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 'Python':
    if i=='h':
        break
    print(i)
```

```
P
y
t
```

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 'Python':
    if i=='h':
        continue
    print(i)
```

```
P
y
t
o
n
```

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

In [4]:
```
if 10>5:
   pass
```

In [5]:
```
for i in 'Python':
    if i=='h':
        pass
    print(i)
```

```
P
y
t
h
o
n
```

In [ ]:

In [ ]:

Built-in Functions

In [22]:
```
#print
print("Hello")
```

```
Hello
```

In [23]:
```
#max
max(20,45,67,89,25)
```

Out[23]: 89

In [24]:
```
#min
min(20,45,67,89,25)
```

Out[24]: 20

In [ ]:

In [25]:
```
#sum
sum([20,45,67,89,25])
```

Out[25]: 246

In [26]:
```
#len
len([20,45,67,89,25])
```

Out[26]: 5

In [27]:
```
len("Mango")
```

Out[27]: 5

In [28]:
```
#type
type(10)
```

Out[28]: int

In [31]:
```
#range
list(range(1,11,1))
```

Out[31]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

In [32]:
```
#eval - evaluate
a,b,c = eval(input('Enter values : '))
print('a = ',a)
print('b = ',b)
print('c = ',c)
```

```
a =  10
b =  cat
c =  2.4
```

In [33]:
```
type(a),type(b),type(c)
```

Out[33]: (int, str, float)

In [34]:
```
eval("10+20+30")
```

Out[34]: 60

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 argument
def greet():        #function declaration
    print("Hello")  #function definition
```

In [7]:
```
greet()  #function call
```

```
Hello
```

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

```
Hello
```

In [ ]:
```

```

In [9]:
```
#one argument
def greet(name):
    print('Hello', name)
```

In [10]:
```
greet('Aadil')
```

```
Hello Aadil
```

In [11]:
```
greet('Keshav')
```

```
Hello Keshav
```

In [ ]:
```

```

In [12]:
```
#return - it is a keyword which is use to return the result of a function
return
```

```
  Cell In[12], line 2
    return
    ^
SyntaxError: 'return' outside function
```

In [13]:
```
def greet(name):
    return "Hey " + name
```

In [14]:
```
greet('Sweety')
```

Out[14]:
```
'Hey Sweety'
```

In [ ]:
```

```

In [15]:
```
#two arguments
def add(num1,num2):
    return num1+num2
```

In [16]:
```
add(10,20)
```

Out[16]:
```
30
```

In [18]:
```
add(20,40,50)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[18], line 1
----> 1 add(20,40,50)

TypeError: add() takes 2 positional arguments but 3 were given
```

In [19]:
```
#Multiple Argument
def add(*args):
    return sum(args)
```

In [21]:
```
add(20,30,47,89,46)
```

Out[21]: 232

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [14]:
```
#Multiple arguments
```

In [ ]:

In [ ]:

In [15]:
```
#Even or Odd
```

In [ ]:

In [ ]:

In [ ]: