1. **Required Arguments**

Arguments that must be passed to a function in the correct positional order.

2. **Keyword Arguments**

Arguments passed by explicitly specifying the parameter name, allowing order flexibility.

3. **Default Arguments**

Arguments that assume a default value if no value is provided during the function call.

4. **Arbitrary Arguments (`*args`)**

Used to pass a variable number of **positional** arguments to a function.

5. **Arbitrary Keyword Arguments (`**kwargs`)**

Used to pass a variable number of **keyword** arguments to a function.

Required Arguments

Required arguments are parameters that must be passed to the function in the correct positional order.If not provided,Python will raise a TypeError.

In [1]:
```
#required arguements
def sum(x,y):
    return x+y
```

In [37]:
```
print(sum(10,20))
#print(sum())# this is error this empty not working only upper input works
```

```
30
```

Default Arguments

Default arguements are parameters that assume a default value if a value is not provided in the function call.They are specified by assigning a value to the parameter in the function defination.

In [38]:
```
def sum(x=0,y=0):
    return x+y
```

In [39]:
```
sum()
sum(234,34)
```

Out[39]: 268

In [40]:
```
#USE CASE
def greetings(fname,lname='',occup=''):

    print(f'Hi! This side {fname} {lname}.My occup is {occup}')
```

In [41]:
```
greetings('vipul','pandey','data science')
```

```
Hi! This side vipul pandey.My occup is data science
```

In [42]:
```
greetings('vipul','data science') #problem answer in keyword arguments
```

```
Hi! This side vipul data science.My occup is
```

Keyword Arguments

keyword arguement are passed to a functions by explicity naming the parameters and assigning them values in the function call. This allows you to specify values in the function call.This allows you to specify values for only some parameters while relying on default values of others , and to pass arguments in a different order.

In [43]:
```
def call(x=10,y=20):
    # return x,y
    print('x=',x)
    print('y=',y)
```

In [44]:
```
call(y=30)
```

```
x= 10
y= 30
```

In [45]:
```
#reordering
call(y=0,x=100)
```

```
x= 100
y= 0
```

In [46]:
```
greetings(fname='vipul',occup='data science')
```

```
Hi! This side vipul .My occup is data science
```

Variable-length arguments

Variable length arguments allow a function to accept an arbitary numbers of arguments.They have defined using *args for naon-keyword arguments and **kwargs for keyword arguments

*args

*args allows a function to accept any numbers of positional arguements.The arguements are passed as tuple.(iterable)

In [47]:
```
#args
# *
def funct1(*args):
    print(args)
    print(type(args))
```

In [48]:
```
funct1(13,25,5416,656,6161,61,51,15416,5162)
```

```
(13, 25, 5416, 656, 6161, 61, 51, 15416, 5162)
<class 'tuple'>
```

In [49]:
```
#mean
def mean(*args):
    sum=0
    for i in args:
        sum+=i
    #print(sum)
    return sum/len(args)
```

In [50]:
```
mean(12,6265,622,65692,5,25,152,626,25562,895)
```

Out[50]:
```
9985.6
```

In [51]:
```
mean(10,20,30,40,50,60,70)
```

Out[51]:
```
40.0
```

**kwargs

**kwargs allows a function to accept numbers of keyword arguements.The arguements are passed a dictionary

In [52]:
```
def funct2(**kwargs):
```

```
    print(kwargs)
    print(type(kwargs))
```

In [53]:
```
funct2(name='vipul',desig='Data scientist')
```

```
{'name': 'vipul', 'desig': 'Data scientist'}
<class 'dict'>
```

In [54]:
```
def info(**kwargs):
    # print()
    global var1
    var1 = kwargs
    for i,j in kwargs.items():
        print(i,'\t',j)
```

In [55]:
```
info(name='vipul',age=19,crs='DS')
```

```
name	 vipul
age	 19
crs	 DS
```

In [56]:
```
var1
```

Out[56]:
```
{'name': 'vipul', 'age': 19, 'crs': 'DS'}
```

In [57]:
```
var1.items()
```

Out[57]:
```
dict_items([('name', 'vipul'), ('age', 19), ('crs', 'DS')])
```

In [ ]:
```

```