In [1]:
```
def greet():
    print("Good Morning")
greet()
```

```
Good Morning
```

In [2]:
```
def add(x,y):
    c=x+y
    print(c)
```

In [9]:
```
def add(x,y): #x,y formal argument
    c=x+y
    print(c)
add(5,6) #actual argument
```

```
11
```

In [4]:
```
def add(x,y):
    c=x+y
    print(c)
add(5,6)
add(5,8)
add(5,9)
```

```
11
13
14
```

In [8]:
```
def add(x,y):
    c=x+y
    return c
add(5,6)
add(5,8)
add(5,9)
```

Out[8]:
```
14
```

Understanding Return vs Print in Python Functions: The `return` statement doesn't actually print anything to the console by itself. Instead, it:

1. Ends the function execution
2. Sends a value back to the caller
3. Only returns one time per function call

When you run this code in a Jupyter notebook, you only see the result of the last function call. If we use print(), we would see all the values because `print` explicitly sends output to the console each time it's called.

In [10]:
```
def greet():
    print("Good Morning")
greet()

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

```
Good Morning
11
```

In [11]:
```python
def greet():
    print("Good Morning")
def add(x,y):
    c=x+y
    return c
greet()
add(11,13)
```

```
Good Morning
```

Out[11]:
```
24
```

In [14]:
```python
def greet():
    print("Good Morning")

def add(x,y):
    c=x+y
    print(c) #if we use return here then we get 2 outputs only

def sub(x,y):
    z=x-y
    return z #better to use print() statement, while we call multiple functions

greet()
add(11,13)
sub(10,4)
```

```
Good Morning
24
```

Out[14]:
```
6
```

In [17]:
```python
def add_sub(x,y):
    c= x+y
    d= x-y
    return c, d

add_sub(4,5)
```

Out[17]:
```
(9, -1)
```

In [18]:
```python
def add_sub(x,y):
    c= x+y
    d= x-y
    return c, d

result = add_sub(4,5)
print(type(result)) #type is tuple bcz we are using 1 variable to store the outp
```

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

In [22]:
```python
def add_sub(x,y):
    c= x+y
    d= x-y
    return c, d

result, result1 = add_sub(4,5)

print(result)
print(result1)

print(type(result))
print(type(result1)) #type is int bcz 2 variables stores 2 output values
```

```
9
-1
<class 'int'>
<class 'int'>
```

Actual Argument and It's Types

1. Positional Arguments

In [23]:
```
def person(name,age):
    print(name)
    print(age)
person('Gopal',30)
```

```
Gopal
30
```

In [25]:
```
def person(name,age):
    print(name)
    print(age)
person(30,'Gopal') #doesn't throw an error as it takes the position
```

```
30
Gopal
```

In [26]:
```
def person(name,age):
    print(name)
    print(age)
person('Gopal')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[26], line 4
      2     print(name)
      3     print(age)
----> 4 person('Gopal')

TypeError: person() missing 1 required positional argument: 'age'
```

In [27]:
```
def person(name,age):
    print(name)
    print(age)
person(30)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[27], line 4
      2     print(name)
      3     print(age)
----> 4 person(30)

TypeError: person() missing 1 required positional argument: 'age'
```

- The error is displayed as missing 1 required positional argument: 'age' in both the scenarios, but it doesn't check what type of argument is missing.
- When we pass only 1 argument, Whatever the data is provided it will be assigned to the 1st argument.

Keyword Argument

In [28]:
```
def person(name,age):
    print(name)
    print(age)
person(age=30, name='Gopal') #we need to define the arguments with the keywords
```

```
Gopal
30
```

In [29]:
```
def person(name,age):
    print(name)
    print(age)
person(age=30, name='Gopal', '20L') #we need to pass only 2 keyword values
```

```
  Cell In[29], line 4
    person(age=30, name='Gopal', '10L') #we need to pass only 2 keyword values
                                      ^
SyntaxError: positional argument follows keyword argument
```

In [34]:
```
def person(name,age, salary):
    print("name:", name)
    print("Age:",age)
    print("Salary:",salary)
person(age=30, name='Gopal', salary='20L')
```

```
name: Gopal
Age: 30
Salary: 20L
```

Default Argument

In [35]:
```
def person(name,age=18): #defining the default value for age
    print(name)
    print(age)
person('Gopal')
```

```
Gopal
18
```

In [37]:
```
def person(name,age=18): #defining the default value for age
    print(name)
    print(age)
person('Gopal', 30) #when we pass the value, it over rides the defalut value
```

```
Gopal
30
```

Variable Length Argument (Arg)

In [38]:
```python
def add(x,*y): #when we mention * before an argument it becomes a tuple
    c=x+y
    print(c)
add(2,3,4,5) #x refers to 2, but *y refers to all the values
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[38], line 4
      2     c=x+y
      3     print(c)
----> 4 add(2,3,4,5)

Cell In[38], line 2, in add(x, *y)
      1 def add(x,*y): #when we mention * before an argument it becomes a tuple
----> 2     c=x+y
      3     print(c)

TypeError: unsupported operand type(s) for +: 'int' and 'tuple'
```

In [39]:
```python
def sum(a, *b): # 1st argument is fixed but for 2nd argument
    #c = a+b
    print(type(a))
    print(type(b))

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

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

In [40]:
```python
def sum(a, *b): # 1st argument is fixed & we fetch each value from the tuple & w
    c = a
    for i in b:
        c = c + i
    print(c)

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

```
26
```

In [41]:
```python
def sum(a, *b): # 1st argument is fixed & we fetch each value from the tuple & w
    c = a
    for i in b:
        c = c + i
    print(c)

sum(5,6,7,8,0,10, 100, 1000, -45)
```

```
1091
```

Keyword Variable Length Argument (Kwarg)

In [42]:
```python
def person(name,*data):
    print('name')
    print(data)

person('ALEX', age = 36, home_place ='southcity', mob =987767)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[42], line 5
      2     print('name')
      3     print(data)
----> 5 person('ALEX', age = 36, home_place ='southcity', mob =987767)

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

In [43]:
```
def person(name,**data): #for kwarg we use **
    print('name')
    print(data)

person('ALEX', age = 36, home_place ='southcity', mob =987767)
```

```
name
{'age': 36, 'home_place': 'southcity', 'mob': 987767}
```

In [44]:
```
def person(name,**data): #for kwarg we use **
    print(name)
    print(data)

person('ALEX', age = 36, home_place ='southcity', mob =987767)
```

```
ALEX
{'age': 36, 'home_place': 'southcity', 'mob': 987767}
```

In [45]:
```
def person(name,**data):
    print('name')
    print(data)

person('ALEX', age = 36, home_place ='southcity', mob =987767, city = ['usa', 'u
```

```
name
{'age': 36, 'home_place': 'southcity', 'mob': 987767, 'city': ['usa', 'uk']}
```

In [ ]: