers

# Python - Control Flow

By default, the instructions in a computer program are executed in a sequential manner, from top to bottom, or from start to end. However, such sequentially executing programs can perform only simplistic tasks. We would like the program to have a decision-making ability, so that it performs different steps depending on different conditions.

Normally, there are two type of control flow statements in any programming language and Python also supports them.

## Decision-making Statements

Decision making statements are used in the Python programs to make them able to decide which of the alternative group of instructions to be executed, depending on value of a certain Boolean expression

#Example for discount on purchase

discount=0

amount=int(input("Enter the amount: "))

if amount>1000:

    discount=amount\*10/100

    print(f"The 10% discount on amount {amount} is ={discount}")

else:

    print("Sorry! No discount is applicable")

# Python if-else Statement

#program for discount if elif else

"""

The discount structure used in an earlier

example is modified to different slabs of discount −

20% on amount exceeding 10000,

10% for amount between 5-10000,

5% if it is between 1 to 5000.

no discount if amount<1000"""

discount=0

amount=int(input("Enter the amount: "))

if amount>10000:

    discount=amount\*20/100

    print(f"The 20% discount on amount {amount} is ={discount}")

elif amount>5000:

    discount=amount\*10/100

    print(f"The 10% discount on amount {amount} is ={discount}")

elif amount>1000:

      discount=amount\*5/100

      print(f"The 5% discount on amount {amount} is ={discount}")

else:

    print("Sorry! no discount applicable")

print("Payable amount is= ",amount-discount)

#program for discount if else

discount=0

amount=int(input("Enter the amount: "))

if amount>10000:

    discount=amount\*20/100

    print(f"The 20% discount on amount {amount} is ={discount}")

else:

    if amount>5000:

        discount=amount\*10/100

        print(f"The 10% discount on amount {amount} is ={discount}")

    else:

        if amount>1000:

            discount=amount\*5/100

            print(f"The 5% discount on amount {amount} is ={discount}")

        else:

            print("Sorry! no discount applicable")

print("Payable amount is= ",amount-discount)

# 1.3 Nested If Statements

"""

There may be a situation when you want to check for another condition after a condition resolves to true.

In such a situation,

you can use the nested if construct."""

num=int(input("Enter the number: "))

if num%2==0:

    if num%3==0:

        print("divisible by 3 and 2")

    else:

        print("divisible by 2 not divisible by 3")

else:

    if num%3==0:

        print("divisible by 3 not divisible by 2 ")

    else:

        print("not divisible by 2 not divisible by 3")

# Python - Match-Case Statement

A **match** statement takes an expression and compares its value to successive patterns given as one or more case blocks. The usage is more similar to pattern matching in languages like Rust or Haskell than a switch statement in C or C++.

def weekday(n):

    match n:

        case 0: return "Monday"

        case 1: return "Tuesday"

        case 2: return "Wednesday"

        case 3: return "Thursday"

        case 4: return "Friday"

        case 5: return "Saturday"

        case 6: return "Sunday"

        case \_: return "invalid day number"

print(weekday(0))

print(weekday(3))

print(weekday(8))

#Combined Cases

"""

Sometimes, there may be a situation where for more thanone cases,

a similar action has to be taken. For this, you can combine cases

with the OR operator represented by "|" symbol"""

def access(user):

    match user:

        case "admin" | "Manager": return "Full access"

        case "Guest": return "Limited access"

        case  \_:return "No access"

print(access("Manager"))

print(access("Guest"))

print(access("Ravi"))

#List as the Argument

"""

Since Python can match the expression against any literal,

you can use a list as a case value. Moreover,

for variable number of items in the list,

they can be parsed to a sequence with "\*" operator."""

def greeting(details):

    match details:

        case [time,name]:

            return (f"Good {time} {name}!")

        case [time,\*names]:

            msg=""

            for name in names:

                msg+=(f"Good {time} {name}!\n")

            return msg

print(greeting(["Morning","shary"]))

print(greeting(["Afternoon","Khan"]))

print(greeting(["Evening","Umair","Fahad","Ali"]))

#Using "if" in "Case" Clause

"""

Normally Python matches an expression against literal cases.

However, it allows you to include if statement in

the case clause for conditional computation of match variable."""

def interest(detials):

    match detials:

        case [amount,duration] if amount<10000:

            return amount\*10\*duration/100

        case [amount,duration] if amount>=10000:

            return amount\*duration\*15/100

print("Interest=",interest([5000,5]))

print("Interest= ",interest([12000,3]))

# Python for Loops

The **for** loop in Python has the ability to iterate over the items of any sequence, such as a list, tuple or a string.

### **Syntax**

for iterating\_var in sequence:

statements(s)

If a sequence contains an expression list, it is evaluated first. Then, the first item (at 0th index) in the sequence is assigned to the iterating variable iterating\_var.

Next, the statements block is executed. Each item in the list is assigned to iterating\_var, and the statement(s) block is executed until the entire sequence is exhausted.

#Using "for" with a String

"""

A string is a sequence of Unicode letters,

each having a positional index. The following

example compares each character and displays

if it is not a vowel ('a', 'e', 'I', 'o' or 'u')"""

zen='''

Beautiful is better than ugly.

Explicit is better than implicit.

Simple is better than complex.

Complex is better than complicated.

'''

for char in zen:

    if char not in "aeiou":

        print(char,end='')

# Using "for" with a Tuple

"""

Python's tuple object is also an indexed sequence,

and hence we can traverse its items with a for loop.

Example

In the following example, the for loop traverses

a tuple containing integers and returns the total of all numbers."""

numbers=(1,2,3,4,5,6,7,8,9,10)

total=0

for num in numbers:

    total+=num

print("\nSum of all numbers:",total)

#Using "for" with a List

"""

Python's list object is also an indexed sequence,

and hence we can traverse its items with a for loop.

Example

In the following example, the for loop traverses

a list containing

integers and prints only those which are divisible by 2"""

lists=[2,4,6,7,2,9,8,16,18]

total=[]

for num in lists:

    if num%2==0:

        print("divisible by 2 numbers in lists: ",(num))

        total.append(num)

print("All divisible by 2 numbers in lists: ",list(total))

#Using "for" with a Range Object

"""

Python's buil-in range() function returns a range object.

Python's range object is an iterator which generates an integer with each iteration.

The object contains integrrs from start to stop, separated by step parameter.

Syntax

The range() function has the following syntax −"""

numbers=range(5)

print("List:",list(numbers))

numbers=range(1,10)

print("List:",list(numbers))

numbers=range(0,20,2)

print("Even List: ",list(numbers))

for num in range(5):

    print(num,end=' ')

print()

for num in range(1,10):

    print(num,end=' ')

print()

for num in range(0,20,2):

    print(num,end=' ')

print()

#Factorial of a Number

"""

Factorial is a product of all numbers from 1 to that number say n.

It can also be defined as product of 1, 2, up to n."""

#initialize fact 1

fact=1

#factorial of 5

n=5

for x in range(1,n+1):

    fact\*=x

   # print(fact)

print(f"factorial of {n} is {fact}")

#Using "for" Loop with Sequence Index

"""

To iterate over a sequence, we can obtain the

list of indices using the range() function

"""

numbers=[10,20,30,40,50,60,70]

indexes=range(len(numbers))

for index in indexes:

    print("Index: ",index,"numbers: ",numbers[index])

# Using "for" with Dictionaries

"""

Unlike a list, tuple or a string, dictionary data type in Python is not a sequence, as the items do not have a positional index. However,

traversing a dictionary is still possible with different techniques.

Running a simple for loop over

the dictionary object traverses the keys used in it."""

dicts={10:"Ten",20:"Twenty",30:"Thirty",40:"Forty"}

for x in dicts:

    #print(x)#show only keys

    print(x," : ",dicts[x])#show data with keys

#return a tuple of keys with data

for y in dicts.items():

    print(y)

#return without tuple same data

for x,y in dicts.items():

    print(x," : ",y)

# Python while Loops

Normally, flow of execution of steps in a computer program goe from start to end. However, instead of the next step, if the flow is redirected towards any earlier step, it constitutes a loop.

A **while** loop statement in Python programming language repeatedly executes a target statement as long as a given boolean expression is true.

#Python while Loops

count=0

while count<5:

    count+=1

    print(f"Iteration no.{count}")

print("End of while loop")

#Python break Statement

"""

It terminates the current loop and resumes

execution at the next statement

The break statement can be used in both

while and for loops.

If you are using nested loops,

the break statement stops the execution of the

innermost loop and start executing the next line

of code after the block."""

#break in for loop

letters="Python"

for letter in letters:

    if letter=="h":

        break

    print("Current letter: ",letter)

#break in while loop

var=10

while var>0:

    print("Current variable value: ",var)

    var-=1

    #var=var-1

    if var==5:

        break

print("Good bye!")

#Python Continue Statement

"""

The continue statement in Python returns the control to the

beginning of the current loop. When encountered,

the loop starts next iteration without executing the remaining

statements in the current iteration.

The continue statement can

be used in both while and for loops."""

#continue in for loop

letters="Python"

for letter in letters:

    if letter=="h":

        continue

    print("Current letter: ",letter)

#continue in while loop

var=10

while var>0:

    var-=1

    #var=var-1

    if var==5:

        continue

    print("Current variable value: ",var)

print("Good bye!")

# Python - Functions

A Python function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

A top-to-down approach towards building the processing logic involves defining blocks of independent reusable functions. A Python function may be invoked from any other function by passing required data (called **parameters** or **arguments**). The called function returns its result back to the calling environment.

## Types of Python Functions

Python provides the following types of functions −

* Built-in functions
* Functions defined in built-in modules
* User-defined functions

## Defining a Function in Python

You can define custom functions to provide the required functionality. Here are simple rules to define a function in Python.

* Function blocks begin with the keyword **def** followed by the function name and parentheses ( ( ) ).
* Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
* The first statement of a function can be an optional statement; the documentation string of the function or docstring.
* The code block within every function starts with a colon (:) and is indented.
* The statement **return [expression]** exits a function, optionally passing back an expression to the caller. A **return** statement with no arguments is the same as return None.

#create function

def function\_Name():

    "this is doc string of greetings function"

    print("Hello word!")

    return

function\_Name()

#calling a function

def printme(str):

    "This prints a passed string into this function"

    print(str)

    return;

#now you can call printme function

printme("\ni am first call to user defined function!")

printme("Again second call to the same function")

#Pass by Reference vs Value

"""

The function calling mechanism of Python differs from that of C and C++. There are two main function calling mechanisms:

    Call by Value and Call by Reference."""

#Pass string variable

def testfunction(arg):

    print("Id inside the function: ",id(arg))

var="Hello"

print("\nId before passing: ",id(var))

testfunction(var)

#Pass numaric varaible

"""

The behaviour also depends on whether the passed object is mutable or immutable. Python numeric object is immutable. When a numeric object is passed, and then the function changes the value of the formal argument, it actually creates a new object in the memory,

leaving the original variable unchanged."""

def testfuntion2(arg):

    print("Id inside the function: ",id(arg))

    arg=arg+1

    print("new object after increment",arg,id(arg))

var=10

print("\nId before passing: ",id(var))

testfuntion2(var)

print("value after function call: ", var)

#passing list or dictionary

def listFunction(arg):

    arg=arg.append(100)

    print("inside the function: ",arg)

    print("Inside and modify the list in function: ",id(arg))

list=[10,20,30,40]

print("\nList after function call :",list)

print("\nId before passing: ",id(list))

listFunction(list)

print("List after function call :",list)

print("Id after the function calling: ",id(list))

#Function Arguments

"""

The process of a function often depends on certain

data provided to it while calling it.

While defining a function, you must give a

list of variables in which the data passed to it is collected.

The variables in the parentheses are called formal arguments.

When the function is called, value to each of the formal arguments must be provided.

Those are called actual arguments."""

def string(name):

    "This is docString of greetting function  "

    print(f"\nHello {name}")

    return

string("Shary ")

string("Khan  ")

string("jan")

#Function with Return Value

"""

The return keyword as the last statement in function

definition indicates end of function block,

and the program flow goes back to the calling function.

Although reduced indent after the last statement in the block also implies return

but using explicit return is a good practice.

Along with the flow control, the function can also return value of an expression to the calling function.

The value of returned expression can be stored in a variable for further processing."""

def add(a,b):#step 1,step5

    result=a+b#step 6

    return result#step 7

n1=10#step2

n2=20#step3

result=add(n1,n2)#step 4,8

print(f"Addition of {n1} + {n2} is = {result}")#step 9

## Types of Function Arguments

Based on how the arguments are declared while defining a Python function, they are classified into the following categories −

* Positional or required arguments
* Keyword arguments
* Default arguments
* Positional-only arguments
* Keyword-only arguments
* Arbitrary or variable-length arguments

In the next few chapters, we will discuss these function arguments at length.

## Order of Arguments

A function can have arguments of any of the types defined above. However, the arguments must be declared in the following order −

* The argument list begins with the positional-only args, followed by the slash (/) symbol.
* It is followed by regular positional args that may or may not be called as keyword arguments.
* Then there may be one or more args with default values.
* Next, arbitrary positional arguments represented by a variable prefixed with single asterisk, that is treated as tuple. It is the next.
* If the function has any keyword-only arguments, put an asterisk before their names start. Some of the keyword-only arguments may have a default value.
* Last in the bracket is argument with two asterisks \*\* to accept arbitrary number of keyword arguments.

# 1. Arguments

# Information can be passed into functions as arguments.

#Parameters or Arguments?

"""

The terms parameter and argument can be used for the same thing: information that are passed into a function.

From a function's perspective:

A parameter is the variable listed inside the parentheses in the function definition.

An argument is the value that is sent to the function when it is called."""

# 1.1 Number argument

"""

By default, a function must be called with the correct number of arguments.

Meaning that if your function expects 2 arguments,

you have to call the function with 2 arguments, not more, and not less.

"""

def student(fname,lname):

    print(fname+ " " +lname)

#calling

student("Shary","Khan")

# 1.2 Default Aurguments

"""

Python allows to define a function with default value assigned to one or more formal arguments.

Python uses the default value for such an argument if no value is passed to it.

If any value is passed, the default value is overridden with the actual value passed.

"""

def printinf(name,age=35):

    " this prints a passed info this function  "

    print("Name: ",name)

    print("Age: ",age)

    return

#now calling function

printinf(name="Shary",age=17)#using passing age

printinf(name="Shahriyar")#when we dont pass age then using 35

#Example 2

def percent(phy,maths,maxmarks=200):

    var=(phy+maths)\*100/maxmarks

    return var

#callin point 1

phy=60

maths=70

result=percent(phy,maths)#no pass maxmarks so use defaul

print("Percentage: ",result)

#calling point 2

phy=40

maths=46

result=percent(phy,maths,100)

print("Percentage: ",result)

# 1.3 Python - Keyword Arguments

"""

Python allows to pass function arguments in the form of keywords

which are also called named arguments.

Variables in the function definition are used as keywords.

When the function is called,

you can explicitly mention the name and its value."""

def keyword\_arg(name,age):

    print("Name: ",name)

    print("Age: ",age)

#Now calling function by positional arguments

keyword\_arg("S\_Khan",30)

#calling with keyword arguments

keyword\_arg(name="Shary",age=50)

#the positional arguments must be before the keyword arguments while using mixed calling.

#keyword\_arg(name="Shary1",11)#error

# 1.4 Python - Keyword-Only Arguments

"""

You can use the variables in formal argument list as keywords to pass value.

Use of keyword arguments is optional.

But, you can force the function be given arguments by keyword only.

You should put an

astreisk (\*) before the keyword-only arguments list."""

def inter(amt,\*,rate):

    var=amt\*rate/100

    return var

#intrest=inter(1000,10) error

inttrest=inter(1000,rate=10)

print(f"Intrest on {1000} is ={inttrest}")

# 1.5 Python - Positional Arguments

"""

The list of variables declared in the parentheses at

the time of defining a function are the formal

arguments. A function may be defined with any number of formal arguments.

While calling a function −

All the arguments are required

The number of actual arguments must be equal to the number of formal arguments.

Formal arguments are positional. They Pick up values in the order of definition.

The type of arguments must match.

Names of formal and actual arguments need not be same."""

def add(x,y):

   z=x+y

   print ("x={} y={} x+y={}".format(x,y,z))

a=10

b=20

add(a,b)

# 1.6 Python - Arbitrary Arguments

"""

You may want to define a function that is able to accept arbitrary or variable number of arguments.

Moreover, the arbitrary number of arguments might be positional or keyword arguments.

An argument prefixed with a single asterisk \* for arbitrary positional arguments.

An argument prefixed with two asterisks \*\* for arbitrary keyword arguments."""

def add(\*args):

    s=0

    for x in args:

        s=s+x

    return s

#calling and pass a tuple

result=add(10,20,30,40)

print("Sum of all items: ",result)

"""The args variable prefixed with "\*" stores all the values passed to it. H

ere, args becomes a tuple.

We can run a loop over its items to add the numbers."""

result=add(1,2,3,4,5)

print("Sum of all numbers: ",result)

#Example 2

def avg(first,\*rest):

    second=max(rest)

    return (first+second)/2

result=avg(40,30,50,34)

print("average of first and best test: ",result)

#Explain  Example 2

"""

Following call to avg() function passes first value to the required argument first,

and the remaining values to a tuple named rest.

We then find the maximum and use it to calculate the average.

"""

#Example

"""

The following code is an example of a function with arbitrary keyword arguments.

The addr() function has an argument \*\*kwargs which is able to accept any number

of address elements like name, city, phno, pin, etc.

Inside the function kwargs dictionary of kw:value pairs is traversed using items() method."""

def addr(\*\*kwargs):

    for k,v in kwargs.items():

        print(f"{k}:{v}")

print("\n\nPassed two keyword args")

addr(Name="Jhon",City="Peshawer")

print("Passed four keywords args")

addr(Name="Shary",City="Charsadda",ph\_no="1234456",pin="4455")

#Example

"""

Imagine a case where science and maths are mandatory subjects,

in addition to which student may choose any number of elective subjects.

The following code defines a percent() function where marks in science and marks are stored in required arguments,

and the marks in variable number of elective subjects in \*\*optional argument."""

def test(maths,sci,\*\*Optional):

    print("\n\nMaths: ",maths)

    print("Sci: ",sci)

    #add maths and sci marks in S

    s=maths+sci

    #for loop for reaminning subject values

    for k,v in Optional.items():

        print(f"{k} : {v}")

        s=s+v

        return s/len(Optional)+2

    #call function with args

results=test(maths=80,sci=75,Eng=70,Hist=65,Geo=72)

print("Percentage: ",results)

# Python - Variable Scope

## Python Variable Scope

A variable in Python is a symbols name to the object in computer's memory. Python works on the concept of namespaces to define the context for various identifiers such as functions, variables etc. A namespace is a collection of symbolic names defined in the current context.

Python provides the following types of namespaces −

* **Built-in namespace** contains built-in functions and built-in exceptions. They are loaded in the memory as soon as Python interpreter is loaded and remain till the interpreter is running.
* **Global namespace** contains any names defined in the main program. These names remain in memory till the program is running.
* **Local namespace** contains names defined inside a function. They are available till the function is running.

#Python globals() Function

"""

Python's standard library includes a built-in function globals(). It returns a dictionary

of symbols currently available in global namespace."""

name="Shary"

marks=69

result=True

def myfunction():

    a=10

    b=20

    return a+b

print(globals())

#show the usag of global variable in python

#global variable

x=5

y=10

def add():

    sum=x+y

    return sum

print(add())

#Python's standard library includes a built-in function locals().

"""

It returns a dictionary of symbols currently available in the local namespace of the function.

Modify the above script to print dictionary of

global and local namespaces from within the function."""

name="Shary Khan"

marks=89

result=True

def myfunction1():

    a=10

    b=20

    c=a+b

    print("\n\nGlobals(): ",globals())

    print("\n\nLocals(): ",locals())

    print(locals().get('a'))#shows 10

    return c

myfunction1()

#USING get() method and index op for both global and locals function return

print(globals()["name"])#display shary

#Namespace Conflict in Python

"""

If a variable of same name is present in global as well as local scope,

Python interpreter gives priority to the one in local namespace."""

marks=50 #This is a gloabal variable

def myfunction2():

    marks=80 #this is a local variable

    print("\n\nLocal marks: ",marks) #print local marks

myfunction2()

print("Global marks:",marks) #print global marks

#Modify global variable

"""T

o modify a global variable, you can either update it with a dictionary syntax,

or use the global keyword to refer it before modifying."""

var1=100 #global

var2=50

def myfunction3():

    #local=marks+30 #UnboundLocal error

    #so modify global values two methods

    globals()['var1']=globals()['var1']+20

    #also this

    global var2

    var2=var2+10

myfunction3()

print("var1: ",var1,"\nvar2: ",var2)#show global variables with modify values

# Python - Function Annotations

The function annotation feature of Python enables you to add additional explanatory metada about the arguments declared in a function definition, and also the return data type.

Although you can use the docstring feature of Python for documentation of a function, it may be obsolete if certain changes in the function's prototype are made. Hence, the annotation feature was introduced in Python as a result of PEP 3107.

The annotations are not considered by Python interpreter while executing the function. They are mainly for the Python IDEs for providing a detailed documentation to the programmer.

Annotations are any valid Python expressions added to the arguments or return data type. Simplest example of annotation is to prescribe the data type of the arguments. Annotation is mentioned as an expression after putting a colon in front of the argument.

def myfunction(a: int, b: int):

c = a+b

return c

#Example 1 annotations

"""

Annotations are ignored at runtime,

but are helpful for the IDEs and static type checker libraries such as mypy."""

def myfunction(a: int, b: int):

    c=a+b

    return c

print(myfunction(10,20))

print(myfunction("Hello ","Khan"))

#You can give annotation for the return data type as wel

def myfunction(a:int, b: int) ->int:

    c=a+b

    return c

"""

As using the data type as annotation is ignored at runtime,

you can put any expression which acts as the metadata for the arguments.

"""

#arbitrary expression as annotation as in following example −

def total(x: 'marks in Physics', y:'marks in English'):

    return x+y

#The \_\_annotations\_\_ attribute itself is a dictionary in which arguments are keys and anootations their values.

def myfunction(a: "physics", b:"Maths" = 20) -> int:

   c = a+b

   return c

print (myfunction.\_\_annotations\_\_)

#You may have arbitrary positional and/or arbitrary keyword arguments for a function. Annotations can be given for them also.

def myfunction(\*args:'arbitrary\_args', \*\*kwargs: 'arbitrary\_keyword\_args') -> int:

   pass

print (myfunction.\_\_annotations\_\_)

# Python - Modules

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

The concept of module in Python further enhances the modularity. You can define more than one related functions together and load required functions. A module is a file containing definition of functions, classes, variables, constants or any other Python object. Contents of this file can be made available to any other program. Python has the import keyword for this purpose.

Built in Modules

Python's standard library comes bundled with a large number of modules. They are called built-in modules. Most of these built-in modules are written in C (as the reference implementation of Python is in C), and pre-compiled into the library. These modules pack useful functionality like system-specific OS management, disk IO, networking, etc.

Here is a select list of built-in modules −

|  |  |
| --- | --- |
| **Sr.No.** | **Name & Brief Description** |
| 1 | **os**  This module provides a unified interface to a number of operating system functions. |
| 2 | **string**  This module contains a number of functions for string processing |
| 3 | **re**  This module provides a set of powerful regular expression facilities. Regular expression (RegEx), allows powerful string search and matching for a pattern in a string |
| 4 | **math**  This module implements a number of mathematical operations for floating point numbers. These functions are generally thin wrappers around the platform C library functions. |
| 5 | **cmath**  This module contains a number of mathematical operations for complex numbers. |
| 6 | **datetime**  This module provides functions to deal with dates and the time within a day. It wraps the C runtime library. |
| 7 | **gc**  This module provides an interface to the built-in garbage collector. |
| 8 | **asyncio**  This module defines functionality required for asynchronous processing |
| 9 | **Collections**  This module provides advanced Container datatypes. |
| 10 | **Functools**  This module has Higher-order functions and operations on callable objects. Useful in functional programming |
| 11 | **operator**  Functions corresponding to the standard operators. |
| 12 | **pickle**  Convert Python objects to streams of bytes and back. |
| 13 | **socket**  Low-level networking interface. |
| 14 | **sqlite3**  A DB-API 2.0 implementation using SQLite 3.x. |
| 15 | **statistics**  Mathematical statistics functions |
| 16 | **typing**  Support for type hints |
| 17 | **venv**  Creation of virtual environments. |
| 18 | **json**  Encode and decode the JSON format. |
| 19 | **wsgiref**  WSGI Utilities and Reference Implementation. |
| 20 | **unittest**  Unit testing framework for Python. |
| 21 | **random**  Generate pseudo-random numbers |

User Defined Modules

Any text file with .py extension and containing Python code is basically a module. It can contain definitions of one or more functions, variables, constants as well as classes. Any Python object from a module can be made available to interpreter session or another Python script by import statement. A module can also include runnable code.

#Example 1

import math

print("Square root of 100: ",math.sqrt(100))

#create module

def hello(name):

     print("Hi {}! how are you? ".format(name))

     return

#The import Statement

"""

In Python, the import keyword has been provided to load a Python object from one module.

The object may be a function, class, a variable etc.

If a module contains multiple definitions,

all of them will be loaded in the namespace.

"""

#import modules from another program

# import mymodules as md

# print("Sum: ",md.sum(10,10))

# print("average: ",md.avg(10,10))

# print("power: ",md.power(10,2))

#list all the function names (or variable names) in a module. The dir() function:

# x=dir(md)

# print("The all function names (or variable names) in mymodules: ",x)

#The from ... import Statement

"""

The import statement will load all the resources of the module in the current namespace.

It is possible to import specific objects from a module by using this syntax

"""

from mymodules import  sum,avg

print("Sum : ",sum(10,40))

print("Average: ",avg(10,40))

"""

Note: When importing using the from keyword, do not use the module name when referring to elements in the module

"""

## The from...import \* Statement

It is also possible to import all the names from a module into the current namespace by using the following import statement −

from modname import \*

This provides an easy way to import all the items from a module into the current namespace; however, this statement should be used sparingly.

## Module Attributes

In Python, a module is an object of module class, and hence it is characterized by attributes.

Following are the module attributes −

* \_\_file\_\_ returns the physical name of the module.
* \_\_package\_\_ returns the package to which the module belongs.
* \_\_doc\_\_ returns the docstring at the top of the module if any
* \_\_dict\_\_ returns the entire scope of the module
* \_\_name\_\_ returns the name of the module

"The doc string of mymodules"

def sum(x,y):

    c=x+y

    return print("Additon of two numbers: ",c)

def avg(x,y):

     return x+y/2

def power(x,y):

     return x\*\*y

"""

1. \_\_name\_\_ is a special variable in Python. When a Python script is executed,

\_\_name\_\_ is set to "\_\_main\_\_" if the script is being run as the main program.

If the script is imported as a module into another script,

\_\_name\_\_ is set to the name of the module (not "\_\_main\_\_").

2. if \_\_name\_\_ == "\_\_main\_\_": checks if the script is being run as the main program.

If true, the code inside the block will be executed.

3. Inside the block, print("M\_Sum: ", sum(4, 5)) is calling a function named sum with the arguments 4 and 5.

However, there's a potential issue here. The built-in sum function in

Python takes an iterable (like a list) as its argument, not individual values.

If you want to sum these two values,

you might want to use the + operator instead:"""

if \_\_name\_\_=="\_\_main\_\_":

    print("M\_Sum: ",sum(4,5))

#Example 1

import math

print("Square root of 100: ",math.sqrt(100))

#create module

def hello(name):

     print("Hi {}! how are you? ".format(name))

     return

#The import Statement

"""

In Python, the import keyword has been provided to load a Python object from one module.

The object may be a function, class, a variable etc.

If a module contains multiple definitions,

all of them will be loaded in the namespace.

"""

#import modules as alies

import mymodules as md

print("Sum: ",md.sum(10,10))

print("average: ",md.avg(10,10))

print("power: ",md.power(10,2))

#list all the function names (or variable names) in a module. The dir() function:

x=dir(md)

print("The all function names (or variable names) in mymodules: ",x)

#The from ... import Statement

"""

The import statement will load all the resources of the module in the current namespace.

It is possible to import specific objects from a module by using this syntax

"""

#Example From ... import statement

from mymodules import  sum,avg

print("Sum : ",sum(10,40))

print("Average: ",avg(10,40))

"""

Note: When importing using the from keyword, do not use the module name when referring to elements in the module

"""

#Example Module Attributes

import mymodules

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

#1. \_\_file\_\_ returns the physical name of the module.

print("\_\_file\_\_attribute: ",mymodules.\_\_file\_\_)# c:\Users\Shahriyar Khan\Desktop\Python\_Prac\Ch3\_Function\mymodules.py

#2. \_\_package\_\_ returns the package to which the module belongs.

print("\_\_package\_\_attribute: ",mymodules.\_\_package\_\_)#

#3. \_\_doc\_\_ returns the docstring at the top of the module if any

print("\_\_doc\_\_attribute: ",mymodules.\_\_doc\_\_)#The doc string of mymodules

#4. \_\_dict\_\_ returns the entire scope of the module

#print("\_\_dict\_\_attribute: ",mymodules.\_\_dict\_\_)

#5. \_\_name\_\_ returns the name of the module

print("\_\_name\_\_attribute: ",mymodules.\_\_name\_\_)#The doc string of mymodules

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

#The reload() Function

"""

Sometimes you may need to reload a module, especially when working with the interactive interpreter session of Python.

Even if you edit the test.py file and save it, the function loaded in the memory won't update.

You need to reload it, using reload() function in imp module.

"""

import imp

imp.reload(mymodules)

mymodules.sum(5,6)

# Python - Built-in Functions

As of Python 3.11.2 version, there are 71 built-in functions in Pyhthon. The list of built-in functions is given below −

|  |  |
| --- | --- |
| **Sr.No.** | **Function & Description** |
| 1 | **abs()**  Returns absolute value of a number |
| 2 | **aiter()**  Returns an asynchronous iterator for an asynchronous iterable |
| 3 | **all()**  Returns true when all elements in iterable is true |
| 4 | **anext()**  Returns the next item from the given asynchronous iterator |
| 5 | **any()**  Checks if any Element of an Iterable is True |
| 6 | **ascii()**  Returns String Containing Printable Representation |
| 7 | **bin()**  Converts integer to binary string |
| 8 | **bool()**  Converts a Value to Boolean |
| 9 | **breakpoint()**  This function drops you into the debugger at the call site and calls sys.breakpointhook() |
| 10 | **bytearray()**  returns array of given byte size |
| 11 | **bytes()**  returns immutable bytes object |
| 12 | **callable()**  Checks if the Object is Callable |
| 13 | **chr()**  Returns a Character (a string) from an Integer |
| 14 | **classmethod()**  Returns class method for given function |
| 15 | **compile()**  Returns a code object |
| 16 | **complex()**  Creates a Complex Number |
| 17 | **delattr()**  Deletes Attribute From the Object |
| 18 | **dict()**  Creates a Dictionary |
| 19 | **dir()**  Tries to Return Attributes of Object |
| 20 | **divmod()**  Returns a Tuple of Quotient and Remainder |
| 21 | **enumerate()**  Returns an Enumerate Object |
| 22 | **eval()**  Runs Code Within Program |
| 23 | **exec()**  Executes Dynamically Created Program |
| 24 | **filter()**  Constructs iterator from elements which are true |
| 25 | **float()**  Returns floating point number from number, string |
| 26 | **format()**  Returns formatted representation of a value |
| 27 | **frozenset()**  Returns immutable frozenset object |
| 28 | **getattr()**  Returns value of named attribute of an object |
| 29 | **globals()**  Returns dictionary of current global symbol table |
| 30 | **hasattr()**  Returns whether object has named attribute |
| 31 | **hash()**  Returns hash value of an object |
| 32 | **help()**  Invokes the built-in Help System |
| 33 | **hex()**  Converts to Integer to Hexadecimal |
| 34 | **id()**  Returns Identify of an Object |
| 35 | **input()**  Reads and returns a line of string |
| 36 | **int()**  Returns integer from a number or string |
| 37 | **isinstance()**  Checks if a Object is an Instance of Class |
| 38 | **issubclass()**  Checks if a Class is Subclass of another Class |
| 39 | **iter()**  Returns an iterator |
| 40 | **len()**  Returns Length of an Object |
| 41 | **list()**  Creates a list in Python |
| 42 | **locals()**  Returns dictionary of a current local symbol table |
| 43 | **map()**  Applies Function and Returns a List |
| 44 | **max()**  Returns the largest item |
| 45 | **memoryview()**  Returns memory view of an argument |
| 46 | **min()**  Returns the smallest value |
| 47 | **next()**  Retrieves next item from the iterator |
| 48 | **object()**  Creates a featureless object |
| 49 | **oct()**  Returns the octal representation of an integer |
| 50 | **open()**  Returns a file object |
| 51 | **ord()**  Returns an integer of the Unicode character |
| 52 | **pow()**  Returns the power of a number |
| 53 | **print()**  Prints the Given Object |
| 54 | **property()**  Returns the property attribute |
| 55 | **range()**  Returns a sequence of integers |
| 56 | **repr()**  Returns a printable representation of the object |
| 57 | **reversed()**  Returns the reversed iterator of a sequence |
| 58 | **round()**  Rounds a number to specified decimals |
| 59 | **set()**  Constructs and returns a set |
| 60 | **setattr()**  Sets the value of an attribute of an object |
| 61 | **slice()**  Returns a slice object |
| 62 | **sorted()**  Returns a sorted list from the given iterable |
| 63 | **staticmethod()**  Transforms a method into a static method |
| 64 | **str()**  Returns the string version of the object |
| 65 | **sum()**  Adds items of an Iterable |
| 66 | **super()**  Returns a proxy object of the base class |
| 67 | **tuple()**  Returns a tuple |
| 68 | **type()**  Returns the type of the object |
| 69 | **vars()**  Returns the \_\_dict\_\_ attribute |
| 70 | **zip()**  Returns an iterator of tuples |
| 71 | **\_\_import\_\_()**  Function called by the import statement |

## Built-in Mathematical Functions

Following mathematical functions are built into the Python interpreter, hence you don't need to import them from any module.

|  |  |
| --- | --- |
| **Sr.No.** | **Function & Description** |
| 1 | [**abs() function**](https://www.tutorialspoint.com/python/abs_function.htm)  The abs() function returns the absolute value of x, i.e. the positive distance between x and zero. |
| 2 | [**max() function**](https://www.tutorialspoint.com/python/max_function.htm)  The max() function returns the largest of its arguments or largest number from the iterable (list or tuple). |
| 3 | [**min() function**](https://www.tutorialspoint.com/python/min_function.htm)  The function min() returns the smallest of its arguments i.e. the value closest to negative infinity, or smallest number from the iterable (list or tuple) |
| 4 | [**pow() function**](https://www.tutorialspoint.com/python/pow_function.htm)  The pow() function returns x raised to y. It is equivalent to x\*\*y. The function has third optional argument mod. If given, it returns (x\*\*y) % mod value |
| 5 | [**round() Function**](https://www.tutorialspoint.com/python/round_function.htm)  round() is a built-in function in Python. It returns x rounded to n digits from the decimal point. |
| 6 | [**sum() function**](https://www.tutorialspoint.com/python/sum_function.htm)  The sum() function returns the sum of all numeric items in any iterable (list or tuple). An optional start argument is 0 by default. If given, the numbers in the list are added to start value. |

# Python - Strings

In Python, a string is an immutable sequence of Unicode characters. Each character has a unique numeric value as per the UNICODE standard. But, the sequence as a whole, doesn't have any numeric value even if all the characters are digits. To differentiate the string from numbers and other identifiers, the sequence of characters is included within single, double or triple quotes in its literal representation. Hence, 1234 is a number (integer) but '1234' is a string.

#String

"""

A string is a non-numeric data type."""

# 1.1 String Length

"""

To get the length of a string, use the len() function.

"""

print(f"Length of {string} =",len(string))

# 1.2 Remove Whitespace

"""The strip() method removes any whitespace from the beginning or the end"""

name="    sharykhan"

print(f"{name} \n",name.strip())

#

# 1.3 replace

# 1.5 Python Slicing Strings

"""

In Python, a string is an ordered sequence of Unicode characters.

Each character in the string has a unique index in the sequence.

The index starts with 0. First character in the string has its positional index 0.

The index keeps incrementing towards the end of string.

"""

string="Shahriyar"

print(string[0])#output 1st letter S

print(string[0:6])#output shahri

print(string[6::-1])#output  yirhahS opposite the above

print(string[0:-1])#Shahriya

#start from 0 to 6 but 6 not include

print(string[:6])#Shahri

#start from 6 to end of string

print(string[6:])#yar

"""

In Python, string is an immutable object. The object is immutable

if it cannot be modified in-place, once stored in a certain memory location.

You can retrieve any character from the string with the help of its index,

but you cannot replace it with another character. In our example,

character Y is at index 7 in HELLO PYTHON. Try to replace Y with y and see what happens.

"""

#string[0]='s' #str' object does not support item assignment

#print(string)

"""

The left operand must be smaller than the operand on right,

for getting a substring of the original string.

Python doesn't raise any error,

if the left operand is greater, bu returns a null string.

"""

print(string[6:2])#empty

print("[:5][:2]: ",string[:5][:2])#Sh because using and op those show wich in both

# 1.6 Python - Modify Strings

"""

In Python, a string (object of str class) is of immutable type.

An immutable object is the one which can be modified in place,

one created in the memory. Hence, unlike a list, any character in the sequence cannot be overwritten,

nor can we insert or append characters to it unless we use certain string method that returns a new string object.

"""

#tricks as a workaround to modify a string.

"""

Converting a String to a List

Since both string and list objects are sequences, they are interconvertible.

Hence, if we cast a string object to a list, modify the list either by

insert(), append() or remove() methods and convert the list back to a string,

to get back the modified version."""

s1="Shahriyar Khan"

print("Orignal String ",s1)

# 1.6.1 convert string to list then modify string

list1=list(s1)

print("convert string to list :",list1)

list1.insert(0,'s')

print("Modify list: ",list1)

s1=''.join(list1)

print("modify String is: ",s1)

#1.6.2 Using Uppercase and lower case

#String in upper case

print(f"{string} is in Upper case: ",string.upper())

#String in lower case

print(f"{string} is in lower case: ",string.lower())

# 1.6.3 Using the Array Module

"""

To modify a string, construct an array object. Python standard library includes array module.

We can have an array of Unicode type from a string variable."""

#   Example

import array as ar

s1="Word"

sAr=ar.array('u',s1)

"""perform array operations such as append, insert, remove etc. Let us insert L before the character D"""

sAr.insert(3,'L')

print("Modified array: ",sAr)

"""Now, with the help of tounicode() method, get back the modified string"""

print("Orignal string: ",s1)

s1=sAr.tounicode()

print("Modified String: ",s1)

# 1.6.4 Using the StringIO Class

"""

Python's io module defines the classes to handle streams.

The StringIO class represents a text stream using an in-memory text buffer.

A StringIO object obtained from a string behaves like a File object.

Hence we can perform read/write operations on it.

The getvalue() method of StringIO class returns a string."""

import io

s2="Shahriyar"

print("Orignal String :",s2)

sIO=io.StringIO(s2)#Convert string to Io class

sIO.seek(9)

sIO.write("Khan")

s2=sIO.getvalue()

print("Modified String: ",s2)

#Python - String Concatenation

"""

The "+" operator is well-known as an addition operator,

returning the sum of two numbers.

However, the "+" symbol acts as string concatenation operator in Python.

It works with two string operands, and results in the concatenation of the two.

The characters of the string on the right of plus symbol are appended to the string on its left.

Result of concatenation is a new string.

"""

str1="Shahriyar"

str2=" Khan"

print("String 1:",str1)

print("String 2:",str2)

str3=str1+str2

print("String 3:",str3)

# Using (\*) operator for repitation

"""

Another symbol \*, which we normally use for multiplication of two numbers,

can also be used with string operands. Here, \* acts as a repetition operator in Python.

One of the operands must be an integer, and the second a string.

The operator concatenates multiple copies of the string. """

#The "\*" operator has a higher precedence over the "+" operator.

str4=str1+str2\*3

print("String 4: ",str4)

str5=("\t"+str1+str2)\*3

print("String 5:",str5.strip())

# 1.7 Python - String Formatting

String formatting is the process of building a string representation dynamically by inserting the value of numeric expressions in an already existing string. Python's string concatenation operator doesn't accept a non-string operand. Hence, Python offers following string formatting techniques −

* [Using % operator for substitution](https://www.tutorialspoint.com/python/string_formatting_operator.htm)
* [Using format() method of **str** class](https://www.tutorialspoint.com/python/using_the_format_method.htm)
* [Using f-string syntax](https://www.tutorialspoint.com/python/using_f_string_formatting.htm)
* [Using String Template class](https://www.tutorialspoint.com/python/using_string_template_class.htm)

# 1.7.1 String Formatting Operator

One of Python's coolest features is the string format operator %. This operator is unique to strings and makes up for the pack of having functions from C's printf() family. Format specification symbols (%d %c %f %s etc) used in C language are used as placeholders in a string.

Following is a simple example −

print ("My name is %s and weight is %d kg!" % ('Zara', 21))

My name is Zara and weight is 21 kg!

It will produce the following **output** −

My name is Zara and weight is 21 kg!

Here is the list of complete set of symbols which can be used along with % −

|  |  |
| --- | --- |
| **Sr.No** | **Format Symbol & Conversion** |
| 1 | **%c**  character |
| 2 | **%s**  string conversion via str() prior to formatting |
| 3 | **%i**  signed decimal integer |
| 4 | **%d**  signed decimal integer |
| 5 | **%u**  unsigned decimal integer |
| 6 | **%o**  octal integer |
| 7 | **%x**  hexadecimal integer (lowercase letters) |
| 8 | **%X**  hexadecimal integer (UPPERcase letters) |
| 9 | **%e**  exponential notation (with lowercase 'e') |
| 10 | **%E**  exponential notation (with UPPERcase 'E') |
| 11 | **%f**  floating point real number |
| 12 | **%g**  the shorter of %f and %e |
| 13 | **%G**  the shorter of %f and %E |

Other supported symbols and functionality are listed in the following table −

|  |  |
| --- | --- |
| **Sr.No.** | **Format Symbol & Conversion** |
| 1 | **\***  argument specifies width or precision |
| 2 | **-**  left justification |
| 3 | **+**  display the sign |
| 4 | **<sp>**  leave a blank space before a positive number |
| 5 | **#**  add the octal leading zero ( '0' ) or hexadecimal leading '0x' or '0X', depending on whether 'x' or 'X' were used. |
| 6 | **0**  pad from left with zeros (instead of spaces) |
| 7 | **%**  '%%' leaves you with a single literal '%' |
| 8 | **(var)**  mapping variable (dictionary arguments) |
| 9 | **m.n.**  m is the minimum total width and n is the number of digits to display after the decimal point (if appl.) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Format Specifier Name | Symbol/ Notation | Description | Data type | Range | Size |
| %d or %i | Decimal integer | Signed integer in base 10 | int | -2147483648 to 2147483647 | 4 bytes |
| %f | Float | Floating point number with six digits of precision | float | 1.2E-38 to 3.4E+38 | 4 bytes |
| %If | Long double | Floating point number with extended precision | long double | 3.4E-4932 to 1.1E+4932 | 10 or 16 bytes |
| %c | Character | Single character | char | -128 to 127 | 1 byte |
| %s | String | String of characters | char[] | - | - |
| %p | Pointer | Address in memory | void \* | - | 4 or 8 bytes |
| %Id | Long integer | Signed long integer | signed long | -2147483648 to 2147483647 | 4 bytes |
| %lu | Unsigned Long | Unsigned long integer | unsigned long | 0 to 4294967295 | 4 bytes |
| %lld | Long Long | Signed long long integer | long long | -9223372036854775808 to 9223372036854775807 | 8 bytes |
| %llu | Unsigned Long Long | Unsigned long long integer | unsigned long long | 0 to 18446744073709551615 | 8 bytes |
| %x | Hexadecimal | Unsigned integer in base 16 | unsigned int | 0 to 4294967295 | 4 bytes |
| %E | Scientific notation | Floating point number in scientific notation | double | 2.2E-308 to 1.8E+308 | 8 bytes |
| %o | Octal | Unsigned integer in base 8 | unsigned int | 0 to 4294967295 | 4 bytes |
| %u | Unsigned Decimal | Unsigned integer in base 10 | unsigned int | 0 to 4294967295 | 4 bytes |
| %hd | Short | Short signed integer | short | -32768 to 32767 | 2 bytes |
| %m | Error message | Error message corresponding to the error number in the argument | int | - | - |
| %n | Output assignment | Stores the number of characters written so far into the pointer argument | int \* | - | - |
| %hu | Unsigned Short | Short unsigned integer | unsigned short | 0 to 65535 | 2 bytes |
| %hd | Short | Short signed integer | short | -32768 to 32767 | 2 bytes |

# 1.7 String Formatting Operator

# 1.7.1 Using % operator for substitution

"""

One of Python's coolest features is the string format operator %. This operator is unique to strings and makes up for the pack of having functions from C's printf() family.

Format specification symbols (%d %c %f %s etc)

used in C language are used as placeholders in a string."""

#Using %s for string and d% or %i for sin

print("My name is %s and weight is %o  kg! height is %d"%('Shary',64,34))

#using %u for (unsigned decimal integer base10) and %o for (Octal integer base 8)

print("width :%u and height is: %o (Octal integer base 8)"%(42,42))

#Using %x for (Hexadecimal int lowercase letter) and %X for uppercase

print("{42} in Hexadecimal integer in lowercase:%x\n {50} in Hexadecimal intger in Uppercase :%X"%(42,50))

#using %e and %E: Exponential notation (with lowercase 'e' or uppercase 'E')

print("{12345.67} is:%e {12345678.543} is:%E"%(12345.67,12345678.543))

#%f: Floating point real number

print("{12.4567} in float round 2 decimal point:%.2f"%(12.4567))

# 1.7.2 Using the format() Method

"""Python 3.0, introduced format() method to str class for handling complex string formatting more efficiently."""

"""The string itself contains placeholders {} in which values of variables are successively inserted."""

name="Shary"

age=23

print("My name is {} and age is {} years".format(name,age))

"""You can use variables as keyword arguments to format() method and use the variable name as the placeholder in the string."""

print("My name is {name} and age is {age} years".format(name="SharyKhan",age=25))

"""You can also specify C style formatting symbols. Only change is using ":" instead of %. For example, instead of %s use {:s} and instead of %d use (:d}"""

print("My name is {:s} and age is {:d} years".format(name,age))

# 1.7.3 Using f-string Formatting

"""

With the version 3.6, Python introduced a new string formatting method,

f-strings or Literal String Interpolation.

With this formatting method you can use embedded Python expressions inside string constants.

Python f-strings are a faster, more readable, more concise, and less error prone.

The string starts with a 'f' prefix, and one or more place holders are inserted in it,

whose value is filled dynamically."""

name1="Umair"

height="5.6 cm"

fstring=f"My name is {name} and height is {height}"

print(fstring)

"""The placeholders can be populated by dictionary values."""

user={'name':"Fahad",'age':40}

print(f"your name is {user['name']} and age is {user['age']}")

"""The = character is used for self debugging the expression in f-string."""

price=20

quantity=5

print(F"Total: {price\*quantity=}")

"""The f-strings can display numbers with hexadecimal, octal and scientific notation"""

num=30

print(f"Hexadecimal: {num:x}")

print(f"Octal: {num:o}")

print(f"Scientific notation :{num:e}")

# 1.7.4 Using String Template Class

"""

Python's standard library has string module.

It provides functionalities to perform different string operations.

The Template class in string module provides an alternative method to format the strings dynamically.

One of the benefits of Template class is to be able to customize the formatting rules."""

from string import Template

temp\_str="My name is $name and i am $age years old "

temp\_obj=Template(temp\_str)

ret=temp\_obj.substitute(name="Ali",age=34)

print(ret)

"""We can also unpack the key-value pairs from a dictionary to substitute the values."""

names={'name':"Muhammad",'age':25,"height":"5.6 cm"}

temp\_str="My name is $name and i am $age years old and $height "

temp\_obj=Template(temp\_str)

ret1=temp\_obj.substitute(\*\*names)

print(ret1)

# 1.8 Python - Escape Characters

In Python, a string becomes a raw string if it is prefixed with "r" or "R" before the quotation symbols. Hence 'Hello' is a normal string whereas r'Hello' is a raw string.

there is no difference between the two. However, when the escape character is embedded in the string, the normal string actually interprets the escape sequence, whereas the raw string doesn't process the escape character.

|  |  |
| --- | --- |
| **Sr.No** | **Escape Sequence & Meaning** |
| 1 | **\<newline>**  Backslash and newline ignored |
| 2 | **\\**  Backslash (\) |
| 3 | **\'**  Single quote (') |
| 4 | **\"**  Double quote (") |
| 5 | **\a**  ASCII Bell (BEL) |
| 6 | **\b**  ASCII Backspace (BS) |
| 7 | **\f**  ASCII Formfeed (FF) |
| 8 | **\n**  ASCII Linefeed (LF) |
| 9 | **\r**  ASCII Carriage Return (CR) |
| 10 | **\t**  ASCII Horizontal Tab (TAB) |
| 11 | **\v**  ASCII Vertical Tab (VT) |
| 12 | **\ooo**  Character with octal value ooo |
| 13 | **\xhh**  Character with hex value hh |

#Example

"""when a normal string is printed the escape character '\n' is processed to introduce a newline"""

normal="Hello\nWord"

print("Normal:",normal)

"""However, because of the raw string operator 'r' the effect of escape character is not translated as per its meaning."""

raw="Hello\nWord"

print("Raw: ",raw)

#Example for all

# 1.8.1 ignore \

s='this string will not include \

blackslashes or newline character.'

print(s)

# 1.8.2 Escape blacksalsh \\

s="The \\ character is called blackslash"

print(s)

#1.8.3 Escape single quote 'Python'

s='Hello \'Python \''

print(s)

#1.8.4 Escape double quote "Python"

s='Hello \"Python\"'

print(s)

#1.8.5 escape \b to generate ASCII backspace

s='Hel\blo'

print (s)

# 1.8.6 ASCII Bell character

s='Hello\a'

print (s)

#1.8.7 Horizontal tab

s='Hello\tPython'

print (s)

# 1.8.7 form feed

s= "hello\fworld"

print (s)

#1.8.8 Octal notation

s="\101"

print(s)

# 1.8.9 Hexadecimal notation

s="\x41"

print (s)

# 1.9 Python - String Methods

Python's built-in **str** class defines different methods. They help in manipulating strings. Since string is an immutable object, these methods return a copy of the original string, performing the respective processing on it.

The string methods can be classified in following categories −

* [Case conversion](https://www.tutorialspoint.com/python/case_conversion.htm)
* [Alignment](https://www.tutorialspoint.com/python/alignment_methods.htm)
* [Split and join](https://www.tutorialspoint.com/python/split_and_join.htm)
* [Boolean](https://www.tutorialspoint.com/python/boolean_string_methods.htm)
* [Find and replace](https://www.tutorialspoint.com/python/find_and_replace.htm)
* Formatting
* [Translate](https://www.tutorialspoint.com/python/translate_methods.htm)

# Case Conversion Methods

This category of built-in methods of Python's **str** class deal with the conversion of alphabet characters in the string object. Following methods fall in this category −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | [**capitalize()**](https://www.tutorialspoint.com/python/capitalize_method.htm)  Capitalizes first letter of string |
| 2 | [**casefold()**](https://www.tutorialspoint.com/python/casefold_method.htm)  Converts all uppercase letters in string to lowercase. Similar to lower(), but works on UNICODE characters alos |
| 3 | [**lower()**](https://www.tutorialspoint.com/python/lower_method.htm)  Converts all uppercase letters in string to lowercase. |
| 4 | [**swapcase()**](https://www.tutorialspoint.com/python/swapcase_method.htm)  Inverts case for all letters in string. |
| 5 | [**title()**](https://www.tutorialspoint.com/python/title_method.htm)  Returns "titlecased" version of string, that is, all words begin with uppercase and the rest are lowercase. |
| 6 | [**upper()**](https://www.tutorialspoint.com/python/upper_method.htm)  Converts lowercase letters in string to uppercase. |

Let us discuss these methods with examples –

#1.9.1 Case Conversion Methods

#Example 1 Capitalize() method

"""The capitalize() method returns a string with only its first character capitalized."""

var="hi how are you"

mvar=var.capitalize()

print("Orignal String: ",var)

print("Modify String: ",mvar)

#Example 2 casefold() Method

"""

The casefold() method is a new addition to string methods. The .casefold() method returns a copy of a string with all characters in lowercase.

It is similar to .lower(), but whereas that method deals purely with ASCII text, .casefold() can also convert Unicode characters"""

var="I AM LIVING IN PAKISTAN"

mvar=var.casefold()

print("\nOrignal String: ",var)

print("Modify String: ",mvar)

#Example 3 lower() Method

mvar=var.lower()

print("\nOrignal String: ",var)

print("Modify String: ",mvar)

#Example 4 swapcase() Method

var2="This Is Python Practices"

mvar1=var2.swapcase()

print("\nOrignal String: ",var2)

print("Modify String: ",mvar1)

#Example 5 title and upper()

# 1.9.2 Alignment Methods

Following methods in the **str** class control the alignment of characters within the string object.

|  |  |
| --- | --- |
| **Sr.No.** | **Methods & Description** |
| 1 | [**center(width, fillchar)**](https://www.tutorialspoint.com/python/center_method.htm)  Returns a string padded with fillchar with the original string centered to a total of width columns. |
| 2 | [**ljust(width[, fillchar])**](https://www.tutorialspoint.com/python/ljust_method.htm)  Returns a space-padded string with the original string left-justified to a total of width columns. |
| 3 | [**rjust(width,[, fillchar])**](https://www.tutorialspoint.com/python/rjust_method.htm)  Returns a space-padded string with the original string right-justified to a total of width columns. |
| 4 | [**expandtabs(tabsize = 8)**](https://www.tutorialspoint.com/python/expandtabs_method.htm)  Expands tabs in string to multiple spaces; defaults to 8 spaces per tab if tabsize not provided. |
| 5 | [**zfill (width)**](https://www.tutorialspoint.com/python/zfill_method.htm)  Returns original string leftpadded with zeros to a total of width characters; intended for numbers, zfill() retains any sign given (less one zero). |

Here is the detailed explanation of each of the above methods –

# 1.9.2 Alignment Methods

"""Following methods in the str class control the alignment of characters within the string object.

Syntax

Following is the syntax for center() method −

"""

#Example 1 center(width[, fillchar])

var = "This Is String Example....WOW!!!"

var1 = var.center(40, '\*')

print ("original string:", var)

print ("Centered string:", var1)

#Example 2 ljust(width[, fillchar]) Method

var = "This Is String Example....WOW!!!"

var1 = var.ljust(40, '\*')

print ("\noriginal string:", var)

print ("Left justified string:", var1)

#Example 3 rjust(width[, fillchar]) Method

var = "This Is String Example....WOW!!!"

var1 = var.rjust(40, '\*')

print ("\noriginal string:", var)

print ("Rigth justified string:", var1)

#Example 4 expandtabs(tabsize = 8) Method

var = "this is\tstring example....wow!!!"

var1 = var.expandtabs()

var2 = var.expandtabs(24)

print ("\noriginal string:", var)

print ("string with default expanded tab:", var1)

print ("string with tripple expanded tab:", var2)

#Example zfill (width) Method

var="Hello python"

var1=var.center(40)

var2=var.ljust(40, '\*')

var3=var.rjust(40, '\*')

print ("original string: ", var)

print ("centered:", var1)

print ("left justified: ", var2)

print ("right justified:", var3)

var="Hello\tPython"

var4=var.expandtabs(16)

print ("capitalized:",var4)

var=-1234.50

var5=str(var).zfill(10)

print ("zfilled:", var5)

# 1.9.3 Split and Join Methods

Python has the following methods to perform split and join operations −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | [**lstrip()**](https://www.tutorialspoint.com/python/lstrip_method.htm)  Removes all leading whitespace in string. |
| 2 | [**rstrip()**](https://www.tutorialspoint.com/python/rstrip_method.htm)  Removes all trailing whitespace of string. |
| 3 | [**strip()**](https://www.tutorialspoint.com/python/strip_method.htm)  Performs both lstrip() and rstrip() on string |
| 4 | [**rsplit()**](https://www.tutorialspoint.com/python/rsplit_method.htm)  Splits the string from the end and returns a list of substrings |
| 5 | [**split()**](https://www.tutorialspoint.com/python/split_method.htm)  Splits string according to delimiter (space if not provided) and returns list of substrings. |
| 6 | [**splitlines()**](https://www.tutorialspoint.com/python/splitlines_method.htm)  Splits string at NEWLINEs and returns a list of each line with NEWLINEs removed. |
| 7 | [**partition()**](https://www.tutorialspoint.com/python/partition_method.htm)  Splits the string in three string tuple at the first occurrence of separator |
| 8 | [**rpartition()**](https://www.tutorialspoint.com/python/rpartition_method.htm)  Splits the string in three string tuple at the ladt occurrence of separator |
| 9 | **join()**  Concatenates the string representations of elements in sequence into a string, with separator string. |
| 10 | [**removeprefix()**](https://www.tutorialspoint.com/python/removeprefix_method.htm)  Returns a string after removing the prefix string |
| 11 | [**removesuffix()**](https://www.tutorialspoint.com/python/removesuffix_method.htm)  Returns a string after removing the suffix string |

# 1.9.3 Split and join method

#usign strip() for remove the white spases of start working of lstrip() and rstrip()

Svar="         Shahriyar khan"

Rvar=Svar.strip()

print("Orignal   String: ",Svar)

print("Istripped string: ",Rvar)

#using lstrip(remove spaces from start) and rstrip(from end)

num="0001234400"

num2=num.lstrip('0')

num3=num.rstrip('0')

print("Orignal String : ",num)

print("lStripped String: ",num2)

print("rStripped String: ",num3)

#using Split() This method returns a list of strings.

name="My name is shahriyar khan"

print("Orignal string: ",name)

print("list of split strings: ",name.split())

print("First three word split :",name.split(" ",3))

#using splitlines() This method returns a list of lines in the string

name1='''my

name

is

shahriyar

khan'''

splitline=name1.splitlines()

print("Orignal string: ",name1)

print("Splitlinse String in list: ",splitline)

#also use this

print("Splitlinse String in list: ",name1.splitlines(True))

#using removesuffic and removeprifex

print("Remove prefix 'my': ",name.removeprefix('My'))

# 1.9.4 Find and Replace Methods

Following are the Find and Replace methods in Python −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | [**count(sub, beg ,end)**](https://www.tutorialspoint.com/python/count_method.htm)  Counts how many times sub occurs in string or in a substring of string if starting index beg and ending index end are given. |
| 2 | [**find(sub, beg, end)**](https://www.tutorialspoint.com/python/find_method.htm)  Determine if sub occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise. |
| 3 | [**index(sub, beg, end)**](https://www.tutorialspoint.com/python/index_method.htm)  Same as find(), but raises an exception if str not found. |
| 4 | [**replace(old, new [, max])**](https://www.tutorialspoint.com/python/replace_method.htm)  Replaces all occurrences of old in string with new or at most max occurrences if max given. |
| 5 | [**rfind(sub, beg, end)**](https://www.tutorialspoint.com/python/rfind_method.htm)  Same as find(), but search backwards in string. |
| 6 | [**rindex( sub, beg, end)**](https://www.tutorialspoint.com/python/rindex_method.htm)  Same as index(), but search backwards in string. |
| 7 | [**startswith(sub, beg, end)**](https://www.tutorialspoint.com/python/startswith_method.htm)  Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring sub; returns true if so and false otherwise. |
| 8 | [**endswith(suffix, beg, end)**](https://www.tutorialspoint.com/python/endswith_method.htm)  Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise. |

# 1.9.5 Boolean String Methods

Following methods in str class return True or False.

|  |  |
| --- | --- |
| **Sr.No.** | **Methods & Description** |
| 1 | [**isalnum()**](https://www.tutorialspoint.com/python/isalnum_method.htm)  Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise. |
| 2 | [**isalpha()**](https://www.tutorialspoint.com/python/isalpha_method.htm)  Returns true if string has at least 1 character and all characters are alphabetic and false otherwise. |
| 3 | [**isdigit()**](https://www.tutorialspoint.com/python/isdigit_method.htm)  Returns true if the string contains only digits and false otherwise. |
| 4 | [**islower()**](https://www.tutorialspoint.com/python/islower_method.htm)  Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise. |
| 5 | [**isnumeric()**](https://www.tutorialspoint.com/python/isnumeric_method.htm)  Returns true if a unicode string contains only numeric characters and false otherwise. |
| 6 | [**isspace()**](https://www.tutorialspoint.com/python/isspace_method.htm)  Returns true if string contains only whitespace characters and false otherwise. |
| 7 | [**istitle()**](https://www.tutorialspoint.com/python/istitle_method.htm)  Returns true if string is properly "titlecased" and false otherwise. |
| 8 | [**isupper()**](https://www.tutorialspoint.com/python/isupper_method.htm)  Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise. |
| 9 | [**isascii()**](https://www.tutorialspoint.com/python/isascii_method.htm)  Returns True is all the characters in the string are from the ASCII character set |
| 10 | [**isdecimal()**](https://www.tutorialspoint.com/python/isdecimal_method.htm)  Checks if all the characters are decimal characters |
| 11 | [**isidentifier()**](https://www.tutorialspoint.com/python/isidentifier_method.htm)  Checks whether the string is a valid Python identifier |
| 12 | [**isprintable()**](https://www.tutorialspoint.com/python/isprintable_method.htm)  Checks whether all the characters in the string are printable |

Let us discuss these methods with examples −

# 1.9.6 Find and Replace Methods

Following are the Find and Replace methods in Python −

|  |  |
| --- | --- |
| **Sr.No.** | **Method & Description** |
| 1 | [**count(sub, beg ,end)**](https://www.tutorialspoint.com/python/count_method.htm)  Counts how many times sub occurs in string or in a substring of string if starting index beg and ending index end are given. |
| 2 | [**find(sub, beg, end)**](https://www.tutorialspoint.com/python/find_method.htm)  Determine if sub occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise. |
| 3 | [**index(sub, beg, end)**](https://www.tutorialspoint.com/python/index_method.htm)  Same as find(), but raises an exception if str not found. |
| 4 | [**replace(old, new [, max])**](https://www.tutorialspoint.com/python/replace_method.htm)  Replaces all occurrences of old in string with new or at most max occurrences if max given. |
| 5 | [**rfind(sub, beg, end)**](https://www.tutorialspoint.com/python/rfind_method.htm)  Same as find(), but search backwards in string. |
| 6 | [**rindex( sub, beg, end)**](https://www.tutorialspoint.com/python/rindex_method.htm)  Same as index(), but search backwards in string. |
| 7 | [**startswith(sub, beg, end)**](https://www.tutorialspoint.com/python/startswith_method.htm)  Determines if string or a substring of string (if starting index beg and ending index end are given) starts with substring sub; returns true if so and false otherwise. |
| 8 | [**endswith(suffix, beg, end)**](https://www.tutorialspoint.com/python/endswith_method.htm)  Determines if string or a substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise. |

# 1.9.6  Find and Replace Methods

#Example using count(sub,start,end)

uni="Abasyn university"

print("Orignal string: ",uni)

print(f"Count i {uni}: ",uni.count('i'))

print(f"Count 'i' in first 10 character {uni}: ",uni.count('i',0,10))

#Example find() Method

print("i is found at: ",uni.find('i'))

print("sity is found at: ",uni.find('sity',0,10))

#example replace() method

print("sity is replace with city:",uni.replace('sity','city'))

# 1.9.7 Translation Methods

## maketrans()

The maketrans() method returns a mapping table. It maps each character from astr to a character at same index in bstr. The mapping table returned by this method may be used by translate() method to replace the characters.

### **Syntax**

var.maketrans(astr, bstr, cstr)

### **Parameters**

* **astr** − This can be either dictionary or string. If only one parameter is supplied, this parameter must be a dictionary. If two or more parameters are given, this parameter has to be a string specifying the characters to be replaced.
* **bstr** − This is the string having corresponding mapping character.
* **cstr** − Optional. The string parameter specifies the characters to remove in the string.

### **Return Value**

This method returns a translate table to be used translate() function.

# 1.9.5 Translation Methods

# maketrans()

"""

The maketrans() method returns a mapping table.

It maps each character from astr to a character at same index in bstr.

The mapping table returned by this method may be used by translate() method to replace the characters.

Syntax

var.maketrans(astr, bstr, cstr)"""

name3="Explicit is better than implicit "

#translation table creation

table=name3.maketrans({"i":"I"})#his line creates a translation table using the maketrans method. The dictionary {"i": "I"} specifies that every occurrence of the letter 'i' in the string should be replaced with 'I'. The resulting table variable holds the translation table.

#Printing Original String and Translation Table:

print("\nOrignal String :",name3)

print("Translation table: ",table)

#String Translation:

var3=name3.translate(table)#This line applies the translation table to the original string using the translate method. It replaces every 'i' with 'I' according to the specified translation rule.

print("Translated string: ",var3)

#Second example

table1=name3.maketrans("than","THEN")

print("Tranlation table: ",table1)

var4=name3.translate(table1)

print("Translated string: ",var4)

#Third Example

#where 'i' is mapped to 'a', and 's' is removed.

tableone = name3.maketrans("is","as", "s")

print ("original string:", name3)

print ("translation table:", tableone)#This line uses the translate method to apply the translation table to the original string (var). It replaces 'i' with 'a' and removes 's' based on the specified translation rules."""

var5=name3.translate(tableone)

print ("Translated string", var5)

# Ch#5 Python - Lists

List is one of the built-in data types in Python. A Python list is a sequence of comma separated items, enclosed in square brackets [ ]. The items in a Python list need not be of the same data type.

Following are some examples of Python lists −

list1 = ["Rohan", "Physics", 21, 69.75]

list2 = [1, 2, 3, 4, 5]

list3 = ["a", "b", "c", "d"]

list4 = [25.50, True, -55, 1+2j]

In Python, a list is a sequence data type. It is an ordered collection of items. Each item in a list has a unique position index, starting from 0.

A list in Python is similar to an array in C, C++ or Java. However, the major difference is that in C/C++/Java, the array elements must be of same type. On the other hand, Python lists may have objects of different data types.

A Python list is mutable. Any item from the list can be accessed using its index, and can be modified. One or more objects from the list can be removed or added. A list may have same item at more than one index positions.

## Python List Operations

In Python, List is a sequence. Hence, we can concatenate two lists with "+" operator and concatenate multiple copies of a list with "\*" operator. The membership operators "in" and "not in" work with list object.

|  |  |  |
| --- | --- | --- |
| **Python Expression** | **Results** | **Description** |
| [1, 2, 3] + [4, 5, 6] | [1, 2, 3, 4, 5, 6] | Concatenation |
| ['Hi!'] \* 4 | ['Hi!', 'Hi!', 'Hi!', 'Hi!'] | Repetition |
| 3 in [1, 2, 3] | True | Membership |

# 5.1 Python - Access List Items

In Python, a list is a sequence. Each object in the list is accessible with its index. The index starts from 0. Index or the last item in the list is "length-1". To access the values in a list, use the square brackets for slicing along with the index or indices to obtain value available at that index.

The slice operator fetches one or more items from the list. Put index on square brackets to retrieve item at its position.

obj = list1[i]

# 5.1 Access List Items

#Example 1

list1=["Shary","Umair",1,2,3,4,6.7]

list2=[1,2,3,4,5]

print("Item at index1 in list1: ",list1[1])#umair

print("Item at index4 in list2: ",list1[4])#3

#Example 2 Python allows negative index to be used with any sequence type. The "-1" index refers to the last item in the list.

print("item at index 0 in list1:",list1[-1])#6.7

print("item at index 0 in list2:",list1[-3])#3

#Example 3 Slicing

"""The slice operator extracts a sublist from the original list."""

print("item from index 0 to 2 :",list1[0:3])#Shary and umair

print("item from index 4 to -1 :",list2[3:-1])#4

#Example 4

print("items from start 1 to end in list1: ",list1[1:])

print("items from index 0 to 2 in list2: ",list2[:3])#1 2 3

# 5.2 Python - Change List Items

List is a mutable data type in Python. It means, the contents of list can be modified in place, after the object is stored in the memory. You can assign a new value at a given index position in the list

### **Syntax**

list1[i] = newvalue

#Example 1 we change the value at index 2 of the given list.

list1=[1,2,3,4,5,6,7]

print("Orignal List: ",list1)

#changing the index 2 value with 8

list1[2]=8

print("List after changing value at index 2:",list1)#[1, 2, 8, 4, 5, 6, 7]

#Example 2  items at index 0 and 2 are replaced by items in another sublist.

name=['S','H','a','r','y']

print("orignal list: ",name)

list2=['s','h']

name[0:2]=list2

print("list after changing with sublist: ",name)# ['s', 'h', 'a', 'r', 'y']

#Example 3

"""If the source sublist has more items than the slice to be replaced,

the extra items in the source will be inserted. Take a look at the following code"""

num1=[1,2,3,7,8,9]

print("orignal string: ",num1)

num2=[4,5,6]

num1[3:4]=num2

print('list after changing with sublist: ',num1)#[1, 2, 3, 4, 5, 6, 8, 9]

#Example 4

"""

If the sublist with which a slice of original list is to be replaced, has lesser items,

the items with match will be replaced and rest of the items in original list will be removed."""

list3=['a','b','c','d']

print("Orignal list: ",list3)

list3[1:3]='X'

print("list after changing b and c with X:",list3)

# 5.3 Python - Add List Items

There are two methods of the **list** class, append() and insert(), that are used to add items to an existing list.

#Add list items

#Example 1  The append() method adds the item at the end of an existing list

list1=[1,2,3,4,5]

print("orignal list: ",list1)

list1.append(6)

print("list after appending: ",list1)#[1,2,3,4,5,6]

#Example 2 The insert() method inserts the item at a specified index in the list.

list2=["database","HCI",1,2,3,4]

print("Orignal list: ",list2)

list2.insert(2,"MAD")

print("List after inserting at index 2 :",list2)

list2.insert(-1,"English")

print("List after inserting at index 4 :",list2)

# 5.4 Python - Remove List Items

The list class methods **remove()** and **pop()** both can remove an item from a list. The difference between them is that remove() removes the object given as argument, while pop() removes an item at the given index.

In Python, **remove**, **pop**, and **del** are methods used to remove elements from data structures, but they have different use cases and behaviors.

1. **remove() method**:
   * Used with lists.
   * Removes the first occurrence of a specified value.
   * Raises a **ValueError** if the specified value is not found in the list.
   * Syntax: **list.remove(value)**

**pop() method**:

* Used with lists.
* Removes and returns the element at the specified index.
* If no index is provided, it removes and returns the last element.
* Raises an **IndexError** if the index is out of range.
* Syntax: **removed\_element = list.pop([index])**

**del statement**:

* Used to delete elements from lists, dictionaries, or variables.
* Can delete elements based on index or slices.
* Doesn't return the removed element(s); it simply removes them.
* Syntax: **del list[index]** or **del list[start:stop]**

#Using the remove method

# shows how you can use the remove() method to remove list items −

list1=["Shary","khan",1,2,3,4.5]

print("Orignal string: ",list1)

#remove is use for given specific item delete

list1.remove('khan')

print("List after removing : ",list1)

#Using the pop() Method

"""

shows how you can use the pop() method to remove list items −

"""

print("\Origna string :",list1)

#pop using index

list1.pop(2)

print("List after popping index 2: ",list1)

#Using the "del" Keyword

""" Python has the "del" keyword that deletes any Python object from the memory."""

print("Orignal string: ",list1)

#del to delete the list items

del list1[0]

print("List after delating item at index 0: ",list1)

#we can also delete sreies of items like slicing

del list1[0:2]

print("List after delete item from index 0 to 2 : ",list1)

# 5.5 Python - Loop Lists

You can traverse the items in a list with Python's **for** loop construct. The traversal can be done, using list as an iterator or with the help of index.

### **Syntax**

Python list gives an iterator object. To iterate a list, use the for statement as follows −

for obj in list:

. . .

. . .

#Example 1

list1=[1,2,3,4,7,6,5]

for num in list1:

    print(num,end=' ')

#Example 2

"""

To iterate through the items in a list, obtain the range object of integers "0" to "len-1"""

list2=[5,10,15,20,25]

indexes=range(len(list2))

for i in indexes:

    print(f"List[{i}]: {list2[i]}")

# 5.6 Python - List Comprehension

List comprehension is a very powerful programming tool. It is similar to set builder notation in mathematics. It is a concise way to create new list by performing some kind of process on each item on existing list. List comprehension is considerably faster than processing a list by **for** loop.

#Example 1

"""Suppose we want to separate each letter in a string and put all non-vowel letters in a list object."""

string="Shahriyar khan"

char=[]

vowels='aeiou'

for x in string:

    if x not in vowels:

        char.append(x)

print("List without vowels: ",char)

#List Comprehension Technique

"""

We can easily get the same result by a list comprehension technique. A general usage of list comprehension is as follows −"""

chars=[char for char in 'Shahriyar khan ' if char not in 'aioue']

print("List :",char)

#Example 2

"""

Use list comprehension to build a list of squares of numbers between 1 to 10"""

num=[x\*x for x in range(1,11)]

print(num)

#Nested Loops in List Comprehension

"""

all combinations of items from two lists in the form of a tuple are added in a third list object."""

list1=[1,2,3,4]

list2=[4,5,6,7]

comblst=[(x,y) for x in list1 for y in list2]

print("combinations of items from two lists:  \n",comblst)

#Condition in List Comprehension

"""

The following statement will create a list of all even numbers between 1 to 20."""

list1=[x for x in range(1,21) if x%2==0]

print("List of even numbers: ",list1)

# 5.7 Python - Sort Lists

The sort() method of list class rearranges the items in ascending or descending order with the use of lexicographical ordering mechanism. The sorting is in-place, in the sense the rearrangement takes place in the same list object, and that it doesn't return a new object.

### **Syntax**

list1.sort(key, reverse)

### **Parameters**

* **Key** − The function applied to each item in the list. The return value is used to perform sort. Optional
* **reverse** − Boolean value. If set to True, the sort takes place in descending order. Optional

#Example 1

"""

Now let's take a look at some examples to understand how we can sort lists in Python

−"""

list1=['Physics','Biology','chemistry','maths']

print("List before sort: ",list1)

list1.sort()

print("List after sorted list: ",list1)

list2=[10,16,9,5,24]

print("List before sort: ",list2)

list2.sort()

print("List after sorted : ",list2)

#Example 2

"""

the str.lower() method is used as key parameter in sort() method.

"""

list3=["shahriyar","Ali","Muhammad","Umair","Hamsafer"]

print("Orignal list: ",list3)

list3.sort(key=str.lower)

print("list after sorted: ",list3)

#Example 3

"""

The myfunction() uses % operator to return the remainder,

based on which the sort is done."""

def myfunction(x):

    result=x%10

    return result

list4=[23,17,46,51,90]

print("Orignal list: ",list4)

list4.sort(key=myfunction)

print("Sorted list: ",list4)

# 5.8 Python - Copy Lists

In Python, a variable is just a label or reference to the object in the memory. Hence, the assignment "lst1 = lst" refers to the same list object in the memory. Take a look at the following example −

#Example 1

list1=[10,20]

print("list1: ",list1,"Id(list)",id(list1))

list2=list1 #same memeory address because only refrence

print("list2: ",list2,"Id(list)",id(list2))

"""As a result, if we update "lst", it will automatically reflect in "lst1". Change lst[0] to 100"""

list1[0]=100

print("list1: ",list1,"Id(list)",id(list1))

print("list2: ",list2,"Id(list)",id(list2))#Hence, we can say that "lst1" is not the physical copy of "lst".

#Using the Copy Method of List Class

"""

Python's list class has a copy() method to create a new physical copy of a list object."""

list3=[1,2,10,34]

list4=list3.copy()

print("List3 :",list3,"id(list3):",id(list3))

print("List4 :",list4,"id(list4):",id(list4))

# 5.9 Python - Join Lists

In Python, List is classified as a sequence type object. It is a collection of items, which may be of different data types, with each item having a positional index starting with 0. You can use different ways to join two Python lists.

All the sequence type objects support concatenation operator, with which two lists can be joined.

#Example 1 (+) used for joined two list

list1=[10,20,30,40,50,60]

list2=["Khan","Umair","Ali"]

list3=list1+list2

print("Joined List1(+): ",list3)

#Example 2  use the augmented concatenation operator with "+=" symbol to append L2 to L1

#the above joined also

list1+=list2

print("\nJoined List2(+=): ",list1)

#Also using Extend() method

list2.extend(list1)

print("\nJoined List3(Extend): \n",list2)

#Using append() method to joined two lists

list5=[1,2,3,4,5]

list6=['one','two','three','four','five']

for x in list6:

    list5.append(x)

print("Joined list (append): ",list5)

# 5.10 Python - List Methods

Python's **list** class includes the following methods using which you can add, update, and delete list items −

|  |  |
| --- | --- |
| **Sr.No** | **Methods & Description** |
| 1 | **list.append(obj)**  Appends object obj to list |
| 2 | [**list.clear()**](https://www.tutorialspoint.com/python/python_clear_a_List.htm)  Clears the contents of list |
| 3 | **list.copy()**  Returns a copy of the list object |
| 4 | [**list.count(obj)**](https://www.tutorialspoint.com/python/python_counting_the_list_items.htm)  Returns count of how many times obj occurs in list |
| 5 | [**list.extend(seq)**](https://www.tutorialspoint.com/python/python_extending_a_list.htm)  Appends the contents of seq to list |
| 6 | [**list.index(obj)**](https://www.tutorialspoint.com/python/python_finding_the_index_list_item.htm)  Returns the lowest index in list that obj appears |
| 7 | **list.insert(index, obj)**  Inserts object obj into list at offset index |
| 8 | **list.pop(obj=list[-1])**  Removes and returns last object or obj from list |
| 9 | **list.remove(obj)**  Removes object obj from list |
| 10 | [**list.reverse()**](https://www.tutorialspoint.com/python/python_reverse_a_list.htm)  Reverses objects of list in place |
| 11 | **list.sort([func])**  Sorts objects of list, use compare func if given |

#Example 1 clear() methods  clears only contents of list not removed from memory

# Removes all elements from the list, making it empty.

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

print("before clear list: ",list)

list.clear()

print("After clear the list: ",list)

#del is used to delete list permanantly

del list

print("After delete the list: ",list)

#Example list count() method

list2=[4,3,5,2,3]

print("Count of 3: ",list2.count(3))

#Example list Extend method

list2.extend(["Khan","Ali"])

print("After extended the list: ",list2)

#Example list index() method

print("First index of 'Khan': ",list2.index('Khan'))

#Example list insert() method

# insert using for specific index insertion

list2.insert(0,'Fahad')

print("After inserted list: ",list2)

#Example list pop()  Removes the element at the specified index (or the last element if no index is provided) and returns it.

print("Popped list: ",list2.pop(0))

print(list2)

#Example list append()

# Appends a single element at the end of the list.

list2.append(33)

print("Appended list: ",list2)

#Example list Extend()   multple elements from another list append

list2.extend([44,55,66])

print("Extended List: ",list2)

#Example list remove() method

#Removes the first occurrence of a specified value.  Removes the specified value from the list. Raises a ValueError if the value is not found.

print("Orignal list: ",list2)

list2.remove(3)

print("After first occurense removing from list: ",list2)

#Example list reverse() method

list2.reverse()

print("Reversed the list: ",list2)

#Example 1 Python program to find unique numbers in a given list."""

list1=[1,2,3,2,4,4,5,6,5,2]

u\_nums=[]

for x in list1:

    if x not in u\_nums:

        u\_nums.append(x)

print("Orignal list: ",list1)

print("Unique numbers in list: ",u\_nums)

#Example 2  Python program to find sum of all numbers in a list.

total=0

for x in u\_nums:

    total+=x

print("Orignal list: ",u\_nums)

print("Sum of all elements in List: ",total)

#Example 3 Python program to create a list of 5 random integers.

import random

list2=[]

for i in range(5):

     x=random.randint(0,100)

     list2.append(x)

print("Random List: ",list2)

#Example 4 Python program to remove all odd numbers from a list.

l\_Mix=[1,2,3,4,5,6,7,8,9]

l\_even=[]

for i in l\_Mix:

    if i%2==0:

        l\_even.append(i)

print("Orignal list: ",l\_Mix)

print("Even list: ",l\_even)

#Example 5 Python program non-numeric items in a list in a separate list.

list3=['a','b','c','d',1,2,3]

l\_num=[]

for i in list3:

    if not isinstance (i,(int,float)) :

        l\_num.append(i)

print("Orignal list: ",list3)

print("Non numaric item in a list: ",l\_num)

#Example 6 Python program to sort a list of strings on the number of alphabets in each word.

def sort\_by\_alphaCount(input\_l):

    sorted\_list=sorted(input\_l,key=lambda x: sum(c.isalpha() for c in x))

    return sorted\_list

list4=['apple','Banana','Orange','graps','mangoes']

sorted\_list=sort\_by\_alphaCount(list4)

print("Orignal list: ",list4)

print("Sorted list: ",sorted\_list)

#Example 7 Python program to create a list of integers representing each character in a string

def char\_to\_int(input\_list):

    #using list comprehension to convert character to ASCII values

    int\_list=[ord(char) for char in input\_list]

    return int\_list

mylist="Shahriyar khan"

print("Orignal list:",mylist)

reultlist=char\_to\_int(mylist)

print("list of ASCII values: ",reultlist)

#Example 8 Python program to find numbers common in two lists.

#Method 1 using intersection method

l1=[1,2,3,4,5,6]

l2=[2,3,6,4,8,9]

#Using list comprehension

c\_list=[x for x in l1 if x in l2]

print("List1 : ",l1)

print("List2: ",l2)

print("Commen in both list: ",c\_list)

#using for loop

c\_list2=[]

for x in l1:

    if x in l2:

        c\_list2.append(x)

print("Common in both list(for): ",c\_list2)

# Ch# 6 Python - Tuples

Tuple is one of the built-in data types in Python. A Python tuple is a sequence of comma separated items, enclosed in parentheses (). The items in a Python tuple need not be of same data type.

Following are some examples of Python tuples −

tup1 = ("Rohan", "Physics", 21, 69.75)

tup2 = (1, 2, 3, 4, 5)

tup3 = ("a", "b", "c", "d")

tup4 = (25.50, True, -55, 1+2j)

In Python, tuple is a sequence data type. It is an ordered collection of items. Each item in the tuple has a unique position index, starting from 0.

In C/C++/Java array, the array elements must be of same type. On the other hand, Python tuple may have objects of different data types.

Python tuple and list both are sequences. One major difference between the two is, Python list is mutable, whereas tuple is immutable. Although any item from the tuple can be accessed using its index, and cannot be modified, removed or added.

## Python Tuple Operations

In Python, Tuple is a sequence. Hence, we can concatenate two tuples with + operator and concatenate multiple copies of a tuple with "\*" operator. The membership operators "in" and "not in" work with tuple object.

|  |  |  |
| --- | --- | --- |
| **Python Expression** | **Results** | **Description** |
| (1, 2, 3) + (4, 5, 6) | (1, 2, 3, 4, 5, 6) | Concatenation |
| ('Hi!',) \* 4 | ('Hi!', 'Hi!', 'Hi!', 'Hi!') | Repetition |
| 3 in (1, 2, 3) | True | Membership |

Note that even if there is only one object in a tuple, you must give a comma after it. Otherwise, it is treated as a string.

# 6.1 Python - Access Tuple Items

In Python, Tuple is a sequence. Each object in the list is accessible with its index. The index starts from "0". Index or the last item in the tuple is "length-1". To access values in tuples, use the square brackets for slicing along with the index or indices to obtain value available at that index.

The slice operator fetches one or more items from the tuple.

#Example 1  by index

tuple1=("Shahy","khan","Ali",1,2,3)

tuple2=(1,2,3,4,5)

print("Item at 1 index in tuple1: ",tuple1[1])

print("Item at 3 index in tuple2: ",tuple2[3])

#Example 2 by negative index

print("Item at last index in tuple : ",tuple1[-1])

#Example 3 Extracting a Subtuple from a Tuple

# The slice operator extracts a subtuple from the original tuple.

print("Item from index 0 to 3 in tuple1: ",tuple1[0:4])

print("Item from index 0 to 3 in tuple1: ",tuple1[0:-2])

# 6.2 Python - Update Tuples

In Python, tuple is an immutable data type. An immutable object cannot be modified once it is created in the memory.

it is not possible to update a tuple. Therefore, the tuple class doesn't provide methods for adding, inserting, deleting, sorting items from a tuple object, as the list class.

**How to Update a Python Tuple?**

You can use a work-around to update a tuple. Using the list() function, convert the tuple to a list, perform the desired append/insert/remove operations and then parse the list back to tuple object.

#Example tuple i simmutable so we convert to list

tuple1=(1,2,3,4,5,6)

print("tuple before update: ",tuple1,"id(): ",id(tuple1))

#Convert tuple to dictionary

list1=list(tuple1)

#now we can modify

#changing value of index 0

list1[0]=10

#appending value at the end

list1.append(77)

#sorted in decending order

list1.reverse()

print("Update List: ",list1)

#list convert to tuple

tuple1=tuple(list1)

print("Updated tuple: ",tuple1,"id():",id(tuple1))

# 6.3 Python - Unpack Tuple Items

The term "unpacking" refers to the process of parsing tuple items in individual variables. In Python, the parentheses are the default delimiters for a literal representation of sequence object.

#Example 1 To store tuple items in individual variables, use multiple variables on the left of assignment operator

tuple1=(2,4,6,8)

a,b,c,d=tuple1

print("a:",a," b:",b," C:",c," d:",d)

#Example 2 If the number of variables is more or less than the length of tuple, Python raises a ValueError.

# a1,b1=tuple1

# print("a1:",a1,"b1:",b1)

#the "\*" symbol is used for unpacking. Prefix "\*" to "y", as

a1,\*b1=tuple1

print("a1:",a1,"b1: ",b1)

#Example 3  the tuple contains 6 values and variables to be unpacked are 3. We prefix "\*" to the second variable.

tuple2=(1,2,3,4,5,6)

x,\*y,z=tuple2

print("x:",x,"y:",y,"z:",z)

# 6.4 Python - Loop Tuples

You can traverse the items in a tuple with Python's **for** loop construct. The traversal can be done, using tuple as an iterator or with the help of index.

#Example 1 The following example shows a simple Python for loop construct −

tuple1=(2,3,5,7,8,9)

for i in tuple1:

    print(i,end=' ')

#Example 2 To iterate through the items in a tuple, obtain the range object of integers "0" to "len-1"

indexes=range(len(tuple1))

for i in indexes:

    print("tuple1[{}]:".format(i),tuple1[i])

# 6.5 Python - Join Tuples

In Python, a Tuple is classified as a sequence type object. It is a collection of items, which may be of different data types, with each item having a positional index starting with 0. Although this definition also applies to a list, there are two major differences in list and tuple. First, while items are placed in square brackets in case of List (example: [10,20,30,40]), the tuple is formed by putting the items in parentheses (example: (10,20,30,40)).

In Python, a Tuple is an immutable object. Hence, it is not possible to modify the contents of a tuple one it is formed in the memory.

tuple1=(10,20,30,40)

tuple2=("one","two","Three","Four")

t3=tuple1+tuple2

print("Joined tuple: ",t3)

#Example 2 You can also use the augmented concatenation operator with the "+=" symbol to append T2 to T1

tuple1+=tuple2

#Example 3

"""

The same result can be obtained by using the extend() method.

Here, we need cast the two tuple objects to lists,

extend so as to add elements from one list to another,

and convert the joined list back to a tuple."""

#Convert to list

l1=list(tuple1)

l2=list(tuple2)

l1.extend(l2)

tResult=tuple(l1)

print("The Extended tuple: ",tResult)

#Example 4 Python's built-in sum() function also helps in concatenating tuples. We use an expression

t1=(1,2,3)

t2=("a","b")

t3=sum((t1,t2),())

print("Joined tuple: ",t3)

#Example 5 A slightly complex approach for merging two tuples is using list comprehension

#convert to list

l1,l2=list(t1),list(t2)

#using list comprehension

l3=[y for x in [l1,l2] for y in x]

t3=tuple(l3)

print("Joined tuple (list comprehension): ",t3)

#Example 6 You can run a for loop on the items in second loop, convert each item in a single item tuple and concatenate it to first tuple with the "+=" operator

for i in t2:

    t1+=(i,)

print("joined tuple(for loop): ",t1)

# 6.6 Python - Tuple Methods

Since a tuple in Python is immutable, the tuple class doesn't define methods for adding or removing items. The tuple class defines only two methods.

|  |  |
| --- | --- |
| **Sr.No** | **Methods & Description** |
| 1 | **tuple.count(obj)**  Returns count of how many times obj occurs in tuple |
| 2 | **tuple.index(obj)**  Returns the lowest index in tuple that obj appears |

#Example of index

t1=(1,2,6,4,5,6,5,5)

print("Tuple1: ",t1)

x=t1.index(6)

print("First index of 6: ",x)

#Example Counting Tuple Items The count() method in tuple class returns the number of times a given object occurs in the tuple.

c=t1.count(5)

print("Count of 5: ",c)

#Example 1 Python program to find unique numbers in a given tuple

t1=(1,2,2,3,4,3,5,6,5,7,7,8,9,2)

t2=()

for x in t1:

    if x not in t2:

        t2+=(x,)

print("Orignal tuple: ",t1)

print("Unique numbers: ",t2)

#Example 2 Example 2 Python program to find sum of all numbers in a tuple −

sum=0

for i in t1:

    sum+=i

print("Sum of all numbers in tuple: ",sum)

#Example 3 Python program to create a tuple of 5 random integers −

import random

t3=()

for i in range(5):

    x=random.randint(0,100)

    t1+=x,

print("Tuple: ",t1)

#Question 1 Python program to remove all duplicates numbers from a list.

t1=(1,2,2,3,7,4,4,1,8)

t2=()

for i in t1:

    if i not in t2:

        t2+=(i,)

print("Tuple without duplicates: ",t2)

#Question 2 Python program to sort a tuple of strings on the number of alphabets in each word.

def num\_of\_alpha(tS):

    sorted\_t = sorted(tS, key=lambda x: sum(y.isalpha() for y in x))

    return sorted\_t

input\_t = ("Shahriyar", "Umair", "Fahad", "Ali")

result = num\_of\_alpha(input\_t)

print("Original tuple:", input\_t)

print("Sorted tuple:", result)

#Question 3 Python program to prepare a tuple of non-numeric items from a given tuple.

# 7 Python - Sets

A set is one of the built-in data types in Python. In mathematics, set is a collection of distinct objects. Set data type is Python's implementation of a set. Objects in a set can be of any data type.

Set in Python also a collection data type such as list or tuple. However, it is not an ordered collection, i.e., items in a set or not accessible by its positional index. A set object is a collection of one or more immutable objects enclosed within curly brackets {}.

#Example 1

s1={"Rohan","Physics",1,2,3,4.5}

s2={'a','b',True,-55,1+4j}

print(s1)

print(s2)

#set() Function

"""

set() is one of the built-in functions. It takes any sequence object (list, tuple or string) as argument and returns a set object"""

l1=[1,2,4,5]

t1=('a','b','c')

s1="Shahriyar khan"

print(set(s1))

print(set(l1))

print(set(t1))

#Example 3

"""

Set is a collection of distinct objects. Even if you repeat an object in the collection,

only one copy is retained in it"""

s2={1,2,2,3,3,4,5}

print(s2)

# 7.1 Python - Access Set Items

Since set is not a sequence data type, its items cannot be accessed individually as they do not have a positional index (as in list or tuple). Set items do not have a key either (as in dictionary) to access. You can only traverse the set items using a **for** loop.

#Example 1

langs={'C','C+','JAVA','Python'}

#Printin one by one

for lang in langs:

    print(lang)

#Example 2 Python's membership operators let you check if a certain item is available in the set.

print('c' in langs)#false

print('C+' in langs)#True

# 7.2 Python - Add Set Items

Even if a set holds together only immutable objects, set itself is mutable. We can add new items in it with any of the following ways −

## add() Method

The add() method in set class adds a new element. If the element is already present in the set, there is no change in the set.

### **Syntax**

set.add(obj)

### **Parameters**

* **obj** − an object of any immutable type.

#Example

langs={"C",'C+','Java','Python'}

print("Orignal set: ",langs)

langs.add("Php")

print("Modify Set: ",langs)

#update() Method

"""

The update() method of set class includes the items of the set given as argument.

If elements in the other set has one or more items that are already existing,

they will not be included."""

#Example 2

lang2={'Perl','Pascal','C+'}

langs.update(lang2)

print("Update Set: ",langs)

#Example 3

"""

The update() method also accepts any sequence object as argument. Here,

a tuple is the argument for update() method."""

t1=(1,2,3)

langs.update(t1)

print(langs)

#Example 4

"""

Example

In this example, a set is constructed from a string, and another string is used as argument for update() method.

"""

set1=set("Hello")

set1.update("World")

print("From string to set: ",set1)

#union() Method

"""The union() method of set class also combines the unique items from two sets, but it returns a new set object."""

langs3=langs.union(lang2)

print("Union of two sets: ",langs3)

#Example

""" a set is constructed from a string, and another string is used as argument for union() method."""

set3=(set("Hello"))

set3.union("World")

print(set3)

# 7.3 Python - Remove Set Items

Python's set class provides different methods to remove one or more items from a set object.

## remove() Method

The remove() method removes the given item from the set collection, if it is present in it. However, if it is not present, it raises KeyError.

#Example The intersection() method returns a set object, retaining only those items common in itself and obj.

s1={1,2,3,4,5}

s2={3,4,5,6,7}

s3=s1.intersection(s2)

print("S1: ",s1," S2: ",s2)

print("The common in two sets: ",s3)

s5=s2 & s1

print("Common Item in two sets: ",s5)

#symmetric\_difference\_update() method

"""

The symmetric difference between two sets is the collection of all the uncommon items, rejecting the common elements.

The symmetric\_difference\_update() method updates a set with

symmetric difference between itself and the set given as argument.

"""

s1.symmetric\_difference\_update(s2)

print("S1 after running symmetric difference: ",s1)

#symmetric\_difference() Method

"""

The symmetric\_difference() method in set class is similar to symmetric\_difference\_update() method,

except that it returns a new set object that holds all the items from two sets minus the common items."""

s4=s1.symmetric\_difference(s2)

print("Symmetrics diff: ",s4)

#also worked ^

s6=s1^s2

print("Symmetrics diff(^): ",s6)

# 7.4 Python - Loop Sets

A set in Python is not a sequence, nor is it a mapping type class. Hence, the objects in a set cannot be traversed with index or key. However, you can traverse each item in a set using a **for** loop.

#Example 1

langs={'C','C++','C#',"Python","java"}

for i in langs:

    print(i)

#Example 2

"""

The following example shows how you can run a for loop over the elements of one set, and use the add() method of set class to add in another set."""

s1={1,2,3,4,5}

s2={5,6,7,8,9}

for i in s2:

    s1.add(i)

print("Added two Sets: ",s1)

# 7.5 Python - Join Sets

In Python, a Set is an ordered collection of items. The items may be of different types. However, an item in the set must be an immutable object. It means, we can only include numbers, string and tuples in a set and not lists. Python's set class has different provisions to join set objects.

#Using the "|" Operator

"""

The "|" symbol (pipe) is defined as the union operator. It performs the A∪B operation and returns a set of items in A, B or both.

Set doesn't allow duplicate items."""

s1={1,2,2,2,3,4,5,6}

s2={6,7,8,9}

s3=s1|s2

print("The Union of two sets: ",s3)

#Using the union() Method

"""

The set class has union() method that performs the same operation as | operator. It returns a set object that holds all items in both sets, discarding duplicates."""

s4=s1.union(s2)

print("The Union of two sets(u): ",s4)

#Using the update() Method

"""The update() method also joins the two sets, as the union() method.

However it doen't return a new set object.

Instead, the elements of second set are added in first,

duplicates not allowed."""

s1.update(s2)

print("The updated set: ",s1)

#Using the unpacking Operator

"""the "\*" symbol is used as unpacking operator. The unpacking operator internally assign each element in a collection to a separate variable."""

s11={1,2}

s5=\*s11,\*s2

print("Unpacked sets: ",s5)

# 7.6 Python - Copy Sets

The copy() method in set class creates a shallow copy of a set object.

### **Syntax**

set.copy()

### **Return Value**

The copy() method returns a new set which is a shallow copy of existing set.

lang1 = {"C", "C++", "Java", "Python"}

print ("lang1: ", lang1, "id(lang1): ", id(lang1))

lang2 = lang1.copy()

print ("lang2: ", lang2, "id(lang2): ", id(lang2))

lang1.add("PHP")

print ("After updating lang1")

print ("lang1: ", lang1, "id(lang1): ", id(lang1))

print ("lang2: ", lang2, "id(lang2): ", id(lang2))

# 7.7 Python - Set Operators

In the Set Theory of Mathematics, the union, intersection, difference and symmetric difference operations are defined. Python implements them with following operators −

## Union Operator (|)

The union of two sets is a set containing all elements that are in A or in B or both. For example,

{1,2}∪{2,3}={1,2,3}

Python uses the "|" symbol as a union operator. The following example uses the "|" operator and returns the union of two sets

s1={1,2,3,4,5,5}

s2={1,3,6,7,8}

s3=s1|s2

#also use union method

s3=s1.union(s2)

print(f"{s1} U {s2}= {s3}")

#Intersection Operator (&)

"""

The intersection of two sets AA and BB, denoted by A∩B, consists of all elements that are both in A and B. For example,"""

s5=s1 & s1

print(f"{s1} intersection {s2}={s5}")

#Difference Operator (-)

"""

The difference (subtraction) is defined as follows. The set A−B consists of elements that are in A but not in B. For example,"""

s6=s1-s2

s7=s2-s1

print(f"Difference of s1-s2 \n {s1} - {s2}={s6}")

print(f"Difference of s1-s2 \n {s2} - {s1}={s7}")

#Symmetric Difference Operator

"""

The symmetric difference of A and B is denoted by "A Δ B" and is defined by

A Δ B = (A − B) ⋃ (B − A)

Python uses the "^" symbol as a symbolic difference operator."""

s8=s1^s2

print(f"S\_Difference of s1-s2:\n{s1} ^ {s2}= {s8}")

# 7.8 Python - Set Methods

Following methods are defined in Python's set class −

|  |  |
| --- | --- |
| **Sr.No.** | **Methods & Description** |
| 1 | **add()**  Add an element to a set. |
| 2 | **clear()**  Remove all elements from this set. |
| 3 | **copy()**  Return a shallow copy of a set. |
| 4 | **difference()**  Return the difference of two or more sets as a new set. |
| 5 | **difference\_update()**  Remove all elements of another set from this set. |
| 6 | **discard()**  Remove an element from a set if it is a member. |
| 7 | **intersection()**  Return the intersection of two sets as a new set. |
| 8 | **intersection\_update()**  Update a set with the intersection of itself and another. |
| 9 | **isdisjoint()**  Return True if two sets have a null intersection. |
| 10 | **issubset()**  Return True if another set contains this set. |
| 11 | **issuperset()**  Return True this set contains another set. |
| 12 | **pop()**  Remove and return an arbitrary set element |
| 13 | **remove()**  Remove an element from a set; it must be a member. |
| 14 | **symmetric\_difference()**  Return the symmetric difference of two sets as a new set. |
| 15 | **symmetric\_difference\_update()**  Update a set with the symmetric difference of itself and another. |
| 16 | **union()**  Return the union of sets as a new set. |
| 17 | **update()**  Update a set with the union of itself and others. |

#Example 1 Add()

s1={1,2,3,4}

s1.add(5)

s1.add("khan")

s11=(6,7)

s1.add(s11)

print("Added set: ",s1)

#Example 2 clear() Remove all elements from this set.

s1.clear()

print("Set: ",s1)

#Example 3 discard() Remove an element from a set if it is a member.

s2={2,4,5,6,8}

s2.discard(8)

print("Set: ",s2)

#Example 4 isdisjoint()

s1={1,2,3,4}

s2={5,6,2}

s3=s1.isdisjoint(s2)

print(f"{s1} is {s2}:{s3} ")

#Example 5 issubset()

s5=s1.issubset(s2)

print(f"{s1} issubset {s2}:{s5} ")

#Example 6 pop() Remove and return an arbitrary set element

s9=s2.pop()

print("Poped Element: ",s9)#deleted 2

#Example 7 remove() Remove an element from a set; it must be a member.

s10=s1.remove(1)

print("Set: ",s10)

#Example 1 Python program to find common elements in two lists with the help of set operations −"""

s1={1,2,3,4,5}

s2={1,2,5,6,7}

s3=s1&s2

print(f"The Common in both sets:\n {s1} & {s2}= {s3}")

#Example 2 Python program to check if a set is a subset of another −"""

s1={1,2,3,4,5,6,7}

s2={1,3,5,7}

s4=s2.issubset(s1)

print(f"{s2} is the subset of {s1} ={s4}")

#Example 3Python program to obtain a list of unique elements in a list −

l1=[1,2,2,3,4,3,4,5]

print("Oringal list: ",l1)

s1=set(l1)

print("Unique Set: ",s1)

#Exercise Programs

# 1. Python program to find the size of a set object

print(f"The size of set1 {s1}= {len(s1)} ")

# 2. Python program that splits a set into two based on odd/even numbers.

s\_num={1,2,3,4,5,6,7,8,9,10}

s\_even=set()

s\_odd=set()

for i in s\_num:

    if i%2==0:

        s\_even.add(i)

    else:

        s\_odd.add(i)

print(f"Set {s\_num}\n Set\_Even: {s\_even}\n Set\_Odd: {s\_odd}")

# 3. Python program to remove all negative numbers from a set.

s1={1,2,3,4,-1,-2,-3,-4}

s2=set()

for i in s1:

    if i>0:

        s2.add(i)

print("\n\nOrignal Set: ",s1)

print("Set without negative numbers: ",s2)

# 4. Python program to build another set with absolute value of each number in a set.

set1={1,2,3,-3,0,-1,-1.1,-0.5}

s\_abs=set()

for i in set1:

    j=abs(i)

    s\_abs.add(j)

print(f"Absulote set: ",s\_abs)

#also

abs\_set={abs(num) for num in set1}

print(f"Absulote set: ",abs\_set)

#Python program to remove all strings from a set which has elements of different types.

mixed\_S={1,"a",2,"b",3,"c"}

filter\_S={num for num in mixed\_S if not isinstance(num,str)}

print("Mixed set: ",mixed\_S)

print("Filterd set: ",filter\_S)

# Ch#8 Python - Dictionaries

Dictionary is one of the built-in data types in Python. Python's dictionary is example of mapping type. A mapping object 'maps' value of one object with another.

In a language dictionary we have pairs of word and corresponding meaning. Two parts of pair are key (word) and value (meaning). Similarly, Python dictionary is also a collection of key:value pairs. The pairs are separated by comma and put inside curly brackets {}.

To establish mapping between key and value, the colon ':' symbol is put between the two.

#Example 1

capital={"Kpk":"Peshawer","Pakistan":"Islamabad","Punjab":"Lahore","Sindh":"Karachi"}

print(capital)

#Example 2

number={10:"Ten",20:"Twenty",30:"Therty",40:"Fourty"}

print(number)

#Example 1

"""

Only a number, string or tuple can be used as key.

All of them are immutable. You can use an object of any type as the value. Hence following definitions of dictionary are also valid −"""

d1={"Fruit":["Mango,Banana"],"Flower":["Rose","Lotu"]}

d2={("Pakistan,USA"):"Countries",("Islamabad","New York"):"Capitals"}

print("Dict: ",d1)

print("Dict: ",d2)

#Example 2 Python doesn't accept mutable objects such as list as key, and raises TypeError.

d1 = {["Mango","Banana"]:"Fruit", "Flower":["Rose", "Lotus"]}

Python Dictionary Operators

In Python, following operators are defined to be used with dictionary operands. In the example, the following dictionary objects are used.

d1 = {'a': 2, 'b': 4, 'c': 30}

d2 = {'a1': 20, 'b1': 40, 'c1': 60}

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Example** |
| dict[key] | Extract/assign the value mapped with key | print (d1['b']) retrieves 4  d1['b'] = 'Z' assigns new value to key 'b' |
| dict1|dict2 | Union of two dictionary objects, retuning new object | d3=d1|d2 ; print (d3)  {'a': 2, 'b': 4, 'c': 30, 'a1': 20, 'b1': 40, 'c1': 60} |
| dict1|=dict2 | Augmented dictionary union operator | d1|=d2; print (d1)  {'a': 2, 'b': 4, 'c': 30, 'a1': 20, 'b1': 40, 'c1': 60} |

# 8.1 Python - Access Dictionary Items

## Using the "[ ]" Operator

A dictionary in Python is not a sequence, as the elements in dictionary are not indexed. Still, you can use the square brackets "[ ]" operator to fetch the value associated with a certain key in the dictionary object.

#Example 1

capital={"Kpk":"Peshawer","Pakistan":"Islamabad","Punjab":"Lahore","Sindh":"Karachi"}

print("Capital of pakistan: ",capital["Pakistan"])

print("Capital of Punjab: ",capital["Sindh"])

#Using the get() Method

"""The get() method in Python's dict class returns the value mapped to the given key."""

print("Capital of Punjab: ",capital.get("Punjab"))

"""The get() method accepts an optional string argument. If it is given, and if the key is not found,

this string becomes the return value."""

print("Capital of Kashmir: ",capital.get("Kashmir","Not found"))

# 8.2 Python - Change Dictionary Items

Apart from the literal representation of dictionary, where we put comma-separated key:value pairs in curly brackets, we can create dictionary object with built-in dict() function.

## Empty Dictionary

Using dict() function without any arguments creates an empty dictionary object. It is equivalent to putting nothing between curly brackets.

#Example Empty Dic

d1=dict()

d2={}#both are dic same output

print(d1)

print(d2)

#Dictionary from List of Tuples

"""

The dict() function constructs a dictionary from a list or tuple of two-item tuples. First item in a tuple is treated as key, and the second as its value."""

d1=dict([('a',100),('b',200)])

d2=dict((('a',"One"),("b","Two")))

print("\nd1: ",d1)

print("d2: ",d2)

#Dictionary from Keyword Arguments

"""

The dict() function can take any number of keyword arguments with name=value pairs.

It returns a dictionary object with the name as key and associates it to the value."""

d3=dict(a=10,b=20,c=30)

d4=dict(d=40,e=50)

print("\nd3: ",d3)

print("d4: ",d4)

# 8.3 Python - Add Dictionary Items

## Using the Operator

The "[]" operator (used to access value mapped to a dictionary key) is used to update an existing key-value pair as well as add a new pair.

marks={"Shary":80,"Fahad":85,"Umair":87}

print("Marks dictionary before update: ",marks)

#Updation

marks["Umair"]=90

print("Marks dictionary after update: ",marks)

#Add new item with pair of key

marks["Ali"]=60

print("Marks dictionary after update: ",marks)

# 1. Using the update() Method

"""You can use the update() method in dict class in three different ways:"""

# 1.2 Update with Another Dictionary

"""In this case, the update() method's argument is another dictionary.

Value of keys common in both dictionaries is updated.

For new keys, key-value pair is added in the existing dictionary"""

#create another dic

marks1={"Muhammad":100,"Yaseen":86,"Ubaid":78}

#Update the first dict

marks.update(marks1)

print("Marks dictionary after update: ",marks)

# 1.3 Update with Iterable

"""

If the argument to update() method is a list or tuple of two item tuples,

an item each for it is added in the existing dictionary,

or updated if the key is existing."""

marks2=[("Yasir",60),("Hamad",70)]

marks1.update(marks2)

print("\nMarks1 dictionary after update: ",marks1)

#1.4 Update with Keyword Arguments

"""

Third version of update() method accepts list of keyword arguments in name=value format. New k-v pairs are added,

or value of existing key is updated."""

marks1.update(Sana=44,Shary=88)

print("\nMarks1 dictionary after update: ",marks1)

#Using the Unpack Operator

"""

The "\*\*" symbol prefixed to a dictionary object unpacks it to a list of tuples, each tuple with key and value.

Two dict objects are unpacked and merged together and obtain a new dictionary."""

newmarks={\*\*marks1,\*\*marks}

print("\nNewmarks: ",newmarks)

#Using the Union Operator (|)

"""It updates existing keys in dict object on left, and adds new key-value pairs to return a new dict object."""

newmarks2=marks|marks1

#Also mark|=marks1

print("Union of two dicts: \n",newmarks2)

# 8.4 Python - Remove Dictionary Items

## Using del Keyword

Python's **del** keyword deletes any object from the memory. Here we use it to delete a key-value pair in a dictionary.

#Example 1

d1={"D":"Database","H":"HCI","J":"Java","P":"Python"}

print("Dict1: ",d1)

#remove items

del d1["H"]

print("After removing items from d1:\n ",d1)

#remove full dict

# del d1

# print("After removes d1: ",d1)

#Using pop() Method

"""The pop() method of dict class causes an element with the specified key to be removed from the dictionary."""

v1=d1.pop("J")

print("Remove item key J: ",v1)

#Return value The pop() method returns the value of the specified key after removing the key-value pair.

#Using popitem() Method

"""

The popitem() method in dict() class doesn't take any argument. It pops out the last inserted key-value pair, and returns the same as a tuple"""

v2=d1.popitem()

print("d1 dict After pop op:  ",d1)

print("value Poped:  ",v2)

#Add some items

d1.update(DS="Data Scienc",Op="Operating system")

print("UPdate d1: \n",d1)

#Using clear() Method

"""

The clear() method in dict class removes all the elements from the dictionary object and returns an empty object."""

d1.clear()

print("After clear the d1: ",d1)

# 8.5 Python - Dictionary View Objects

The items(), keys() and values() methods of dict class return view objects. These views are refreshed dynamically whenever any change occurs in the contents of their source dictionary object.

## items() Method

The items() method returns a dict\_items view object. It contains a list of tuples, each tuple made up of respective key, value pairs.

Obj = dict.items()

### **Return value**

The items() method returns dict\_items object which is a dynamic view of (key,value) tuples.

#EXample 1 items() Method

numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}

obj = numbers.items()

print("the type of obj: ",type(obj))

print("obj: ",obj)

#update the dict

numbers.update({50:"Fifty"})

print("View automaticaly update: ",obj)

#keys() Method

"""

The keys() method of dict class returns dict\_keys object which is a list of all keys defined in the dictionary.

It is a view object, as it gets automatically updated whenever any update action is done on the dictionary object"""

obj1=numbers.keys()

print("The keys of number dict: ",obj1)

#values() Method

"""The values() method returns a view of all the values present in the dictionary. The object is of dict\_value type, which gets automatically updated."""

obj2=numbers.values()

print("The values of dict: ",obj2)

# 8.6 Python - Loop Dictionaries

Unlike a list, tuple or a string, dictionary data type in Python is not a sequence, as the items do not have a positional index. However, traversing a dictionary is still possible with different techniques.

#Example 1

"""

Running a simple for loop over the dictionary object traverses the keys used in it."""

numbers = {10:"Ten", 20:"Twenty", 30:"Thirty",40:"Forty"}

for x in numbers:

#    print (x)

   """Once we are able to get the key, its associated value can be easily accessed either by using square brackets operator or with get() method"""

   print(x,":",numbers[x])

#Example 3

"""

The dict\_items object is a list of key-value tuples over which a for loop can be run as follows:"""

for x in numbers.items():

   print(x)

#Example 5

"""

Similarly, the collection of keys in dict\_keys object can be iterated over"""

print("\n\n")

for x in numbers.keys():

   print(x,":",numbers[x])

#Example 6

print("\n\n")

l=len(numbers)

for x in range(l):

   print(list(numbers.keys())[x], ":",list(numbers.values())[x])

# 8.7 Python - Copy Dictionaries

Since a variable in Python is merely a label or reference to an object in the memory, a simple assignment operator will not create copy of object.

#Example 1

"""

In this example, we have a dictionary "d1" and we assign it to another variable "d2". If "d1" is updated, the changes also reflect in "d2"""

d1={"a":11,"b":22,"c":33}

d2=d1

print("id: ",id(d1),"dict: ",d1)

print("id: ",id(d2),"dict: ",d2)

d1["d"]=100

print("id: ",id(d1),"dict: ",d1)

print("id: ",id(d2),"dict: ",d2)

#Example 2

"""To avoid this, and make a shallow copy of a dictionary, use the copy() method instead of assignment."""

print("\n\n")

d3=d1.copy()

print("\nid: ",id(d1),"dict: ",d1)

print("id: ",id(d3),"dict: ",d3)

d1["E"]=120

print("\nid: ",id(d1),"dict: ",d1)

print("id: ",id(d3),"dict: ",d3)

"""

Output

When "d1" is updated, "d2" will not change now because "d2" is the copy of dictionary object,

not merely a reference.

"""

# 8.8 Python - Nested Dictionaries

A Python dictionary is said to have a nested structure if value of one or more keys is another dictionary. A nested dictionary is usually employed to store a complex data structure.

#Example 1

"""

You can also constitute a for loop to traverse nested dictionary, as in the previous section."""

markslist={

    "Shary": {"HCI":80,"maths":70},

    "Umair": {"HCI":67,"maths":60},

    "Fahad":{"HCI":87,"maths":50}

}

for k,v in markslist.items():

    print(k," : ",v)

#Example 2

"""

It is possible to access value from an inner dictionary with [] notation or get() method."""

#print shary maths marks

print("Shary maths marks: ",markslist.get("Shary")['maths'])

#Also from this tecnique geting

obj=markslist["Shary"]

print("Shary HCI marks: ",obj.get('HCI'))

# 8.9 Python - Dictionary Methods

A dictionary in Python is an object of the built-in **dict** class, which defines the following methods −

|  |  |
| --- | --- |
| **Sr.No.** | **Method and Description** |
| 1 | [**dict.clear()**](https://www.tutorialspoint.com/python/dictionary_clear_method.htm)  Removes all elements of dictionary dict. |
| 2 | [**dict.copy()**](https://www.tutorialspoint.com/python/dictionary_copy_method.htm)  Returns a shallow copy of dictionary dict. |
| 3 | [**dict.fromkeys()**](https://www.tutorialspoint.com/python/dictionary_fromkeys_method.htm)  Create a new dictionary with keys from seq and values set to value. |
| 4 | [**dict.get(key, default=None)**](https://www.tutorialspoint.com/python/dictionary_get_method.htm)  For key key, returns value or default if key not in dictionary. |
|  |  |
| 5 | [**dict.has\_key(key)**](https://www.tutorialspoint.com/python/dictionary_has_key_method.htm)  Returns true if a given key is available in the dictionary, otherwise it returns a false. |
| 6 | [**dict.items()**](https://www.tutorialspoint.com/python/dictionary_items_method.htm)  Returns a list of dict's (key, value) tuple pairs. |
| 7 | [**dict.keys()**](https://www.tutorialspoint.com/python/dictionary_keys_method.htm)  Returns list of dictionary dict's keys. |
| 8 | **dict.pop()**  Removes the element with specified key from the collection |
| 9 | **dict.popitem()**  Removes the last inserted key-value pair |
| 10 | [**dict.setdefault(key, default=None)**](https://www.tutorialspoint.com/python/dictionary_setdefault_method.htm)  Similar to get(), but will set dict[key]=default if key is not already in dict. |
| 11 | [**dict.update(dict2)**](https://www.tutorialspoint.com/python/dictionary_update_method.htm)  Adds dictionary dict2's key-values pairs to dict. |
| 12 | [**dict.values()**](https://www.tutorialspoint.com/python/dictionary_values_method.htm)  Returns list of dictionary dict's values. |

#EXample 1 clear()

dict = {'Name': 'Zara', 'Age': 7};

print ("Start Len : %d" % len(dict))

dict.clear()

print("End len: %d"% len(dict))

#Example 2 method fromkeys()

"""

Python dictionary method fromkeys() creates a new dictionary

with keys from seq and values set to value.

"""

#Example 3 method get()

"""Python dictionary method get() returns a value for the given key. If key is not available then returns default value None"""

dict1 = {'Name': 'Zara', 'Age': 7};

print("\nValue :%d " % dict1.get("Age"))

print("\nValue :%s " % dict1.get("Shary","No here"))

#Example 4  dict.has\_key(key) method

"""

Returns true if a given key is available in the dictionary, otherwise it returns a false."""

#not working .has\_key method for this condition use in op

print("\nValue :%s" % ('Age' in dict1))

#Example5 dict.items() method

#Returns a list of dict's (key, value) tuple pairs."""

print("Return a list of dict(key,value) tuple pairs: \n",dict1.items())

#Example 6 dict.keys() Returns list of dictionary dict's keys.

print("Return a list of keys: \n",dict1.keys())

# 8.10 Python - Dictionary Exercises

### **Example 1**

Python program to create a new dictionary by extracting the keys from a given dictionary.

#Example 1

"""

Python program to create a new dictionary by extracting the keys from a given dictionary"""

d1 = {"one":11, "two":22, "three":33, "four":44, "five":55}

keys={"two","one"}

d2={}

for k in keys:

    d2[k]=d1[k]

print("Orignal dic: \n",d1)

print("new dict: \n",d2)

#Example 2

"""

Python program to convert a dictionary to list of (k,v) tuples."""

l1=list(d1.items())

print("\nList of tuple: \n",l1)

#Example 3

"""

Python program to remove keys with same values in a dictionary."""

D1 = {"one":"eleven", "2":2, "three":3, "11":"eleven", "four":44, "two":2}

val=list(D1.values()) #all values

unique\_val=[v for v in val if val.count(v)==1]#unique values

d2={}

for k,v in D1.items():

     if v in unique\_val:

         d={k:v}

         d2.update(d)

print("dict with unique value: ",d2)

#Example 4 Python program to sort list of dictionaries by values

# Sample list of dictionaries

D11 = [

    {'name': 'John', 'age': 30, 'score': 95},

    {'name': 'Alice', 'age': 25, 'score': 85},

    {'name': 'Bob', 'age': 35, 'score': 92},

]

sorted\_D1=sorted(D11,key=lambda x:x['score'])

for i in sorted\_D1:

    print(i)

#Python program to extract dictionary with each key having non-numeric value from a given dictionary.

dict1={1:10,2:20,3:30,4:40,5:"khan",6:"ali"}

for k,v in dict1.items():#taking all element

    if not isinstance(v,(int,float)):#check condition the pair value is not int or float

        print(k,":",v)#key and value print

#Python program to build a dictionary from list of two item (k,v) tuples

#creating list of tuples

list1=[('HCI',40),('DSA',70),('MAD',80),('MPL',60)]

#convert list of tuple to dict

dict2=dict(list1)

print("Convert list to dict: ",dict2)

#Python program to merge two dictionary objects, using unpack operator.

#using the above dict1 and dict2 to combine

merg\_dict={\*\*dict1,\*\*dict2}

print("merge dict: ",merg\_dict)

# Ch#9 Python - Arrays

Python's standard data types **list, tuple** and **string** are sequences. A sequence object is an ordered collection of items. Each item is characterized by incrementing index starting with zero. Moreover, items in a sequence need not be of same type. In other words, a list or tuple may consist of items of different data type.

This feature is different from the concept of an array in C or C++. In C/C++, an array is also an indexed collection of items, but the items must be of similar data type. In C/C++, you have an array of integers or floats, or strings, but you cannot have an array with some elements of integer type and some of different type. A C/C++ array is therefore a homogenous collection of data types.

Python's standard library has array module. The array class in it allows you to construct an array of three basic types, integer, float and Unicode characters.

Arrays are sequence types and behave very much like lists, except that the type of objects stored in them is constrained.

Python array type is decided by a single character Typecode argument. The type codes and the intended data type of array is listed below −

|  |  |  |
| --- | --- | --- |
| **typecode** | **Python data type** | **Byte size** |
| 'b' | signed integer | 1 |
| 'B' | unsigned integer | 1 |
| 'u' | Unicode character | 2 |
| 'h' | signed integer | 2 |
| 'H' | unsigned integer | 2 |
| 'i' | signed integer | 2 |
| 'I' | unsigned integer | 2 |
| 'l' | signed integer | 4 |
| 'L' | unsigned integer | 4 |
| 'q' | signed integer | 8 |
| 'Q' | unsigned integer | 8 |
| 'f' | floating point | 4 |
| 'd' | floating point | 8 |

import array as arr

a=arr.array('i' ,[1,2,3])

print(type(a))#type of arrayclass

print(a)

b=arr.array('u','Shary')

print(type(b))

print("b :",b)

c=arr.array('d',[1.1,1.2,1.3,1.4])

print("C :",c)

# 9.1 Python - Access Array Items

Since the array object behaves very much like a sequence, you can perform indexing and slicing operation with it.

import array as arr

a=arr.array('i',[1,2,3])

#indexing

print("at index 1 value:  ",a[1])

#sclicing

print("sclicing",a[1:])

#Changing Array Items

"""

You can assign value to an item in the array just as you assign a value to item in a list."""

a[1]=10

print("updated array: ",a)

# a[1]='A'

# print("Updated: ",a[1])

# Changing Array Items

# You can assign value to an item in the array just as you assign a value to item in a list.

# 9.2 Python - Add Array Items

## The append() Method

The append() method adds a new element at the end of given array.

### **Syntax**

array.append(v)

### **Parameters**

* **v** − new value is added at the end of the array. The new value must be of the same type as datatype argument used while declaring array object.

#Example 1

import array as b

newArray=b.array('i',[1,2,3])

newArray.append(4)#append in lasts

print(newArray)

#The insert() Method

"""

The array class also defines insert() method. It is possible to insert a new element at the specified index."""

#Example 2

newArray.insert(1,10)#insert at index 1

print(newArray)

#The extend() Method

"""

The extend() method in array class appends all the elements from another array of same typecode."""

#Example 3

newArray2=b.array('i',[5,6,7,8])

newArray.extend(newArray2)

print("Extended array: ",newArray)

# 9.4 Python - Remove Array Items

The array class defines two methods with the help of which we can remove an element from the array. It has remove() and pop() methods

## array.remove() Method

The remove() method removes the first occurrence of a given value from the array

### **Syntax**

array.remove(v)

### **Parameters**

* **v** − The value to be removed from the array

#Example 1

#array.remove() Method

"""

The remove() method removes the first occurrence of a given value from the array"""

import array as b

newArray=b.array('i',[1,2,3,4])

print("Orignal array: ",newArray)

newArray.remove(1)

print("After remove the item 1: ",newArray)

#Example 2

#array.pop() Method

"""

The pop() method removes an element at the specified index from the array, and returns the removed element."""

newArray.pop(2)#poped 4

print("after Poped element: ",newArray)

# 9.5 Python - Loop Arrays

Since the array object behaves like a sequence, you can iterate through its elements with the help of **for** loop or **while** loop.

### **"for" Loop with Array**

Take a look at the following example −

import array as arr

a = arr.array('d', [1, 2, 3])

for x in a:

print (x)

It will produce the following **output** −

1.0

2.0

3.0

### **"while L oop with Array**

The following example shows how you can loop through an array using a **while** loop −

import array as arr

a=arr.array('d',[1,2,3,4,5])

l=len(a)#length of array

idx=0

while idx<l:

    print(a[idx])

    idx+=1

print("\n")

#"for Loop with Array I ndex

l1=len(a)

for i in range(l1):

    print(a[i])

#You will get the same output as in the first example.

# 9.5 Python - Copy Arrays

Python's built-in sequence types i.e. list, tuple and string are indexed collection of items. However, unlike arrays in C/C++, Java etc. , they are not homogenous, in the sense the elements in these types of collection may be of different types. Python's array module helps you to create object similar to Java like arrays. In this chapter, we discuss how to copy an array object to another.

Python arrays can be of string, integer or float type. The array class constructor is used as follows −

import array

obj = array.array(typecode[, initializer])

The typecode may be a character constant representing the data type.

We can assign an array to another by the assignment operator.

a = arr.array('i', [1, 2, 3, 4, 5])

b=a.copy()

However, such assignment doesn't create a new array in the memory. In Python, a variable is just a label or reference to the object in the memory. So, a is the reference to an array, and so is b. Check the id() of both a and b. Same value of id confirms that simple assignment doesn't create a copy

import array as arr

a = arr.array('i', [1, 2, 3, 4, 5])

b=a

print (id(a), id(b))

It will produce the following **output** −

2771967068656 2771967068656

Because "a" and "b" refer to the same array object, any change in "a" will reflect in "b" too −

a[2]=10

print (a,b)

It will produce the following **output** −

array('i', [1, 2, 10, 4, 5]) array('i', [1, 2, 10, 4, 5])

To create another physical copy of an array, we use another module in Python library, named copy and use deepcopy() function in the module. A deep copy constructs a new compound object and then, recursively inserts copies into it of the objects found in the original.

#Example 1

import array as arr

#create array

array=arr.array('i',[1,2,3,4,5,6])

#assign array to another by the assingment op

newarray=array#it refeence same object

print("Id: ",id(array),"Array: ",array)

print("Id: ",id(newarray),"Array: ",newarray)

#change in array reflect to newarray

array[3]=10

print("\nId: ",id(array),"Array: ",array)

print("Id: ",id(newarray),"Array: ",newarray)

#physical copy of an array named copy and use deepcopy() function in the module

import copy

newarray=copy.deepcopy(array)

print("\nafter copy the array:\n id: ",id(array),"Array: ",array)

print("after copy the array:\n id: ",id(newarray),"Array: ",newarray)

#chang in array not reflect newarray

array[4]=102

print("\nafter Changing the array:\n id: ",id(array),"Array: ",array)

print("after changing the array:\n id: ",id(newarray),"Array: ",newarray)

# 9.6 Python - Reverse Arrays

In this chapter, we shall explore the different ways to rearrange the given array in the reverse order of the index. In Python, array is not one of the built-in data types. However, Python's standard library has array module. The array class helps us to create a homogenous collection of string, integer or float types.

The **syntax** used for creating array is −

import array

obj = array.array(typecode[, initializer])

Let us first create an array consisting of a few objects of **int** type −

#Example1

import array as ar

newarray=ar.array('i',[1,2,3,4,5,6,7])

"""

    The array class doesn't have any built-in method to reverse array. Hence, we have to use another array. An empty array "b" is declared as follows −

"""

newarray2=ar.array('i')

"""

Next, we traverse the numbers in array "a" in reverse order, and append each element to the "b" array −

"""

for i in range(len(newarray)-1,-1,-1):

    newarray2.append(i)

print("Orignal array:",newarray)

print("Exteded array is: ",newarray2)

#Example 2

"""

We can also reverse the sequence of numbers in an array using the reverse() method in list class.

List is a built-in type in Python.

We have to first transfer the contents of an array to a list with tolist() method of array class −

"""

a=ar.array('i',[10,5,15,2,3,4,5])

print("\n\n orignal array: ",a)

b=a.tolist()

print("convert to list: ",b)

#We can call the reverse() method now −

b.reverse()

print("Reverse the list: ",b)

"""If we now convert the list back to an array, we get the array with reversed order,"""

a=ar.array('i')

a.fromlist(b)

print("Reverse the orignal array: ",a)

# 9.7 Python - Sort Arrays

Python's array module defines the array class. An object of array class is similar to the array as present in Java or C/C++. Unlike the built-in Python sequences, array is a homogenous collection of either strings, or integers, or float objects.

The array class doesn't have any function/method to give a sorted arrangement of its elements. However, we can achieve it with one of the following approaches −

* Using a sorting algorithm
* Using the sort() method from List
* Using the built-in sorted() function

Let's discuss each of these methods in detail.

## Using a Sorting Algorithm

We shall implement the classical **bubble sort algorithm** to obtain the sorted array. To do it, we use two nested loops and swap the elements for rearranging in sorted order.

#Example 1 sorting array using bubble sort a sorting alg

import array as ary

#create array

CreateArray=ary.array('i',[9,2,4,10,30,20,5,7,8])

print("Orignal array: ",CreateArray)

#length

for i in range(0,len(CreateArray)):

    for j in range(i+1,len(CreateArray)):

        if CreateArray[i]>CreateArray[j]:

            temp=CreateArray[i]

            CreateArray[i]=CreateArray[j]

            CreateArray[j]=temp

print("Sorted array:",CreateArray)

#Using the sort() Method from List

"""

Even though array doesn't have a sort() method, Python's built-in List class does have a sort method.

We shall use it in the next example.

First, declare an array and obtain a list object from it, using tolist() metho"""

from array import array as arr

a = arr.array('i', [10,5,15,4,6,20,9])

b=a.tolist()

b.sort()

a = arr.array('i')

a.fromlist(b)

print (a)

## Using the Builtin sorted() Function

The third technique to sort an array is with the sorted() function, which is a built-in function.

The **syntax** of sorted() function is as follows −

sorted(iterable, reverse=False)

The function returns a new list containing all items from the iterable in ascending order. Set reverse parameter to True to get a descending order of items.

The sorted() function can be used along with any iterable. Python array is an iterable as it is an indexed collection. Hence, an array can be used as a parameter to sorted() function.

#Example 3  Using the Builtin sorted() Function

a1=ary.array("i",[4,5,6,9,2,1,3])

a1=ary.array('i',sorted(a1))

print("Sort array(sorted func): ",a1)

# 9.8 Python - Join Arrays

In Python, array is a homogenous collection of Python's built in data types such as strings, integer or float objects. However, array itself is not a built-in type, instead we need to use the array class in Python's built-in array module.

## First Method

To join two arrays, we can do it by appending each item from one array to other.

#Example 1 First method

import array as ary

a=ary.array('i',[1,2,3,4,5])

b=ary.array('i',[6,7,8,9])

"""Run a for loop on the array "b". Fetch each number from "b" and append it to array "a" with the following loop statement −"""

for i in range(len(b)):

    a.append(b[i])

print("The two array are join(append): ",a)

#Example 2 Second Method

"""Using another method to join two arrays, first convert arrays to list objects −"""

a1=a.tolist()

b1=b.tolist()

c=a1+b1

result=ary.array('i')

result.fromlist(c)

print("joined array(tolist): ",result)

#Example 3 Third Method

"""

We can also use the extend() method from the List class to append elements from one list to another.

First, convert the array to a list and then call the extend() method to merge the two lists −"""

a.extend(b)

print("Extended array: ",a)

# 9.9 Python - Array Methods

## array.reverse() Method

Like the sequence types, the array class also supports the reverse() method which rearranges the elements in reverse order.

### **Syntax**

array.reverse()

### **Parameters**

This method has no parameters

### **Example**

import array as arr

a = arr.array('i', [1, 2, 3, 4, 5])

a.reverse()

print (a)

It will produce the following **output** −

array('i', [5, 4, 3, 2, 1])

The array class also defines the following useful methods.

## array.count() Method

The count() method returns the number of times a given element occurs in the array.

### **Syntax**

array.count(v)

### **Parameters**

* **v** − The value whose occurrences are to be counted

### **Return value**

The count() method returns an integer corresponding the number of times v appears in the array.

#Example 1  array.count()

import array as ary

a=ary.array('i',[1,2,3,4,5,8,9,10,2,3,2,1,2])

c=a.count(2)

print("Count of 2: ",c)

#Example 2 array.index() method

"""The index() method in array class finds the position of first occurrence of a given element in the array."""

ind=a.index(4)

print("index of 4 is: ",ind)

#Example 3 array.fromlist() Method

"""The fromlist() method appends items from a Python list to the array object.

i − The list, items of which are appended to the array. All items in the list must be of same arrtype.

"""

ary1=ary.array('i',[1,2,3,4,5])

list1=[5,6,7,8]

#appending list into array

ary1.fromlist(list1)

print("After the  appended list  array is: \n",ary1)

#Example 4 array.tofile() Method

"""

The tofile() method in array class writes all items (as machine values) in the array to the file object f.

Syntax array.tofile(f)

Parameters

f − the file object obtained with open() function. The file to be opened in wb mode."""

f=open('list.txt','wb')

ary.array('i',[10,20,30,40,50]).tofile(f)

f.close()

"""

Output

After running the above code, a file named as "list.txt" will be created in the current directory.

"""

#Example 5  array.fromfile() Method

"""

The fromfile() method reads a binary file and appends specified number of items to the array object.

Syntax array.fromfile(f, n)

Parameters

f − The file object referring to a disk file opened in rb mode

n − number of items to be appended"""

ary2=ary.array('i',[1,2,3,4,5])

f=open('list.txt','rb')#open the file for read only

ary2.fromfile(f,5)#append the file all value to this array

print(ary2)

# 9.10 Python - Array Exercises

## Example 1

# Ch#10 Python - File Handling

When we use any computer application, some data needs to be provided. Data is stored in computer's main memory (RAM) until the application is running. Thereafter, memory contents from RAM are erased.

We would like to store it in such a way that it can be retrieved whenever required in a persistent medium such as a disk file.

Python uses built-in **input()** and **print()** functions to perform standard input/output operations. Python program interacts with these IO devices through standard stream objects stdin and stdout defined in sys module.

The input() function reads bytes from a standard input stream device i.e. keyboard. Hence both the following statements read input from the user.

name = input()

#is equivalent to

import sys

name = sys.stdin.readline()

The print() function on the other hand, sends the data towards standard output stream device, i.e., the display monitor. It is a convenience function emulating write() method of stdout object.

print (name)

#is equivalent to

import sys

sys.stdout.write(name)

Any object that interacts with input and output steam is called File object. Python's built-in function open() returns a file object.

## The open() Function

This function creates a file object, which would be utilized to call other support methods associated with it.

### **Syntax**

file object = open(file\_name [, access\_mode][, buffering])

Here are the parameter details −

* **file\_name** − The file\_name argument is a string value that contains the name of the file that you want to access.
* **access\_mode** − The access\_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. A complete list of possible values is given below in the table. This is an optional parameter and the default file access mode is read (r).
* **buffering** − If the buffering value is set to 0, no buffering takes place. If the buffering value is 1, line buffering is performed while accessing a file. If you specify the buffering value as an integer greater than 1, then buffering action is performed with the indicated buffer size. If negative, the buffer size is the system default (default behavior).

## File Opening Modes

Following are the file opening modes −

|  |  |
| --- | --- |
| **Sr.No.** | **Modes & Description** |
| 1 | **r**  Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode. |
| 2 | **rb**  Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode. |
| 3 | **r+**  Opens a file for both reading and writing. The file pointer placed at the beginning of the file. |
| 4 | **rb+**  Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file. |
| 5 | **w**  Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing. |
| 6 | **b**  Opens the file in binary mode |
| 7 | **t**  Opens the file in text mode (default) |
| 8 | **+**  open file for updating (reading and writing) |
| 9 | **wb**  Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing. |
| 10 | **w+**  Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing. |
| 11 | **wb+**  Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing. |
| 12 | **a**  Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing. |
| 13 | **ab**  Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing. |
| 14 | **a+**  Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing. |
| 15 | **ab+**  Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing. |
| 16 | **x**  open for exclusive creation, failing if the file already exists |

Once a file is opened and you have one file object, you can get various information related to that file.

# 10.1 Python - Write to File

To write data to a file in Python, you need to open a file. Any object that interacts with input and output steam is called File object. Python's built-in function open() returns a file object.

fileObject = open(file\_name [, access\_mode][, buffering])

After you obtain the file object with the open() function, you can use the write() method to write any string to the file represented by the file object. It is important to note that Python strings can have binary data and not just text.

The write() method does not add a newline character ('\n') to the end of the string.

### **Syntax**

fileObject.write(string)

Here, passed parameter is the content to be written into the opened file.

### **Example**

# Open a file

fo = open("foo.txt", "w")

fo.write( "Python is a great language.\nYeah its great!!\n")

# Close opened file

fo.close()

The above method would create foo.txt file and would write given content in that file and finally it would close that file. The program shows no output as such, although if you would open this file with any text editor application such as Notepad, it would have the following content −

Python is a great language.

Yeah its great!!

## Writing in Binary Mode

By default, read/write operation on a file object are performed on text string data. If we want to handle files of different other types such as media (mp3), executables (exe), pictures (jpg) etc., we need to add 'b' prefix to read/write mode.

Following statement will convert a string to bytes and write in a file.

f=open('test.bin', 'wb')

data=b"Hello World"

f.write(data)

f.close()

Conversion of text string to bytes is also possible using encode() function.

data="Hello World".encode('utf-8')

## Appending to a File

When any existing file is opened in 'w' mode to store additional text, its earlier contents are erased. Whenever a file is opened with write permission, it is treated as if it is a new file. To add data to an existing file, use 'a' for append mode.

### **Syntax**

fileobject = open(file\_name,"a")

### **Example**

# Open a file in append mode

fo = open("foo.txt", "a")

text = "TutorialsPoint has a fabulous Python tutorial"

fo.write(text)

# Close opened file

fo.close()

When the above program is executed, no output is shown, but a new line is appended to foo.txt. To verify, open with a text editor.

Python is a great language.

Yeah its great!!

TutorialsPoint has a fabulous Python tutorial

## Using the w+ Mode

When a file is opened for writing (with 'w' or 'a'), it is not possible to perform write operation at any earlier byte position in the file. Th 'w+' mode enables using write() as well as read() methods without closing a file. The File object supports seek() unction to rewind the stream to any desired byte position.

Following is the syntax for seek() method −

fileObject.seek(offset[, whence])

### **Parameters**

* **offset** − This is the position of the read/write pointer within the file.
* **whence** − This is optional and defaults to 0 which means absolute file positioning, other values are 1 which means seek relative to the current position and 2 means seek relative to the file's end.

Let us use the seek() method to show how simultaneous read/write operation on a file can be done.

### **Example**

The following program opens the file in w+ mode (which is a read-write mode), adds some data. The it seeks a certain position in file and overwrites its earlier contents with new text.

# Open a file in read-write mode

fo=open("foo.txt","w+")

fo.write("This is a rat race")

fo.seek(10,0)

data=fo.read(3)

fo.seek(10,0)

fo.write('cat')

fo.close()

### **Output**

If we open the file in Read mode (or seek the starting position while in w+ mode), and read the contents, it shows –

#Example 1 open file

fo=open("foo.txt",'wb')

print("Name of the file: ",fo.name)

print("Closed or not: ",fo.closed)

print("Opinning mode: ",fo.mode)

fo.close()

# 10 .1 Python - Write to File

# open a file

fo=open("foo.txt","w")

fo.write("Python is a great language.\nYeah its great!!\n")

#Close file

fo.close()

# 10.1.2 Writing in Binary Mode

"""

By default, read/write operation on a file object are performed on text string data.

If we want to handle files of different other types such as media (mp3), executables (exe),

pictures (jpg) etc., we need to add 'b' prefix to read/write mode.

Following statement will convert a string to bytes and write in a file.

"""

f=open("test.bin","wb")

data=(b"Hello word") #b is for Opens the file in binary mode

f.write(data)

f.close()

"""Conversion of text string to bytes is also possible using encode() function."""

data1='Hello word'.encode('utf-8')

print(data1)

#10.1.3  Appending to a File

"""

When any existing file is opened in 'w' mode to store additional text, its earlier contents are erased.

Whenever a file is opened with write permission, it is treated as if it is a new file.

To add data to an existing file, use 'a' for append mode.

"""

#Open a file in append mode

append\_f=open("foo.txt","a")

text="TutorialPoint has a fabulous python tutorial"

append\_f.write(text)

#close file

append\_f.close()

# 10.1.4 Using the w+ Mode

"""

When a file is opened for writing (with 'w' or 'a'),

it is not possible to perform write operation at any earlier byte position in the file.

Th 'w+' mode enables using write() as well as read() methods without closing a file.

The File object supports seek() unction to rewind the stream to any desired byte position.

"""

#Open a file in read write mode

# fo3=open("foo.txt","w+")

# fo3.write("This is a rat race")

"""

Parameters

offset : This is the position of the read/write pointer within the file.

whence : This is optional and defaults to 0 which means absolute file positioning,

other values are 1 which means seek relative to the current position and 2 means seek relative to the file's end.

    """

# fo3.seek(10,0)#offset 10 and whence 0

# data=fo3.read(3)

# fo3.seek(10,0)#seek() method to show how simultaneous read/write operation on a file can be done.

# fo3.write('cat')

# fo3.close()

# 10.2 Python - Read Files

To programmatically read data from a file using Python, it must be opened first. Use the built-in open() function −

file object = open(file\_name [, access\_mode][, buffering])

Here are the parameter details −

* **file\_name** − The file\_name argument is a string value that contains the name of the file that you want to access.
* **access\_mode** − The access\_mode determines the mode in which the file has to be opened, i.e., read, write, append, etc. This is an optional parameter and the default file access mode is read (r).

These two statements are identical −

fo = open("foo.txt", "r")

fo = open("foo.txt")

To read data from the opened file, use read() method of the File object.

**It is important to note that Python strings can have binary data apart from the text data.**

### **Syntax**

fileObject.read([count])

### **Parameters**

* **count** − Number of bytes to be read.

Here, passed parameter is the number of bytes to be read from the opened file. This method starts reading from the beginning of the file and if count is missing, then it tries to read as much as possible, maybe until the end of file.

### **Example**

# Open a file

fo = open("foo.txt", "r")

text = fo.read()

print (text)

# Close the opened file

fo.close()

It will produce the following **output** −

Python is a great language.

Yeah its great!!

## Reading in Binary Mode

By default, read/write operation on a file object are performed on text string data. If we want to handle files of different other types such as media (mp3), executables (exe), pictures (jpg) etc., we need to add 'b' prefix to read/write mode.

Assuming that the **test.bin** file has already been written with binary mode.

f=open('test.bin', 'wb')

data=b"Hello World"

f.write(data)

f.close()

We need to use 'rb' mode to read binary file. Returned value of read() method is first decoded before printing

f=open('test.bin', 'rb')

data=f.read()

print (data.decode(encoding='utf-8'))

It will produce the following **output** −

Hello World

## Read Integer Data from F ile

In order to write integer data in a binary file, the integer object should be converted to bytes by to\_bytes() method.

n=25

n.to\_bytes(8,'big')

f=open('test.bin', 'wb')

data=n.to\_bytes(8,'big')

f.write(data)

To read back from a binary file, convert the output of read() function to integer by using the from\_bytes() function.

f=open('test.bin', 'rb')

data=f.read()

n=int.from\_bytes(data, 'big')

print (n)

## Read Float Data from File

For floating point data, we need to use **struct** module from Python's standard library.

import struct

x=23.50

data=struct.pack('f',x)

f=open('test.bin', 'wb')

f.write(data)

Unpacking the string from read() function to retrieve the float data from binary file.

f=open('test.bin', 'rb')

data=f.read()

x=struct.unpack('f', data)

print (x)

## Using the r+ M ode

When a file is opened for reading (with 'r' or 'rb'), it is not possible to write data in it. We need to close the file before doing other operation. In order to perform both operations simultaneously, we have to add '+' character in the mode parameter. Hence 'w+' or 'r+' mode enables using write() as well as read() methods without closing a file.

The File object also supports the seek() function to rewind the stream to read from any desired byte position.

Following is the syntax for seek() method −

fileObject.seek(offset[, whence])

### **Parameters**

* **offset** − This is the position of the read/write pointer within the file.
* **whence** − This is optional and defaults to 0 which means absolute file positioning, other values are 1 which means seek relative to the current position and 2 means seek relative to the file's end.

Let us use the seek() method to show how to read data from a certain byte position.

### **Example**

This program opens the file in w+ mode (which is a read-write mode), adds some data. The it seeks a certain position in file and overwrites its earlier contents with new text.

fo=open("foo.txt","r+")

fo.seek(10,0)

data=fo.read(3)

print (data)

fo.close()

It will produce the following **output** −

rat

## Python Simultaneous Read/Write

When a file is opened for writing (with 'w' or 'a'), it is not possible to read from it and vice versa. Doing so throws UnSupportedOperation error. We need to close the file before doing other operation.

In order to perform both operations simultaneously, we have to add '+' character in the mode parameter. Hence 'w+' or 'r+' mode enables using write() as well as read() methods without closing a file. The File object also supports the seek() unction to rewind the stream to any desired byte position.

## The seek() Method

The method seek() sets the file's current position at the offset. The whence argument is optional and defaults to 0, which means absolute file positioning, other values are 1 which means seek relative to the current position and 2 means seek relative to the file's end.

There is no return value. Note that if the file is opened for appending using either 'a' or 'a+', any seek() operations will be undone at the next write.

If the file is only opened for writing in append mode using 'a', this method is essentially a no-op, but it remains useful for files opened in append mode with reading enabled (mode 'a+').

If the file is opened in text mode using 't', only offsets returned by tell() are legal. Use of other offsets causes undefined behavior.

Note that not all file objects are seekable.

### **Syntax**

Following is the syntax for seek() method −

fileObject.seek(offset[, whence])

### **Parameters**

* **offset** − This is the position of the read/write pointer within the file.
* **whence** − This is optional and defaults to 0 which means absolute file positioning, other values are 1 which means seek relative to the current position and 2 means seek relative to the file's end.

Let us use the seek() method to show how simultaneous read/write operation on a file can be done.

The following program opens the file in w+ mode (which is a read-write mode), adds some data. The it seeks a certain position in file and overwrites its earlier contents with new text.

### **Example**

# Open a file in read-write mode

fo=open("foo.txt","w+")

fo.write("This is a rat race")

fo.seek(10,0)

data=fo.read(3)

fo.seek(10,0)

fo.write('cat')

fo.seek(0,0)

data=fo.read()

print (data)

fo.close()

### **Output**

This is a cat race

# 10.2 Python - Read Files

"""

To programmatically read data from a file using Python, it must be opened first. Use the built-in open() function

These two statements are identica

"""

fo4=open("foo.txt","r")

fo4=open("foo.txt")

#Example 1

#Open a file

fo1=open("foo.txt","r")

print("Read the foo file: ",fo1.read())

#close file

fo1.close()

#10.2.1 Reading in Binary Mode

"""

By default, read/write operation on a file object are performed on text string data.

If we want to handle files of different other types such as media (mp3), executables (exe),

pictures (jpg) etc., we need to add 'b' prefix to read/write mode.

Assuming that the test.bin file has already been written with binary mode

    """

fo2=open("test.bin","rb")

print("Read the test.bin  file: ",fo2.read())

fo2.close()

#Output: b'Hello word'

"""

We need to use 'rb' mode to read binary file. Returned value of read() method is first decoded before printing

    """

#the above code also writen

fo3=open("test.bin","rb")

data2=fo3.read()

print(data.decode(encoding='utf-8'))

#Output: Hello word

#10.2.3 Read Integer Data from F ile

"In order to write integer data in a binary file, the integer object should be converted to bytes by to\_bytes() method."""

n=23

n.to\_bytes(8,'big')#n.to\_bytes(8, 'big'): This method converts the integer n to a bytes object of length 8 (using 8 bytes) and specifies the byte order as 'big endian' (most significant byte first). The result is stored in the variable data.

fo4=open('test.bin',"wb")#Opens the file 'test.bin' in binary write mode

data=n.to\_bytes(8,"big")

fo4.write(data)# Writes the bytes data to the binary file.

fo4.close()

"""To read back from a binary file, convert the output of read() function to integer by using the from\_bytes() function."""

fo5=open('test.bin','rb')

data=fo5.read()

n=int.from\_bytes(data,"big")#Converts the bytes data back to an integer.

#The 'big' parameter indicates the byte order, specifying 'big endian'.

print(n)

fo5.close()

#important point

"""

he term "big" refers to the byte order, specifically "big endian." Endianness refers to the order in which bytes are stored in computer memory. In 'big endian

big:  ' the most significant byte (the one with the highest value) is stored at the lowest memory address.

little:  The alternative is 'little endian,' where the least significant byte is stored at the lowest memory address.

"""

#10.2.4 Read Float Data from File

"""

For floating point data, we need to use struct module from Python's standard library."""

import struct

x=23.4

data3=struct.pack('f',x)#This function packs the floating-point number x into a bytes object using the format specifier 'f' (indicating a 4-byte float).

                                    #The result is stored in the variable data3."""

fo6=open('test2.bin','wb')

fo6.write(data3) # Writes the packed binary data to the file.

fo6.close()

"""Unpacking the string from read() function to retrieve the float data from binary file."""

fo7=open('test2.bin','rb')

data4=fo7.read()

x=struct.unpack('f',data4)#Unpacks the binary data using the format

print(x)

#10.2.5  Using the r+ M ode

"""

When a file is opened for reading (with 'r' or 'rb'), it is not possible to write data in it.

We need to close the file before doing other operation.

In order to perform both operations simultaneously, we have to add '+' character in the mode parameter.

Hence 'w+' or 'r+' mode enables using write() as well as read() methods without closing a file."""

#Example 1

"""This program opens the file in w+ mode (which is a read-write mode), adds some data. The it seeks a certain position in file and overwrites its earlier contents with new text."""

fo8=open('foo.txt','r+')

fo8.seek(6,0)#it means start reading from 7 char and read total 11 char

data5=fo8.read(11)

print(data5)

fo8.close()

#Example 2

#open a file in read write mode

""" Opens the file 'foo1.txt' in write mode with the ability to read ('w+'). If the file does not exist, it will be created. If it does exist, its contents will be truncated."""

fo9=open('foo1.txt','w+')

fo9.write("This is rat race")#Write the string to the file

fo9.seek(8,0)#Moves the file cursor to the 8th byte from the beginning of the file (absolute seek).

data61=fo9.read(3)#Reads 3 bytes of data starting from the current file cursor position and stores it in the variable data61.

print(data61)

fo9.seek(8,0)#Moves the file cursor back to the 8th byte from the beginning of the file.

fo9.write('cat')

fo9.seek(0,0)#Moves the file cursor to the beginning of the file.

data6=fo9.read()#Reads the entire content of the file and stores it in the variable data6.

print(data6)

fo9.close()

# 10.3 Python - Renaming and Deleting Files

Python **os** module provides methods that help you perform file-processing operations, such as renaming and deleting files.

To use this module, you need to import it first and then you can call any related functions.

## rename() Method

The rename() method takes two arguments, the current filename and the new filename.

### **Syntax**

os.rename(current\_file\_name, new\_file\_name)

# 10.4 Python - Directories

All files are contained within various directories, and Python has no problem handling these too. The **os** module has several methods that help you create, remove, and change directories.

## The mkdir() Method

You can use the mkdir() method of the os module to create directories in the current directory. You need to supply an argument to this method, which contains the name of the directory to be created.

### **Syntax**

# 10.4.1 mkdir() method

#Example 1

"""Following is an example to create a directory test in the current directory −"""

import os

#Create a directory "test"

#os.mkdir("test1")

#10.4.2 The chdir() Method

"""

You can use the chdir() method to change the current directory.

The chdir() method takes an argument, which is the name of the directory that you want to make the current directory."""

#Example 2

"""

Following is an example to go into "/home/newdir" directory −"""

import os

#Create a directory to "/home/newdir"

os.chdir("E:\DCIM")

#10.4.3The getcwd() Method

"""The getcwd() method displays the current working directory."""

#Example 3

"""Following is an example to give current directory """

Current\_Dir=os.getcwd()

print("current working directory :",Current\_Dir )

#10.4.4 The rmdir() Method

"""

The rmdir() method deletes the directory, which is passed as an argument in the method.

Before removing a directory, all the contents in it should be removed."""

#Example 4

"""Following is an example to remove the "/tmp/test" directory."""

#os.rmdir("test")

#List Directories and Files in Python

list\_Dir=os.listdir(Current\_Dir)

print(list\_Dir)

#Also

print(os.listdir("E:\\"))

# 10.5 Python - File Methods

A file object is created using open() function. The file class defines the following methods with which different file IO operations can be done. The methods can be used with any file like object such as byte stream or network stream.

|  |  |
| --- | --- |
| **Sr.No.** | **Methods & Description** |
| 1 | [**file.close()**](https://www.tutorialspoint.com/python/file_close.htm)  Close the file. A closed file cannot be read or written any more. |
| 2 | [**file.flush()**](https://www.tutorialspoint.com/python/file_flush.htm)  Flush the internal buffer, like stdio's fflush. This may be a no-op on some file-like objects. |
| 3 | [**file\_fileno()**](https://www.tutorialspoint.com/python/file_fileno.htm)  Returns the integer file descriptor that is used by the underlying implementation to request I/O operations from the operating system. |
| 4 | [**file.isatty()**](https://www.tutorialspoint.com/python/file_isatty.htm)  Returns True if the file is connected to a tty(-like) device, else False. |
| 5 | [**next(file)**](https://www.tutorialspoint.com/python/next_file.htm)  Returns the next line from the file each time it is being called. |
| 6 | [**file.read([size])**](https://www.tutorialspoint.com/python/file_read_size.htm)  Reads at most size bytes from the file (less if the read hits EOF before obtaining size bytes). |
| 7 | [**file.readline([size])**](https://www.tutorialspoint.com/python/file_readline_size.htm)  Reads one entire line from the file. A trailing newline character is kept in the string. |
| 8 | [**file.readlines([sizehint])**](https://www.tutorialspoint.com/python/file_readlines_sizehint.htm)  Reads until EOF using readline() and return a list containing the lines. If the optional sizehint argument is present, instead of reading up to EOF, whole lines totalling approximately sizehint bytes (possibly after rounding up to an internal buffer size) are read. |
| 9 | [**file.seek(offset[, whence])**](https://www.tutorialspoint.com/python/file_seek_offset_whence.htm)  Sets the file's current position |
| 10 | [**file.tell()**](https://www.tutorialspoint.com/python/file_tell.htm)  Returns the file's current position |
| 11 | [**file.truncate([size])**](https://www.tutorialspoint.com/python/file_truncate_size.htm)  Truncates the file's size. If the optional size argument is present, the file is truncated to (at most) that size. |
| 12 | [**file.write(str)**](https://www.tutorialspoint.com/python/file_write_str.htm)  Writes a string to the file. There is no return value. |
| 13 | [**file.writelines(sequence)**](https://www.tutorialspoint.com/python/file_writelines_sequence.htm)  Writes a sequence of strings to the file. The sequence can be any iterable object producing strings, typically a list of strings. |

Let us go through the above methods briefly.

#EXample 1 fileno() method

# Open a file

fo = open("fo.txt", "wb")

print ("Name of the file: ", fo.name)

fid = fo.fileno()

print ("File Descriptor: ", fid)

# Close the opened file

fo.close()

#EXample 2 isatty() method

"""The method isatty() returns True if the file is connected

(is associated with a terminal device) to a tty(-like) device, else False."""

# Open a file

fo = open("foo.txt", "wb")

print ("Name of the file: ", fo.name)

ret = fo.isatty()

print ("Return value : ", ret)

# Close the opened file

fo.close()

# Python - OS File/Directory Methods

The **os** module provides a big range of useful methods to manipulate files. Most of the useful methods are listed here −

|  |  |
| --- | --- |
| **Sr.No.** | **Methods with Description** |
| 1 | **os.close(fd)**  Close file descriptor fd. |
| 2 | [**os.closerange(fd\_low, fd\_high)**](https://www.tutorialspoint.com/python/os_closerange_method.htm)  Close all file descriptors from fd\_low (inclusive) to fd\_high (exclusive), ignoring errors. |
| 3 | [**os.dup(fd)**](https://www.tutorialspoint.com/python/os_dup_method.htm)  Return a duplicate of file descriptor fd. |
| 4 | [**os.fdatasync(fd)**](https://www.tutorialspoint.com/python/os_fdatasync_method.htm)  Force write of file with filedescriptor fd to disk. |
| 5 | [**os.fdopen(fd[, mode[, bufsize]])**](https://www.tutorialspoint.com/python/os_fdopen_method.htm)  Return an open file object connected to the file descriptor fd. |
| 6 | [**os.fsync(fd)**](https://www.tutorialspoint.com/python/os_fsync_method.htm)  Force write of file with filedescriptor fd to disk. |
| 7 | [**os.ftruncate(fd, length)**](https://www.tutorialspoint.com/python/os_ftruncate_method.htm)  Truncate the file corresponding to file descriptor fd, so that it is at most length bytes in size. |
| 8 | [**os.lseek(fd, pos, how)**](https://www.tutorialspoint.com/python/os_lseek_method.htm)  Set the current position of file descriptor fd to position pos, modified by how. |
| 9 | [**os.open(file, flags[, mode])**](https://www.tutorialspoint.com/python/os_open_method.htm)  Open the file file and set various flags according to flags and possibly its mode according to mode. |
| 10 | [**os.read(fd, n)**](https://www.tutorialspoint.com/python/os_read_method.htm)  Read at most n bytes from file descriptor fd. Return a string containing the bytes read. If the end of the file referred to by fd has been reached, an empty string is returned. |
| 11 | **os.tmpfile()**  Return a new file object opened in update mode (w+b). |
| 12 | [**os.write(fd, str)**](https://www.tutorialspoint.com/python/os_write_method.htm)  Write the string str to file descriptor fd. Return the number of bytes actually written. |