TYPES OF FUNCTIONS:

1. BUILT IN FUNCTION
2. USER-DEFINED FUNCTIONS

3. BUILT-IN FUNCTION:

print(),input(),sqrt(),abs(),

A. USER-DEFINED FUNCTION:

SYNTAX: def function_name(parameters/arguments):

```
'''docstring
'''
statement(s)
```

In [8]:

```
name=str(input("Enter the name:"))


def greet(name):
    """
    This function greets to
    the person passed in as
    a parameter
    """
    print("Hello, " + name + ". Good morning!")
    return()

greet(name)
print(greet.__doc__)
```

```
Enter the name:HAKFHA
Hello, HAKFHA. Good morning!

    This function greets to
    the person passed in as
    a parameter
```

```
def function_name():
  .........
  .........
  .........
.......
.......

function_name()
.......
.......
.......
```

In [10]:
```
for i in range(1,6):
    for j in range(1,6-i):
        print(" ",end="")
    k=i
    for j in range(1,i+1):
        print(k,end="")
        k=k+1
    k=k-2
    for j in range(1,i):
        print(k,end="")
        k=k-1
print()
```

```
    1   232  34543 4567654567898765
```

RETURN STATEMENT

In [13]:
```
def greet():
    print("Hello")

print(greet())
```

```
Hello
None
```

In [19]:
```
def greet():
    return "Hello"

greet()
```

Out[19]:
```
'xyz'
```

In [28]:
```
num=int(input("Enter the number: "))

def squ(num):
    ''' This functions returns
        the square of a given number
    '''
    return num**2

print(squ(num))
print(squ.__doc__)
```

```
Enter the number: 9
81
 This functions returns
       the square of a given number
```

In [1]:
```
#define a function to calculate Simple Interest:

def sim_int(p,r,t):
    si=p*r*t/100
    return si
```

```
s=sim_int(1000,10,3)
print("Simple Interest is=",s)
print("Simple Interest is=",sim_int(500,12,6))
pi=int(input("Enter the Principle: "))
ri=int(input("Enter the Rate: "))
td=int(input("Enter the Time: "))

sim=sim_int(pi,ri,td)
print("Simple Interest is=",sim)
```

```
Simple Interest is= 300.0
Simple Interest is= 360.0
Enter the Principle: 5000
Enter the Rate: 10
Enter the Time: 5
Simple Interest is= 2500.0
```

In [6]:

```
#define a function to check whether a number is divisble by another number

def div(x,y):
    if x%y==0:
        print(x,"is divisble by",y)
    else:
        print(x,"is not divisble by",y)

div(12,4)
div(8,3)
div(9999,3)
```

```
12 is divisble by 4
8 is not divisble by 3
9999 is divisble by 3
```

In [7]:

```
#define a function to check whether a number is divisble by another number

def div(x,y):
    if x%y==0:
        return True
    else:
        return False
x=13
y=4
t=div(x,y)
print(x,"is divisble",y,":",t)
```

```
13 is divisble 4 : False
```

In [8]:

```
#define a function to count the number of digit for a given integer number

def count_digit(n):
    count=0
    while n>0:
        n=n//10
        count=count+1
    return count
print("Number of digit: ",count_digit(12431241489612489))
print("Number of digit: ",count_digit(124))
```

```
Number of digit:  17
Number of digit:  3
```

In [9]:

```
#Write a program to calculate value of nCr
#step-1
#define a function to compute factorial

def fact(x):
    f=1
    for i in range(1,x+1):
        f=f*i
    print("Factorial of",x,"is=",f)

#step-2
#receive input from the user n and r

n=int(input("Enter the value of n in nCr: "))
r=int(input("Enter the value of r in nCr: "))

#step-3
#calculating nCr with the help of fact function

c=fact(n)/(fact (r) * fact(n-r))

#step-4
#printing the final output

print("value of",n,"C",r,"is",c)
```

```
Enter the value of n in nCr: 5
Enter the value of r in nCr: 2
Factorial of 5 is= 120
Factorial of 2 is= 2
Factorial of 3 is= 6
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9980/2248361591.py in <module>
     18 #calculating nCr with the help of fact function
     19
---> 20 c=fact(n)/(fact (r) * fact(n-r))
     21
     22 #step-4

TypeError: unsupported operand type(s) for *: 'NoneType' and 'NoneType'
```

In [10]:

```
#Write a program to calculate value of nCr
#step-1
#define a function to compute factorial

def fact(x):
    f=1
    for i in range(1,x+1):
        f=f*i
    return f

#step-2
#receive input from the user n and r
```

```
n=int(input("Enter the value of n in nCr: "))
r=int(input("Enter the value of r in nCr: "))

#step-3
#calculating nCr with the help of fact function

c=fact(n)/(fact (r) * fact(n-r))

#step-4
#printing the final output

print("value of",n,"C",r,"is",c)
```

```
Enter the value of n in nCr: 10
Enter the value of r in nCr: 5
value of 10 C 5 is 252.0
```

In [11]:

```
print(sim_int(1000,10))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9980/922703614.py in <module>
----> 1 print(sim_int(1000,10))

TypeError: sim_int() missing 1 required positional argument: 't'
```

In [12]:

```
print(sim_int(1000,10,5,78))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9980/2376813292.py in <module>
----> 1 print(sim_int(1000,10,5,78))

TypeError: sim_int() takes 3 positional arguments but 4 were given
```

In [19]:

```
#Function to understand the concept of pass by reference in mutable data type

def change(x): #x is formal parameter
    x[1]='123'
    print("Value inside the function",x)
    return

y=[10,20,30] #y is a list
print("Before changing values are:",y)
change(y) #y is an actual paramter
print("After changing the values are:",y)
```

```
Before changing values are: [10, 20, 30]
Value inside the function [10, '123', 30]
After changing the values are: [10, '123', 30]
```

In [1]:

```
#Function to understand the concept of pass by reference in immutable data types

def change(x): #x is formal parameter
    x='mohit'
    print("String inside the function: ",x)
    return
```

```
y='NIET' #y is a string
print("Before changing string is:",y)
change(y) #y is an actual paramter
print("After changing string is:",y)
```

```
Before changing string is: NIET
String inside the function:  mohit
After changing string is: NIET
```

In [3]:

```
#Function to understand the concept of pass by reference in immutable data types

def change(x): #x is formal parameter
    x[1]='mohit'
    print("String inside the function: ",x)
    return

y=(1,'regional',333,'lion') #y is tuple
print("Before changing string is:",y)
change(y) #y is an actual paramter
print("After changing string is:",y)
```

```
Before changing string is: (1, 'regional', 333, 'lion')
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_4932/48792912.py in <module>
      8 y=(1,'regional',333,'lion') #y is tuple
      9 print("Before changing string is:",y)
---> 10 change(y) #y is an actual paramter
     11 print("After changing string is:",y)

~\AppData\Local\Temp/ipykernel_4932/48792912.py in change(x)
      2
      3 def change(x): #x is formal parameter
----> 4     x[1]='mohit'
      5     print("String inside the function: ",x)
      6     return

TypeError: 'tuple' object does not support item assignment
```

In [4]:

```
#Function to understand the concept of pass by reference in immutable data types

def change(x): #x is formal parameter
    x=777
    print("Number inside the function: ",x)
    return

y=10 #y is a number
print("Before changing number is:",y)
change(y) #y is an actual paramter
print("After changing number is:",y)
```

```
Before changing number is: 10
Number inside the function:  777
After changing number is: 10
```

TYPES OF ACTUAL ARGUMENTS:

1. Required /Positional arguments
2. Keyword arguments
3. Default arguments
4. Variable-length arguments

REQUIRED POSITIONAL ARGUMENTS: Required arguments are the arguments passed to a function in correct positional order.The number of arguments in the function call should match exactly with the function definition.

In [5]:
```
# Demonstration of required positional arguments
# All the required positional arguments are necessary to be passed at the time of fun

def pos_arg(a,b,c):
    print(a,b,c)
    return

pos_arg(5,10,15) #All the parameters are passed
```

```
5 10 15
```

In [7]:
```
# Demonstration of required positional arguments
# All the required positional arguments are necessary to be passed at the time of fun

def pos_arg(a,b,c):
    print(a,b,c)
    return

pos_arg(5,10) #Less parameters are passed as per the mentioned in definition
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_4932/2920327708.py in <module>
      6     return
      7
----> 8 pos_arg(5,10) #Less parameters are passed as per the mentioned in definition

TypeError: pos_arg() missing 1 required positional argument: 'c'
```

In [8]:
```
# Demonstration of required positional arguments
# All the required positional arguments are necessary to be passed at the time of fun

def pos_arg(a,b,c):
    print(a,b,c)
    return

pos_arg(5,10,15,20) #More parameters are passed as per the mentioned in definition
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_4932/1748090421.py in <module>
      6     return
      7
----> 8 pos_arg(5,10,15,20) #More parameters are passed as per the mentioned in defin
ition

TypeError: pos_arg() takes 3 positional arguments but 4 were given
```

KEYWORD ARGUMENTS:

1.Keyword arguments are passed at the time of function call! 2.The caller identifies the arguments by the parameter name which are mentioned as keyword at time of function call. 3.This allows to place arguments out of order because the Python interpreter can use the keywords provided to match the values with parameters.

Note: A non keyword/positional argument can be followed by keyword argument, but keyword argument can not be followed by non keyword/positional argument.

In [20]:

```
# Demo of keyword arguments:
def keyword(a,b,c): # keyword arguments are not defined in function definition
        print("a=",a)
        print("b=",b)
        print("c=",c)
        return

keyword(1,2,3) #passed as a required positional arguments
keyword(a=10,b=20,c=30) #all parameters are passed as keywords arguments
keyword(100,c=300,b=400) #keyword argumnets following required positional arguments t
```

```
a= 1
b= 2
c= 3
a= 10
b= 20
c= 30
a= 100
b= 400
c= 300
```

In [4]:

```
# Demo of keyword arguments:
def keyword(a,b,c): # keyword arguments are not defined in function definition
        print("a=",a)
        print("b=",b)
        print("c=",c)
        return

keyword(100,b=300,a=400) #order must be maintained in the mixture of reqpos and keywo
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_10796/2184526207.py in <module>
      6         return
      7
----> 8 keyword(100,b=300,a=400) #order must be maintained in the mixture of reqpos a
nd keyword

TypeError: keyword() got multiple values for argument 'a'
```

In [11]:

```
# Demo of keyword arguments:
def keyword(a,b,c): # keyword arguments are not defined in function definition
        print("a=",a)
        print("b=",b)
        print("c=",c)
        return

keyword(a=100,b=200,300) #positional arguments can not follow keywords arguments
```

```
  File "C:\Users\callage\AppData\Local\Temp/ipykernel_9816/1405506763.py", line 8
    keyword(a=100,b=200,300) #positional arguments can not follow keywords arguments
                           ^
SyntaxError: positional argument follows keyword argument
```

In [14]:

```
#Usage of keyword arguments
def person(name,age):
    print(name)
    print(age-5)


person(age=28,name='aakash')
```

```
aakash
23
```

In [7]:

```
#Usage of keyword arguments

print(5,6,7,end="++",sep='**')
print()
print(5,6,7,sep="##",end='@@')
```

```
5**6**7++
5##6##7@@
```

In [ ]:

```
print(obj,sep,end,file,flush) #all the parameters have some default value associated
```

DEFAULT ARGUMENT:

1.When an argument is given a default value while defining a function, it will be called a default argument. 2.The default value will be considered if a value is not passed in the function call for that argument.

In [18]:

```
#Demo of default argument
```

```
def project(name,language='python'):
    print("project",name,"is developed using",language)
    return

project("Online exam system","java") # Required Positional Argument
project(language='C++',name="Reservation System") # Keyword Argument
project("Election Data Analysis") #Required Positional Argument
project(language="Java") #One positional arguments is missing
```

```
project Online exam system is developed using java
project Reservation System is developed using C++
project Election Data Analysis is developed using python
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_10796/1889896714.py in <module>
      8 project(language='C++',name="Reservation System") # Keyword Argument
      9 project("Election Data Analysis") #Required Positional Argument
---> 10 project(language="Java") #One positional arguments is missing

TypeError: project() missing 1 required positional argument: 'name'
```

VARIABLE LENGTH ARGUMENTS:

1.NON-KEYWORDS ARGUMENTS 2.KEYWORDS ARGUMENTS

Python provides a facility to define a function even when number of arguments are not fixed. A function having variable number of arguments can be defined by using variable-length-argument.

This argument is prefixed with a special character asterisk (* or **) in function definition !

VARIABLE LENGTH ARGUMENTS

You may need to process a function for more arguments than you specified while defining the function. These arguments are called variable-length arguments and are not named in the function definition, unlike required and default arguments.

Syntax: Syntax for a function with non-keyword variable arguments is this –

def function_name(*var_args_tuple): "function_docstring" function_body return (expression)

An asterisk (*) is placed before the variable name that holds the values of all non-keyword variable arguments. This tuple remains empty if no additional arguments are specified during the function call.

NON-KEYWORDS VAR. LENGTH ARGUMENTS:

This argument will hold all the unspecified non-keyword arguments in the function definition by packing them in a tuple. Non-keyword arguments will be prefixed with single asterisk in the function definition.

Ex: *var_args

In [1]:
```
#demo of non-keyword variable length argument:

def var_len(*var): #variable length arguments can recieve zero or more than zero para
    print("All the parameters are packed in tuple:",var) # Tuple will be created for
    print("Accessing tuple elements one by one: ")
    for v in var: #Loop can be used to access elements of tuple one by one
        print (v)
    return

var_len(10)
var_len(10,20,30,40,50)
```

```
All the parameters are packed in tuple: (10,)
Accessing tuple elements one by one:
10
All the parameters are packed in tuple: (10, 20, 30, 40, 50)
Accessing tuple elements one by one:
10
20
30
40
50
```

In [2]:
```
var_len()
```

```
All the parameters are packed in tuple: ()
Accessing tuple elements one by one:
```

In [5]:
```
#fixing the number of arguments to at least one with variable length argument

def var_len1(pos,*var): #here one positional argument is required so function can rec
    print("One position arguments: ",pos) #we have to manage positional argument sepe
    print("Remaining parameters are packed in tuple: ",var) #a tuple will created for
    print("Acessing tuple element one by one: ")
    for v in var: #Loop can be used to access elements of tuple one by one
        print(v)
    return

var_len1()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_4456/3094496204.py in <module>
      9     return
     10
---> 11 var_len1()

TypeError: var_len1() missing 1 required positional argument: 'pos'
```

In [4]:
```
var_len1(100,200,300,400,500)
```

```
One position arguments:  100
Remaining parameters are packed in tuple:  (200, 300, 400, 500)
Acessing tuple element one by one:
200
300
400
500
```

In [1]:
```
# A function to add all the numbers provided by user at the time of
# function calling.
# function must receive at least one parameter

def add_param(first,*num):
    add=first #first value which is passed through req positional arg. is added
    for i in num:
        add=add+i
    return add

x=add_param(10,20,40,50)
print("Addition of the numbers is: ",x)
print("Addition of the numbers is: ",add_param(5))
print("Addition of the numbers is: ",add_param())
```

```
Addition of the numbers is:  120
Addition of the numbers is:  5
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9816/3241582378.py in <module>
     12 print("Addition of the numbers is: ",x)
     13 print("Addition of the numbers is: ",add_param(5))
---> 14 print("Addition of the numbers is: ",add_param())

TypeError: add_param() missing 1 required positional argument: 'first'
```

In [2]:
```
add_param(10,20,30,40,first=100)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9816/3236942631.py in <module>
----> 1 add_param(10,20,30,40,first=100)

TypeError: add_param() got multiple values for argument 'first'
```

In [3]:
```
add_param(first=100,10,20,30,40)
```

```
  File "C:\Users\callage\AppData\Local\Temp/ipykernel_9816/14697281.py", line 1
    add_param(first=100,10,20,30,40)
                                   ^
SyntaxError: positional argument follows keyword argument
```

In [6]:
```
#Create a function to compute hcf of the given natural numbers:

def hcf_var(*num):
    if len(num)==0: #when there is no parameter is passed in function calling
        print("HCF is not possible")
```

```
        return
    else:
        low=num[0] #initialize min value of the tuple
        for i in num: #finding actual minimum in the tuple
            if low>i:
                low=i
        for i in range(low,0,-1):
            for v in num:
                if v%i!=0:
                    break
            else:
                return i
print("HCF= ",hcf_var(10))
print("HCF= ",hcf_var(12,20,))
print("HCF= ",hcf_var(20,12,8))
print("HCF= ",hcf_var())
```

```
HCF=  10
HCF=  4
HCF=  4
HCF is not possible
HCF=  None
```

In [8]:

```
#Create a function to compute hcf of the given natural numbers without else:

def hcf_var(*num):
    if len(num)==0: #when there is no parameter is passed in function calling
        print("HCF is not possible")
        return


    low=num[0] #initialize min value of the tuple
    for i in num: #finding actual minimum in the tuple
        if low>i:
            low=i
    for i in range(low,0,-1):
        for v in num:
            if v%i!=0:
                break
        else:
            return i


print("HCF= ",hcf_var(10))
print("HCF= ",hcf_var(12,20,))
print("HCF= ",hcf_var(20,12,8))
print("HCF= ",hcf_var())
```

```
HCF=  10
HCF=  4
HCF=  4
HCF is not possible
HCF=  None
```

KEYWORD VARIABLE LENGTH ARGUMENT:

1.This argument will hold all the unspecified keyword arguments in the function definition by packing them in a dictionary. Keyword-arguments will be prefixed with double asterisk in the function definition. Ex: **var_kwargs

In [9]:
```
#Demo of variable length keyword arguments

def var_key(**vark): #It accept all the zero or more keyword arguments
    #all the keyword arguments will be packed in a dictionary as key
    #and value pair under the name vark
    print("All the keyword arguments are: ",vark)
    return

var_key(a=10,b=20,c=30)
var_key() #It will create a blank dictionary
var_key(10)
```

```
All the keyword arguments are:  {'a': 10, 'b': 20, 'c': 30}
All the keyword arguments are:  {}
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9816/418519626.py in <module>
      9 var_key(a=10,b=20,c=30)
     10 var_key() #It will create a blank dictionary
---> 11 var_key(10)

TypeError: var_key() takes 0 positional arguments but 1 was given
```

In [15]:
```
#Accessing individual keyword arg in a function def having var. len. keywords

def var_key1(**vark1):
    for v in vark1: #for loop will iterate through only keys of the dictionary
        print(v,vark1[v]) #values can be accessed by indexing on keys
    return
var_key1(a=10,b=20,c=30)
```

```
a 10
b 20
c 30
```

STUDENT CHECK

1. Create a function to convert distance from feet to inches to cm.
2. Create a function to find the smallest number among three numbers.
3. Create a function to reverse a number and also check whether number is pallindrome or not.
4. Create a function to check whether a number is prime or not.

In [1]:
```
#Program to compute hcf/gcd of two numbers:

a=int(input("Enter the first number: "))
b=int(input("Enter the second number: "))

if a<b:
    h=a
```

```
else:
    h=b

while True:
    if a%h==0 and b%h==0:
        print(h,"is HCF of",a,b)
        break
    h=h-1
```

```
Enter the first number: 6
Enter the second number: 9
3 is HCF of 6 9
```

In [2]:

```
def hcf(a,b):
    if a<b:
        h=a
    else:
        h=b
    while True:
        if(a%h==0 and b%h==0):
            return h
        h=h-1

a=int(input("Enter the first number: "))
b=int(input("Enter the second number: "))
z=hcf(a,b)
print(z,"is HCF of",a,b)
```

```
Enter the first number: 8
Enter the second number: 12
4 is HCF of 8 12
```

In [5]:

```
def prime(n):
    for i in range(2,n):
        if n%i==0:
            print(n,"is not a prime number")
            break
    else:
        print(n,"is a prime number")
    return
x=int(input("Enter a number: "))
prime(x)
```

```
Enter a number: 9
9 is not a prime number
```

SCOPE OF VARIABLE IN PYTHON:

In programming languages, variables need to be defined before using them. These variables can only be accessed in the area where they are defined, this is called scope. You can think of this as a block where you can access variables.

All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.

The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python:

1.Local Variable 2.Global Variable

GLOBAL VS LOCAL VARIABLES

Variables that are defined inside a function body have a local scope, and those defined outside have a global scope.

This means that local variables can be accessed only inside the function in which they are declared, whereas global variables can be accessed throughout the program body by all functions. When you call a function, the variables declared inside it are brought into scope.

In [6]:
```
total = 0 # This is global variable.
# Function definition is here
def sum( arg1, arg2 ):
   # Add both the parameters and return them."
   total = arg1 + arg2; # Here total is local variable.
   print("Inside the function local total : ", total)
   return total
# Now you can call sum function
sum( 10, 20 )
print("Outside the function global total : ", total)
```

```
Inside the function local total :  30
Outside the function global total :  0
```

In [10]:
```
#Scope of variable
a=10 #it is a global variable as it is not defined inside the body of function
def scope():
    print("Value of global a is accessed inside the function scope:",a)
    a=20 # once we define a variable inside the function body it becomes local
    #so same name global can not be access from the function.

    print("Modified value of inside the function: ",a)
    return

scope()
print("VAlue of outside the function: ",a)
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_11504/2267888031.py in <module>
      9     return
     10
---> 11 scope()
     12 print("VAlue of outside the function: ",a)

~\AppData\Local\Temp/ipykernel_11504/2267888031.py in scope()
      2 a=10 #it is a global variable as it is not defined inside the body of functio
n
      3 def scope():
----> 4     print("Value of global a is accessed inside the function scope:",a)
      5     a=20 # once we define a variable inside the function body it becomes loca
l
      6     #so same name global can not be access from the function.

UnboundLocalError: local variable 'a' referenced before assignment
```

In [8]:

```
#Scope of variable
a=10 #it is a global variable as it is not defined inside the body of function
def scope():
    a=20 # once we define a variable inside the function body it becomes local
    #so same name global can not be access from the function.
    print("Value of global a is accessed inside the function scope:",a)

    print("Modified value of inside the function: ",a)
    return

scope()
print("VAlue of outside the function: ",a)
```

```
Value of global a is accessed inside the function scope: 20
Modified value of inside the function:  20
VAlue of outside the function:  10
```

In [13]:

```
#How to modify a global variable inside the body of function it becomes
#We  have to write global statement

a=50 #global variable

def scope():
    global a # it allow to access and modify the global variable inside the function
    a=20
    print("Value of a inside the function: ",a)
    print(id(a))
    return

scope()
print("Value of outside the function: ",a)
print(id(a))
```

```
Value of a inside the function:  20
1591599170384
Value of outside the function:  20
1591599170384
```

In [1]:

```
def fun():
```

```
    x=100
    print("Inside function x= ",x)
    return

fun()
print("Outside the function x= ",x) # A local variable can not be accessed  from outs
# as it is no longer exist after the function exit.
```

```
Inside function x=  100
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_12152/1923083856.py in <module>
      5
      6 fun()
----> 7 print("Outside the function x= ",x)

NameError: name 'x' is not defined
```

In [2]:

```
# A local variablle can not be accessed outside even if it is being
#returned by function as only it's value is returned

def fun():
    x=500
    print("Inside the function x= ",x)
    return x

fun()
print("Outside the function x=",x)
```

```
Inside the function x=  500
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_3272/1005135858.py in <module>
      8
      9 fun()
---> 10 print("Outside the function x=",x)

NameError: name 'x' is not defined
```

RECURSION IN PYTHON:

Recursion is the process of defining something in terms of itself.

In Python, it's also possible for a function to call itself! A function that calls itself is said to be recursive, and the technique of employing a recursive function is called recursion.

It may seem peculiar for a function to call itself, but many types of programming problems are best expressed recursively. When you bump up against such a problem, recursion is an indispensable tool for you to have in your toolkit.

What is Recursion?

The word recursion comes from the Latin word recurrere, meaning to run or hasten back, return, revert, or recur.

A recursive definition is one in which the defined term appears in the definition itself. Self-referential situations often crop up in real life, even if they aren't immediately recognizable as such. For example, suppose you wanted to describe the set of people that make up your ancestors. You could describe them this way:

Eg.: Your ancestors = (your parents) + (your parent's ancestors)

Recursion techniques in Python:

When you call a function in Python, the interpreter creates a new local namespace so that names defined within that function don't collide with identical names defined elsewhere. One function can call another, and even if they both define objects with the same name, it all works out fine because those objects exist in separate namespaces.

In [1]:
```
# Recursive function for countdown.

def countdown(n):
    print(n)
    if n==0:            # Recursion termination
        return
    else:
        countdown(n-1)  #Recursive call

n=int(input("Enter your countdown number: "))
countdown(n)
```

```
Enter your countdown number: 7
7
6
5
4
3
2
1
0
```

print(obj,sep,end,file,flush) #all the parameters have some default value associated with them 1.obj="ANything you wanted to show as your results" 2.sep="Space" 3.end="Null character or New Line" 4.file="File where you wanted to show your result" 5.flush="Flush by default is set to 0 i.e, it will store your result in memory but if you set flush=1 then your result will be flusded out from the memory".

In [ ]:
```
#Program to print the following Series:
#1, 2, 3, 6, 9, 18, 27, 54, ... upto n terms

n=int(input("Enter the number:"))
a=1
b=2
for i in range(1,n+1):
    if(i%2==1):
        print(a,end=',')
        a=a*3
```

```
    else:
        print(b,end =',')
        b=b*3
```

In [ ]:

```
#Program to print the following Series:
#1, 2, 3, 6, 9, 18, 27, 54, ... upto n terms

n=int(input("Enter the number:"))
a=0
b=1
c=2

print(b,c,end=',')
for i in range(3,n+1):
    if i%2==0:
        d=a+b+c
    else:
        d=b+c
    print(d,end=',')
    a=b
    b=c
    c=d
```

In [ ]:

```
#Program to print the following Series:
#2, 15, 41, 80, 132, 197, 275, 366, 470, 587

n=int(input("Enter the range of number(Limit):"))
i=1
value=2
while(i<=n):
    print(value,end=",")
    value+=i*13
    i+=1
```

In [4]:

```
a = 2
b = 13
n =int(input("Enter a no. - "))
for i in range(1, n+1, 1):
    print(a,end=',')
    a = a + b*i
```

```
Enter a no. - 9
2,15,41,80,132,197,275,366,470,
```

In [3]:

```
#Program to add n natural number using recursion

def rec_add(n):
    if n==1:  #base condition
        return 1
    else:
        return rec_add(n-1)+n #recursive condition
```

```
n=int(input("Enter the term: "))
print(rec_add(n))
```

```
Enter the term: 5
15
```

In [8]:

```
#Program to compute factorial of a given number using recursion

def fact(n):
    if n==1: #base case
        return 1
    else:
        return n*fact(n-1)  #Recursive Case(Self Refrential Case)

x=int(input("Enter a number: "))
print("Factorial: ",fact(x))
```

```
Enter a number: 5
Factorial:  120
```

In [6]:

```
# A recursive function to calculate exponential (power) of a given numbe (x**y)

def exp(x,y):
    if y==0:  #Base Case
        return 1
    else:
        return x*exp(x,y-1)  #Recursive Case

x=int(input("Enter the number: "))
y=int(input("Enter the power: "))
print ("Power of",x,"is",exp(x,y))
```

```
Enter the number: 2
Enter the power: 3
Power of 2 is 8
```

In [ ]:

```
# W.A.P. to calculate sum of digits of a given number using recursion.
# W.A.P. to reverse a given number using recursion.
# W.A.P. to display n terms of fibonacci series using recursion.
```

In [3]:

```
#Program to calculate sum of digits of a given number using recursion.

def rec_digit(n):
    if n==0:
        return 0
    else:
        return n%10 + rec_digit(n//10)

n=int(input("Enter the digit: "))
print("Sum of digit: ",rec_digit(n))
```

```
Enter the digit: 978978574758
Sum of digit:  84
```

In [3]:

```
# W.A.P. to reverse a given number using recursion.
```

```
def rev(n,r=0):
    if n==0:
        return r
    else:
        return rev(n//10,r*10+n%10)

n=int(input("Enter the number: "))
print("Reverse of the number is: ",rev(n))
```

```
Enter the number: 654
Reverse of the number is:  456
```

In [1]:

```
# W.A.P. to display n terms of fibonacci series using recursion.

def fib(n):
    if n==1:    #base case
        return 0
    elif n==2:  #base case
        return 1
    else:
        return fib(n-1)+fib(n-2)  #Recursive Case

def fib_series(n):
    for i in range(1,n+1):
        print(fib(i),end=',')

n=int(input("Enter the number of term: "))
(fib_series(n))
```

```
Enter the number of term: 20
0,1,1,2,3,5,8,13,21,34,55,89,144,233,377,610,987,1597,2584,4181,
```

In [9]:

```
#Program to generate fibonacci Series using recursion.

def fibonacci(n):
    if(n <= 2):   #base case
        return n
    else:
        return(fibonacci(n-1) + fibonacci(n-2))  #recursive case
n = int(input("Enter number of terms:"))
print("Fibonacci sequence:")
for i in range(1,n+1):
    print(fibonacci(i),end=',')
```

```
Enter number of terms:10
Fibonacci sequence:
1,2,3,5,8,13,21,34,55,89,
```

FUNCTION AS AN OBJECT

Python is an object-oriented programming language. Unlike procedure-oriented programming, where the main emphasis is on functions, object-oriented programming stresses on objects.

An object is simply a collection of data (variables) and methods (functions) that act on those data.

FUNCTION NAME IS ALSO A VARIABLE WHICH IS AN OBJECT OF FUNCTION CLASS. IT CAN BE RE-ASSIGNED TO ANOTHER VARIABLE(NAME).

In [4]:
```
def fun(text):
    return text + " students"

print(type(fun))  # fun is an object of builtin class function

fun_obj=fun  # reference of one object can be assigned to another variable
#fun_obj can also be used as the function name

print(fun("hello")) #calling through name fun
print(fun_obj("HIIIII"))  # calling same function through name fun_obj
```

```
<class 'function'>
hello students
HIIIII students
```

NESTED FUNCTION:

FUNCTION DEFINED INSIDE THE BODY OF ANOTHER FUNCTION IS CALLED "NESTED FUNCTION".

In [5]:
```
#Example of nested function

def outer (num):   #outer function
    def inner(n):  #inner function
        n=n+1
        return n
    return n # n can not be accessed as it is local to inner

print(outer(10))
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_4872/4165270793.py in <module>
      7     return n # n can not be accessed as it is local to inner
      8
----> 9 print(outer(10))

~\AppData\Local\Temp/ipykernel_4872/4165270793.py in outer(num)
      5         n=n+1
      6         return n
----> 7     return n # n can not be accessed as it is local to inner
      8
      9 print(outer(10))

NameError: name 'n' is not defined
```

In [6]:
```
#Example of nested function

def outer (num):   #outer function
    def inner(n):  #inner function
        n=n+1
```

```
        return n
    return

print(inner(10))  #inner function can not be accessed from outside as it is local to
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_4872/3671796946.py in <module>
      7     return
      8
----> 9 print(inner(10))  #inner function can not be accessed from outside as it is l
ocal to other function

NameError: name 'inner' is not defined
```

NON-LOCAL IN PYTHON:

Nonlocal variables are used in nested functions whose local scope is not defined. This means that the variable can be neither in the local nor the global scope.

Let's see an example of how a nonlocal variable is used in Python.

We use nonlocal keywords to create nonlocal variables.

Note : If we change the value of a nonlocal variable, the changes appear in the local variable.

In [10]:

```
#Inner can be accessed from the body of outer function

def outer (num):   #outer function
    def inner(n):  #inner function
        n=n+1     #local for inner function
        return n
    res=inner(num)
    return res

n=int(input("Enter the number you want to increment by one: "))
print(outer(n))
```

```
Enter the number you want to increment by one: 77
78
```

In [11]:

```
#Understanding non-local variable

x=10  #global variable
def outer():
    y=10 # Local for outer function
         # y is non-local variable for inner
    def inner():
        print("Value of non-local variable y inside inner function= ",y)
        # a non-local variable can be accessed from inner function
        return
    inner()
    return
outer()
```

```
Value of non-local variable y inside inner function=  10
```

In [14]:
```
# Same name local variable will be created once we define a non-local vaiable
# Inside the inner function

def outer():
    y=100 #Local for outer function
          # y is non-local variable to inner
    def inner():
        y=200 # Once we define inside a function it become local
        print("Value of y inside the inner function:",y)
        return
    inner()
    print("Value of y inside the outer function:",y)
    #non-local variable can not be modified directly in inner function
    return
outer()
```

```
Value of y inside the inner function: 200
Value of y inside the outer function: 100
```

In [15]:
```
# To modify a non-local variable in inner function
# A non-local statement will be used

def outer():
    y=444 #Local to outer function
          # y is a non-local variable for inner
    def inner():
        nonlocal y
        y=555 # once we define a variable inside a function it becomes local
        print("VAlue of y inside the inner function: ",y)
        return
    print("Befor calling value of y inside the inner functon: ",y)
    inner()# call of inner\
    print("AFter calling value of y inside outer function: ",y)
    # non-local ariable can not be modified directly in inner function
    return
outer()
```

```
Befor calling value of y inside the inner functon:  444
VAlue of y inside the inner function:  555
AFter calling value of y inside outer function:  555
```

Passing a function as an argument to another function:

In [17]:
```
def one(text):
    s= text + "one"
    return s
def two(text):
    s=text + "two"
    return s
def three (fun): #fun is an argument which has to be a function
    result=fun("Function as an argument inside: ") #function passed as argument is ca
    print(result)
    return
```

```
three(one)  #function one is passed as argument
three(two)  #function two is passed as argument
```

```
Function as an argument inside: one
Function as an argument inside: two
```

FUNCTION RETURNING OTHER FUNCTION

In [1]:

```
#in order to use facility of inner function we must call it in the body
#of the outer function in nested function scenario.

def greetings(): #outer function
    def say_hi(): #inner function
        return "Hiii How are you??"
    return
print(greetings()) #call of outer
```

```
None
```

In [3]:

```
def greetings(): #outer function
    def say_hi(): #inner function
        return "Hiii How are you??"
    msg=say_hi() #call of inner function inside the outer body
    return msg
print(greetings()) #call of outer
```

```
Hiii How are you??
```

In [4]:

```
#Accessing inner function from outside

def greetings(): #outer function
    def say_hi(): #inner function
        return "Hiii How are you??"
    msg=say_hi() #call of inner function inside the outer body
    return msg
print(greetings()) #call of outer
print(say_hi()) # it is not allowed as inner function is a local resource of
                # outer function so it can't be accessed from outside.
```

```
Hiii How are you??
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_3964/166790671.py in <module>
      7     return msg
      8 print(greetings()) #call of outer
----> 9 print(say_hi())

NameError: name 'say_hi' is not defined
```

In [5]:

```
#Accessing inner function from outside by returning it from outer function

def greetings():  #outer function
    def say_hi(): #inner function
        return "Hiiii!!!! How are you??"
```

```
    return say_hi # It can be returned as an object

hi=greetings() #assisgning returned reference by outer function in some name
print(say_hi()) #It is still a local resources
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_3964/2418066946.py in <module>
      7
      8 hi=greetings() #assisgning returned reference by outer function in some name
----> 9 print(say_hi()) #It is still a local resources

NameError: name 'say_hi' is not defined
```

ANONYMOUS FUNCTION OR LAMBDA FUNCTION:

In Python, an "anonymous function" is a function that is defined without a name.

While normal functions are defined using the def keyword in Python, anonymous functions are defined using the lambda keyword.

Hence, anonymous functions are also called lambda functions.

FEATURES OF LAMBDA FUNCTION:

1.Lambda functions all take a single argument.

2.In the definition of the lambdas, the arguments don’t have parentheses around them.

3.Multi-argument functions (functions that take more than one argument) are expressed in Python lambdas by listing arguments and separating them with a comma (,) but without surrounding them with parentheses:

USE OF LAMBDA FUNCTION:

We use lambda functions when we require a nameless function for a short period of time.

In Python, we generally use it as an argument to a higher-order function (a function that takes in other functions as arguments). Lambda functions are used along with built-in functions like filter(), map() etc. Lambda functions are frequently used with higher-order functions, which take one or more functions as arguments or return one or more functions.

Python exposes higher-order functions as built-in functions or in the standard library. Examples include map(), filter(), functools.reduce(), as well as key functions like sort(), sorted(), min(), and max().

In [2]:

```
def add(x,y):
    z=x+y
    return z

print(add(5,3))
```

```
8
```

In [6]:

```
#Addition using anonymous/lambda function

#syntax "lambda" arguments: expression

s=lambda x,y:x+y #lambda function for addition
#Lambda function definition returns a function at the time of compilation.
#returned function reference can be assigned in some variable name.
#this variable can be further used to call the defined lambda function.
```

In [7]:

```
#checking the type of defined lambda function

print(type(s))  #s has the reference of defined lambda function
```

```
<class 'function'>
```

In [8]:

```
#calling the lambda function through its referenced variable
#s has the reference of defined lambda function..
#so s can be called like a normal function by passing parameters mentioned in lambda
#expression written in lambda function will be evaluated and it return the value.

print(s(7,5))
```

```
12
```

In [9]:

```
#Square of a number using lambda

lambda x:(x*x)

#on compilation it has returned the created function
```

Out[9]:

```
<function __main__.<lambda>(x)>
```

In [10]:

```
#Square of a number using lambda

sq=lambda x:(x*x)
print(sq)
```

```
<function <lambda> at 0x000002551222C820>
```

In [11]:

```
#Square of a number using lambda

sq=lambda x:(x*x)
print(sq())
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9408/954540975.py in <module>
      2
      3 sq=lambda x:(x*x)
----> 4 print(sq())

TypeError: <lambda>() missing 1 required positional argument: 'x'
```

In [15]:

```
#Square of a number using lambda

sq=lambda x:(x*x)
print(sq(x=8))   #Parameter passed as keyword argument.
print(sq(5))     #Parameter passed as positional argument.
```

```
64
25
```

In [1]:

```
# A lambda function to calculate simple interest

si=lambda p,r,t: p*r*t/100
print(si(1000,5,5))
print(si(r=10,p=5000,t=5))
```

```
250.0
2500.0
```

In [3]:

```
#Lambda with default arguments:

sid=lambda p,r,t=5: p*r*t/100
print(sid(1000,5,10))
print(sid(5000,5))
```

```
500.0
1250.0
```

In [7]:

```
#Lambda function with user-defined function:

def multiplier(n): #user defined function
    return lambda a:a*n

doubler=multiplier(2)
print(doubler(10))
print(doubler(5))
tripler=multiplier(3)
print(tripler(10))
```

```
20
10
30
```

In [8]:

```
x=10
y=15
z=x+y
if x>y:
    print (z)
```

```
else:
    print (x-y)
```

```
-5
```

In [ ]:

```
#Using "If-else" statement with lambda function.
#it's not advisable to use lambda for conditional statement.
#A conditional statement becomes very confusing when it's written like expression.
```

In [9]:

```
greater=lambda x,y: x if x>y else y

print("Greater number is:",greater(12,8))
print("Greater number is:",greater(12,20))
```

```
Greater number is: 12
Greater number is: 20
```

In [10]:

```
even_odd=lambda x:print("Even") if x%2==0 else print("Odd")

even_odd(12)
even_odd(7)
even_odd(9)
```

```
Even
Odd
Odd
```

In [11]:

```
#Lambda with non-keyword variable length arguments

var=lambda *v: print(v)

var()
var(1)
var(1,2,3,4,5)
```

```
()
(1,)
(1, 2, 3, 4, 5)
```

In [14]:

```
#Lambda with keyword variable length arguments

kvar=lambda **k: print(k)

kvar()
kvar(a=1,b=2,c=3,d=4,e=5)
```

```
{}
{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}
```

What are map(), filter() and reduce() functions in Python?

As mentioned earlier, map(), filter() and reduce() are inbuilt functions of Python. These functions enable the functional programming aspect of Python. In functional programming, the arguments passed are the only factors that decide upon the output. These functions can take any other function as a parameter and can be supplied to other functions as parameters as well.

MAP() FUNCTION:

The map() function is a type of higher-order function. As mentioned earlier, this function takes another function as a parameter along with a sequence of iterables and returns an output after applying the function to each iterable present in the sequence.

Syntax: map(function,iterables)

Here, the function defines an expression that is in turn applied to the iterables. The map function can take user-defined functions as well as lambda functions as a parameter.

User-defined functions can be sent to the map() method. The user or programmer is the only one who can change the parameters of these functions. For example look at the code mention below:

In [4]:
```
def cs(a):
    return a*a
x=map(cs,(1,2,3,4,5,6,7,8,9))

print(x)

print(tuple(x))
```

```
<map object at 0x0000024BDD8BC700>
(1, 4, 9, 16, 25, 36, 49, 64, 81)
```

In the above code you can see: x is a map object, as you can see. The map function is displayed next, which takes "cs()" as a parameter and then applies "a * a" to all 'iterables'. As a result, all iterables' values are multiplied by themselves before being returned.

In [6]:
```
#map with lambda

output=map(lambda x:x+3,[1,2,3,4,5])
print(output)
print(list(output))
```

```
<map object at 0x0000024BDD8BF670>
[4, 5, 6, 7, 8]
```

In [9]:
```
def doubler(x):
    return x*2

li=[10,20,30,40,50]

new_list=map(doubler,li)
new_list=list(new_list)
print(new_list)
```

```
[20, 40, 60, 80, 100]
```

In [19]:
```
keshav=[11,22,33,44,55]
do_li=list(map((lambda x:x*2),keshav))
print(do_li)
```

```
[22, 44, 66, 88, 110]
```

In [11]:
```
# map function with more than one iterables
#simple interest with map function

pli=[1000,2000,3000,4000,5000]  #principal amount
rli=[5.5,5,7,1.5,8,4]   #rate of intereset
tli=[2,3,4,4,6] #time

si_li=list(map((lambda p,r,t:p*r*t/100),pli,rli,tli))

print(si_li)
```

```
[110.0, 300.0, 840.0, 240.0, 2400.0]
```

In [14]:
```
#Map function can also use built-in or library function as arguments

num=['10','20','30','40','50']
#a map function to convert each element of the above list into integer
#so each elements will be typecasted in integer by using int function

num_li=list(map(int,num))
print(num_li)
```

```
[10, 20, 30, 40, 50]
```

In [24]:
```
#Program to calculate length of a string using map function
#built-in function "len()" will be used in map

st_li=("amartya","kush","aslam","shivangi","diksha","kavya")
st_len=tuple(map(len,st_li))

print(st_len)
```

```
(7, 4, 5, 8, 6, 5)
```

FILTER FUNCTION:

1. The filter() function is used to create an output list consisting of values for which the function returns true.
2. The syntax of it is as follows: filter(function,iterables).
3. Function: function that test if elements of an iterable return true or false.
4. If None,then the function defaults to identity function which returns false if any elements are false.
5. Iterables: Iterable which is to be filtered,could be set,list,tuples or containers of any iterators.

In [26]:
```
#Filter function() with user-defined function:

def bekar_class(x):
    if x>=5:
        return x
y = filter(bekar_class, (1,2,3,4,5,6,7,8))
print(y)
print(list(y))

#As you can see, y is the filter object and the list
#is a list of values that are true for the condition (x>=5).
```

```
<filter object at 0x0000024BDD8BFFA0>
[5, 6, 7, 8]
```

In [27]:
```
y = filter(lambda x: (x>=5), (1,2,3,4,5,6,7,8))
print(list(y))
```

```
[5, 6, 7, 8]
```

In [2]:
```
#Filter function returns the elements of seq which produce True as a result
#for the condtion in function

seq=[True,False,True,True,False,False,None]
res=list(filter(lambda x:x,seq))
print(res)
```

```
[True, True, True]
```

In [4]:
```
#Filter out all the positive integers from a given list

cs=[-22,1,-34,45,67,-98,78,11,-65,65]

cs_obj=list(filter(lambda x:x>0,cs))
print("Positive Numbers are: ",cs_obj)
```

```
Positive Numbers are:  [1, 45, 67, 78, 11, 65]
```

REDUCE() FUNCTION:

1. Python's reduce() implement a mathematical technique commonly known as "Folding" or "Reduction". It is used to reduce a list of items to a single cumulative value.
2. The reduce() function in Python takes in a function and an iterable as an argument.
3. The function applied to the first two items in an iteable and generate a partial result.
4. The partial result,together with the third item in the iterable to generate another partial result and the process is repeated untill the iterable is exhausted and then return a single cumulative value.
5. Reduce() function is available in "functools module". So before using it we must import in our scope. "from functools import reduce"

6. Syntax: reduce(function,iterable,[intializer]). Intializer[]: optional argument that provides a seed value to the computation or reduction.

In [10]:
```
#Example reduce() function:

#Sum of all elements in a list
from functools import reduce

morning=[2,4,5,6,7,8,1,10]

cs=reduce(lambda x,y:x+y,morning)
print("Sum of elements in list is: ",cs)
```

```
Sum of elements in list is:  43
```

In [13]:
```
from functools import reduce

l=[10,20,30,40]

s=reduce(lambda x,y:x+y,l,1000) #third argument will be seed for the cumulative resul

print(s)
```

```
1100
```

In [14]:
```
#Sum of only even numbers using reduce in a given list

from functools import reduce

bolo=[1,2,3,4,5,6,7,8,9,10,11,12]

result=reduce(lambda x,y: x+y if y%2==0 else x,bolo,0)

print("Sum of even numbers in list: ",result)
```

```
Sum of even numbers in list:  42
```

In [15]:
```
#To find the max value in a list using reduce

li2=[22,33,4,5,77,99,1,43]

mx=reduce(lambda x,y: x if x>y else y,li2)

print(mx)
```

```
99
```

USING MAP(),FILTER() AND REDUCE() TOGETHER:

When you use the map() function within filter() function, the iterables are first operated upon by the map function and then the condition of filter() is applied to them.

Using filter() within map():

In [17]:
```
c= map(lambda x: x+x,filter(lambda x: (x>=3),(1,2,3,4,5,6,7,8)))
print(list(c))
```

```
[6, 8, 10, 12, 14, 16]
```

Using map() within filter():

In [18]:
```
c=filter(lambda x:(x>=3),map(lambda x:x+x,(1,2,3,4,5,6,7,8)))
print(list(c))
```

```
[4, 6, 8, 10, 12, 14, 16]
```

Using map() and filter() within reduce():

The output of the internal functions is reduced to the condition supplied to the reduce() function.

In [21]:
```
from functools import reduce

d=reduce(lambda x,y: x+y,map(lambda x:x+x,filter(lambda x:(x>=3),(1,2,3,4,5,6,7,8))))
print(d)
```

```
66
```

In [4]:
```
from functools import reduce
cs=[1,2,3,4,5,6,7,8]
sum_lk=reduce(lambda x,y: x+y,list(filter(lambda x:x%2==0,list(map(int,cs)))))
print (sum_lk)
```

```
20
```

In [6]:
```
#filter all the prime numbers within a given range and compute its summation.

from functools import reduce

def prime(n):
    if n<=1:
        return False
    for i in range(2,n):
        if n%i==0:
            return False
    else:
        return True

#input range

x=int(input("Enter start point of the range: "))
y=int(input("Enter end point of the range: "))

cs=list(range(x,y+1))  #creation of list for a given range
```

```
#filterout all the prime numbers in a given range
prime_cs=list(filter(prime,cs))
print("All the prime within the given range is:",prime_cs)

#reduce all the prime numbers into sum

sum_cs=reduce(lambda x,y:x+y,prime_cs)
print("Sum of prime numbers is:",sum_cs)
```

```
Enter start point of the range: 10
Enter end point of the range: 50
All the prime within the given range is: [11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
Sum of prime numbers is: 311
```

In [11]:

```
#Find all the cubes of each element in a list which are divisible by 3

jordar=[10,20,30,40,50,60,70,80,90,100]

#first map all the elements with its cube
cube_jordar=list(map(lambda x:x*x*x,jordar))
print("Cubes are: ",cube_jordar)

#filterout cubes which are divisible by 3

div_jordar=list(filter(lambda x: x%3==0,cube_jordar))

print("Divisible by 3: ",div_jordar)

#combined map within filter

div_jordar=list(filter(lambda x: x%3==0,list(map (lambda x: x*x*x,jordar))))
print(div_jordar)
```

```
Cubes are:  [1000, 8000, 27000, 64000, 125000, 216000, 343000, 512000, 729000, 100000
0]
Divisible by 3:  [27000, 216000, 729000]
[27000, 216000, 729000]
```

MATH MODULE:

In [1]:

```
import math
dir(math)
```

Out[1]:
```
['__doc__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'acos',
 'acosh',
 'asin',
 'asinh',
 'atan',
 'atan2',
 'atanh',
 'ceil',
 'comb',
 'copysign',
 'cos',
 'cosh',
 'degrees',
 'dist',
 'e',
 'erf',
 'erfc',
 'exp',
 'expm1',
 'fabs',
 'factorial',
 'floor',
 'fmod',
 'frexp',
 'fsum',
 'gamma',
 'gcd',
 'hypot',
 'inf',
 'isclose',
 'isfinite',
 'isinf',
 'isnan',
 'isqrt',
 'lcm',
 'ldexp',
 'lgamma',
 'log',
 'log10',
 'log1p',
 'log2',
 'modf',
 'nan',
 'nextafter',
 'perm',
 'pi',
 'pow',
 'prod',
 'radians',
 'remainder',
 'sin',
 'sinh',
 'sqrt',
 'tan',
 'tanh',
```

```
'tau',
'trunc',
'ulp']
```

