Break Statement : The break is a keyword in python which is used to bring the program control out of the loop.

In [1]:
```
for i in range(7):
    if i==5:
        break
    print(i)
```

```
0
1
2
3
4
```

Continue Statement : the continue keyword return control of the iteration to the beginning of the Python for loop or Python while loop. All remaining lines in the prevailing iteration of the loop are skipped by the continue keyword, which returns execution to the beginning of the next iteration of the loop.

In [2]:
```
for i in range(7):
    if i==5:
        continue
    print(i)
```

```
0
1
2
3
4
6
```

Pass Statement : The Python pass statement to write empty loops. Pass is also used for empty control statements, functions, and classes.

In [3]:
```
for i in range(3,10):
```

```
  Cell In[3], line 1
    for i in range(3,10):
                         ^
SyntaxError: incomplete input
```

In [5]:
```
for i in range(8,10):
    pass
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Functions : A reusable piece of code. Block of statements that does some specific task and returns something.

- By including functions, we can prevent repeating the same code block repeatedly in a program.
- Python functions, once defined, can be called many times and from anywhere in a program.
- If our Python program is large, it can be separated into numerous functions which is simple to track.
- The key accomplishment of Python functions is we can return as many outputs as we want with different arguments.

Syntax of Python Function:

```
def function_name( parameters ):
   # code block
```

In [6]:
```
#no return no arguemnt
def hello():  #function declaration
    print('helloooo da students') #function definition
```

In [8]:
```
hello() #function call
```

```
helloooo da students
```

In [9]:
```
#with arguemnt no return
#single argument
def Abc(a):
    print(f"value is {a}")
```

In [10]:
```
Abc(6)
```

```
value is 6
```

In [11]:
```
Abc(5,8)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[11], line 1
----> 1 Abc(5,8)

TypeError: Abc() takes 1 positional argument but 2 were given
```

In [12]:
```
#2 arguments
def add(a,b):
    print(a+b)
```

In [13]:
```
add(5,7)
```

```
12
```

In [17]:
```
#with arguement with return
#multiple arguements
def sum1(*a):
    return sum(a)
```

In [18]:
```
sum1(4,5,6,7,4,535,354,354,353,5,32523,53,53,53,654,7,567)
```

Out[18]:
```
35537
```

In [21]:
```
#no arguement with return
def multiply():
    a = int(input('Enter a number :'))
    b = int(input('Enter another number :'))
    return f"Multiplication of {a} and {b} is {a*b}"
```

In [22]:
```
multiply()
```

```
Enter a number :6
Enter another number :9
```

Out[22]:
```
'Multiplication of 6 and 9 is 54'
```

PIL:PYTHON IMAGING LIBRARY

In [24]:
```
from PIL import Image
```

In [25]:
```
c = Image.open(r"C:\Users\msi 1\Downloads\abc.jpg")
c
```

Out[25]:

In [26]:
```
c.format
```

Out[26]:
```
'JPEG'
```

In [27]:
```
c.size
```

Out[27]:
```
(694, 988)
```

In [29]:
```
c.mode
```

Out[29]:
```
'RGB'
```

In [30]:
```
type(c)
```

Out[30]:
```
PIL.JpegImagePlugin.JpegImageFile
```

In [28]:
```
c.rotate(90)
```

Out[28]:

In [31]:
```
c.transpose(Image.FLIP_LEFT_RIGHT)
```

Out[31]:

In [32]:
```
c.transpose(Image.FLIP_TOP_BOTTOM)
```

Out[32]:

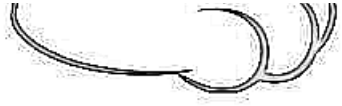
In [33]:
```
from PIL import ImageFilter
```

In [34]:
```
a = c.filter(ImageFilter.BLUR)
a
```

Out[34]:

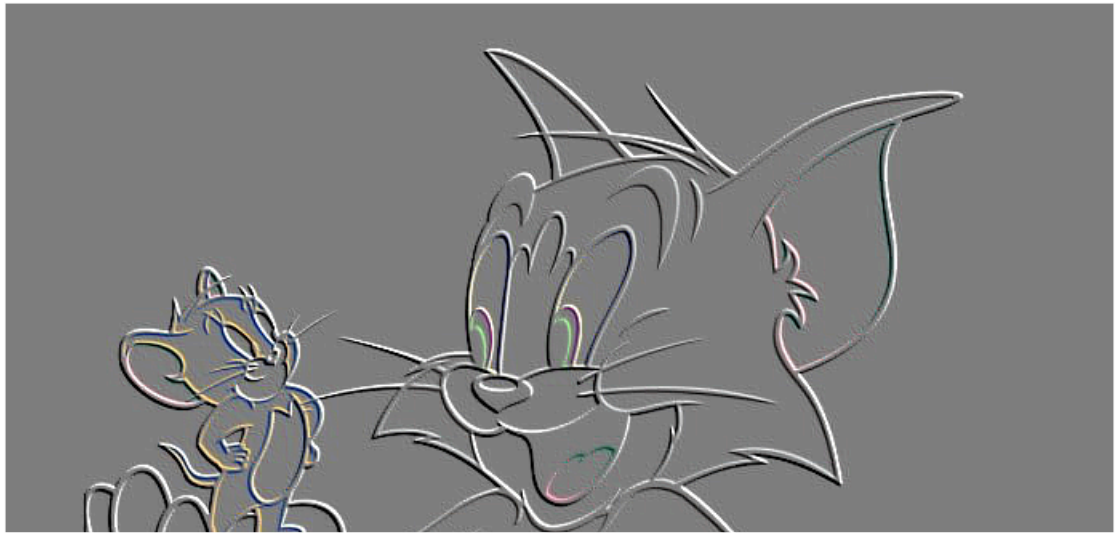
In [35]:
```
a = c.filter(ImageFilter.CONTOUR)
a
```

Out[35]:



In [36]:
```
a = c.filter(ImageFilter.EMBOSS)
a
```

Out[36]:



In [ ]:

In [ ]:

In [ ]:

In [ ]:

Write a function isPrime to print all the prime numbers between 10 - 1000.

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Q. Wapp to check if the entered number is armstrong or not.

```
Armstrong Number: The number that equals the sum of its digits, each raised to a power.
Ex: 153 => 1^3 + 5^3 + 3^3 = 153
```

In [ ]:

In [ ]:

Q Create a function isArmstrong and use the function to find all the armstrong numbers present between 100~2000.

Q. Create a function to validate a password. The criterias for a valid password are:

1. must contain atleast 1 uppercase character
2. must contain atleast 1 lowercase character
3. must contain atleast 1 special character
4. must contain atleast 1 digit
5. length must be >=8

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Recursion : A function that calls itself is called a recursive function.

In [1]:
```
def Factorial(a):
    if a==0:
        return 1
    else:
        return a*Factorial(a-1)
```

In [4]:
```
Factorial(3)
```

Out[4]: 6

In [6]:
```
# Using recursion find the sum of 1 upto n number.
def Sum1(a):
    if a==0:
        return a
    else:
        return a+Sum1(a-1)
```

In [8]:
```
Sum1(10)
```

Out[8]: 55

In [ ]:

In [ ]:

Lambda Function : These are the one liner anonymous functions, that can be stored in a variable.

Syntax:
var = lambda arg : val if condition else val

In [19]:
```
v = lambda a:'welcome to '+a
print(v('cttc'))
```

```
welcome to cttc
```

In [18]:
```
v = lambda a:'values is '+a
print(v(5))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[18], line 2
      1 v = lambda a:'values is '+a
----> 2 print(v(5))

Cell In[18], line 1, in <lambda>(a)
----> 1 v = lambda a:'values is '+a
      2 print(v(5))

TypeError: can only concatenate str (not "int") to str
```

In [21]:
```
f = lambda a,b:a+b
print(f(4,8))
```

```
12
```

In [20]:
```
f = lambda a,b:a*b
print(f(4,8))
```

```
32
```

In [23]:
```
evenodd = lambda a: 'Even' if a%2==0 else 'Odd'
print(f'{evenodd(6)}')
```

```
Even
```

Q. Wapp to check whether a character is alphabet or not using lambda function.

In [28]:
```
char = lambda a:'Alphabet' if a.isalpha() else 'Not alphabet'
char('5')
```

Out[28]:
```
'Not alphabet'
```

Q. Wapp to check whether an entered character is special character or not.

Built-in python functions :

1. sum(seq) :- takes a sequence of number and calculate its sum.
2. len(seq) :- return the length of any sequence.
3. min(seq) :- return the minimum value from the sequence.
4. max(seq) :- return the maximum value from the sequence.
5. map(function,seq) :- apply the function to every element of the sequence.
6. filter(function,seq) :- return all the element from the sequence that satisfy the function.
7. enumerate(seq) :- return the index, value of the sequence.

In [1]:
```
# sum(seq):
sum({1,346,7,4,567,45})
```

Out[1]:
```
970
```

In [43]:
```
var = (34,6,235,313,5467)
sum(var)
```

Out[43]:
```
6055
```

In [29]:
```
#len()
a = 'eye'
len(a)
```

Out[29]:
```
3
```

In [ ]:

In [2]:
```
# max(seq):
var = (34,6675,235,313,5467)
max(var)
```

Out[2]:
```
6675
```

In [45]:
```
# By using the ascii code of this alphabet, maximum value is
# compared.
# A ~ Z --> 65 to 90
# a ~ z --> 95 to 120
max(['A','a','x','R','g'])
```

Out[45]:
```
'x'
```

In [46]:
```
# min(seq):
min(['A','a','x','R','g'])
```

Out[46]:
```
'A'
```

In [9]:
```
# Find & print the quadral of each element in the list.
list1 = [20,12,42,21,67,30,40]
result = []  # empty list to store the output/result
for i in list1:
    result.append(i**4)
print('The quadral of the list element are:')
print(result)
```

```
The quadral of the list element are:
[160000, 20736, 3111696, 194481, 20151121, 810000, 2560000]
```

In [30]:
```
#Find & print the quadral of each element in the list using map.
# function:
quadral = lambda z : z**3

# sequence:
list1 = [20,12,42,21,67,30,40]

# map(function,sequence)
tuple(map(quadral,list1))
```

Out[30]:
```
(8000, 1728, 74088, 9261, 300763, 27000, 64000)
```

Write a Python program to square and cube every number in a given list of integers using Lambda.

Write a Python program to add two given lists using map and lambda.

```
nums1 = [1, 2, 3]
nums2 = [4, 5, 6]
```

In [8]:
```
add()
```

```
hello this is function
```

In [29]:
```
c='!@#$%^&'
a = input()
if a not in c:
    print('a is not a spac')
else:
    print('sp')
```

```
*
a is not a spac
```

In [ ]: