In [1]:
```
def abc():
    print('Good morning Team')
abc()
```

```
Good morning Team
```

In [2]:
```
def greet():
    print('hello')
    print('Good morning')
greet()
```

```
hello
Good morning
```

In [3]:
```
def greet():
    print('hello')
    print('Good morning')
greet()

def greet():
    print('hello')
    print('Good morning')
greet()

def greet():
    print('hello')
    print('Good morning')
greet()
```

```
hello
Good morning
hello
Good morning
hello
Good morning
```

In [4]:
```
def greet():
    print('Hello Good Morning Boss')
greet()
greet()
greet()
greet()
```

```
Hello Good Morning Boss
Hello Good Morning Boss
Hello Good Morning Boss
Hello Good Morning Boss
```

In [5]:
```
def add(x,y):
    c=x+y
    print(c)
add(5,6)
```

```
11
```

In [6]:
```
def add(x,y,z,m):
    c=x+y+z+m
    print(c)
add(1,2,3,4)
```

```
10
```

In [7]:
```
def greet():
    print('Hello')
    print('Good morning')
greet()
```

```
Hello
Good morning
```

In [8]:
```
def add(x,y):
    c=x+y
    print(c)
add(5,6)
```

```
11
```

In [9]:
```
def greet():
    print('Hello')
    print('Good morning')
greet()

def add(x,y):
    c=x+y
    print(c)
add(5,6)
```

```
Hello
Good morning
11
```

In [10]:
```
def greet():
    print('Hello')
    print('Good morning')
def add(x,y):
    c=x+y
    print(c)

greet()
add(5,6)
```

```
Hello
Good morning
11
```

In [11]:
```
def greet():
    print('Hello')
    print('Good morning')
def add(x,y):
    c=x+y
    print(c)
def sub(x,y):
    d=x-y
    print(d)

greet()
add(5,6)
sub(10,2)
```

```
Hello
Good morning
11
8
```

In [12]:
```
def add_sub(x,y):
    c=x+y
    d=x-y
    print(c)
    print(d)
add_sub(10,5)
```

```
15
5
```

In [13]:
```
def add_sub(x,y):
    c=x+y
    d=x-y
    return c,d
add_sub(10,5)
```

Out[13]: (15, 5)

In [14]:
```
def add_sub(x,y):
    c=x+y
    d=x-y
    return c,d
result=add_sub(5,4)
print(result)
```

```
(9, 1)
```

In [15]:
```
def add_sub(x,y):
    c=x+y
    d=x-y
    return c,d
result1,result2=add_sub(5,4)
print(result1,result2)
```

```
9 1
```

In [16]:
```
def add(x,y):  # x,y=FORMAL ARGUMENT
    c=x+y
    print(c)
add(5,6)      #5,6=ACTUAL ARGUMENT
```

```
11
```

FORMAL ARGUMENT & ACTUAL ARGUMENT

In [17]:
```
def person(name,age):
    print(name)
    print(age)
person('nit',23)
```

```
nit
23
```

In [18]:
```
def person(name,age):
    print(name)
    print(age)
person(23,'nit')
```

```
23
nit
```

KEYWORD ARGUMENT

In [19]:
```
def person(name,age):
    print(name)
    print(age+1)
person(age=23,name='nit')
```

```
nit
24
```

In [20]:
```
def person(name,age,city):
    print(name)
    print(age+1)
    print(city)
person(age=23,name='nit',city='hyd')
```

```
nit
24
hyd
```

Position Arguments

In [21]:
```
def introduction(name,age,city):
    print(f"My name is{name},i am {age},&I am from{city}.")
introduction("Arun",25,"hyd")
```

```
My name isArun,i am 25,&I am fromhyd.
```

In [22]:
```
def calculate_area(length,width):
    return length * width
print (calculate_area(10,5))
```

```
50
```

Mixing Positional and default Arguments

In [23]:
```
def greet(name, greeting="Hello"):
    print(f"{greeting}, {name}!")

# Using positional arguments
greet("Arun")          # Hello, Arun!
greet("Arun", "Hi")    # Hi, Arun!
```

```
Hello, Arun!
Hi, Arun!
```

In [24]:
```
def order(item,quantity,price):
    total=quantity*price
```

```
    print(f"You ordered {quantity} {item}(s). Total cost: ₹{total}")
order("pen",10,20)
```

```
You ordered 10 pen(s). Total cost: ₹200
```

In [25]:
```
def order(item, quantity, price):
    total = quantity * price
    print(f"You ordered {quantity} {item}(s). Total cost: ₹{total}")
order("Book", quantity=3, price=150)
order(item="Pen", quantity=10, price=20)
```

```
You ordered 3 Book(s). Total cost: ₹450
You ordered 10 Pen(s). Total cost: ₹200
```

In [26]:
```
def add(**b):
    return b

add(name='hello',age=45,r=6,f=9)
```

Out[26]:
```
{'name': 'hello', 'age': 45, 'r': 6, 'f': 9}
```

In [27]:
```
def add(*b):
    return b

add(1,2,4,'d',543,)
```

Out[27]:
```
(1, 2, 4, 'd', 543)
```

In [28]:
```
def sum(a,*b):
    c = [a]+[b]
    return c

sum(5,6,7,8,9,10)
```

Out[28]:
```
[5, (6, 7, 8, 9, 10)]
```

In [29]:
```
# if i want to define global variabel inside the function
a = 10

def something():
    global a
    b = 15 # 15 is converted to local when user assigned global a
    a=a+1
    print('in function',b)
    print('gloabl variable', a)
something()
# print('out function',a)
```

```
in function 15
gloabl variable 11
```

In [30]:
```
import keyword
keyword.kwlist
```

Out[30]:
```
['False',
 'None',
 'True',
 'and',
 'as',
 'assert',
 'async',
 'await',
 'break',
 'class',
 'continue',
 'def',
 'del',
 'elif',
 'else',
 'except',
 'finally',
 'for',
 'from',
 'global',
 'if',
 'import',
 'in',
 'is',
 'lambda',
 'nonlocal',
 'not',
 'or',
 'pass',
 'raise',
 'return',
 'try',
 'while',
 'with',
 'yield']
```

In [31]:
```
def simple_gen():
    yield "first"
    yield "second"
    yield "third"

gen=simple_gen()

print(next(gen))
print(next(gen))
print(next(gen))
```

```
first
second
third
```

In [32]:
```
def countdown(n):
    while n > 0:
        yield n
        n -= 1

gen =countdown(5)
print(next(gen))
print(next(gen))
```

```
5
4
```

In [33]:
```
next(countdown(5))
```

Out[33]: 5

In [34]:
```
def fib(n):
    a=0
    b=1
    print(a)
    print(b)
    for i in range(n):
        c=a+b
        a=b
        b=c
        print(c)

fib(4)
```

```
0
1
1
2
3
5
```

dt 02-nov

Variable length argument

In [35]:
```
def sum(a,b):
    c =a+b  # in this case only 2 actual argument allow
    return c
sum(5,6)
```

Out[35]: 11

In [36]:
```
def sum(a, *b): # 1st argument is fixed but for 2nd agument
    print(type(a))
    print(type(b))
sum(5,6,7,8)
```

```
<class 'int'>
<class 'tuple'>
```

In [37]:
```
def sum(a,*b):
    c=a

    for i in b:
        c=c+i
    print(c)
sum(5,6,7,8,9,10,100,200,300)
```

```
645
```

In [38]:
```
def sum(a,*b):
    c=a

    for i in b:
        c=c+i
    print(c)
sum(5,6,7,8)
```

```
26
```

1.Positional argument

2.Keyword argument

3.Default argument

4.Variable length argument(* at last arg)| (args)

5.Keyword + Variable Length(Kwargs)

In [39]:
```
def person():
    person('alex',36,'john',9938)
```

In [40]:
```
def person(name,*data):
    print(name)  # args
    print(data)

person('alex',36,'john',9938)
```

```
alex
(36, 'john', 9938)
```

In [41]:
```
def person(name,*data):
    print('name')
    print(data)
person('alex',age=36,home_place='england',mob=99999)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[41], line 4
      2     print('name')
      3     print(data)
----> 4 person('alex',age=36,home_place='england',mob=99999)

TypeError: person() got an unexpected keyword argument 'age'
```

In [ ]:
```
def person(name,**data):
    print('name')  # kwargs
```

```
    print(data)
person('alex',age=36,home_place='england',mob=99999)
```

Global variable vs local variable

In [1]:
```
a=10
print(a)
```

```
10
```

In [2]:
```
a=10

def something():
    b=15
    print('in function',b)
print('out function',a)
```

```
out function 10
```

In [3]:
```
a=10

def something():
    a=15
print('in function',a)
print('out function',a)
```

```
in function 10
out function 10
```

In [ ]:
```
a=10

def something():
    b=15
    print('in function',b)
something()
print('out function',a)
```

In [4]:
```
a=10   # Global varaible

def something():
    b=55  # local variable
    print('in function',b)
something()

print('out function',a)
```

```
in function 55
out function 10
```

In [5]:
```
# if i want to define global variable inside the function
a=10

def something():
    global a
    b=15 # 15 is converted to local when user assigned global
    print('in function',b)
    print('global variable',a)
something()
```

```
print('out function',a)
```

```
in function 15
global variable 10
out function 10
```

In [6]:
```
x=10 # Global variable

def update_x():
    global x  #Declare that  we are using the global variable x
    x +=5     #Modify the Global Variable

update_x()
print(x)
```

```
15
```

In [7]:
```
x=10 # Global variable

def update_x():
    globals() ['x'] += 5  # Access and Modify the global Variable

update_x()
print(x)
```

```
15
```

In [8]:
```
import keyword
keyword.kwlist
```

Out[8]:
```
['False',
 'None',
 'True',
 'and',
 'as',
 'assert',
 'async',
 'await',
 'break',
 'class',
 'continue',
 'def',
 'del',
 'elif',
 'else',
 'except',
 'finally',
 'for',
 'from',
 'global',
 'if',
 'import',
 'in',
 'is',
 'lambda',
 'nonlocal',
 'not',
 'or',
 'pass',
 'raise',
 'return',
 'try',
 'while',
 'with',
 'yield']
```

In [9]:
```
def count(lst):

    lst = [12,3,4,5]

    lst
```

In [10]:
```
def count(lst):
    even=0
    odd=0
    for i in lst:
        if i%2==0:
            even +=1
        else:
            odd +=1
    return even,odd
lst =[1,2,3,4,8,9,10]
even,odd=count(lst)
print(even)
print(odd)
```

```
4
3
```

In [11]:
```
def count(lst):
    even=0
    odd=0
    for i in lst:
        if i%2==0:
            even +=1
        else:
            odd +=1
    return even,odd
lst =[1,2,3,4,8,9,10,11,12,13,115,65]
even,odd=count(lst)
print("even no:{} and odd no:{}".format(even,odd))
```

```
even no:5 and odd no:7
```

In [12]:
```
def fib(n):
    a=0
    b=1

    print(a)
    print(b)

    for i in range(0,n):
        c=a+b
        a=b
        b=c

        print(c)
fib(10)
```

```
0
1
1
2
3
5
8
13
21
34
55
89
```

03 dec class

In [13]:
```
def wish():
    print('hello')
    print('hi')
wish()
```

```
hello
hi
```

In [14]:
```
# def wish():
#     print('hello')
#     print('hi')
```

```
#      wish()
# wish()
```

In [42]:
```
import sys
sys.getrecursionlimit()
```

Out[42]: 3000

In [43]:
```
import sys
sys.setrecursionlimit(200)
print(sys.getrecursionlimit())
```

```
200
```

In [44]:
```
import sys
sys.setrecursionlimit(200)
print(sys.getrecursionlimit())
```

```
200
```

In [45]:
```
import sys

# Set recursion limit
sys.setrecursionlimit(200)

# Get recursion limit
print(sys.getrecursionlimit())
```

```
200
```

In [46]:
```
#import sys
#sys.setrecursionlimit(30)
#print(sys.getrecursionlimit())
#i=0
#def wish():
   # global i
   # i+=1
   # print('hello',i)
    #wish()
#wish()
```

```
ERROR! Session/line number was not unique in database. History logging moved to n
ew session 366
```

In [46]:
```
def fact(n):
    if n==0:
        return 1
    return n* fact(n-1)
result =fact(5)
result
```

```
Unexpected exception formatting exception. Falling back to standard exception
Unexpected exception formatting exception. Falling back to standard exception
Unexpected exception formatting exception. Falling back to standard exception
```

In [46]:
```
def square(a):
    return a*a
square(5)
```

```
Unexpected exception formatting exception. Falling back to standard exception
Unexpected exception formatting exception. Falling back to standard exception
Unexpected exception formatting exception. Falling back to standard exception
ERROR! Session/line number was not unique in database. History logging moved to n
ew session 367
```

In [15]:
```
def square(a):
    return a*a
result=square(5)
print(result)
```

```
25
```

In [16]:
```
f=lambda a:a*a
result=f(5)
result
```

Out[16]: 25

In [17]:
```
f=lambda a,b:a+b
f1=lambda a,b:a-b
result=f(1,4)
result1=f1(4,1)
print(result)
print(result1)
```

```
5
3
```

3-DEC

fibonacci Sequence

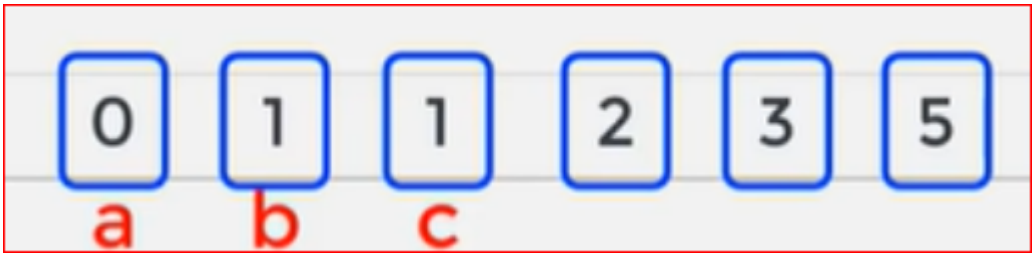
In [18]:
```
def fib(n):
    print(0)
    print(1)
fib(0)
```

```
0
1
```

In [19]:
```
def fib(n):
    print(0)
    print(1)
    print(1)
    print(2)
```

```
    print(3)
fib(0)
```

```
0
1
1
2
3
```



In [20]:
```
def fib(n):
    a=0
    b=1

    print(a)
    print(b)
    for i in range(0,n):
        c=a+b
        a=b
        b=c
        print(c)
fib(10)
```

```
0
1
1
2
3
5
8
13
21
34
55
89
```

In [21]:
```
def fib(n):
    a,b=0,1
    if n==1:
        print(a)
    else:
        print(a)
        print(b)
        for i in range(2,n):
            c=a+b
            a=b
            b=c
            print(c)
fib(4)
```

```
0
1
1
2
```

Factorial of a Number in python

In [22]:
```
def fact(n):
    f=1
    for i in range(1,n+1):
        f=f*i

    return f
x=5
result=fact(x)
print(result)
```

```
120
```

Factorial using Recursion

In [23]:
```
def fact(n):
    if n==0:
        return 1
    return n * fact(n-1)
result=fact(5)
print(result)
```

```
120
```

In [ ]:
```
def fact(n):
    if n == 0:
        return 1
    return n * fact(n-1)

print(fact(5))
```

In [24]:
```
def square(a):
    return a*a
result=square(5)
print(result)
```

```
25
```

lambda function

In [25]:
```
f=lambda a : a*a
f(5)
```

Out[25]: 25

In [26]:
```
f=lambda b:b*b
result=f(6)
print(result)
```

```
36
```

In [27]:
```
add=lambda a,b:a+b
add(1,2)
```

Out[27]:
```
3
```

In [28]:
```
add=lambda a,b:a+b
add(10,20)
```

Out[28]:
```
30
```

In [29]:
```
f=lambda a: a*a
result=f(5)
result
```

Out[29]:
```
25
```

In [30]:
```
f=lambda a,b:a+b
f1=lambda a,b:a-b
result=f(1,4)
result1=f1(4,1)
print(result)
print(result1)
```

```
5
3
```

In [31]:
```
f=lambda a,b:a+b
f1=lambda a,b:a-b
f2=lambda a,b:a*b
result=f(1,4)
result1=f1(4,1)
result2=f2(1,4)
print(result)
print(result1)
print(result2)
```

```
5
3
4
```

filter()

map()

reduce()

In [32]:
```
def is_even(n):
    return n%2==0
nums=[1,2,3,4,5,6,7,8,9]
evens=list(filter(is_even,nums))
print(evens)
```

```
[2, 4, 6, 8]
```

In [33]:
```
def is_odd(n):
    return n%2!=0
nums=[1,2,3,4,5,6,7,8,9]
odd=list(filter(is_odd,nums))
print(odd)
```

```
[1, 3, 5, 7, 9]
```

In [34]:
```
nums=[1,2,3,4,5,6,7,8,9]
evens=list(filter(lambda n:n%2==0,nums))
print(evens)
```

```
[2, 4, 6, 8]
```

In [35]:
```
nums=[1,2,3,4,5,6,7,8,9]
odd=list(filter(lambda n:n%2!=0,nums))
print(odd)
```

```
[1, 3, 5, 7, 9]
```

In [36]:
```
nums=[1,2,3,4,5,6,7,8,9]
evens=list(filter(lambda n:n%2==0,nums))
odd=list(filter(lambda n:n%2!=0,nums))
print(evens)
print(odd)
```

```
[2, 4, 6, 8]
[1, 3, 5, 7, 9]
```

In [37]:
```
nums=[1,2,3,4,5,6,7,8,9]
evens=list(filter(is_even,nums))
double=list(map(lambda n:n*2,evens))
print(evens)
print(double)
```

```
[2, 4, 6, 8]
[4, 8, 12, 16]
```

In [38]:
```
nums=[1,2,3,4,5,6,7,8,9]
evens=list(filter(is_even,nums))

double=list(map(lambda n:n*2,evens))
double1=list(map(lambda n:n+2,evens))
double2=list(map(lambda n:n-2,evens))
print(evens)
print(double)
print(double1)
print(double2)
```

```
[2, 4, 6, 8]
[4, 8, 12, 16]
[4, 6, 8, 10]
[0, 2, 4, 6]
```

In [39]:
```
from functools import reduce
nums=[1,2,3,4,5,6,7,8,9]
evens=list(filter(is_even,nums))
double=list(map(lambda n:n*2,evens))
sums=(reduce(lambda a,b: a+b,double))
print(evens)
```

```
print(double)
print(sums)
```

```
[2, 4, 6, 8]
[4, 8, 12, 16]
40
```

|#Python decorators

In [41]:
```
def div(a,b):
    print(a/b)
div(4,2)
```

```
2.0
```

In [42]:
```
# but what if we pass the value 2,4 in state of 4,2
```

In [43]:
```
def div(a,b):
    print(a/b)
div(2,4)
```

```
0.5
```

In [44]:
```
#i wannts to apply logic hear so the out put will be same
```

In [45]:
```
def div(a,b):

    if a<b:
        a,b =b,a
    print(a/b)
div(2,4)
```

```
2.0
```

In [47]:
```
def div(a,b):
    print(a/b)
def div_decorator(func):
    def inner(a,b):
        if a<b:
            a,b=b,a
        return func(a,b)
    return inner
div =div_decorator(div)
div(2,4)
```

```
2.0
```

In [49]:
```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper
@my_decorator
def say_hello():
    print("Hello!")
say_hello()
```

```
Something is happening before the function is called.
Hello!
Something is happening after the function is called.
```

In [50]:
```
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        #func()
        print("Something is happening after the function is called.")
    return wrapper
@my_decorator
def say_hello():
    print("Hello!")
say_hello()
```

```
Something is happening before the function is called.
Something is happening after the function is called.
```

In [ ]: