(7.10) Meta Classes

```
Class of Class is known Meta Class

Meta means data about data
```

Intoduction

Type is Meta Class of Every Class in Python

```
every class in python is just an object of Type
```

In [12]:
```
print(type(int))
print(type(float))
print(type(list))
```
```
<class 'type'>
<class 'type'>
<class 'type'>
```

In [13]:
```
class MyClass:
    pass

a = MyClass()
print(a)
print(type(MyClass))
```
```
<__main__.MyClass object at 0x000001EA332DB760>
<class 'type'>
```

In [14]:
```
issubclass(MyClass, type)
```
Out[14]:
```
False
```

Object is super class or Base Class or Parent Class of all classes in Python

In [15]:
```
issubclass(MyClass, object)
```
Out[15]:
```
True
```

Meta class defines attributes and behaviours of a Class object

```
() -> evalute an expression

func() -> to call a function

() -> to create tuple (iterable)
```

In [17]:
```
(4+6-2)
```
Out[17]:
```
8
```

```
In [18]:  1, 2, 3, 4, 5
Out[18]: (1, 2, 3, 4, 5)
```

Attributes and behaviour

```
In [32]:
class Y:
    def hi(self):
        print("hi world!")

class X(Y):
    def __new__(cls_name, *args, **kwargs):
        """
        bases -> tuple which contains base classes (parent classes)
        attrs -> {} dictionary which will hold all attributes of class
        """
        print(cls_name)
        print(args)
        print(kwargs)
        self = super(cls_name, X).__new__(cls_name)
        self.author = "Sachin Yadav" # we providing some extra features to each object
        return self
    def __init__(self, *args, **kwargs):
        self.data= args
        self.__dict__.update(kwargs)
    def hello(self):
        print("hello world")
```

```
In [33]:
p = X(304504, name='sachin', age='23')
```

```
<class '__main__.X'>
(304504,)
{'name': 'sachin', 'age': '23'}
```

```
In [34]:
print(p.name)
print(p.age)
print(p.author)
```

```
sachin
23
Sachin Yadav
```

Defining attributes and behaviour of class using type class

type(cls_name, bases, attrs)

```
bases -> tuple which contains base classes (parent classes)
attrs -> {} dictionary which will hold all attributes of class
```

1:

```
In [24]:
class A:
    pass
class C:
    pass
```

```
class B(A, C):
    name = "It's B class" # attributes / data members
    def hello(self): # behaviours / methos / members function
        print("Hello I am an Instace of class B")


print(type(B))
print(type(B()))
a = B()
print(a.name)
a.hello()
```

```
<class 'type'>
<class '__main__.B'>
It's B class
Hello I am an Instace of class B
```

In [25]:

```
B = type('B', (A, C), {'name': "It's B class",
         'hello': lambda self: print("Hello I am an Instance Method of Class B") }) # type

print(type(B))
print(type(B()))
a = B()
print(a.name)
a.hello()
```

```
<class 'type'>
<class '__main__.B'>
It's B class
Hello I am an Instance Method of Class B
```

In [26]:

```
def hello(self):
    print("Hello I am an Instacne Method of class B")

B = type('B', (A, C), {'name': "It's B class",
         'hello': hello }) # type(name, bases, attrs)

print(type(B))
print(type(B()))
a = B()
print(a.name)
a.hello()
```

```
<class 'type'>
<class '__main__.B'>
It's B class
Hello I am an Instacne Method of class B
```

In [27]:

```
q = B()
print(q.name)
q.hello()
```

```
It's B class
Hello I am an Instacne Method of class B
```

2.

In [35]:

```
def init(self, name):
    self.name = name
def get_name(self):
    return self.name
```

```
def set_name(self, new_name):
    self.name = new_name
A = type('A', (), {
    '__init__': init,
    '__str__': lambda self: self.name.title(),
    'get_name': get_name,
    'set_name': set_name
})
a = A('Sachin Yadav')
print(type(A))
print(a)
print(a.get_name())
a.set_name('Rajat Goyal')
print(a)
```

```
<class 'type'>
Sachin Yadav
Sachin Yadav
Rajat Goyal
```

Custom Meta Class

1.