In [2]:

```
help(math)
```

```
Help on built-in module math:

NAME
    math

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

        The result is between 0 and pi.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

        The result is between -pi/2 and pi/2.

    asinh(x, /)
        Return the inverse hyperbolic sine of x.

    atan(x, /)
        Return the arc tangent (measured in radians) of x.

        The result is between -pi/2 and pi/2.

    atan2(y, x, /)
        Return the arc tangent (measured in radians) of y/x.

        Unlike atan(y/x), the signs of both x and y are considered.

    atanh(x, /)
        Return the inverse hyperbolic tangent of x.

    ceil(x, /)
        Return the ceiling of x as an Integral.

        This is the smallest integer >= x.

    comb(n, k, /)
        Number of ways to choose k items from n items without repetition and without
order.

        Evaluates to n! / (k! * (n - k)!) when k <= n and evaluates
        to zero when k > n.

        Also called the binomial coefficient because it is equivalent
        to the coefficient of k-th term in polynomial expansion of the
        expression (1 + x)**n.

        Raises TypeError if either of the arguments are not integers.
        Raises ValueError if either of the arguments are negative.

    copysign(x, y, /)
        Return a float with the magnitude (absolute value) of x but the sign of y.
```

```
        On platforms that support signed zeros, copysign(1.0, -0.0)
        returns -1.0.

    cos(x, /)
        Return the cosine of x (measured in radians).

    cosh(x, /)
        Return the hyperbolic cosine of x.

    degrees(x, /)
        Convert angle x from radians to degrees.

    dist(p, q, /)
        Return the Euclidean distance between two points p and q.

        The points should be specified as sequences (or iterables) of
        coordinates.  Both inputs must have the same dimension.

        Roughly equivalent to:
            sqrt(sum((px - qx) ** 2.0 for px, qx in zip(p, q)))

    erf(x, /)
        Error function at x.

    erfc(x, /)
        Complementary error function at x.

    exp(x, /)
        Return e raised to the power of x.

    expm1(x, /)
        Return exp(x)-1.

        This function avoids the loss of precision involved in the direct evaluation
of exp(x)-1 for small x.

    fabs(x, /)
        Return the absolute value of the float x.

    factorial(x, /)
        Find x!.

        Raise a ValueError if x is negative or non-integral.

    floor(x, /)
        Return the floor of x as an Integral.

        This is the largest integer <= x.

    fmod(x, y, /)
        Return fmod(x, y), according to platform C.

        x % y may differ.

    frexp(x, /)
        Return the mantissa and exponent of x, as pair (m, e).

        m is a float and e is an int, such that x = m * 2.**e.
        If x is 0, m and e are both 0.  Else 0.5 <= abs(m) < 1.0.
```

```
fsum(seq, /)
    Return an accurate floating point sum of values in the iterable seq.

    Assumes IEEE-754 floating point arithmetic.

gamma(x, /)
    Gamma function at x.

gcd(*integers)
    Greatest Common Divisor.

hypot(...)
    hypot(*coordinates) -> value

    Multidimensional Euclidean distance from the origin to a point.

    Roughly equivalent to:
        sqrt(sum(x**2 for x in coordinates))

    For a two dimensional point (x, y), gives the hypotenuse
    using the Pythagorean theorem:  sqrt(x*x + y*y).

    For example, the hypotenuse of a 3/4/5 right triangle is:

        >>> hypot(3.0, 4.0)
        5.0

isclose(a, b, *, rel_tol=1e-09, abs_tol=0.0)
    Determine whether two floating point numbers are close in value.

      rel_tol
        maximum difference for being considered "close", relative to the
        magnitude of the input values
      abs_tol
        maximum difference for being considered "close", regardless of the
        magnitude of the input values

    Return True if a is close in value to b, and False otherwise.

    For the values to be considered close, the difference between them
    must be smaller than at least one of the tolerances.

    -inf, inf and NaN behave similarly to the IEEE 754 Standard.  That
    is, NaN is not close to anything, even itself.  inf and -inf are
    only close to themselves.

isfinite(x, /)
    Return True if x is neither an infinity nor a NaN, and False otherwise.

isinf(x, /)
    Return True if x is a positive or negative infinity, and False otherwise.

isnan(x, /)
    Return True if x is a NaN (not a number), and False otherwise.

isqrt(n, /)
    Return the integer part of the square root of the input.

lcm(*integers)
```

```
        Least Common Multiple.

    ldexp(x, i, /)
        Return x * (2**i).

        This is essentially the inverse of frexp().

    lgamma(x, /)
        Natural logarithm of absolute value of Gamma function at x.

    log(...)
        log(x, [base=math.e])
        Return the logarithm of x to the given base.

        If the base not specified, returns the natural logarithm (base e) of x.

    log10(x, /)
        Return the base 10 logarithm of x.

    log1p(x, /)
        Return the natural logarithm of 1+x (base e).

        The result is computed in a way which is accurate for x near zero.

    log2(x, /)
        Return the base 2 logarithm of x.

    modf(x, /)
        Return the fractional and integer parts of x.

        Both results carry the sign of x and are floats.

    nextafter(x, y, /)
        Return the next floating-point value after x towards y.

    perm(n, k=None, /)
        Number of ways to choose k items from n items without repetition and with ord
er.

        Evaluates to n! / (n - k)! when k <= n and evaluates
        to zero when k > n.

        If k is not specified or is None, then k defaults to n
        and the function returns n!.

        Raises TypeError if either of the arguments are not integers.
        Raises ValueError if either of the arguments are negative.

    pow(x, y, /)
        Return x**y (x to the power of y).

    prod(iterable, /, *, start=1)
        Calculate the product of all the elements in the input iterable.

        The default start value for the product is 1.

        When the iterable is empty, return the start value.  This function is
        intended specifically for use with numeric values and may reject
        non-numeric types.
```

```
    radians(x, /)
        Convert angle x from degrees to radians.

    remainder(x, y, /)
        Difference between x and the closest integer multiple of y.

        Return x - n*y where n*y is the closest integer multiple of y.
        In the case where x is exactly halfway between two multiples of
        y, the nearest even value of n is used. The result is always exact.

    sin(x, /)
        Return the sine of x (measured in radians).

    sinh(x, /)
        Return the hyperbolic sine of x.

    sqrt(x, /)
        Return the square root of x.

    tan(x, /)
        Return the tangent of x (measured in radians).

    tanh(x, /)
        Return the hyperbolic tangent of x.

    trunc(x, /)
        Truncates the Real x to the nearest Integral toward 0.

        Uses the __trunc__ magic method.

    ulp(x, /)
        Return the value of the least significant bit of the float x.

DATA
    e = 2.718281828459045
    inf = inf
    nan = nan
    pi = 3.141592653589793
    tau = 6.283185307179586

FILE
    (built-in)
```

In [3]:
```
print(math.ceil(2.5))
```

```
3
```

In [4]:
```
help(math.ceil)
```

```
Help on built-in function ceil in module math:

ceil(x, /)
    Return the ceiling of x as an Integral.

    This is the smallest integer >= x.
```

In [5]:
```
print(math.factorial(2.5))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_8228/385641981.py in <module>
----> 1 print(math.factorial(2.5))

TypeError: 'float' object cannot be interpreted as an integer
```

In [6]:
```
help(math.factorial)
```

```
Help on built-in function factorial in module math:

factorial(x, /)
    Find x!.

    Raise a ValueError if x is negative or non-integral.
```

In [7]:
```
print(math.factorial(5))
```

```
120
```

In [8]:
```
print(math.floor(2.5))
```

```
2
```

In [9]:
```
print(math.floor(2.5))
print(math.floor(-2.5))
```

```
2
-3
```

In [10]:
```
r=5
print(math.pi)
area=math.pi*r*r
print(area)
```

```
3.141592653589793
78.53981633974483
```

In [11]:
```
help(math.pow)
```

```
Help on built-in function pow in module math:

pow(x, y, /)
    Return x**y (x to the power of y).
```

In [12]:
```
print(math.pow(2,5))
```

```
32.0
```

STASTICS MODULE

In [13]:

```
import statistics as st
dir(st)
```

Out[13]:
```
['Counter',
 'Decimal',
 'Fraction',
 'LinearRegression',
 'NormalDist',
 'StatisticsError',
 '__all__',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 '_coerce',
 '_convert',
 '_exact_ratio',
 '_fail_neg',
 '_find_lteq',
 '_find_rteq',
 '_isfinite',
 '_normal_dist_inv_cdf',
 '_ss',
 '_sum',
 'bisect_left',
 'bisect_right',
 'correlation',
 'covariance',
 'erf',
 'exp',
 'fabs',
 'fmean',
 'fsum',
 'geometric_mean',
 'groupby',
 'harmonic_mean',
 'hypot',
 'itemgetter',
 'linear_regression',
 'log',
 'math',
 'mean',
 'median',
 'median_grouped',
 'median_high',
 'median_low',
 'mode',
 'multimode',
 'namedtuple',
 'numbers',
 'pstdev',
 'pvariance',
 'quantiles',
 'random',
 'repeat',
 'sqrt',
 'stdev',
 'tau',
 'variance']
```

In [14]:
```
help(st)
```

```
Help on module statistics:

NAME
    statistics - Basic statistics module.

MODULE REFERENCE
    https://docs.python.org/3.10/library/statistics.html

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
    This module provides functions for calculating statistics of data, including
    averages, variance, and standard deviation.

    Calculating averages
    --------------------

    ==================  ==================================================
    Function            Description
    ==================  ==================================================
    mean                Arithmetic mean (average) of data.
    fmean               Fast, floating point arithmetic mean.
    geometric_mean      Geometric mean of data.
    harmonic_mean       Harmonic mean of data.
    median              Median (middle value) of data.
    median_low          Low median of data.
    median_high         High median of data.
    median_grouped      Median, or 50th percentile, of grouped data.
    mode                Mode (most common value) of data.
    multimode           List of modes (most common values of data).
    quantiles           Divide data into intervals with equal probability.
    ==================  ==================================================

    Calculate the arithmetic mean ("the average") of data:

    >>> mean([-1.0, 2.5, 3.25, 5.75])
    2.625


    Calculate the standard median of discrete data:

    >>> median([2, 3, 4, 5])
    3.5


    Calculate the median, or 50th percentile, of data grouped into class intervals
    centred on the data values provided. E.g. if your data points are rounded to
    the nearest whole number:

    >>> median_grouped([2, 2, 3, 3, 3, 4])  #doctest: +ELLIPSIS
    2.8333333333...

    This should be interpreted in this way: you have two data points in the class
    interval 1.5-2.5, three data points in the class interval 2.5-3.5, and one in
    the class interval 3.5-4.5. The median of these data points is 2.8333...
```

```
    Calculating variability or spread
    ---------------------------------

    ==================  ============================================
    Function            Description
    ==================  ============================================
    pvariance           Population variance of data.
    variance            Sample variance of data.
    pstdev              Population standard deviation of data.
    stdev               Sample standard deviation of data.
    ==================  ============================================

    Calculate the standard deviation of sample data:

    >>> stdev([2.5, 3.25, 5.5, 11.25, 11.75])  #doctest: +ELLIPSIS
    4.38961843444...

    If you have previously calculated the mean, you can pass it as the optional
    second argument to the four "spread" functions to avoid recalculating it:

    >>> data = [1, 2, 2, 4, 4, 4, 5, 6]
    >>> mu = mean(data)
    >>> pvariance(data, mu)
    2.5


    Statistics for relations between two inputs
    -------------------------------------------

    ==================  ====================================================
    Function            Description
    ==================  ====================================================
    covariance          Sample covariance for two variables.
    correlation         Pearson's correlation coefficient for two variables.
    linear_regression   Intercept and slope for simple linear regression.
    ==================  ====================================================

    Calculate covariance, Pearson's correlation, and simple linear regression
    for two inputs:

    >>> x = [1, 2, 3, 4, 5, 6, 7, 8, 9]
    >>> y = [1, 2, 3, 1, 2, 3, 1, 2, 3]
    >>> covariance(x, y)
    0.75
    >>> correlation(x, y)  #doctest: +ELLIPSIS
    0.31622776601...
    >>> linear_regression(x, y)  #doctest:
    LinearRegression(slope=0.1, intercept=1.5)


    Exceptions
    ----------

    A single exception is defined: StatisticsError is a subclass of ValueError.

CLASSES
    builtins.ValueError(builtins.Exception)
        StatisticsError
    builtins.object
```

```
NormalDist

class NormalDist(builtins.object)
 |  NormalDist(mu=0.0, sigma=1.0)
 |
 |  Normal distribution of a random variable
 |
 |  Methods defined here:
 |
 |  __add__(x1, x2)
 |      Add a constant or another NormalDist instance.
 |
 |      If *other* is a constant, translate mu by the constant,
 |      leaving sigma unchanged.
 |
 |      If *other* is a NormalDist, add both the means and the variances.
 |      Mathematically, this works only if the two distributions are
 |      independent or if they are jointly normally distributed.
 |
 |  __eq__(x1, x2)
 |      Two NormalDist objects are equal if their mu and sigma are both equal.
 |
 |  __hash__(self)
 |      NormalDist objects hash equal if their mu and sigma are both equal.
 |
 |  __init__(self, mu=0.0, sigma=1.0)
 |      NormalDist where mu is the mean and sigma is the standard deviation.
 |
 |  __mul__(x1, x2)
 |      Multiply both mu and sigma by a constant.
 |
 |      Used for rescaling, perhaps to change measurement units.
 |      Sigma is scaled with the absolute value of the constant.
 |
 |  __neg__(x1)
 |      Negates mu while keeping sigma the same.
 |
 |  __pos__(x1)
 |      Return a copy of the instance.
 |
 |  __radd__ = __add__(x1, x2)
 |
 |  __repr__(self)
 |      Return repr(self).
 |
 |  __rmul__ = __mul__(x1, x2)
 |
 |  __rsub__(x1, x2)
 |      Subtract a NormalDist from a constant or another NormalDist.
 |
 |  __sub__(x1, x2)
 |      Subtract a constant or another NormalDist instance.
 |
 |      If *other* is a constant, translate by the constant mu,
 |      leaving sigma unchanged.
 |
 |      If *other* is a NormalDist, subtract the means and add the variances.
 |      Mathematically, this works only if the two distributions are
 |      independent or if they are jointly normally distributed.
 |
```

```
|  __truediv__(x1, x2)
|      Divide both mu and sigma by a constant.
|
|      Used for rescaling, perhaps to change measurement units.
|      Sigma is scaled with the absolute value of the constant.
|
|  cdf(self, x)
|      Cumulative distribution function.  P(X <= x)
|
|  inv_cdf(self, p)
|      Inverse cumulative distribution function.  x : P(X <= x) = p
|
|      Finds the value of the random variable such that the probability of
|      the variable being less than or equal to that value equals the given
|      probability.
|
|      This function is also called the percent point function or quantile
|      function.
|
|  overlap(self, other)
|      Compute the overlapping coefficient (OVL) between two normal distribution
s.
|
|      Measures the agreement between two normal probability distributions.
|      Returns a value between 0.0 and 1.0 giving the overlapping area in
|      the two underlying probability density functions.
|
|          >>> N1 = NormalDist(2.4, 1.6)
|          >>> N2 = NormalDist(3.2, 2.0)
|          >>> N1.overlap(N2)
|          0.8035050657330205
|
|  pdf(self, x)
|      Probability density function.  P(x <= X < x+dx) / dx
|
|  quantiles(self, n=4)
|      Divide into *n* continuous intervals with equal probability.
|
|      Returns a list of (n - 1) cut points separating the intervals.
|
|      Set *n* to 4 for quartiles (the default).  Set *n* to 10 for deciles.
|      Set *n* to 100 for percentiles which gives the 99 cuts points that
|      separate the normal distribution in to 100 equal sized groups.
|
|  samples(self, n, *, seed=None)
|      Generate *n* samples for a given mean and standard deviation.
|
|  zscore(self, x)
|      Compute the Standard Score.  (x - mean) / stdev
|
|      Describes *x* in terms of the number of standard deviations
|      above or below the mean of the normal distribution.
|
|  ----------------------------------------------------------------------
|  Class methods defined here:
|
|  from_samples(data) from builtins.type
|      Make a normal distribution instance from sample data.
|
|  ----------------------------------------------------------------------
```

```
 |  Readonly properties defined here:
 |
 |  mean
 |      Arithmetic mean of the normal distribution.
 |
 |  median
 |      Return the median of the normal distribution
 |
 |  mode
 |      Return the mode of the normal distribution
 |
 |      The mode is the value x where which the probability density
 |      function (pdf) takes its maximum value.
 |
 |  stdev
 |      Standard deviation of the normal distribution.
 |
 |  variance
 |      Square of the standard deviation.

class StatisticsError(builtins.ValueError)
 |  Method resolution order:
 |      StatisticsError
 |      builtins.ValueError
 |      builtins.Exception
 |      builtins.BaseException
 |      builtins.object
 |
 |  Data descriptors defined here:
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from builtins.ValueError:
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Static methods inherited from builtins.ValueError:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from builtins.BaseException:
 |
 |  __delattr__(self, name, /)
 |      Implement delattr(self, name).
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __reduce__(...)
 |      Helper for pickle.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
```

```
 |  __setattr__(self, name, value, /)
 |      Implement setattr(self, name, value).
 |
 |  __setstate__(...)
 |
 |  __str__(self, /)
 |      Return str(self).
 |
 |  with_traceback(...)
 |      Exception.with_traceback(tb) --
 |      set self.__traceback__ to tb and return self.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from builtins.BaseException:
 |
 |  __cause__
 |      exception cause
 |
 |  __context__
 |      exception context
 |
 |  __dict__
 |
 |  __suppress_context__
 |
 |  __traceback__
 |
 |  args

FUNCTIONS
    correlation(x, y, /)
        Pearson's correlation coefficient

        Return the Pearson's correlation coefficient for two inputs. Pearson's
        correlation coefficient *r* takes values between -1 and +1. It measures the
        strength and direction of the linear relationship, where +1 means very
        strong, positive linear relationship, -1 very strong, negative linear
        relationship, and 0 no linear relationship.

        >>> x = [1, 2, 3, 4, 5, 6, 7, 8, 9]
        >>> y = [9, 8, 7, 6, 5, 4, 3, 2, 1]
        >>> correlation(x, x)
        1.0
        >>> correlation(x, y)
        -1.0

    covariance(x, y, /)
        Covariance

        Return the sample covariance of two inputs *x* and *y*. Covariance
        is a measure of the joint variability of two inputs.

        >>> x = [1, 2, 3, 4, 5, 6, 7, 8, 9]
        >>> y = [1, 2, 3, 1, 2, 3, 1, 2, 3]
        >>> covariance(x, y)
        0.75
        >>> z = [9, 8, 7, 6, 5, 4, 3, 2, 1]
        >>> covariance(x, z)
        -7.5
        >>> covariance(z, x)
```

```
    -7.5

fmean(data)
    Convert data to floats and compute the arithmetic mean.

    This runs faster than the mean() function and it always returns a float.
    If the input dataset is empty, it raises a StatisticsError.

    >>> fmean([3.5, 4.0, 5.25])
    4.25

geometric_mean(data)
    Convert data to floats and compute the geometric mean.

    Raises a StatisticsError if the input dataset is empty,
    if it contains a zero, or if it contains a negative value.

    No special efforts are made to achieve exact results.
    (However, this may change in the future.)

    >>> round(geometric_mean([54, 24, 36]), 9)
    36.0

harmonic_mean(data, weights=None)
    Return the harmonic mean of data.

    The harmonic mean is the reciprocal of the arithmetic mean of the
    reciprocals of the data.  It can be used for averaging ratios or
    rates, for example speeds.

    Suppose a car travels 40 km/hr for 5 km and then speeds-up to
    60 km/hr for another 5 km. What is the average speed?

        >>> harmonic_mean([40, 60])
        48.0

    Suppose a car travels 40 km/hr for 5 km, and when traffic clears,
    speeds-up to 60 km/hr for the remaining 30 km of the journey. What
    is the average speed?

        >>> harmonic_mean([40, 60], weights=[5, 30])
        56.0

    If ``data`` is empty, or any element is less than zero,
    ``harmonic_mean`` will raise ``StatisticsError``.

linear_regression(x, y, /)
    Slope and intercept for simple linear regression.

    Return the slope and intercept of simple linear regression
    parameters estimated using ordinary least squares. Simple linear
    regression describes relationship between an independent variable
    *x* and a dependent variable *y* in terms of linear function:

        y = slope * x + intercept + noise

    where *slope* and *intercept* are the regression parameters that are
    estimated, and noise represents the variability of the data that was
    not explained by the linear regression (it is equal to the
    difference between predicted and actual values of the dependent
```

```
    variable).

    The parameters are returned as a named tuple.

    >>> x = [1, 2, 3, 4, 5]
    >>> noise = NormalDist().samples(5, seed=42)
    >>> y = [3 * x[i] + 2 + noise[i] for i in range(5)]
    >>> linear_regression(x, y)  #doctest: +ELLIPSIS
    LinearRegression(slope=3.09078914170..., intercept=1.75684970486...)

mean(data)
    Return the sample arithmetic mean of data.

    >>> mean([1, 2, 3, 4, 4])
    2.8

    >>> from fractions import Fraction as F
    >>> mean([F(3, 7), F(1, 21), F(5, 3), F(1, 3)])
    Fraction(13, 21)

    >>> from decimal import Decimal as D
    >>> mean([D("0.5"), D("0.75"), D("0.625"), D("0.375")])
    Decimal('0.5625')

    If ``data`` is empty, StatisticsError will be raised.

median(data)
    Return the median (middle value) of numeric data.

    When the number of data points is odd, return the middle data point.
    When the number of data points is even, the median is interpolated by
    taking the average of the two middle values:

    >>> median([1, 3, 5])
    3
    >>> median([1, 3, 5, 7])
    4.0

median_grouped(data, interval=1)
    Return the 50th percentile (median) of grouped continuous data.

    >>> median_grouped([1, 2, 2, 3, 4, 4, 4, 4, 4, 5])
    3.7
    >>> median_grouped([52, 52, 53, 54])
    52.5

    This calculates the median as the 50th percentile, and should be
    used when your data is continuous and grouped. In the above example,
    the values 1, 2, 3, etc. actually represent the midpoint of classes
    0.5-1.5, 1.5-2.5, 2.5-3.5, etc. The middle value falls somewhere in
    class 3.5-4.5, and interpolation is used to estimate it.

    Optional argument ``interval`` represents the class interval, and
    defaults to 1. Changing the class interval naturally will change the
    interpolated 50th percentile value:

    >>> median_grouped([1, 3, 3, 5, 7], interval=1)
    3.25
    >>> median_grouped([1, 3, 3, 5, 7], interval=2)
    3.5
```

```
    This function does not check whether the data points are at least
    ``interval`` apart.

median_high(data)
    Return the high median of data.

    When the number of data points is odd, the middle value is returned.
    When it is even, the larger of the two middle values is returned.

    >>> median_high([1, 3, 5])
    3
    >>> median_high([1, 3, 5, 7])
    5

median_low(data)
    Return the low median of numeric data.

    When the number of data points is odd, the middle value is returned.
    When it is even, the smaller of the two middle values is returned.

    >>> median_low([1, 3, 5])
    3
    >>> median_low([1, 3, 5, 7])
    3

mode(data)
    Return the most common data point from discrete or nominal data.

    ``mode`` assumes discrete data, and returns a single value. This is the
    standard treatment of the mode as commonly taught in schools:

        >>> mode([1, 1, 2, 3, 3, 3, 3, 4])
        3

    This also works with nominal (non-numeric) data:

        >>> mode(["red", "blue", "blue", "red", "green", "red", "red"])
        'red'

    If there are multiple modes with same frequency, return the first one
    encountered:

        >>> mode(['red', 'red', 'green', 'blue', 'blue'])
        'red'

    If *data* is empty, ``mode``, raises StatisticsError.

multimode(data)
    Return a list of the most frequently occurring values.

    Will return more than one result if there are multiple modes
    or an empty list if *data* is empty.

    >>> multimode('aabbbbbbbbcc')
    ['b']
    >>> multimode('aabbbbccddddeeffffgg')
    ['b', 'd', 'f']
    >>> multimode('')
    []
```

```
    pstdev(data, mu=None)
        Return the square root of the population variance.

        See ``pvariance`` for arguments and other details.

        >>> pstdev([1.5, 2.5, 2.5, 2.75, 3.25, 4.75])
        0.986893273527251

    pvariance(data, mu=None)
        Return the population variance of ``data``.

        data should be a sequence or iterable of Real-valued numbers, with at least o
ne
        value. The optional argument mu, if given, should be the mean of
        the data. If it is missing or None, the mean is automatically calculated.

        Use this function to calculate the variance from the entire population.
        To estimate the variance from a sample, the ``variance`` function is
        usually a better choice.

        Examples:

        >>> data = [0.0, 0.25, 0.25, 1.25, 1.5, 1.75, 2.75, 3.25]
        >>> pvariance(data)
        1.25

        If you have already calculated the mean of the data, you can pass it as
        the optional second argument to avoid recalculating it:

        >>> mu = mean(data)
        >>> pvariance(data, mu)
        1.25

        Decimals and Fractions are supported:

        >>> from decimal import Decimal as D
        >>> pvariance([D("27.5"), D("30.25"), D("30.25"), D("34.5"), D("41.75")])
        Decimal('24.815')

        >>> from fractions import Fraction as F
        >>> pvariance([F(1, 4), F(5, 4), F(1, 2)])
        Fraction(13, 72)

    quantiles(data, *, n=4, method='exclusive')
        Divide *data* into *n* continuous intervals with equal probability.

        Returns a list of (n - 1) cut points separating the intervals.

        Set *n* to 4 for quartiles (the default).  Set *n* to 10 for deciles.
        Set *n* to 100 for percentiles which gives the 99 cuts points that
        separate *data* in to 100 equal sized groups.

        The *data* can be any iterable containing sample.
        The cut points are linearly interpolated between data points.

        If *method* is set to *inclusive*, *data* is treated as population
        data.  The minimum value is treated as the 0th percentile and the
        maximum value is treated as the 100th percentile.
```

```
    stdev(data, xbar=None)
        Return the square root of the sample variance.

        See ``variance`` for arguments and other details.

        >>> stdev([1.5, 2.5, 2.5, 2.75, 3.25, 4.75])
        1.0810874155219827

    variance(data, xbar=None)
        Return the sample variance of data.

        data should be an iterable of Real-valued numbers, with at least two
        values. The optional argument xbar, if given, should be the mean of
        the data. If it is missing or None, the mean is automatically calculated.

        Use this function when your data is a sample from a population. To
        calculate the variance from the entire population, see ``pvariance``.

        Examples:

        >>> data = [2.75, 1.75, 1.25, 0.25, 0.5, 1.25, 3.5]
        >>> variance(data)
        1.3720238095238095

        If you have already calculated the mean of your data, you can pass it as
        the optional second argument ``xbar`` to avoid recalculating it:

        >>> m = mean(data)
        >>> variance(data, m)
        1.3720238095238095

        This function does not check that ``xbar`` is actually the mean of
        ``data``. Giving arbitrary values for ``xbar`` may lead to invalid or
        impossible results.

        Decimals and Fractions are supported:

        >>> from decimal import Decimal as D
        >>> variance([D("27.5"), D("30.25"), D("30.25"), D("34.5"), D("41.75")])
        Decimal('31.01875')

        >>> from fractions import Fraction as F
        >>> variance([F(1, 6), F(1, 2), F(5, 3)])
        Fraction(67, 108)

DATA
    __all__ = ['NormalDist', 'StatisticsError', 'correlation', 'covariance...

FILE
    c:\users\callage\appdata\local\programs\python\python310\lib\statistics.py
```

In [15]:

```
cs=[10,20,30,40,50]
print(st.mean(cs))
```

```
30
```

In [17]:

```
help(st.mode)
```

```
Help on function mode in module statistics:

mode(data)
    Return the most common data point from discrete or nominal data.

    ``mode`` assumes discrete data, and returns a single value. This is the
    standard treatment of the mode as commonly taught in schools:

        >>> mode([1, 1, 2, 3, 3, 3, 3, 4])
        3

    This also works with nominal (non-numeric) data:

        >>> mode(["red", "blue", "blue", "red", "green", "red", "red"])
        'red'

    If there are multiple modes with same frequency, return the first one
    encountered:

        >>> mode(['red', 'red', 'green', 'blue', 'blue'])
        'red'

    If *data* is empty, ``mode``, raises StatisticsError.
```

In [18]:
```
st.mode([1,2,3,4,2,2,1,5,2,5])
```

Out[18]:
```
2
```

In [19]:
```
help (st.median)
```

```
Help on function median in module statistics:

median(data)
    Return the median (middle value) of numeric data.

    When the number of data points is odd, return the middle data point.
    When the number of data points is even, the median is interpolated by
    taking the average of the two middle values:

    >>> median([1, 3, 5])
    3
    >>> median([1, 3, 5, 7])
    4.0
```

In [20]:
```
st.median([1,4,6,5,3,7,9])
```

Out[20]:
```
5
```

In [21]:
```
help (st.stdev)
```

```
Help on function stdev in module statistics:

stdev(data, xbar=None)
    Return the square root of the sample variance.

    See ``variance`` for arguments and other details.

    >>> stdev([1.5, 2.5, 2.5, 2.75, 3.25, 4.75])
    1.0810874155219827
```

RANDOM MODULE

In [22]:

```
import random as ra
dir(ra)
```

Out[22]:
```
['BPF',
 'LOG4',
 'NV_MAGICCONST',
 'RECIP_BPF',
 'Random',
 'SG_MAGICCONST',
 'SystemRandom',
 'TWOPI',
 '_ONE',
 '_Sequence',
 '_Set',
 '__all__',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 '_accumulate',
 '_acos',
 '_bisect',
 '_ceil',
 '_cos',
 '_e',
 '_exp',
 '_floor',
 '_index',
 '_inst',
 '_isfinite',
 '_log',
 '_os',
 '_pi',
 '_random',
 '_repeat',
 '_sha512',
 '_sin',
 '_sqrt',
 '_test',
 '_test_generator',
 '_urandom',
 '_warn',
 'betavariate',
 'choice',
 'choices',
 'expovariate',
 'gammavariate',
 'gauss',
 'getrandbits',
 'getstate',
 'lognormvariate',
 'normalvariate',
 'paretovariate',
 'randbytes',
 'randint',
 'random',
 'randrange',
 'sample',
 'seed',
```

```
'setstate',
'shuffle',
'triangular',
'uniform',
'vonmisesvariate',
'weibullvariate']
```

In [23]:

```
help(ra)
```

```
Help on module random:

NAME
    random - Random variable generators.

MODULE REFERENCE
    https://docs.python.org/3.10/library/random.html

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
        bytes
        -----
               uniform bytes (values between 0 and 255)

        integers
        --------
               uniform within range

        sequences
        ---------
               pick random element
               pick random sample
               pick weighted random sample
               generate random permutation

        distributions on the real line:
        ------------------------------
               uniform
               triangular
               normal (Gaussian)
               lognormal
               negative exponential
               gamma
               beta
               pareto
               Weibull

        distributions on the circle (angles 0 to 2pi)
        ---------------------------------------------
               circular uniform
               von Mises

    General notes on the underlying Mersenne Twister core generator:

    * The period is 2**19937-1.
    * It is one of the most extensively tested generators in existence.
    * The random() method is implemented in C, executes in a single Python step,
      and is, therefore, threadsafe.

CLASSES
    _random.Random(builtins.object)
        Random
            SystemRandom

    class Random(_random.Random)
```

```
|  Random(x=None)
|
|  Random number generator base class used by bound module functions.
|
|  Used to instantiate instances of Random to get generators that don't
|  share state.
|
|  Class Random can also be subclassed if you want to use a different basic
|  generator of your own devising: in that case, override the following
|  methods:  random(), seed(), getstate(), and setstate().
|  Optionally, implement a getrandbits() method so that randrange()
|  can cover arbitrarily large ranges.
|
|  Method resolution order:
|      Random
|      _random.Random
|      builtins.object
|
|  Methods defined here:
|
|  __getstate__(self)
|      # Issue 17489: Since __reduce__ was defined to fix #759889 this is no
|      # longer called; we leave it here because it has been here since random w
as
|      # rewritten back in 2001 and why risk breaking something.
|
|  __init__(self, x=None)
|      Initialize an instance.
|
|      Optional argument x controls seeding, as for Random.seed().
|
|  __reduce__(self)
|      Helper for pickle.
|
|  __setstate__(self, state)
|
|  betavariate(self, alpha, beta)
|      Beta distribution.
|
|      Conditions on the parameters are alpha > 0 and beta > 0.
|      Returned values range between 0 and 1.
|
|  choice(self, seq)
|      Choose a random element from a non-empty sequence.
|
|  choices(self, population, weights=None, *, cum_weights=None, k=1)
|      Return a k sized list of population elements chosen with replacement.
|
|      If the relative weights or cumulative weights are not specified,
|      the selections are made with equal probability.
|
|  expovariate(self, lambd)
|      Exponential distribution.
|
|      lambd is 1.0 divided by the desired mean.  It should be
|      nonzero.  (The parameter would be called "lambda", but that is
|      a reserved word in Python.)  Returned values range from 0 to
|      positive infinity if lambd is positive, and from negative
|      infinity to 0 if lambd is negative.
|
```

```
|  gammavariate(self, alpha, beta)
|      Gamma distribution.  Not the gamma function!
|
|      Conditions on the parameters are alpha > 0 and beta > 0.
|
|      The probability distribution function is:
|
|                  x ** (alpha - 1) * math.exp(-x / beta)
|        pdf(x) =  --------------------------------------
|                    math.gamma(alpha) * beta ** alpha
|
|  gauss(self, mu, sigma)
|      Gaussian distribution.
|
|      mu is the mean, and sigma is the standard deviation.  This is
|      slightly faster than the normalvariate() function.
|
|      Not thread-safe without a lock around calls.
|
|  getstate(self)
|      Return internal state; can be passed to setstate() later.
|
|  lognormvariate(self, mu, sigma)
|      Log normal distribution.
|
|      If you take the natural logarithm of this distribution, you'll get a
|      normal distribution with mean mu and standard deviation sigma.
|      mu can have any value, and sigma must be greater than zero.
|
|  normalvariate(self, mu, sigma)
|      Normal distribution.
|
|      mu is the mean, and sigma is the standard deviation.
|
|  paretovariate(self, alpha)
|      Pareto distribution.  alpha is the shape parameter.
|
|  randbytes(self, n)
|      Generate n random bytes.
|
|  randint(self, a, b)
|      Return random integer in range [a, b], including both end points.
|
|  randrange(self, start, stop=None, step=1)
|      Choose a random item from range(start, stop[, step]).
|
|      This fixes the problem with randint() which includes the
|      endpoint; in Python this is usually not what you want.
|
|  sample(self, population, k, *, counts=None)
|      Chooses k unique random elements from a population sequence or set.
|
|      Returns a new list containing elements from the population while
|      leaving the original population unchanged.  The resulting list is
|      in selection order so that all sub-slices will also be valid random
|      samples.  This allows raffle winners (the sample) to be partitioned
|      into grand prize and second place winners (the subslices).
|
|      Members of the population need not be hashable or unique.  If the
|      population contains repeats, then each occurrence is a possible
```

```
|      selection in the sample.
|
|      Repeated elements can be specified one at a time or with the optional
|      counts parameter.  For example:
|
|          sample(['red', 'blue'], counts=[4, 2], k=5)
|
|      is equivalent to:
|
|          sample(['red', 'red', 'red', 'red', 'blue', 'blue'], k=5)
|
|      To choose a sample from a range of integers, use range() for the
|      population argument.  This is especially fast and space efficient
|      for sampling from a large population:
|
|          sample(range(10000000), 60)
|
|  seed(self, a=None, version=2)
|      Initialize internal state from a seed.
|
|      The only supported seed types are None, int, float,
|      str, bytes, and bytearray.
|
|      None or no argument seeds from current time or from an operating
|      system specific randomness source if available.
|
|      If *a* is an int, all bits are used.
|
|      For version 2 (the default), all of the bits are used if *a* is a str,
|      bytes, or bytearray.  For version 1 (provided for reproducing random
|      sequences from older versions of Python), the algorithm for str and
|      bytes generates a narrower range of seeds.
|
|  setstate(self, state)
|      Restore internal state from object returned by getstate().
|
|  shuffle(self, x, random=None)
|      Shuffle list x in place, and return None.
|
|      Optional argument random is a 0-argument function returning a
|      random float in [0.0, 1.0); if it is the default None, the
|      standard random.random will be used.
|
|  triangular(self, low=0.0, high=1.0, mode=None)
|      Triangular distribution.
|
|      Continuous distribution bounded by given lower and upper limits,
|      and having a given mode value in-between.
|
|      http://en.wikipedia.org/wiki/Triangular_distribution
|
|  uniform(self, a, b)
|      Get a random number in the range [a, b) or [a, b] depending on rounding.
|
|  vonmisesvariate(self, mu, kappa)
|      Circular data distribution.
|
|      mu is the mean angle, expressed in radians between 0 and 2*pi, and
|      kappa is the concentration parameter, which must be greater than or
|      equal to zero.  If kappa is equal to zero, this distribution reduces
```

```
 |      to a uniform random angle over the range 0 to 2*pi.
 |
 |  weibullvariate(self, alpha, beta)
 |      Weibull distribution.
 |
 |      alpha is the scale parameter and beta is the shape parameter.
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  __init_subclass__(**kwargs) from builtins.type
 |      Control how subclasses generate random integers.
 |
 |      The algorithm a subclass can use depends on the random() and/or
 |      getrandbits() implementation available to it and determines
 |      whether it can generate random integers from arbitrarily large
 |      ranges.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes defined here:
 |
 |  VERSION = 3
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from _random.Random:
 |
 |  getrandbits(self, k, /)
 |      getrandbits(k) -> x.  Generates an int with k random bits.
 |
 |  random(self, /)
 |      random() -> x in the interval [0, 1).
 |
 |  ----------------------------------------------------------------------
 |  Static methods inherited from _random.Random:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.

class SystemRandom(Random)
 |  SystemRandom(x=None)
 |
 |  Alternate random number generator using sources provided
 |  by the operating system (such as /dev/urandom on Unix or
 |  CryptGenRandom on Windows).
 |
 |   Not available on all systems (see os.urandom() for details).
 |
 |  Method resolution order:
 |      SystemRandom
 |      Random
 |      _random.Random
 |
```

```
|      builtins.object
|
|  Methods defined here:
|
|  getrandbits(self, k)
|      getrandbits(k) -> x.  Generates an int with k random bits.
|
|  getstate = _notimplemented(self, *args, **kwds)
|
|  randbytes(self, n)
|      Generate n random bytes.
|
|  random(self)
|      Get the next random number in the range [0.0, 1.0).
|
|  seed(self, *args, **kwds)
|      Stub method.  Not used for a system random number generator.
|
|  setstate = _notimplemented(self, *args, **kwds)
|
|  ----------------------------------------------------------------------
|  Methods inherited from Random:
|
|  __getstate__(self)
|      # Issue 17489: Since __reduce__ was defined to fix #759889 this is no
|      # longer called; we leave it here because it has been here since random w
as
|      # rewritten back in 2001 and why risk breaking something.
|
|  __init__(self, x=None)
|      Initialize an instance.
|
|      Optional argument x controls seeding, as for Random.seed().
|
|  __reduce__(self)
|      Helper for pickle.
|
|  __setstate__(self, state)
|
|  betavariate(self, alpha, beta)
|      Beta distribution.
|
|      Conditions on the parameters are alpha > 0 and beta > 0.
|      Returned values range between 0 and 1.
|
|  choice(self, seq)
|      Choose a random element from a non-empty sequence.
|
|  choices(self, population, weights=None, *, cum_weights=None, k=1)
|      Return a k sized list of population elements chosen with replacement.
|
|      If the relative weights or cumulative weights are not specified,
|      the selections are made with equal probability.
|
|  expovariate(self, lambd)
|      Exponential distribution.
|
|      lambd is 1.0 divided by the desired mean.  It should be
|      nonzero.  (The parameter would be called "lambda", but that is
|      a reserved word in Python.)  Returned values range from 0 to
```

```
|      positive infinity if lambd is positive, and from negative
|      infinity to 0 if lambd is negative.
|
|  gammavariate(self, alpha, beta)
|      Gamma distribution.  Not the gamma function!
|
|      Conditions on the parameters are alpha > 0 and beta > 0.
|
|      The probability distribution function is:
|
|                  x ** (alpha - 1) * math.exp(-x / beta)
|        pdf(x) =  --------------------------------------
|                    math.gamma(alpha) * beta ** alpha
|
|  gauss(self, mu, sigma)
|      Gaussian distribution.
|
|      mu is the mean, and sigma is the standard deviation.  This is
|      slightly faster than the normalvariate() function.
|
|      Not thread-safe without a lock around calls.
|
|  lognormvariate(self, mu, sigma)
|      Log normal distribution.
|
|      If you take the natural logarithm of this distribution, you'll get a
|      normal distribution with mean mu and standard deviation sigma.
|      mu can have any value, and sigma must be greater than zero.
|
|  normalvariate(self, mu, sigma)
|      Normal distribution.
|
|      mu is the mean, and sigma is the standard deviation.
|
|  paretovariate(self, alpha)
|      Pareto distribution.  alpha is the shape parameter.
|
|  randint(self, a, b)
|      Return random integer in range [a, b], including both end points.
|
|  randrange(self, start, stop=None, step=1)
|      Choose a random item from range(start, stop[, step]).
|
|      This fixes the problem with randint() which includes the
|      endpoint; in Python this is usually not what you want.
|
|  sample(self, population, k, *, counts=None)
|      Chooses k unique random elements from a population sequence or set.
|
|      Returns a new list containing elements from the population while
|      leaving the original population unchanged.  The resulting list is
|      in selection order so that all sub-slices will also be valid random
|      samples.  This allows raffle winners (the sample) to be partitioned
|      into grand prize and second place winners (the subslices).
|
|      Members of the population need not be hashable or unique.  If the
|      population contains repeats, then each occurrence is a possible
|      selection in the sample.
|
|      Repeated elements can be specified one at a time or with the optional
```

```
|      counts parameter.  For example:
|
|          sample(['red', 'blue'], counts=[4, 2], k=5)
|
|      is equivalent to:
|
|          sample(['red', 'red', 'red', 'red', 'blue', 'blue'], k=5)
|
|      To choose a sample from a range of integers, use range() for the
|      population argument.  This is especially fast and space efficient
|      for sampling from a large population:
|
|          sample(range(10000000), 60)
|
|  shuffle(self, x, random=None)
|      Shuffle list x in place, and return None.
|
|      Optional argument random is a 0-argument function returning a
|      random float in [0.0, 1.0); if it is the default None, the
|      standard random.random will be used.
|
|  triangular(self, low=0.0, high=1.0, mode=None)
|      Triangular distribution.
|
|      Continuous distribution bounded by given lower and upper limits,
|      and having a given mode value in-between.
|
|      http://en.wikipedia.org/wiki/Triangular_distribution
|
|  uniform(self, a, b)
|      Get a random number in the range [a, b) or [a, b] depending on rounding.
|
|  vonmisesvariate(self, mu, kappa)
|      Circular data distribution.
|
|      mu is the mean angle, expressed in radians between 0 and 2*pi, and
|      kappa is the concentration parameter, which must be greater than or
|      equal to zero.  If kappa is equal to zero, this distribution reduces
|      to a uniform random angle over the range 0 to 2*pi.
|
|  weibullvariate(self, alpha, beta)
|      Weibull distribution.
|
|      alpha is the scale parameter and beta is the shape parameter.
|
|  ----------------------------------------------------------------------
|  Class methods inherited from Random:
|
|  __init_subclass__(**kwargs) from builtins.type
|      Control how subclasses generate random integers.
|
|      The algorithm a subclass can use depends on the random() and/or
|      getrandbits() implementation available to it and determines
|      whether it can generate random integers from arbitrarily large
|      ranges.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from Random:
|
|  __dict__
```

```
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes inherited from Random:
 |
 |  VERSION = 3
 |
 |  ----------------------------------------------------------------------
 |  Static methods inherited from _random.Random:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.

FUNCTIONS
    betavariate(alpha, beta) method of Random instance
        Beta distribution.

        Conditions on the parameters are alpha > 0 and beta > 0.
        Returned values range between 0 and 1.

    choice(seq) method of Random instance
        Choose a random element from a non-empty sequence.

    choices(population, weights=None, *, cum_weights=None, k=1) method of Random inst
ance
        Return a k sized list of population elements chosen with replacement.

        If the relative weights or cumulative weights are not specified,
        the selections are made with equal probability.

    expovariate(lambd) method of Random instance
        Exponential distribution.

        lambd is 1.0 divided by the desired mean.  It should be
        nonzero.  (The parameter would be called "lambda", but that is
        a reserved word in Python.)  Returned values range from 0 to
        positive infinity if lambd is positive, and from negative
        infinity to 0 if lambd is negative.

    gammavariate(alpha, beta) method of Random instance
        Gamma distribution.  Not the gamma function!

        Conditions on the parameters are alpha > 0 and beta > 0.

        The probability distribution function is:

                    x ** (alpha - 1) * math.exp(-x / beta)
          pdf(x) =  --------------------------------------
                      math.gamma(alpha) * beta ** alpha

    gauss(mu, sigma) method of Random instance
        Gaussian distribution.

        mu is the mean, and sigma is the standard deviation.  This is
        slightly faster than the normalvariate() function.

        Not thread-safe without a lock around calls.
```

```
getrandbits(k, /) method of Random instance
    getrandbits(k) -> x.  Generates an int with k random bits.

getstate() method of Random instance
    Return internal state; can be passed to setstate() later.

lognormvariate(mu, sigma) method of Random instance
    Log normal distribution.

    If you take the natural logarithm of this distribution, you'll get a
    normal distribution with mean mu and standard deviation sigma.
    mu can have any value, and sigma must be greater than zero.

normalvariate(mu, sigma) method of Random instance
    Normal distribution.

    mu is the mean, and sigma is the standard deviation.

paretovariate(alpha) method of Random instance
    Pareto distribution.  alpha is the shape parameter.

randbytes(n) method of Random instance
    Generate n random bytes.

randint(a, b) method of Random instance
    Return random integer in range [a, b], including both end points.

random() method of Random instance
    random() -> x in the interval [0, 1).

randrange(start, stop=None, step=1) method of Random instance
    Choose a random item from range(start, stop[, step]).

    This fixes the problem with randint() which includes the
    endpoint; in Python this is usually not what you want.

sample(population, k, *, counts=None) method of Random instance
    Chooses k unique random elements from a population sequence or set.

    Returns a new list containing elements from the population while
    leaving the original population unchanged.  The resulting list is
    in selection order so that all sub-slices will also be valid random
    samples.  This allows raffle winners (the sample) to be partitioned
    into grand prize and second place winners (the subslices).

    Members of the population need not be hashable or unique.  If the
    population contains repeats, then each occurrence is a possible
    selection in the sample.

    Repeated elements can be specified one at a time or with the optional
    counts parameter.  For example:

        sample(['red', 'blue'], counts=[4, 2], k=5)

    is equivalent to:

        sample(['red', 'red', 'red', 'red', 'blue', 'blue'], k=5)

    To choose a sample from a range of integers, use range() for the
```

```
        population argument.  This is especially fast and space efficient
        for sampling from a large population:

            sample(range(10000000), 60)

    seed(a=None, version=2) method of Random instance
        Initialize internal state from a seed.

        The only supported seed types are None, int, float,
        str, bytes, and bytearray.

        None or no argument seeds from current time or from an operating
        system specific randomness source if available.

        If *a* is an int, all bits are used.

        For version 2 (the default), all of the bits are used if *a* is a str,
        bytes, or bytearray.  For version 1 (provided for reproducing random
        sequences from older versions of Python), the algorithm for str and
        bytes generates a narrower range of seeds.

    setstate(state) method of Random instance
        Restore internal state from object returned by getstate().

    shuffle(x, random=None) method of Random instance
        Shuffle list x in place, and return None.

        Optional argument random is a 0-argument function returning a
        random float in [0.0, 1.0); if it is the default None, the
        standard random.random will be used.

    triangular(low=0.0, high=1.0, mode=None) method of Random instance
        Triangular distribution.

        Continuous distribution bounded by given lower and upper limits,
        and having a given mode value in-between.

        http://en.wikipedia.org/wiki/Triangular_distribution

    uniform(a, b) method of Random instance
        Get a random number in the range [a, b) or [a, b] depending on rounding.

    vonmisesvariate(mu, kappa) method of Random instance
        Circular data distribution.

        mu is the mean angle, expressed in radians between 0 and 2*pi, and
        kappa is the concentration parameter, which must be greater than or
        equal to zero.  If kappa is equal to zero, this distribution reduces
        to a uniform random angle over the range 0 to 2*pi.

    weibullvariate(alpha, beta) method of Random instance
        Weibull distribution.

        alpha is the scale parameter and beta is the shape parameter.

DATA
    __all__ = ['Random', 'SystemRandom', 'betavariate', 'choice', 'choices...

FILE
    c:\users\callage\appdata\local\programs\python\python310\lib\random.py
```

In [29]:
```
print(ra.random())
```

```
0.06902171704106841
```

In [30]:
```
help(ra.random)
```

```
Help on built-in function random:

random() method of random.Random instance
    random() -> x in the interval [0, 1).
```

In [33]:
```
help(ra.randint)
```

```
Help on method randint in module random:

randint(a, b) method of random.Random instance
    Return random integer in range [a, b], including both end points.
```

In [44]:
```
print(ra.randint(1,9))
```

```
6
```

In [45]:
```
help(ra.choice)
```

```
Help on method choice in module random:

choice(seq) method of random.Random instance
    Choose a random element from a non-empty sequence.
```

In [54]:
```
print(ra.choice([11,22,3,44,55,66]))
print(ra.choice("Computer Science"))
print(ra.choice(('a','b','c','d')))
```

```
66
t
a
```

In [55]:
```
help(ra.shuffle)
```

```
Help on method shuffle in module random:

shuffle(x, random=None) method of random.Random instance
    Shuffle list x in place, and return None.

    Optional argument random is a 0-argument function returning a
    random float in [0.0, 1.0); if it is the default None, the
    standard random.random will be used.
```

In [56]:
```
mh=[12,22,33,44,55,66]
print(id(mh))
ra.shuffle(mh)
print(id(mh))
print(mh)
```

```
1905228688576
1905228688576
[55, 33, 44, 22, 66, 12]
```

OS MODULE

In [57]:
```
import os
dir(os)
```

Out[57]:
```
['DirEntry',
 'F_OK',
 'GenericAlias',
 'Mapping',
 'MutableMapping',
 'O_APPEND',
 'O_BINARY',
 'O_CREAT',
 'O_EXCL',
 'O_NOINHERIT',
 'O_RANDOM',
 'O_RDONLY',
 'O_RDWR',
 'O_SEQUENTIAL',
 'O_SHORT_LIVED',
 'O_TEMPORARY',
 'O_TEXT',
 'O_TRUNC',
 'O_WRONLY',
 'P_DETACH',
 'P_NOWAIT',
 'P_NOWAITO',
 'P_OVERLAY',
 'P_WAIT',
 'PathLike',
 'R_OK',
 'SEEK_CUR',
 'SEEK_END',
 'SEEK_SET',
 'TMP_MAX',
 'W_OK',
 'X_OK',
 '_AddedDllDirectory',
 '_Environ',
 '__all__',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 '_check_methods',
 '_execvpe',
 '_exists',
 '_exit',
 '_fspath',
 '_get_exports_list',
 '_walk',
 '_wrap_close',
 'abc',
 'abort',
 'access',
 'add_dll_directory',
 'altsep',
 'chdir',
 'chmod',
 'close',
 'closerange',
```

```
 'cpu_count',
 'curdir',
 'defpath',
 'device_encoding',
 'devnull',
 'dup',
 'dup2',
 'environ',
 'error',
 'execl',
 'execle',
 'execlp',
 'execlpe',
 'execv',
 'execve',
 'execvp',
 'execvpe',
 'extsep',
 'fdopen',
 'fsdecode',
 'fsencode',
 'fspath',
 'fstat',
 'fsync',
 'ftruncate',
 'get_exec_path',
 'get_handle_inheritable',
 'get_inheritable',
 'get_terminal_size',
 'getcwd',
 'getcwdb',
 'getenv',
 'getlogin',
 'getpid',
 'getppid',
 'isatty',
 'kill',
 'linesep',
 'link',
 'listdir',
 'lseek',
 'lstat',
 'makedirs',
 'mkdir',
 'name',
 'open',
 'pardir',
 'path',
 'pathsep',
 'pipe',
 'popen',
 'putenv',
 'read',
 'readlink',
 'remove',
 'removedirs',
 'rename',
 'renames',
 'replace',
 'rmdir',
```

```
'scandir',
'sep',
'set_handle_inheritable',
'set_inheritable',
'spawnl',
'spawnle',
'spawnv',
'spawnve',
'st',
'startfile',
'stat',
'stat_result',
'statvfs_result',
'strerror',
'supports_bytes_environ',
'supports_dir_fd',
'supports_effective_ids',
'supports_fd',
'supports_follow_symlinks',
'symlink',
'sys',
'system',
'terminal_size',
'times',
'times_result',
'truncate',
'umask',
'uname_result',
'unlink',
'unsetenv',
'urandom',
'utime',
'waitpid',
'waitstatus_to_exitcode',
'walk',
'write']
```

In [58]:

```
help(os)
```

```
Help on module os:

NAME
    os - OS routines for NT or Posix depending on what system we're on.

MODULE REFERENCE
    https://docs.python.org/3.10/library/os.html

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
    This exports:
      - all functions from posix or nt, e.g. unlink, stat, etc.
      - os.path is either posixpath or ntpath
      - os.name is either 'posix' or 'nt'
      - os.curdir is a string representing the current directory (always '.')
      - os.pardir is a string representing the parent directory (always '..')
      - os.sep is the (or a most common) pathname separator ('/' or '\\')
      - os.extsep is the extension separator (always '.')
      - os.altsep is the alternate pathname separator (None or '/')
      - os.pathsep is the component separator used in $PATH etc
      - os.linesep is the line separator in text files ('\r' or '\n' or '\r\n')
      - os.defpath is the default search path for executables
      - os.devnull is the file path of the null device ('/dev/null', etc.)

    Programs that import and use 'os' stand a better chance of being
    portable between different platforms.  Of course, they must then
    only use functions that are defined by all platforms (e.g., unlink
    and opendir), and leave all pathname manipulation to os.path
    (e.g., split and join).

CLASSES
    builtins.Exception(builtins.BaseException)
        builtins.OSError
    builtins.object
        nt.DirEntry
    builtins.tuple(builtins.object)
        nt.times_result
        nt.uname_result
        stat_result
        statvfs_result
        terminal_size

    class DirEntry(builtins.object)
     |  Methods defined here:
     |
     |  __fspath__(self, /)
     |      Returns the path for the entry.
     |
     |  __repr__(self, /)
     |      Return repr(self).
     |
     |  inode(self, /)
     |      Return inode of the entry; cached per entry.
     |
     |  is_dir(self, /, *, follow_symlinks=True)
```

```
 |      Return True if the entry is a directory; cached per entry.
 |
 |  is_file(self, /, *, follow_symlinks=True)
 |      Return True if the entry is a file; cached per entry.
 |
 |  is_symlink(self, /)
 |      Return True if the entry is a symbolic link; cached per entry.
 |
 |  stat(self, /, *, follow_symlinks=True)
 |      Return stat_result object for the entry; cached per entry.
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  __class_getitem__(...) from builtins.type
 |      See PEP 585
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  name
 |      the entry's base filename, relative to scandir() "path" argument
 |
 |  path
 |      the entry's full path name; equivalent to os.path.join(scandir_path, entr
y.name)

    error = class OSError(Exception)
 |  Base class for I/O related errors.
 |
 |  Method resolution order:
 |      OSError
 |      Exception
 |      BaseException
 |      object
 |
 |  Built-in subclasses:
 |      BlockingIOError
 |      ChildProcessError
 |      ConnectionError
 |      FileExistsError
 |      ... and 7 other subclasses
 |
 |  Methods defined here:
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __reduce__(...)
 |      Helper for pickle.
 |
 |  __str__(self, /)
 |      Return str(self).
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate signature.
 |
```

```
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  characters_written
 |
 |  errno
 |      POSIX exception code
 |
 |  filename
 |      exception filename
 |
 |  filename2
 |      second exception filename
 |
 |  strerror
 |      exception strerror
 |
 |  winerror
 |      Win32 exception code
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from BaseException:
 |
 |  __delattr__(self, name, /)
 |      Implement delattr(self, name).
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __setattr__(self, name, value, /)
 |      Implement setattr(self, name, value).
 |
 |  __setstate__(...)
 |
 |  with_traceback(...)
 |      Exception.with_traceback(tb) --
 |      set self.__traceback__ to tb and return self.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from BaseException:
 |
 |  __cause__
 |      exception cause
 |
 |  __context__
 |      exception context
 |
 |  __dict__
 |
 |  __suppress_context__
 |
 |  __traceback__
 |
 |  args

class stat_result(builtins.tuple)
 |  stat_result(iterable=(), /)
```

```
|
|  stat_result: Result from stat, fstat, or lstat.
|
|  This object may be accessed either as a tuple of
|    (mode, ino, dev, nlink, uid, gid, size, atime, mtime, ctime)
|  or via the attributes st_mode, st_ino, st_dev, st_nlink, st_uid, and so on.
|
|  Posix/windows: If your platform supports st_blksize, st_blocks, st_rdev,
|  or st_flags, they are available as attributes only.
|
|  See os.stat for more information.
|
|  Method resolution order:
|      stat_result
|      builtins.tuple
|      builtins.object
|
|  Methods defined here:
|
|  __reduce__(...)
|      Helper for pickle.
|
|  __repr__(self, /)
|      Return repr(self).
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  st_atime
|      time of last access
|
|  st_atime_ns
|      time of last access in nanoseconds
|
|  st_ctime
|      time of last change
|
|  st_ctime_ns
|      time of last change in nanoseconds
|
|  st_dev
|      device
|
|  st_file_attributes
|      Windows file attribute bits
|
|  st_gid
|      group ID of owner
|
|  st_ino
|      inode
|
|  st_mode
|      protection bits
```

```
|
|  st_mtime
|      time of last modification
|
|  st_mtime_ns
|      time of last modification in nanoseconds
|
|  st_nlink
|      number of hard links
|
|  st_reparse_tag
|      Windows reparse tag
|
|  st_size
|      total size, in bytes
|
|  st_uid
|      user ID of owner
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __match_args__ = ('st_mode', 'st_ino', 'st_dev', 'st_nlink', 'st_uid',...
|
|  n_fields = 18
|
|  n_sequence_fields = 10
|
|  n_unnamed_fields = 3
|
|  ----------------------------------------------------------------------
|  Methods inherited from builtins.tuple:
|
|  __add__(self, value, /)
|      Return self+value.
|
|  __contains__(self, key, /)
|      Return key in self.
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(self, key, /)
|      Return self[key].
|
|  __getnewargs__(self, /)
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __hash__(self, /)
|      Return hash(self).
|
|  __iter__(self, /)
```

```
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |
 |      Raises ValueError if the value is not present.
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from builtins.tuple:
 |
 |  __class_getitem__(...) from builtins.type
 |      See PEP 585

class statvfs_result(builtins.tuple)
 |  statvfs_result(iterable=(), /)
 |
 |  statvfs_result: Result from statvfs or fstatvfs.
 |
 |  This object may be accessed either as a tuple of
 |    (bsize, frsize, blocks, bfree, bavail, files, ffree, favail, flag, namema
x),
 |  or via the attributes f_bsize, f_frsize, f_blocks, f_bfree, and so on.
 |
 |  See os.statvfs for more information.
 |
 |  Method resolution order:
 |      statvfs_result
 |      builtins.tuple
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __reduce__(...)
 |      Helper for pickle.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  ----------------------------------------------------------------------
```

```
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  f_bavail
|
|  f_bfree
|
|  f_blocks
|
|  f_bsize
|
|  f_favail
|
|  f_ffree
|
|  f_files
|
|  f_flag
|
|  f_frsize
|
|  f_fsid
|
|  f_namemax
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __match_args__ = ('f_bsize', 'f_frsize', 'f_blocks', 'f_bfree', 'f_bav...
|
|  n_fields = 11
|
|  n_sequence_fields = 10
|
|  n_unnamed_fields = 0
|
|  ----------------------------------------------------------------------
|  Methods inherited from builtins.tuple:
|
|  __add__(self, value, /)
|      Return self+value.
|
|  __contains__(self, key, /)
|      Return key in self.
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(self, key, /)
```

```
 |      Return self[key].
 |
 |  __getnewargs__(self, /)
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |
 |      Raises ValueError if the value is not present.
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from builtins.tuple:
 |
 |  __class_getitem__(...) from builtins.type
 |      See PEP 585

class terminal_size(builtins.tuple)
 |  terminal_size(iterable=(), /)
 |
 |  A tuple of (columns, lines) for holding terminal window size
 |
 |  Method resolution order:
 |      terminal_size
 |      builtins.tuple
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __reduce__(...)
 |      Helper for pickle.
 |
```

```
|  __repr__(self, /)
|      Return repr(self).
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  columns
|      width of the terminal window in characters
|
|  lines
|      height of the terminal window in characters
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __match_args__ = ('columns', 'lines')
|
|  n_fields = 2
|
|  n_sequence_fields = 2
|
|  n_unnamed_fields = 0
|
|  ----------------------------------------------------------------------
|  Methods inherited from builtins.tuple:
|
|  __add__(self, value, /)
|      Return self+value.
|
|  __contains__(self, key, /)
|      Return key in self.
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(self, key, /)
|      Return self[key].
|
|  __getnewargs__(self, /)
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __hash__(self, /)
|      Return hash(self).
|
|  __iter__(self, /)
|      Implement iter(self).
```

```
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |
 |      Raises ValueError if the value is not present.
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from builtins.tuple:
 |
 |  __class_getitem__(...) from builtins.type
 |      See PEP 585

class times_result(builtins.tuple)
 |  times_result(iterable=(), /)
 |
 |  times_result: Result from os.times().
 |
 |  This object may be accessed either as a tuple of
 |    (user, system, children_user, children_system, elapsed),
 |  or via the attributes user, system, children_user, children_system,
 |  and elapsed.
 |
 |  See os.times for more information.
 |
 |  Method resolution order:
 |      times_result
 |      builtins.tuple
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __reduce__(...)
 |      Helper for pickle.
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  ----------------------------------------------------------------------
 |  Static methods defined here:
```

```
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  children_system
|      system time of children
|
|  children_user
|      user time of children
|
|  elapsed
|      elapsed time since an arbitrary point in the past
|
|  system
|      system time
|
|  user
|      user time
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __match_args__ = ('user', 'system', 'children_user', 'children_system'...
|
|  n_fields = 5
|
|  n_sequence_fields = 5
|
|  n_unnamed_fields = 0
|
|  ----------------------------------------------------------------------
|  Methods inherited from builtins.tuple:
|
|  __add__(self, value, /)
|      Return self+value.
|
|  __contains__(self, key, /)
|      Return key in self.
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(self, key, /)
|      Return self[key].
|
|  __getnewargs__(self, /)
|
|  __gt__(self, value, /)
|      Return self>value.
|
|  __hash__(self, /)
```

```
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |
 |      Raises ValueError if the value is not present.
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from builtins.tuple:
 |
 |  __class_getitem__(...) from builtins.type
 |      See PEP 585

class uname_result(builtins.tuple)
 |  uname_result(iterable=(), /)
 |
 |  uname_result: Result from os.uname().
 |
 |  This object may be accessed either as a tuple of
 |    (sysname, nodename, release, version, machine),
 |  or via the attributes sysname, nodename, release, version, and machine.
 |
 |  See os.uname for more information.
 |
 |  Method resolution order:
 |      uname_result
 |      builtins.tuple
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __reduce__(...)
 |      Helper for pickle.
 |
 |  __repr__(self, /)
 |      Return repr(self).
```

```
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
|
|  ----------------------------------------------------------------------
|  Data descriptors defined here:
|
|  machine
|      hardware identifier
|
|  nodename
|      name of machine on network (implementation-defined)
|
|  release
|      operating system release
|
|  sysname
|      operating system name
|
|  version
|      operating system version
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __match_args__ = ('sysname', 'nodename', 'release', 'version', 'machin...
|
|  n_fields = 5
|
|  n_sequence_fields = 5
|
|  n_unnamed_fields = 0
|
|  ----------------------------------------------------------------------
|  Methods inherited from builtins.tuple:
|
|  __add__(self, value, /)
|      Return self+value.
|
|  __contains__(self, key, /)
|      Return key in self.
|
|  __eq__(self, value, /)
|      Return self==value.
|
|  __ge__(self, value, /)
|      Return self>=value.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __getitem__(self, key, /)
|      Return self[key].
|
|  __getnewargs__(self, /)
|
|  __gt__(self, value, /)
```

```
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __mul__(self, value, /)
 |      Return self*value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  __rmul__(self, value, /)
 |      Return value*self.
 |
 |  count(self, value, /)
 |      Return number of occurrences of value.
 |
 |  index(self, value, start=0, stop=9223372036854775807, /)
 |      Return first index of value.
 |
 |      Raises ValueError if the value is not present.
 |
 |  ----------------------------------------------------------------------
 |  Class methods inherited from builtins.tuple:
 |
 |  __class_getitem__(...) from builtins.type
 |      See PEP 585

FUNCTIONS
    _exit(status)
        Exit to the system with specified status, without normal exit processing.

    abort()
        Abort the interpreter immediately.

        This function 'dumps core' or otherwise fails in the hardest way possible
        on the hosting operating system.  This function never returns.

    access(path, mode, *, dir_fd=None, effective_ids=False, follow_symlinks=True)
        Use the real uid/gid to test for access to a path.

          path
            Path to be tested; can be string, bytes, or a path-like object.
          mode
            Operating-system mode bitfield.  Can be F_OK to test existence,
            or the inclusive-OR of R_OK, W_OK, and X_OK.
          dir_fd
            If not None, it should be a file descriptor open to a directory,
```

```
                  and path should be relative; path will then be relative to that
                  directory.
                effective_ids
                  If True, access will use the effective uid/gid instead of
                  the real uid/gid.
                follow_symlinks
                  If False, and the last element of the path is a symbolic link,
                  access will examine the symbolic link itself instead of the file
                  the link points to.

            dir_fd, effective_ids, and follow_symlinks may not be implemented
              on your platform.  If they are unavailable, using them will raise a
              NotImplementedError.

            Note that most operations will use the effective uid/gid, therefore this
              routine can be used in a suid/sgid environment to test if the invoking user
              has the specified access to the path.

        chdir(path)
            Change the current working directory to the specified path.

            path may always be specified as a string.
            On some platforms, path may also be specified as an open file descriptor.
              If this functionality is unavailable, using it raises an exception.

        chmod(path, mode, *, dir_fd=None, follow_symlinks=True)
            Change the access permissions of a file.

              path
                Path to be modified.  May always be specified as a str, bytes, or a path-
    like object.
                On some platforms, path may also be specified as an open file descriptor.
                If this functionality is unavailable, using it raises an exception.
              mode
                Operating-system mode bitfield.
              dir_fd
                If not None, it should be a file descriptor open to a directory,
                and path should be relative; path will then be relative to that
                directory.
              follow_symlinks
                If False, and the last element of the path is a symbolic link,
                chmod will modify the symbolic link itself instead of the file
                the link points to.

            It is an error to use dir_fd or follow_symlinks when specifying path as
              an open file descriptor.
            dir_fd and follow_symlinks may not be implemented on your platform.
              If they are unavailable, using them will raise a NotImplementedError.

        close(fd)
            Close a file descriptor.

        closerange(fd_low, fd_high, /)
            Closes all file descriptors in [fd_low, fd_high), ignoring errors.

        cpu_count()
            Return the number of CPUs in the system; return None if indeterminable.

            This number is not equivalent to the number of CPUs the current process can
            use.  The number of usable CPUs can be obtained with
```

```
    ``len(os.sched_getaffinity(0))``

device_encoding(fd)
    Return a string describing the encoding of a terminal's file descriptor.

    The file descriptor must be attached to a terminal.
    If the device is not a terminal, return None.

dup(fd, /)
    Return a duplicate of a file descriptor.

dup2(fd, fd2, inheritable=True)
    Duplicate file descriptor.

execl(file, *args)
    execl(file, *args)

    Execute the executable file with argument list args, replacing the
    current process.

execle(file, *args)
    execle(file, *args, env)

    Execute the executable file with argument list args and
    environment env, replacing the current process.

execlp(file, *args)
    execlp(file, *args)

    Execute the executable file (which is searched for along $PATH)
    with argument list args, replacing the current process.

execlpe(file, *args)
    execlpe(file, *args, env)

    Execute the executable file (which is searched for along $PATH)
    with argument list args and environment env, replacing the current
    process.

execv(path, argv, /)
    Execute an executable path with arguments, replacing current process.

    path
      Path of executable file.
    argv
      Tuple or list of strings.

execve(path, argv, env)
    Execute an executable path with arguments, replacing current process.

    path
      Path of executable file.
    argv
      Tuple or list of strings.
    env
      Dictionary of strings mapping to strings.

execvp(file, args)
    execvp(file, args)
```

```
    Execute the executable file (which is searched for along $PATH)
    with argument list args, replacing the current process.
    args may be a list or tuple of strings.

execvpe(file, args, env)
    execvpe(file, args, env)

    Execute the executable file (which is searched for along $PATH)
    with argument list args and environment env, replacing the
    current process.
    args may be a list or tuple of strings.

fdopen(fd, mode='r', buffering=-1, encoding=None, *args, **kwargs)
    # Supply os.fdopen()

fsdecode(filename)
    Decode filename (an os.PathLike, bytes, or str) from the filesystem
    encoding with 'surrogateescape' error handler, return str unchanged. On
    Windows, use 'strict' error handler if the file system encoding is
    'mbcs' (which is the default encoding).

fsencode(filename)
    Encode filename (an os.PathLike, bytes, or str) to the filesystem
    encoding with 'surrogateescape' error handler, return bytes unchanged.
    On Windows, use 'strict' error handler if the file system encoding is
    'mbcs' (which is the default encoding).

fspath(path)
    Return the file system path representation of the object.

    If the object is str or bytes, then allow it to pass through as-is. If the
    object defines __fspath__(), then return the result of that method. All other
    types raise a TypeError.

fstat(fd)
    Perform a stat system call on the given file descriptor.

    Like stat(), but for an open file descriptor.
    Equivalent to os.stat(fd).

fsync(fd)
    Force write of fd to disk.

ftruncate(fd, length, /)
    Truncate a file, specified by file descriptor, to a specific length.

get_exec_path(env=None)
    Returns the sequence of directories that will be searched for the
    named executable (similar to a shell) when launching a process.

    *env* must be an environment variable dict or None.  If *env* is None,
    os.environ will be used.

get_handle_inheritable(handle, /)
    Get the close-on-exe flag of the specified file descriptor.

get_inheritable(fd, /)
    Get the close-on-exe flag of the specified file descriptor.

get_terminal_size(...)
```

```
    Return the size of the terminal window as (columns, lines).

    The optional argument fd (default standard output) specifies
    which file descriptor should be queried.

    If the file descriptor is not connected to a terminal, an OSError
    is thrown.

    This function will only be defined if an implementation is
    available for this system.

    shutil.get_terminal_size is the high-level function which should
    normally be used, os.get_terminal_size is the low-level implementation.

getcwd()
    Return a unicode string representing the current working directory.

getcwdb()
    Return a bytes string representing the current working directory.

getenv(key, default=None)
    Get an environment variable, return None if it doesn't exist.
    The optional second argument can specify an alternate default.
    key, default and the result are str.

getlogin()
    Return the actual login name.

getpid()
    Return the current process id.

getppid()
    Return the parent's process id.

    If the parent process has already exited, Windows machines will still
    return its id; others systems will return the id of the 'init' process (1).

isatty(fd, /)
    Return True if the fd is connected to a terminal.

    Return True if the file descriptor is an open file descriptor
    connected to the slave end of a terminal.

kill(pid, signal, /)
    Kill a process with a signal.

link(src, dst, *, src_dir_fd=None, dst_dir_fd=None, follow_symlinks=True)
    Create a hard link to a file.

    If either src_dir_fd or dst_dir_fd is not None, it should be a file
      descriptor open to a directory, and the respective path string (src or dst)
      should be relative; the path will then be relative to that directory.
    If follow_symlinks is False, and the last element of src is a symbolic
      link, link will create a link to the symbolic link itself instead of the
      file the link points to.
    src_dir_fd, dst_dir_fd, and follow_symlinks may not be implemented on your
      platform.  If they are unavailable, using them will raise a
      NotImplementedError.

listdir(path=None)
```

```
        Return a list containing the names of the files in the directory.

        path can be specified as either str, bytes, or a path-like object.  If path i
s bytes,
          the filenames returned will also be bytes; in all other circumstances
          the filenames returned will be str.
        If path is None, uses the path='.'.
        On some platforms, path may also be specified as an open file descriptor;\
          the file descriptor must refer to a directory.
          If this functionality is unavailable, using it raises NotImplementedError.

        The list is in arbitrary order.  It does not include the special
        entries '.' and '..' even if they are present in the directory.

    lseek(fd, position, how, /)
        Set the position of a file descriptor.  Return the new position.

        Return the new cursor position in number of bytes
        relative to the beginning of the file.

    lstat(path, *, dir_fd=None)
        Perform a stat system call on the given path, without following symbolic link
s.

        Like stat(), but do not follow symbolic links.
        Equivalent to stat(path, follow_symlinks=False).

    makedirs(name, mode=511, exist_ok=False)
        makedirs(name [, mode=0o777][, exist_ok=False])

        Super-mkdir; create a leaf directory and all intermediate ones.  Works like
        mkdir, except that any intermediate path segment (not just the rightmost)
        will be created if it does not exist. If the target directory already
        exists, raise an OSError if exist_ok is False. Otherwise no exception is
        raised.  This is recursive.

    mkdir(path, mode=511, *, dir_fd=None)
        Create a directory.

        If dir_fd is not None, it should be a file descriptor open to a directory,
          and path should be relative; path will then be relative to that directory.
        dir_fd may not be implemented on your platform.
          If it is unavailable, using it will raise a NotImplementedError.

        The mode argument is ignored on Windows.

    open(path, flags, mode=511, *, dir_fd=None)
        Open a file for low level IO.  Returns a file descriptor (integer).

        If dir_fd is not None, it should be a file descriptor open to a directory,
          and path should be relative; path will then be relative to that directory.
        dir_fd may not be implemented on your platform.
          If it is unavailable, using it will raise a NotImplementedError.

    pipe()
        Create a pipe.

        Returns a tuple of two file descriptors:
          (read_fd, write_fd)
```

```
popen(cmd, mode='r', buffering=-1)
    # Supply os.popen()

putenv(name, value, /)
    Change or add an environment variable.

read(fd, length, /)
    Read from a file descriptor.  Returns a bytes object.

readlink(path, *, dir_fd=None)
    Return a string representing the path to which the symbolic link points.

    If dir_fd is not None, it should be a file descriptor open to a directory,
    and path should be relative; path will then be relative to that directory.

    dir_fd may not be implemented on your platform.  If it is unavailable,
    using it will raise a NotImplementedError.

remove(path, *, dir_fd=None)
    Remove a file (same as unlink()).

    If dir_fd is not None, it should be a file descriptor open to a directory,
      and path should be relative; path will then be relative to that directory.
    dir_fd may not be implemented on your platform.
      If it is unavailable, using it will raise a NotImplementedError.

removedirs(name)
    removedirs(name)

    Super-rmdir; remove a leaf directory and all empty intermediate
    ones.  Works like rmdir except that, if the leaf directory is
    successfully removed, directories corresponding to rightmost path
    segments will be pruned away until either the whole path is
    consumed or an error occurs.  Errors during this latter phase are
    ignored -- they generally mean that a directory was not empty.

rename(src, dst, *, src_dir_fd=None, dst_dir_fd=None)
    Rename a file or directory.

    If either src_dir_fd or dst_dir_fd is not None, it should be a file
      descriptor open to a directory, and the respective path string (src or dst)
      should be relative; the path will then be relative to that directory.
    src_dir_fd and dst_dir_fd, may not be implemented on your platform.
      If they are unavailable, using them will raise a NotImplementedError.

renames(old, new)
    renames(old, new)

    Super-rename; create directories as necessary and delete any left
    empty.  Works like rename, except creation of any intermediate
    directories needed to make the new pathname good is attempted
    first.  After the rename, directories corresponding to rightmost
    path segments of the old name will be pruned until either the
    whole path is consumed or a nonempty directory is found.

    Note: this function can fail with the new directory structure made
    if you lack permissions needed to unlink the leaf directory or
    file.

replace(src, dst, *, src_dir_fd=None, dst_dir_fd=None)
```

```
    Rename a file or directory, overwriting the destination.

    If either src_dir_fd or dst_dir_fd is not None, it should be a file
      descriptor open to a directory, and the respective path string (src or dst)
      should be relative; the path will then be relative to that directory.
    src_dir_fd and dst_dir_fd, may not be implemented on your platform.
      If they are unavailable, using them will raise a NotImplementedError.

rmdir(path, *, dir_fd=None)
    Remove a directory.

    If dir_fd is not None, it should be a file descriptor open to a directory,
      and path should be relative; path will then be relative to that directory.
    dir_fd may not be implemented on your platform.
      If it is unavailable, using it will raise a NotImplementedError.

scandir(path=None)
    Return an iterator of DirEntry objects for given path.

    path can be specified as either str, bytes, or a path-like object.  If path
    is bytes, the names of yielded DirEntry objects will also be bytes; in
    all other circumstances they will be str.

    If path is None, uses the path='.'.

set_handle_inheritable(handle, inheritable, /)
    Set the inheritable flag of the specified handle.

set_inheritable(fd, inheritable, /)
    Set the inheritable flag of the specified file descriptor.

spawnl(mode, file, *args)
    spawnl(mode, file, *args) -> integer

    Execute file with arguments from args in a subprocess.
    If mode == P_NOWAIT return the pid of the process.
    If mode == P_WAIT return the process's exit code if it exits normally;
    otherwise return -SIG, where SIG is the signal that killed it.

spawnle(mode, file, *args)
    spawnle(mode, file, *args, env) -> integer

    Execute file with arguments from args in a subprocess with the
    supplied environment.
    If mode == P_NOWAIT return the pid of the process.
    If mode == P_WAIT return the process's exit code if it exits normally;
    otherwise return -SIG, where SIG is the signal that killed it.

spawnv(mode, path, argv, /)
    Execute the program specified by path in a new process.

    mode
      Mode of process creation.
    path
      Path of executable file.
    argv
      Tuple or list of strings.

spawnve(mode, path, argv, env, /)
    Execute the program specified by path in a new process.
```

```
    mode
      Mode of process creation.
    path
      Path of executable file.
    argv
      Tuple or list of strings.
    env
      Dictionary of strings mapping to strings.

startfile(...)
    Start a file with its associated application.

    When "operation" is not specified or "open", this acts like
    double-clicking the file in Explorer, or giving the file name as an
    argument to the DOS "start" command: the file is opened with whatever
    application (if any) its extension is associated.
    When another "operation" is given, it specifies what should be done with
    the file.  A typical operation is "print".

    "arguments" is passed to the application, but should be omitted if the
    file is a document.

    "cwd" is the working directory for the operation. If "filepath" is
    relative, it will be resolved against this directory. This argument
    should usually be an absolute path.

    "show_cmd" can be used to override the recommended visibility option.
    See the Windows ShellExecute documentation for values.

    startfile returns as soon as the associated application is launched.
    There is no option to wait for the application to close, and no way
    to retrieve the application's exit status.

    The filepath is relative to the current directory.  If you want to use
    an absolute path, make sure the first character is not a slash ("/");
    the underlying Win32 ShellExecute function doesn't work if it is.

stat(path, *, dir_fd=None, follow_symlinks=True)
    Perform a stat system call on the given path.

      path
        Path to be examined; can be string, bytes, a path-like object or
        open-file-descriptor int.
      dir_fd
        If not None, it should be a file descriptor open to a directory,
        and path should be a relative string; path will then be relative to
        that directory.
      follow_symlinks
        If False, and the last element of the path is a symbolic link,
        stat will examine the symbolic link itself instead of the file
        the link points to.

    dir_fd and follow_symlinks may not be implemented
      on your platform.  If they are unavailable, using them will raise a
      NotImplementedError.

    It's an error to use dir_fd or follow_symlinks when specifying path as
      an open file descriptor.
```

```
strerror(code, /)
    Translate an error code to a message string.

symlink(src, dst, target_is_directory=False, *, dir_fd=None)
    Create a symbolic link pointing to src named dst.

    target_is_directory is required on Windows if the target is to be
      interpreted as a directory.  (On Windows, symlink requires
      Windows 6.0 or greater, and raises a NotImplementedError otherwise.)
      target_is_directory is ignored on non-Windows platforms.

    If dir_fd is not None, it should be a file descriptor open to a directory,
      and path should be relative; path will then be relative to that directory.
    dir_fd may not be implemented on your platform.
      If it is unavailable, using it will raise a NotImplementedError.

system(command)
    Execute the command in a subshell.

times()
    Return a collection containing process timing information.

    The object returned behaves like a named tuple with these fields:
      (utime, stime, cutime, cstime, elapsed_time)
    All fields are floating point numbers.

truncate(path, length)
    Truncate a file, specified by path, to a specific length.

    On some platforms, path may also be specified as an open file descriptor.
      If this functionality is unavailable, using it raises an exception.

umask(mask, /)
    Set the current numeric umask and return the previous umask.

unlink(path, *, dir_fd=None)
    Remove a file (same as remove()).

    If dir_fd is not None, it should be a file descriptor open to a directory,
      and path should be relative; path will then be relative to that directory.
    dir_fd may not be implemented on your platform.
      If it is unavailable, using it will raise a NotImplementedError.

unsetenv(name, /)
    Delete an environment variable.

urandom(size, /)
    Return a bytes object containing random bytes suitable for cryptographic use.

utime(...)
    Set the access and modified time of path.

    path may always be specified as a string.
    On some platforms, path may also be specified as an open file descriptor.
      If this functionality is unavailable, using it raises an exception.

    If times is not None, it must be a tuple (atime, mtime);
        atime and mtime should be expressed as float seconds since the epoch.
    If ns is specified, it must be a tuple (atime_ns, mtime_ns);
        atime_ns and mtime_ns should be expressed as integer nanoseconds
```

```
            since the epoch.
        If times is None and ns is unspecified, utime uses the current time.
        Specifying tuples for both times and ns is an error.

        If dir_fd is not None, it should be a file descriptor open to a directory,
          and path should be relative; path will then be relative to that directory.
        If follow_symlinks is False, and the last element of the path is a symbolic
          link, utime will modify the symbolic link itself instead of the file the
          link points to.
        It is an error to use dir_fd or follow_symlinks when specifying path
          as an open file descriptor.
        dir_fd and follow_symlinks may not be available on your platform.
          If they are unavailable, using them will raise a NotImplementedError.

    waitpid(pid, options, /)
        Wait for completion of a given process.

        Returns a tuple of information regarding the process:
            (pid, status << 8)

        The options argument is ignored on Windows.

    waitstatus_to_exitcode(status)
        Convert a wait status to an exit code.

        On Unix:

        * If WIFEXITED(status) is true, return WEXITSTATUS(status).
        * If WIFSIGNALED(status) is true, return -WTERMSIG(status).
        * Otherwise, raise a ValueError.

        On Windows, return status shifted right by 8 bits.

        On Unix, if the process is being traced or if waitpid() was called with
        WUNTRACED option, the caller must first check if WIFSTOPPED(status) is true.
        This function must not be called if WIFSTOPPED(status) is true.

    walk(top, topdown=True, onerror=None, followlinks=False)
        Directory tree generator.

        For each directory in the directory tree rooted at top (including top
        itself, but excluding '.' and '..'), yields a 3-tuple

            dirpath, dirnames, filenames

        dirpath is a string, the path to the directory.  dirnames is a list of
        the names of the subdirectories in dirpath (excluding '.' and '..').
        filenames is a list of the names of the non-directory files in dirpath.
        Note that the names in the lists are just names, with no path components.
        To get a full path (which begins with top) to a file or directory in
        dirpath, do os.path.join(dirpath, name).

        If optional arg 'topdown' is true or not specified, the triple for a
        directory is generated before the triples for any of its subdirectories
        (directories are generated top down).  If topdown is false, the triple
        for a directory is generated after the triples for all of its
        subdirectories (directories are generated bottom up).

        When topdown is true, the caller can modify the dirnames list in-place
        (e.g., via del or slice assignment), and walk will only recurse into the
```

```
            subdirectories whose names remain in dirnames; this can be used to prune the
            search, or to impose a specific order of visiting.  Modifying dirnames when
            topdown is false has no effect on the behavior of os.walk(), since the
            directories in dirnames have already been generated by the time dirnames
            itself is generated. No matter the value of topdown, the list of
            subdirectories is retrieved before the tuples for the directory and its
            subdirectories are generated.

            By default errors from the os.scandir() call are ignored.  If
            optional arg 'onerror' is specified, it should be a function; it
            will be called with one argument, an OSError instance.  It can
            report the error to continue with the walk, or raise the exception
            to abort the walk.  Note that the filename is available as the
            filename attribute of the exception object.

            By default, os.walk does not follow symbolic links to subdirectories on
            systems that support them.  In order to get this functionality, set the
            optional argument 'followlinks' to true.

            Caution:  if you pass a relative pathname for top, don't change the
            current working directory between resumptions of walk.  walk never
            changes the current directory, and assumes that the client doesn't
            either.

            Example:

            import os
            from os.path import join, getsize
            for root, dirs, files in os.walk('python/Lib/email'):
                print(root, "consumes", end="")
                print(sum(getsize(join(root, name)) for name in files), end="")
                print("bytes in", len(files), "non-directory files")
                if 'CVS' in dirs:
                    dirs.remove('CVS')  # don't visit CVS directories

        write(fd, data, /)
            Write a bytes object to a file descriptor.

    DATA
        F_OK = 0
        O_APPEND = 8
        O_BINARY = 32768
        O_CREAT = 256
        O_EXCL = 1024
        O_NOINHERIT = 128
        O_RANDOM = 16
        O_RDONLY = 0
        O_RDWR = 2
        O_SEQUENTIAL = 32
        O_SHORT_LIVED = 4096
        O_TEMPORARY = 64
        O_TEXT = 16384
        O_TRUNC = 512
        O_WRONLY = 1
        P_DETACH = 4
        P_NOWAIT = 1
        P_NOWAITO = 3
        P_OVERLAY = 2
        P_WAIT = 0
        R_OK = 4
```

```
    SEEK_CUR = 1
    SEEK_END = 2
    SEEK_SET = 0
    TMP_MAX = 2147483647
    W_OK = 2
    X_OK = 1
    __all__ = ['altsep', 'curdir', 'pardir', 'sep', 'pathsep', 'linesep', ...
    altsep = '/'
    curdir = '.'
    defpath = r'.;C:\bin'
    devnull = 'nul'
    environ = environ({'ALLUSERSPROFILE': 'C:\\ProgramData', '...D': 'modu...
    extsep = '.'
    linesep = '\r\n'
    name = 'nt'
    pardir = '..'
    pathsep = ';'
    sep = r'\'
    supports_bytes_environ = False

FILE
    c:\users\callage\appdata\local\programs\python\python310\lib\os.py
```

In [1]:

```
import cspack
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_7608/114026375.py in <module>
----> 1 import cspack

ModuleNotFoundError: No module named 'cspack'
```