Zed-King
GROUP OF INSTITUTIONS
ISO 9001 : 2015 Certified

O-LEVEL (M3-R5.1)
Python Programming By
Mrs. Geeta
Python Instructor
Zed-King Group of Institutions
Kaithal

Zed-King
GROUP OF INSTITUTIONS
ISO 9001 : 2015 Certified

O-LEVEL (M3-R5.1)

Python Programming

Video Tutorial

O-LEVEL (M3-R5.1) Full Video Playlist Available on YouTube Channel **Zed-king Computer Institute Kaithal**

Zed-King
GROUP OF INSTITUTIONS
ISO 9001 : 2015 Certified

Free Online Computer Classes On YouTube Channel

https://www.youtube.com/@zedkinginstitute

For Free PDF Notes

Our Telegram Channel

t.me/boost/Zedking_Institute

Zed-King
GROUP OF INSTITUTIONS
ISO 9001 : 2015 Certified

O-LEVEL (M3-R5.1)
CHAPTER-6

Functions

Function:-

A function is a program that performs a specific task. Python function is a group of statements. Every program has a function. Wherever the statements of the function are needed, the function is called.

"Function is a block of code which only runs when it is called".

Advantages of function:-

- The code written in the function does not have to be written again and again
- Large programs can be divided into small functions
- Function saves the programmer's time, program's space and memory
- If an error occurs in the program, it can be easily removed
- The function can be called again and again where needed

Types of Function in Python:-

There are two types of functions in Python programming language

1. In-Built/Predefined Function
2. User-Defined Function

In-Built/Predefined Function

It is also called library function. The function which is already defined in the python library is called library function. There are many types of library functions in python programming like chr(), id(), print(), ord(), type(), len(), range() etc.

User-Defined Function

These are the functions which are created by the user. Users create these functions according to their requirements. The function which is defined by the user is called user defined function.

Syntax:-

```
def function_name(parameters):

        # function body

        # What the function does goes here

        # write some action

        return result
```

Hint:-

def:- Defining a function in Python begins with the def keyword

Function_name:- We can give any name in it and later we can call the function with the same name.

Parameters:- After the function name, you can pass 0 or more parameters between parentheses () which are given as values, arguments at the time of function calling

Note:- A colon (:) must be placed after the parentheses ()

Function body:- The function body is an indented block of code that contains statements that perform a specific task

Return result:- The result is returned through this

1. wap to print "Hello world" & your name using function.

```
def msg():
    print("Hello World")
msg()
```

```
def name():
    print("AK IT SOLUTION")
name()
```

2. wap to find addition of two number using functions.

```
def add():
    a=int(input("Enter first number "))
    b=int(input("Enter second number "))
    c=a+b
    print("Addition is= ",c)
add()
```

3. wap to check entered number is even number or odd number using function

```
def ch():
    n=int(input("Enter any number "))
    if n%2==0:
        print("Even No")
    else:
        print("Odd No.")
ch()
```

4. wap to find factorial of any number using function

```
def fact():
    n=int(input("Enter any number "))
    fact=1
    while n>0:
        fact=fact*n
        n=n-1
    print("Factorial is : ",fact)
fact()
```

return statement:-

The return statement is used to return the value to the calling function. Upon exit from the called function, the execution control is transferred to the calling function. Return is a keyword which can be used anywhere in the body of the function.

```
def add():
    a=int(input("Enter first number "))
    b=int(input("Enter second number "))
    c=a+b
    return c
res=add()
print("Addition is = ",res)
```

1. **wap to find factorial of given number using return statement.**

Argument:-

The value or variable given inside the bracket ( ) of the function is called argument. It is of two types

1. Actual argument
2. Formal argument

Actual Argument:-

The argument which is written within the bracket ( ) of the calling function is called the actual argument

Formal Argument:-

The argument which is written within the bracket ( ) of the called function is called the formal argument

```
def add(a,b):  ————> formal argument
    c=a+b
    print(c)
add(10,20)  ————> actual argument
```

Note: 10, a में store होगा, 20, b में

Types of argument in python:-

There are four types of arguments in Python

1. Positional or required arguments
2. Default argument
3. Keyword argument
4. Arbitrary or variable-length arguments

Positional Argument:-

This type of argument function is provided in current positional order. If the position of argument in the function argument is changed then the result also changes.

```
def add(a,b):
    c=a+b
    d=a-b
    print(c)
    print(d)
add(20,10)
```

Default Argument:-

By default value can also be provided in positional argument. If no value is given to the function then only default value is passed.

```
def add(a,b=10):

   c=a+b;

   print(c)

add(20)
```

Keyword Argument:-

The argument of the function can also be passed by using the keyword value

```
def info( name, age ):
   print ("Name: ", name)
   print ("Age ", age)
# by keyword arguments
info(name="anil", age = 30)
info(age=25,name='akhilesh')
```

Note:- i. The value will not change if the position changes.

ii. In keyword argument the order of argument is not important but number of argument is important

variable-length arguments:-

Many times we have to define a function which can take any number of (arbitrary) arguments when called. For this, we put an asterisk (*) before the parameter name in the function.

When we do this, we can pass any number of arguments when calling the function and all these arguments are stored as a tuple in the arbitrary parameter of our function.

```
def fun(*num):
    print(num)
fun(10,20)
fun(10,20,30,40,50)
```

Q.1 wap to input a number and print square of value using function

```
def sqr(a):
    n=a**2
    print("Square is: ",n)
num=int(input("Enter any number "))
sqr(num)
```

```
def sqr(a):
    n=a**2
    print("Square is: ",n)
sqr(5)
```

2. wap to count to vowel or consonant of any word entered by user by using function

3. define a function which counts upper case character and lower case character in a word

Type of Variable:-

There are many types of variables

Local variable:-

The variable which is defined inside the body or block of a function. That variable is a local variable of that function or block. The local variable which is defined inside a block or function is used inside that block or function. In the same program, variables can be defined in two different functions of the same name.

```
def fun():
    a=10 ————————-> local variable
    print(a)
fun()
```

Global variable:-

The variable which is not defined inside any function is called a global variable. This variable is usually defined above the function. The global variable can be accessed and modified throughout the program.

```
a=20 ————————> global variable
def fun():
   a=10 ————————> local variable
   print(a)
fun()
print(a)
```

Global keyword:-

In Python, the global keyword is used to declare a local variable as a global variable.

```
def fun():
    global a
    a=10
fun()
print(a)
```

Nested function:-

When a function is defined inside another function, it is called a nested function

```
def fun1():
    print("outer function")
    def fun2():
        print("inner function")
    fun2()
fun1()
```

Output

```
outer function
inner function
```

Note:- 1. Fun2 is a local function of fun1

2. fun2 cannot be directly accessed outside fun1.

Aliasing function:-

You can also give another name to an already available function

```
def al():
    print("welcome to ak it solution")
al()
ali=al
ali()
```

Output

```
welcome to ak it solution
welcome to ak it solution
```

Note:- 1. When a function is aliased, a function definition remains available but that function can be called either by the original function name or by the aliasing function name

2. If the original function is deleted then its definition function can be called from the aliasing function.

Recursion Function:-

जब function स्वयं को call करता है तो उसे recursion function के नाम से जाना जाता है

```
def fun_name():
    .....
    .....
    .....
    .....
    fun_name()  ————> recursion function
fun_name()
```

1: wap to print 1 to 10 number using recursion functions

2: wap to print all even numbers between 1 to 50 using recursion function.

3: wap to find factorial of any number using recursion function

1.

```
def num(a):
    if a<=10:
        print(a," ",end=" ")
        a=a+1
        num(a)
num(1)
```

2.

```
def num(a):
    if a<=50:
        if a%2==0:
            print(a," ",end=" ")
        a=a+1
        num(a)
num(1)
```

3.

```
def factorial(n):
  if n == 1:
      return n
  else:
      return n*factorial(n-1)
num = int(input("Enter any number: "))
if num < 0:
   print("Factorial does not exist for negative numbers")
elif num == 0:
   print("The factorial of 0 is 1")
else:
   print("The factorial of", num, "is", factorial(num))
```

Lambda Functions

Lambda functions are defined anonymously without any name. You assign them to an object and they are used by the same object name. Lambda function is also called anonymous function.

lambda function can take any number of arguments, but can only have one expression.

Lambda Keyword

In Python, normal functions are defined by the def keyword and lambda functions are defined by the lambda keyword.

Syntax: **lambda <arguments> : expression**

- Lambda function allows only one line

```
x=lambda a:a+10
y=x(5)
print(y)
```

Output

15

```
x=lambda a:a+10
print(x(5))
```

Output

15

Note 1:- Arguments are defined after the lambda keyword. Just like normal functions, you can pass any number of arguments as per your wish to lambda functions in Python.

lambda <x1, x2, x3, x4.....xN> : <expression>

```
a=lambda x,y,z:x+y+z
print(a(10,20,30))
```

Output

60

```
x=lambda a,b:a*b
print(x(10,2))
```

Output

20

Note 2:- The expression is defined by putting a colon (:) after the arguments. You can understand this expression like the body of a function.

lambda <x1,x2,x3,....xN> : <single – Expression>

Note 3:- Lambda functions are defined and assigned to an object. After this, you use the lambda function through that object only. This is a function object.

obj = lambda <x1, x2, x3,.....xN> : <expression>

Note 4:- Only one expression can be defined in Lambda functions. The syntax of Lambda functions does not allow multiple expressions to be used. This is why lambda functions are also called one line functions.

Note 5:- In Python, lambda functions are mostly used as arguments in filter(), map() and reduce() functions.

Q 1:- create a lambda function to calculate remainder

```
rem=lambda a,b:a%b
print(rem(10,3))
```

Output

```
1
```

```
rem=lambda a,b:a%b
a=int(input("Enter 1st number "))
b=int(input("Enter 2nd number "))
result=rem(a,b)
print("Remainder is : ",result)
```

Q. 2:- find the Square of a number by using lambda function

```
sq=lambda a:a*a
a=int(input("enter any number "))
res=sq(a)
print("Square is ",res)
```

```
sq=lambda a:a*a
print(sq(5))
```

Output

```
25
```

filter() Function

In Python, filter is a function that is used to extract data from a list by filtering it based on logic. In this function, you pass a function and a list as arguments. The logic is performed with all the items in the list and the item that is in accordance with that logic is stored. Thus, all the items that fulfill that logic are returned.

The filter() function returns an iterator where the items are filtered through a function to test if the item is accepted or not.

```
ages=[5,10,15,18,20,25,35]
def myfunc(x):
    if x<18:
        return False
    else:
        return True
adults=list(filter(myfunc,ages))
for x in adults:
    print(x)
```

Output

18 20 25 35

Using Filter Function with Lambda function

```
ages=[5,10,15,18,20,25,35]
adult=filter(lambda a:a>18,ages)
for x in adult:
    print(x)
```

Output

20 25 35

Question:- Wap to filter the even number given range

```
val=[5,10,13,17,15,18,20,25,30]
def evencheck(x):
    if x%2==0:
        return True
    else:
        return False
even=filter(evencheck,val)
for x in even:
    print(x)
```

Output: 10 18 20 30

Using lambda Function

```
val=[5,10,13,17,15,18,20,25,30]
even=filter(lambda a:a%2==0,val)
for x in even:
    print(x)
```

Output

10 18 20 30

map() Function

The map() function executes a specified function for each item in a iterable. The item is sent to the function as parameter.

Syntax: map(function name ,iterable)

```
a=[5,10,15,20,25,30]
def myfun(a):
    if a%2==0:
        return True
    else:
        return False
def squ(a):
    return a*a
val=list(filter(myfun,a))
square=list(map(squ,val))
print(val)
print(square)
```

output

```
[10, 20, 30]
[100, 400, 900]
```

Without filter function

```
ages=[5,10,15,18,20,25,35]
sqr=list(map(lambda a:a*a,ages))
print(sqr)
```

Output

```
[25, 100, 225, 324, 400, 625, 1225]
```

Using lambda function

```
ages=[5,10,13,15,17,18,20,22,25]
adults=filter(lambda a:a>18,ages)
square=list(map(lambda a:a*a,adults))
print(square)
```

output

```
[400, 484, 625]
```

reduce() Function

This function is used to reduce a sequence of elements to a single value by processing the elements according to a function supplied. It returns a single value. This function is a part of functools module so you have to import it before using

Syntax:- from functools import reduce

Reduce(function name, sequence)

With lambda function

```
from functools import reduce
a=[5,10,15,18]
res=reduce(lambda n,m:n+m,a)
print(res)
```

output

48

Without lambda function

```
from functools import reduce
def myfun(a,b):
    return a+b
val=[10,20,30]
add=reduce(myfun,val)
print("Addition is ",add)
```

output

Addition is 60

1: Finding max number In the list using reduce () function.

Without lambda function

```
from functools import reduce
def myfun(a,b):
    if a>b:
        return a
    else:
        return b
val=[20,10,30,50]
max=reduce(myfun,val)
print("maximum no. is ", max)
```

output

```
maximum no. is  50
```

With lambda function

```
from functools import reduce
val=[20,10,30,60,50]
max=reduce(lambda a,b:a if a>b else b,val)
print("max is ",max)
```

output

```
max is  60
```

Numeric Function:-

1. eval()	2. max()	3. min()	4. pow()	5. round()
6. int()	7. abs()	8. ceil()	9. floor()	10. fabs()
11. sqrt()	12. exp()	13. divmod()	14. factorial()	15. f.mod()
16. f.sum()				

max():-

Python built-in function max() returns the largest of its parameters

Example:-

```
def main():
    a=int(input("Enter first number"))
    b=int(input("Enter second number"))
    c=int(input("Enter third number "))
    print("Largest number is ",max(a,b,c))
main()
```

Output

```
Enter first number10
Enter second number50
Enter third number 40
Largest number is  50
```

min():-

Python built in function min() returns the smallest of its parameters.

Example:-

```
def main():
    a=int(input("Enter first number"))
    b=int(input("Enter second number"))
    c=int(input("Enter third number "))
    print("Minimum number is ",min(a,b,c))
main()
```

Output

```
Enter first number40
Enter second number30
Enter third number 50
Minimum number is  30
```

pow():-

The pow(x,y) function returns the value of x raised to the power of y, i.e. xy.

Both pow() and the double star (**) operator perform exponentiation. ** is an operator and pow() is a built-in function.

Note:-1. ** operator, math.pow() converts both its arguments to type float.

2. use ** or the built in pow() function for computing exact integer powers.

Example:-

```
import math
print(math.pow(2,2))
print(pow(2,2))
print(pow(2,-2))
```

Output

```
4.0
4
0.25
```

round():-

The round() built-in function has a syntax of round (flt,ndig=0). It normally rounds a floating point number to the nearest integral number and returns that result as a float.

Example:-

```
print(round(3.45))

print(round(3.45654,1))

print(round(3.43234,1))
```

Output

```
3

3.5

3.4
```

int():-

It converts integer literal, float literal and string literal into integer number.

Example:-

```
print(int(1))

print(int(5.30))

print(int("10"))
```

Output

```
1

5

10
```

abs():-

It returns the absolute value of the given argument.

Example:-

```
print(abs(-1))

print(abs(10.50))
```

output

```
1

10.5
```

ceil():-

It returns the smallest integer number which is greater than or equal to integer.

Example:-

import math print(math.ceil(35.3)) print(math.ceil(-35.50))	**Output** 36 -35	#Import math library import math #Round a number upward to its nearest integer print(math.ceil(1.4)) # 2 print(math.ceil(5.3)) # 6 print(math.ceil(-5.3)) # -5 print(math.ceil(22.6)) # 23 print(math.ceil(10.0)) # 10

floor():-

Floor() method returns the largest whole number less than or equal to integer.

Example:-

import math	**Output**
print(math.floor(35.3))	35
print(math.floor(-35.50))	-36
print(math.floor(10.0))	10

Zed-King
GROUP OF INSTITUTIONS
ISO 9001 : 2015 Certified

fabs():-

It function returns the absolute value of x.

Example:-

	Output
import math print(math.fabs(-35.3)) print(math.fabs(-35.50)) print(math.fabs(10.0))	35.3 35.5 10.0

sqrt():-

The sqrt() function returns the non-negative square root of x. if x is negative, a domain error occurs.

Example:-

import math	output
print(math.sqrt(25))	5.0
print(math.sqrt(10))	3.1622776601683795
print(math.sqrt(100))	10.0

exp():-

The exp() function returns the exponential number E raised to the power of a.

Example:-

import math print(math.exp(5)) print(math.exp(5.0)) print(math.exp(2.23))	**output** 148.4131591025766 148.4131591025766 9.299866079483586

divmod():-

Divmod() built-in function combines division and modulus operations into a single function call that returns the pair (quotient, remainder) as a tuple.

Example:-

print(divmod(10,3)) def main(): a=int(input("enter first value")) b=int(input("enter Second value")) print("Quotient and Remainder are ",divmod(a,b)) main()	**output** (3, 1) enter first value35 enter Second value3 Quotient and Remainder are (11, 2)

Math.factorial():-

The math.factorial() method returns the factorial of a number.

Note: This method only accepts positive integers.

Example

Find the factorial of a number:

import math	**Output:**
print(math.factorial(9))	362880
print(math.factorial(6))	720
print(math.factorial(12))	479001600
print(math.factorial(5))	120

Math.fsum():-

The math.fsum() method returns the sum of all items in any iterable (tuples, arrays, lists, etc.).

Example

Return the sum of all items:

import math print(math.fsum([1, 2, 3, 4, 5])) print(math.fsum([100, 400, 340, 500])) print(math.fsum([1.7, 0.3, 1.5, 4.5]))	**Output** 15.0 1340.0 8.0

Math.fmod():-

The math.fmod() method returns the remainder (modulo) of x/y.

Example

Return the remainder of x/y:

import math print(math.fmod(20, 4)) print(math.fmod(20, 3)) print(math.fmod(15, 6)) print(math.fmod(-10, 3)) print(math.fmod(0, 0))	**Output:** 0.0 2.0 3.0 -1.0 print(math.fmod(0, 0)) ValueError: math domain error

The random Module:-

Python has a built-in module that you can use to make random numbers.

random() Method

The random() method returns a random floating number between 0 and 1 (0 including, but excluding 1).

Example:-

```
import random
print(random.random())
```

Output:

```
0.73
```

randint() Method

The randint() method returns an integer number selected element from the specified range.

Syntax

random.randint(start, stop)

Parameter Values

Parameter	Description
start	Required. An integer specifying at which position to start.
stop	Required. An integer specifying at which position to end.

Example

Return a number between 3 and 9 (both included):

```
import random
print(random.randint(3,9))
```

Output:

4

randrange() Method

The randrange() method returns a randomly selected element from the specified range.

Syntax

random.randrange(start, stop, step)

Parameter	Description
start	Optional. An integer specifying at which position to start. Default 0
stop	Required. An integer specifying at which position to end.
step	Optional. An integer specifying the incrementation. Default 1

Example

Return a number between 3 and 9:

```
import random
print(random.randrange(3, 9))

#returns a number between 3 (included) and 9 (not included)
```

Output:

5

choice() Method

The choice() method returns a randomly selected element from the specified sequence.

The sequence can be a string, a range, a list, a tuple or any other kind of sequence.

Syntax

random.choice(sequence)

Parameter Values

Parameter	Description
sequence	Required. A sequence like a list, a tuple, a range of numbers etc.

Example

Return a random element from a list:

```
import random
myl = ["apple", "banana", "cherry"]
print(random.choice(myl))
```

Output:

Apple

Example

Return a random character from a string:

```
import random
x = "WELCOME"
print(random.choice(x))
```

Output:

C

shuffle() Method

The shuffle() method takes a sequence, like a list, and reorganize the order of the items.

Note: This method changes the original list, it does not return a new list.

Syntax

random.shuffle(sequence)

Parameter Values

Parameter	Description
sequence	Required. A sequence.
function	Deprecated since Python 3.9. Removed in Python 3.11. Optional. The name of a function that returns a number between 0.0 and 1.0. If not specified, the function random() will be used

Example

Shuffle a list (reorganize the order of the list items):

```
import random
mylist = [“apple”, “banana”, “cherry”]
random.shuffle(mylist)
print(mylist)
```

Output:

[“apple”, “cherry”, “banana”]

uniform() Method

The uniform() method returns a random floating number between the two specified numbers (both included).

Syntax

random.uniform(a, b)

Parameter Values

Parameter	Description
a	Required. A number specifying the lowest possible outcome
b	Required. A number specifying the highest possible outcome

Example

Return a random number between, and included, 20 and 60:

```
import random
print(random.uniform(20, 60))
```

Output:

53.29

Time():-

To get how many ticks have passed since the epoch, you call the python time() method.

Example:

import time print(time.time()) **output** 1687421227.8807673	from datetime import time a=time() print(a) b=time(10,20,40) print(b) **Output** 00:00:00 10:20:40	from datetime import time a=time(10,20,30) print("hour: ",a.hour) print("Minute: ",a.minute) print("Second: ",a.second) **Output:-** hour: 10 Minute: 20 Second: 30

Localtime():-

It function returns the current time according to your geographical location, in a more readable format.

```
import time
print(time.localtime(time.time()))
```

Asctime():-

The time can be formatted by using the asctime() function on time module. It returns the formatted time for the time tuple being passed.

```
import time
print(time.asctime())
```

Output

```
Thu Jun 22 13:52:55 2023
```

Clock():-

It returns the number of seconds elapsed since the first call made to it. It returns the value as a floating number.

```
import time
processing_time = time.clock()
print("Current Processing Time (in sec):", processing_time)
```

Sleep():-

The sleep() method of time module is used to stop the execution of the script for a given amount of time. The output will be delayed for the number of seconds.

```
import time
for i in range(0,10):
    print(i)
    time.sleep(2)
```

Now():-

The now() method to create a datetime object containing the current local date and time. The contains year, month,day, hour, minute, second, and microsecond

```
import datetime
print(datetime.datetime.now())
```

Output

```
2023-06-22 14:44:16.380854
```

Today():-

Today() metho defined in the date class to get a date object containing the current local date.

```
import datetime

print(datetime.date.today())

dat=datetime.date.today()

print(dat)
```

Output

```
2023-06-22

2023-06-22
```

```
from datetime import date

today=date.today()

print("Current Year ",today.year)

print("Current Month ",today.month)

print("Current Day ",today.day)
```

Output

```
Current Year  2023

Current Month  6

Current Day  22
```

Date():-

The date() is a constructor of the date class. It takes three arguments: Year, Month and day.

```
import datetime
d=datetime.date(2020,7,10)
print(d)
```

Output

```
2020-07-10
```

Dir():-

The dir() function to get a list containing all attributes of a module.

```
import datetime

print(dir(datetime))
```

Timestamp():-

You can also create date objects from a timestamp. You can convert a timestamp to date using from timestamp() method.

```
import time
ts = time.time()
print(ts)
```

Output

```
1717608192.4099374
```

Strftime():-

This method takes one or more format codes and returns a formatted string based on it.

```
import datetime
x=datetime.datetime.now()
print(x.year)
print(x.strftime("%a"))
print(x.strftime("%w"))
print(x.strftime("%d"))
print(x.strftime("%b"))
```

Output

```
2024
Wed
3
05
Jun
```

A reference of all the legal format codes:

Directive	Description	Example
%a	Weekday, short version	Wed
%A	Weekday, full version	Wednesday
%w	Weekday as a number 0-6, 0 is Sunday	3
%d	Day of month 01-31	31
%b	Month name, short version	Dec
%B	Month name, full version	December
%m	Month as a number 01-12	12
%y	Year, short version, without century	18
%Y	Year, full version	2018
%H	Hour 00-23	17
%I	Hour 00-12	05
%p	AM/PM	PM
%M	Minute 00-59	41
%S	Second 00-59	08
%f	Microsecond 000000-999999	548513
%z	UTC offset	+0100

%Z	Timezone	CST
%j	Day number of year 001-366	365
%U	Week number of year, Sunday as the first day of week, 00-53	52
%W	Week number of year, Monday as the first day of week, 00-53	52
%c	Local version of date and time	Mon Dec 31 17:41:00 2018
%C	Century	20
%x	Local version of date	12/31/18
%X	Local version of time	17:41:00
%%	A % character	%
%G	ISO 8601 year	2018
%u	ISO 8601 weekday (1-7)	1
%V	ISO 8601 weeknumber (01-53)	01

The Calendar Module:

Python provide a calendar object that contains various methods to work with the calendars.

```
import calendar

print(calendar.month(2022,7))
```

```
     June 2024
Mo Tu We Th Fr Sa Su
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
```

Note:- you can also print the calendar of whole year. The prcal() method of calendar module is used to print the calendar of the whole year.

```
import calendar

print(calendar.prcal(2022))
```

Isleap():-

The isleap() takes a 4 digit year as an argument, and return True if it is a leap year. Otherwise False.

```
import calendar

print(calendar.isleap(2000)) #this is leap year

print(calendar.isleap(2022)) # this is not leap year
```

leapdays():-

the leapdays() returns the total number of leap days from year y1 to year y2.

```
import calendar

print(calendar.leapdays(2000,2022))
```

output:

```
6
```

firstweekday():-

The firstweekday() returns which day is currently set as the first day of the week.

```
import calendar
calendar.setfirstweekday(1)
print(calendar.firstweekday())
```

String Function

पायथन में कई built-in function है जो स्ट्रिंग डेटा टाइप से जुड़े हैं

1. count()	2. find()	3. rfind()	4. capitalize()	5. title()	6. lower()
7. upper()	8. swapcase()	9.islower()	10.isupper()	11. istitle()	12. replace(old,new)
13.strip()	14.lstrip()	15.rstrip()	16. split()	17.partion()	18. join()
19.isspace()	20.isalpha()	21.isdigit()	22.isalnum()	23.startswith()	24. endswith()
25.encode()	26.decode()	27. index()			

Count:-

count returns the number of times a specified value occurs in a string.

Syntax:- **string.count(substring, start, end)**

st="I Love Python programming language" a=st.count("Python") print(a) **Output** 1	st="I Love Python Programming Language" a=st.count("P") print(a) b=st.count('P',5,50) print(b) **Output** 2 2

find:-

The find() method finds the first occurrence of the specified value. The find() method returns -1 if the value is not found. The find() method is almost the same as the index() method, the only difference is that the index() method raises an exception if the value is not found. (See example below)

Syntax:- ***string*.find(*value, start, end*)**

txt = "Hello, welcome to my world." x = txt.find("e") print(x) # 1	txt = "Hello, welcome to my world." x = txt.find("e", 5, 10) print(x) # 8	txt = "Hello, welcome to my world." print(txt.find("q")) # -1 print(txt.index("q")) # valueError

rfind():-

The rfind() method finds the last occurrence of the specified value. The rfind() method returns -1 if the value is not found. The rfind() method is almost the same as the rindex() method. See example below.