In [36]:

```
class A:
    pass

print(issubclass(A, type))
print(issubclass(A, object))
```

```
False
True
```

2.

In [37]:

```
class MyMeta(type):
    pass
print(issubclass(MyMeta, type))
print(issubclass(MyMeta, object))
```

```
True
True
```

In [38]:

```
print(dir(A))
```

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__
ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '__le_
_', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr_
_', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__']
```

3.

In [39]:

```
class MyMeta(type):
    def __new__(cls, cls_name, bases, attrs):
        print(repr(cls_name))
        print(bases)
        print(attrs)
        return type(cls_name, bases, attrs)
```

In [40]:
```
class B:
    pass
```

In [41]:
```
class A(B, metaclass=MyMeta): # bydefault metaclass = type
    name = "Sachin Yadav"
    def hello(self):
        print("Hello World! This is how class are created")
```

```
'A'
(<class '__main__.B'>,)
{'__module__': '__main__', '__qualname__': 'A', 'name': 'Sachin Yadav', 'hello': <function
A.hello at 0x000001EA33383CA0>}
```

4.

In [42]:
```
class Meta(type):
    def __new__(cls, cls_name, bases, attrs):
        print("Creating a Class with attrs : ", attrs)
        return type(cls_name, bases, attrs)
```

In [43]:
```
class A(metaclass=Meta):
    name = 'sachin'
    def hi(self):
        print('hello world')
```

```
Creating a Class with attrs :  {'__module__': '__main__', '__qualname__': 'A', 'name': 'sa
chin', 'hi': <function A.hi at 0x000001EA333BDAF0>}
```

5.

In [44]:
```
class Meta(type):
    def __new__(cls, cls_name, bases, attrs):
        for key, value in attrs.items():
            if not key.startswith('__'):
                if key.isupper():
                    return type(cls_name, bases, attrs)
                else:
                    raise Exception("Can not Create a Class just beacuse some attributes a
```

In [45]:
```
class A(metaclass=Meta):
    name = 'sachin'
    def hello(self):
        print('hi')
```

```
---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_3740/3862264955.py in <module>
----> 1 class A(metaclass=Meta):
      2     name = 'sachin'
      3     def hello(self):
      4         print('hi')

C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_3740/1143832401.py in __new__(cls, cls_nam
e, bases, attrs)
      6                     return type(cls_name, bases, attrs)
      7                        else:
----> 8                            raise Exception("Can not Create a Class just beacuse some attr
ibutes are not uppercased")

Exception: Can not Create a Class just beacuse some attributes are not uppercased
```

In [48]:

```
class A(metaclass=Meta):
    NAME = 'sachin'
    def HELLO(self):
        print("Hi")

a = A()
a.HELLO()
```

```
Hi
```

6.

In [52]:

```
class Meta(type):
    def __new__(cls, cls_name, bases, attrs):
        new_attrs = {}
        for key, value in attrs.items():
            if key.startswith('__'):
                new_attrs[key] = value
            else:
                new_attrs[key.upper()] = value
        return type(cls_name, bases, new_attrs)
```

In [53]:

```
class A(metaclass=Meta):
    name = 'sachin'
    def hello(self):
        print('hi')
```

In [54]:

```
a = A()
print(a.NAME)
a.HELLO()
```

```
sachin
hi
```

In [55]:

```
a.hello
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_3740/4010090673.py in <module>
----> 1 a.hello

AttributeError: 'A' object has no attribute 'hello'
```

In [56]:

```
a.name
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_3740/362388590.py in <module>
----> 1 a.name

AttributeError: 'A' object has no attribute 'name'
```

```
In [ ]:
```