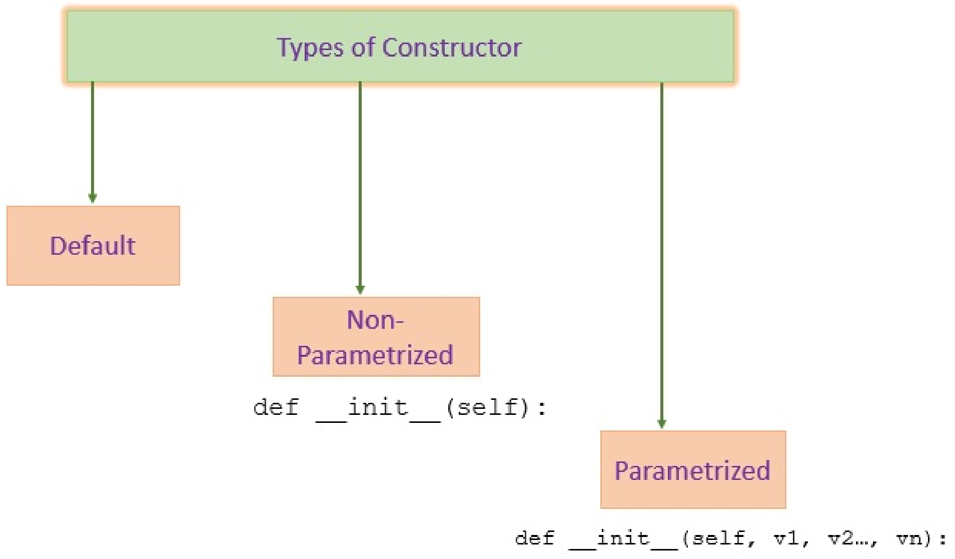
Types of constructor

- Default Constructor
- Non-parametrized Constructor
- Parametrized Constructor



In [3]:
```
#if we forgot to implement or to add constructor, python add constructor on its own as default constructor. It
#example

class car:

    def show(self):
        print("Car can run 1000km")

car1=car()
```

In [4]:
```
car1.show()
```

```
Car can run 1000km
```

Non parametrized

A constructor without any arguments is called a non-parameterized constructor. This type of constructor is used to initialize each object with default values. This constructor doesn't accept the arguments during object creation. Instead, it initializes every object with the same set of values.

In [5]:
```
class Company:

    # no-argument constructor
    def __init__(self):
        self.name = "PYnative"
        self.address = "ABC Street"

    # a method for printing data members
    def show(self):
        print('Name:', self.name, 'Address:', self.address)

# creating object of the class
cmp = Company()

# calling the instance method using the object
cmp.show()
```

```
Name: PYnative Address: ABC Street
```

In [6]:
```
#last is parametrized constructor (normal constructor)
```

A constructor with defined parameters or arguments is called a parameterized constructor. We can pass different values to each object at the time of creation using a parameterized constructor.

In [7]:
```
class Employee:
    # parameterized constructor
    def __init__(self, name, age, salary):
        self.name = name
        self.age = age
        self.salary = salary

    # display object
    def show(self):
        print(self.name, self.age, self.salary)

# creating object of the Employee class
emma = Employee('Emma', 23, 7500)
emma.show()

kelly = Employee('Kelly', 25, 8500)
kelly.show()
```

```
Emma 23 7500
Kelly 25 8500
```

Constructor overloading

If we define multiple constructors then, the interpreter will considers only the last constructor and throws an error if the sequence of the arguments doesn’t match as per the last constructor.

In [17]:
```
#lets see example

class fruit:

    def __init__(self,color):
        self.color=color


    def __init__(self,smell,shape,color):
        self.smell=smell
        self.shape=shape
        self.color=color

    def show(self):
        print("Fruit is so sweet and its shape is",self.shape)

apple=fruit("red")
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[17], line 17
     14     def show(self):
     15         print("Fruit is so sweet and its shape is",self.shape)
---> 17 apple=fruit("red")

TypeError: fruit.__init__() missing 2 required positional arguments: 'shape' and 'color'
```

In [2]:
```
#see its working with directly with second constructor and it thought red as taste and program asking for shape
```

In [18]:
```
banana=fruit('sweet','oval','red')
```

In [19]:
```
print(banana.color)
```

```
red
```

In [20]:
```
banana.show()
```

```
Fruit is so sweet and its shape is oval
```

Constructor Chaining

Constructor chaining is the process of calling one constructor from another constructor.

In [2]:
```
#example

class College:
    def __init__(self,name):
        print("Hello from college class")
        self.name=name

class department(College):
    def __init__(self,departmenthead,name):
        super().__init__(name)
        print("hello from department class")
        self.departmenthaed=departmenthead

#object of department
physics=department("Ram","TexasUNV")
```

```
Hello from college class
hello from department class
```

In [3]:
```
print(physics.name)
```

```
TexasUNV
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js