Syntax:- ***string*.rfind(*value, start, end*)**

txt = "Hello, welcome to my world." x = txt.rfind("e") print(x) # 13	txt = "Hello, welcome to my world." x = txt.rfind("e", 5, 10) print(x) # 8	txt = "Hello, welcome to my world." print(txt.rfind("q")) # -1 print(txt.rindex("q")) # valueError

capitalize():-

The capitalize() method returns a string where the first character is upper case, and the rest is lower case.

Syntax:- ***string*.capitalize()**

txt = "ak it solution!" x = txt.capitalize() print (x) **# Ak it solution!**	txt = "28 is my age." x = txt.capitalize() print (x) **# 28 is my age.**

title():-

The title() method returns a string where the first character in every word is upper case. If the word contains a number or a symbol, the first letter after that will be converted to upper case.

Syntax:- ***string*.title()**

Note: first letter after a non-alphabet letter is converted into a upper case letter:

```
txt = "Welcome to my 2nd lecture"
x = txt.title()
print(x)
```

Output

Welcome To My 2Nd Lecture

```
txt = "hello b2b2b2 and 3g3g3g"
x = txt.title()
print(x)
```

Output

Hello B2B2B2 And 3G3G3G

lower():-

The lower() method returns a string where all characters are lower case. Symbols and Numbers are ignored.

Syntax:- ***string*.lower()**

```
txt = "Hello my FRIENDS"
x = txt.lower()
print(x)
```

Output

hello my friends

upper():-

The upper() method returns a string where all characters are in upper case. Symbols and Numbers are ignored.

Syntax:- ***string*.upper()**

```
txt = "Hello my friends"
x = txt.upper()
print(x)
```

Output

```
HELLO MY FRIENDS
```

swapcase():-

The swapcase() method returns a string where all the upper case letters are lower case and vice versa.

Syntax:- ***string*.swapcase()**

```
txt = "Hello My Name Is AKHILESH"
x = txt.swapcase()
print(x)
```

Output

```
hELLO mY nAME iS akhilesh
```

islower():-

The islower() method returns True if all the characters are in lower case, otherwise False. Numbers, symbols and spaces are not checked, only alphabet characters.

Syntax:- ***string*.islower()**

```
txt = "hello world!"
x = txt.islower()
print(x)
```

Output

True

```
a = "Hello world!"
b = "hello 123"
c = "mynameisAkhilesh"
print(a.islower())
print(b.islower())
print(c.islower())
```

Output

False
True
False

isupper():-

The isupper() method returns True if all the characters are in upper case, otherwise False. Numbers, symbols and spaces are not checked, only alphabet characters.

Syntax:- ***string*.isupper()**

```
txt = "THIS IS NOW!"
x = txt.isupper()
print(x)
```

Output

True

```
a = "Hello World!"
b = "hello 123"
c = "MY NAME IS AKHILESH"
print(a.isupper())
print(b.isupper())
print(c.isupper())
```

Output

False

False

True

istitle():-

The istitle() method returns True if all words in a text start with a upper case letter, AND the rest of the word are lower case letters, otherwise False. Symbols and numbers are ignored.

Syntax:- ***string*.istitle()**

```
txt = "Hello, And Welcome To My World!"
x = txt.istitle()
print(x)
```

Output

```
True
```

```
a = "HELLO, AND WELCOME TO MY WORLD"
b = "Hello"
c = "22 Names"
d = "This Is %'!?"
print(a.istitle())
print(b.istitle())
print(c.istitle())
print(d.istitle())
```

Output

```
False
True
True
True
```

replace():-

The replace() method replaces a specified phrase with another specified phrase.

Syntax:- ***string*.replace(*oldvalue, newvalue, count*)**

```
txt = "I like bananas"
x = txt.replace("bananas", "apples")
print(x)
```

Output

I like apples

```
txt = "one one was a race horse, two two was one too."
x = txt.replace("one", "three")
print(x)
```

Output

three three was a race horse, two two was three too.

```
txt = "one one was a race horse, two two was one too."
x = txt.replace("one", "three", 2)
print(x)
```

Output

three three was a race horse, two two was one too.

strip():-

The strip() method removes any leading, and trailing whitespaces. Leading means at the beginning of the string, trailing means at the end. You can specify which character(s) to remove, if not, any whitespaces will be removed.

Syntax:- ***string*.strip(*characters*)**

txt = " banana " x = txt.strip() print("of all fruits", x, "is my favorite") **Output** of all fruits banana is my favorite	txt = ",,,,,rrttgg.....banana....rrr" x = txt.strip(",.grt") print(x) **Output** banana

lstrip():-

The lstrip() method removes any leading characters (space is the default leading character to remove)

Syntax:- ***string*.lstrip(*characters*)**

txt = " banana " x = txt.lstrip() print("of all fruits", x, "is my favorite") **Output** of all fruits banana is my favorite	txt = ",,,,,ssaaww.....banana" x = txt.lstrip(",.asw") print(x) **Output** banana

rstrip():-

The rstrip() method removes any trailing characters (characters at the end a string), space is the default trailing character to remove.

Syntax:- ***string*.rstrip(*characters*)**

txt = " banana " x = txt.rstrip() print("of all fruits", x, "is my favorite") **Output** of all fruits banana is my favorite	txt = "banana,,,,,ssqqqww....." x = txt.rstrip(",.qsw") print(x) **Output** banana

split():-

The split() method splits a string into a list. You can specify the separator, default separator is any whitespace.

Syntax:- ***string*.split(*separator, maxsplit*)**

Note:-

separator	Optional. Specifies the separator to use when splitting the string. By default any whitespace is a separator
maxsplit	Optional. Specifies how many splits to do. Default value is -1, which is "all occurrences"

txt = "hello, my name is Peter, I am 26 years old" x = txt.split(", ") print(x) **Output** ['hello', 'my name is Peter', 'I am 26 years old']	txt = "apple#banana#cherry#orange" x = txt.split("#") print(x) **Output** ['apple', 'banana', 'cherry', 'orange']

```
txt = "apple#banana#cherry#orange"

# setting the maxsplit parameter to 1, will return a list with 2 elements!

x = txt.split("#", 1)

print(x)
```

Output

['apple', 'banana#cherry#orange']

partition( ):-

The partition() method searches for a specified string, and splits the string into a tuple containing three elements. The first element contains the part before the specified string. The second element contains the specified string. The third element contains the part after the string.

Syntax:- ***string*.partition(*value*)**

```
txt = "I could eat bananas all day"
x = txt.partition("bananas")
print(x)
```

Output

```
('I could eat ', 'bananas', ' all day')
```

Note:- If the specified value is not found, the partition() method returns a tuple containing: 1 – the whole string, 2 – an empty string, 3 – an empty string:

```
txt = "I could eat bananas all day"
x = txt.partition("apples")
print(x)
```

Output

```
('I could eat bananas all day', '', '')
```

join( ):-

The join() method takes all items in an iterable and joins them into one string. A string must be specified as the separator.

Syntax:- ***string*.join(*iterable*)**

myTuple = ("John", "Peter", "Vicky") x = "#".join(myTuple) print(x) **Output** John#Peter#Vicky	myDict = {"name": "John", "country": "Norway"} mySeparator = "TEST" x = mySeparator.join(myDict) print(x) **Output** nameTESTcountry

Note: When using a dictionary as an iterable, the returned values are the keys, not the values.

isspace():-

The isspace() method returns True if all the characters in a string are whitespaces, otherwise False.

Syntax:- ***string*.isspace()**

txt = " " x = txt.isspace() print(x) **Output** True	txt = " s " x = txt.isspace() print(x) **Output** False

isalpha():-

The isalpha() method returns True if all the characters are alphabet letters (a-z). Example of characters that are not alphabet letters: (space)!#%&? etc.

Syntax:- ***string*.isalpha()**

txt = "CompanyX" x = txt.isalpha() print(x) **Output** True	txt = "Company10" x = txt.isalpha() print(x) **Output** False

isdigit():-

The isdigit() method returns True if all the characters are digits, otherwise False. Exponents, like ², are also considered to be a digit.

Syntax:- ***string*.isdigit()**

txt = "50800"
x = txt.isdigit()
print(x)
Output
True
	a = "u0030" #unicode for 0
	b = "u00B2" #unicode for ²
	print(a.isdigit())
	print(b.isdigit())
	Output
	True
	True

isalnum():-

The isalnum() method returns True if all the characters are alphanumeric, meaning alphabet letter (a-z) and numbers (0-9). Example of characters that are not alphanumeric: (space)!#%&? etc.

Syntax:- ***string*.isalnum()**

txt = "Company12" x = txt.isalnum() print(x) **Output** True	txt = "Company 12" x = txt.isalnum() print(x) **Output** False

startswith():-

The startswith() method returns True if the string starts with the specified value, otherwise False.

Syntax:- ***string*.startswith(*value, start, end*)**

Parameter	Description
value	Required. The value to check if the string starts with
start	Optional. An Integer specifying at which position to start the search
end	Optional. An Integer specifying at which position to end the search

```
txt = "Hello, welcome to my world."

x = txt.startswith("Hello")

print(x)
```

Output

True

```
txt = "Hello, welcome to my world."

x = txt.startswith("wel", 7, 20)

print(x)
```

Output

True

endswith():-

The endswith() method returns True if the string ends with the specified value, otherwise False.

Syntax:- ***string*.endswith(*value, start, end*)**

Parameter	Description
value	Required. The value to check if the string starts with
start	Optional. An Integer specifying at which position to start the search
end	Optional. An Integer specifying at which position to end the search

txt = "Hello, welcome to my world." x = txt.endswith(".") print(x) **Output** True	txt = "Hello, welcome to my world." x = txt.endswith("my world.") print(x) **Output** True	txt = "Hello, welcome to my world." x = txt.endswith("my world.", 5, 11) print(x) **Output** False

encode():-

The encode() method encodes the string, using the specified encoding. If no encoding is specified, UTF-8 will be used.

Syntax:- ***string*.encode(encoding=*encoding*, errors=*errors*)**

```
txt = "My name is Ståle"
x = txt.encode()
print(x)
```

Output

```
b'My name is Stxc3xa5le'
```

```
txt = "My name is Ståle"
print(txt.encode(encoding="ascii",errors="backslashreplace"))
print(txt.encode(encoding="ascii",errors="ignore"))
print(txt.encode(encoding="ascii",errors="namereplace"))
print(txt.encode(encoding="ascii",errors="replace"))
print(txt.encode(encoding="ascii",errors="xmlcharrefreplace"))
```

Output:-

```
b'My name is St\xe5le'
b'My name is Stle'
b'My name is St\N{LATIN SMALL LETTER A WITH RING ABOVE}le'
b'My name is St?le'
b'My name is Ståle'
```

decode():-

The python string **decode()** method decodes the string using the codec registered for its encoding. The encoded string can be decoded and the original string can be obtained with the help of this function. This function works based on the parameters specified which are encoding and the error.

Syntax:- **Str.decode(encoding='UTF-8',errors='strict')**

```
str="Hello! Welcome to akitsolution."

str_encoded= str.encode('euc_kr','strict')

print("The encoded string is: ", str_encoded)

str_decoded=str_encoded.decode('euc_kr', 'strict')

print("The decoded string is: ", str_decoded)
```

Output:-

```
The encoded string is:  b'Hello! Welcome to akitsolution.'

The decoded string is:  Hello! Welcome to akitsolution.
```

Index():-

The index() method finds the first occurrence of the specified value. The index() method raises an exception if the value is not found.

The index() method is almost the same as the find() method, the only difference is that the find() method returns -1 if the value is not found.

Syntax:- ***string*.index(*value, start, end*)**

value	Required. The value to search for
start	Optional. Where to start the search. Default is 0
end	Optional. Where to end the search. Default is to the end of the string

txt = "Hello, welcome to my world." x = txt.index("welcome") print(x) **output:-** 7	txt = "Hello, welcome to my world." x = txt.index("e") print(x) **output:-** 1	txt = "Hello, welcome to my world." x = txt.index("e", 5, 10) print(x) **output:-** 8

Note:- 1. Where in the text is the first occurrence of the letter "e"?:

1. Where in the text is the first occurrence of the letter "e" when you only search between position 5 and 10?:
2. If the value is not found, the find() method returns -1, but the index() method will raise an exception:

```
txt = "Hello, welcome to my world."

print(txt.find("q"))

print(txt.index("q"))
```

output:-

```
-1
Traceback (most recent call last):
  File "demo_ref_string_find_vs_index.py", line 4 in <module>
    print(txt.index("q"))
ValueError: substring not found
```

Sys Module

यह module command line arguments को process करने में अहम् भूमिका निभाता है। Command line arguments sys module के argv attribute में list के रूप में store किये जाते है।

```
>>> import sys

>>> print(sys.argv)
```

Sys module में error output redirection और program termination के लिए भी attributes होते है।

- stdin
- stdout
- stderr

stdin:- इसका उपयोग सीधे कमांड लाइन से इनपुट प्राप्त करने के लिए किया जा सकता है।

1. sys.argv :

Python में **script** को दिए गए **agruments(command prompt)**की एक सूची शामिल है। यदि आप **len(sys.argv)** लिखते हैं, तो आपको गिनने होंगे कि कितने तर्क हैं। **sys.argv** का उपयोग उन लोगों द्वारा किया जाता है जो **command-line arguments** के साथ काम करते हैं। **script** के नाम को **Sys.argv[0]** कहा जाता है।

Example-:

```
import sys
print(sys.argv[0])
print("Number of arguments present =",len(sys.argv))
```

2.sys.path-:

यह **function** वर्तमान **system** के **python path** को प्रदर्शित (display) करता है। यह एक **environmental variable** है जिसमें सभी **python module** के लिए **search path** शामिल है।

Example-:Import syssys.path

3) sys.maxsize-:

maxsize() function variable's का **greatest integer return** करता है।

Example-:

```
import sys
sys.maxsize
```

4) sys.version-:

version variable string return करता है जो **python interpreter** के **version number** को **contain** करता है।

Example-:

```
import sys
sys.version
```

5. sys.exit() :

इस **function** का प्रयोग हम **python console** या **command prompt** से बाहर निकलने के लिए करते है। यदि कोई **exception** सुरक्षित रूप से **thrown** की जाती है तो हम यह कह सकते है की **exit function** का उपयोग किसी application से बाहर निकलने के लिए करते है।

os Module

Python का os module operating system से interact करने के लिए बहुत सारे functions provide करता है।python OS module हमे user और operating system के बीच connection establish की सुविधा प्रदान करता है। यह module कई उपयोगी OS फ़ंक्शनप्रदान करता हैपायथन OS मॉड्यूल files and directories के साथ काम करता है।

OS मॉड्यूल के साथ काम करने के लिए, हमें OS मॉड्यूल import करना होगा।

import os

OS Module के सभी फंक्शन को देखने के लिए निचे दिए गए कोड को रन करे

import os

print (dir(os))

OS मॉड्यूल में कुछ function हैं जो नीचे दिए गए हैं:

1. os.name()

यह फ़ंक्शन उस ऑपरेटिंग सिस्टम मॉड्यूल का नाम provide करता है जिसे वह import करता है। वर्तमान में, यह 'posix', 'nt', 'os2', 'ce', 'java' और 'riscos' को पंजीकृत करता है।

Example

```
import os

print(os.name)
```

Output

```
nt
```

2. os.mkdir()

os.mkdir () फ़ंक्शन का उपयोग new directory को create करने के लिए किया जाता है।

```
import os

os.mkdir("E:\newdir")
```

3. os.getcwd()

यह फ़ाइल की current working directory (CWD) को return करता है।

```
import os

print(os.getcwd())
```

4. os.rmdir()

rmdir() फ़ंक्शन specified directory को absolute और relative path से remove कर देता है। सबसे पहले, हमें current working directory को बदलनाहोगा और फिर फ़ोल्डर को delete करना होगा।

```
import os

os.rmdir("d:\newdir")

os.chdir("..")

os.rmdir("newdir")
```