(7.8) Properties, Slots, Monkey Patching

Monkey Patching / Dynamic Binding

We can add methods to class or instance after initiating them (Dynamically)

In [68]:
```
class A:
    pass
```

In [69]:
```
setattr(A, 'msg', 'hello world') # setter property
getattr(A, 'msg') # getter property
```

Out[69]:
```
'hello world'
```

In [74]:
```
a = A()
print(a.msg)
print(getattr(a,'msg'))
print(a.__dict__)

setattr(a, 'name', 'Pankaj')
print(a.name)
a.age = 20
print(a.age)
print(a.__dict__)
```

```
hello world
hello world
{}
Pankaj
20
{'name': 'Pankaj', 'age': 20}
```

In [75]:
```
setattr(A, 'hello', lambda self: print("hello World! from ", self.name))
```

In [76]:
```
a.hello()
```

```
hello World! from  Pankaj
```

Properties

Variables as functions

Methods as variables

are known as Properties

In [46]:
```
help(property)
```

```
Help on class property in module builtins:

class property(object)
```

```
|  property(fget=None, fset=None, fdel=None, doc=None)
|
|  Property attribute.
|
|    fget
|      function to be used for getting an attribute value
|    fset
|      function to be used for setting an attribute value
|    fdel
|      function to be used for del'ing an attribute
|    doc
|      docstring
|
|  Typical use is to define a managed attribute x:
|
|  class C(object):
|      def getx(self): return self._x
|      def setx(self, value): self._x = value
|      def delx(self): del self._x
|      x = property(getx, setx, delx, "I'm the 'x' property.")
|
|  Decorators make defining new properties or modifying existing ones easy:
|
|  class C(object):
|      @property
|      def x(self):
|          "I am the 'x' property."
|          return self._x
|      @x.setter
|      def x(self, value):
|          self._x = value
|      @x.deleter
|      def x(self):
|          del self._x
|
|  Methods defined here:
|
|  __delete__(self, instance, /)
|      Delete an attribute of instance.
|
|  __get__(self, instance, owner, /)
|      Return an attribute of instance, which is of type owner.
|
|  __getattribute__(self, name, /)
|      Return getattr(self, name).
|
|  __init__(self, /, *args, **kwargs)
|      Initialize self.  See help(type(self)) for accurate signature.
|
|  __set__(self, instance, value, /)
|      Set an attribute of instance to value.
|
|  deleter(...)
|      Descriptor to change the deleter on a property.
|
|  getter(...)
|      Descriptor to change the getter on a property.
|
|  setter(...)
|      Descriptor to change the setter on a property.
|
|  ----------------------------------------------------------------------
|  Static methods defined here:
|
|  __new__(*args, **kwargs) from builtins.type
|      Create and return a new object.  See help(type) for accurate signature.
```

```
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __isabstractmethod__
 |
 |  fdel
 |
 |  fget
 |
 |  fset
```

property(fget=None, fset=None, fdel=None, doc=None)

1:

In [39]:
```
class A:
    def __init__(self, x):
        self.__x = x
    def __str__(self):
        return f"A({self.__x})"
    def get_x(self):
        return self.__x
    def set_x(self, value):
        self.__x = value
    def del_x(self):
        del self.__x

    x = property(get_x, set_x, del_x, "X is just a property")
```

In [38]:
```
a = A(100)
print(a.x)

a.x = 200
print(a.x)

del a.x
print(a.x)
```

```
100
200
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_10708/3439183687.py in <module>
      5 print(a.x)
      6
----> 7 print(a.x.doc)
      8
      9 del a.x

AttributeError: 'int' object has no attribute 'doc'
```

2:

In [19]:
```
class A:
    def __init__(self, x):
        self.__x = x
    def __str__(self):
        return f"A({self.__X})"
    @property
    def x(self):
```

```
        pass
    @x.getter
    def x(self):
        return self.__x
    @x.setter
    def x(self, value):
        self.__x = value
    @x.deleter
    def x(self):
        del self.__x
```

In [47]:

```
a = A(100)
print(a.x)

a.x = 200
print(a.x)
```

```
100
200
```

3:

In [49]:

```
class Product:
    def __init__(self, name, price):
        self.__margin = 20
        self.__price = price + (price*self.__margin)/100
        self.name = name
    def __str__(self):
        return f"\nName: {self.name}\nPrice: {self.__price}\n"
    @property
    def price(self):
        pass
    @price.getter
    def price(self):
        return self.__price
    @price.setter
    def price(self, new_price):
        self.__price = new_price + (new_price*self.__margin)/100
    @price.deleter
    def price(self):
        self.__price = 0
```

In [51]:

```
iphone = Product('Iphone 8', 60000)
print(iphone)
print(iphone.price) # getter

iphone.price = 65000 # setter attribute
print(iphone.price) # getter

del iphone.price
print(iphone.price)
```

```
Name: Iphone 8
Price: 72000.0

72000.0
78000.0
0
```

Slots

We use slots to save memory space or limit dynamic binding

We want to off dynamic binding of unnecessory properties

we can force developers to use a fix set of attribues

Case 1: Without Slots

Here We can add properties as we want memory space not limited

1. Demo

In [53]:
```
class A:
    pass
```

In [54]:
```
a = A()
print(a.__dict__)
a.name = 'Pankaj'
a.age = 20
print(a.__dict__)
```

```
{}
{'name': 'Pankaj', 'age': 20}
```

In [55]:
```
print(a.__sizeof__())
print(a.__dict__.__sizeof__())
```

```
32
88
```

In [56]:
```
for i in range(1, 20):
    setattr(a, f'var-{i}', f'value-{i**2}')
```

In [57]:
```
print(a.__sizeof__())
print(a.__dict__.__sizeof__())
```

```
32
216
```

In [58]:
```
a.__dict__
```

Out[58]:
```
{'name': 'Pankaj',
 'age': 20,
 'var-1': 'value-1',
 'var-2': 'value-4',
 'var-3': 'value-9',
 'var-4': 'value-16',
 'var-5': 'value-25',
 'var-6': 'value-36',
 'var-7': 'value-49',
 'var-8': 'value-64',
 'var-9': 'value-81',
 'var-10': 'value-100',
 'var-11': 'value-121',
```

```
'var-12': 'value-144',
'var-13': 'value-169',
'var-14': 'value-196',
'var-15': 'value-225',
'var-16': 'value-256',
'var-17': 'value-289',
'var-18': 'value-324',
'var-19': 'value-361'}
```

2. Class Person

name, age, gender, country

In [59]:
```
class Person:
    def __init__(self, name, age, gender, country):
        self.name = name
        self.age = age
        self.gender = gender
        self.country = country


p1 = Person('Pankaj', 20, 'Male', 'India')
print(p1.name)
print(p1.age)
print(p1.__dict__)
```

```
Pankaj
20
{'name': 'Pankaj', 'age': 20, 'gender': 'Male', 'country': 'India'}
```

In [60]:
```
p1.state = 'Bihar' # it should not be allowed
# you want to off dynamic binding of unnecessory properties

# we can save memory
# we can force developers to use a fix set of attribues
```

In [61]:
```
p1.__dict__
```

Out[61]:
```
{'name': 'Pankaj',
 'age': 20,
 'gender': 'Male',
 'country': 'India',
 'state': 'Bihar'}
```

Case 2: Using Slots

In [64]:
```
class Person:
    __slots__ = [ 'name', 'age', 'gender', 'country']
    # slots will disable dynamic dictonary of an object
    def __init__(self, name, age, gender, country):
        self.name = name
        self.age = age
        self.gender = gender
        self.country = country


p1 = Person('Pankaj', 20, 'Male', 'India')
print(p1.name)
print(p1.age)
```

```
Pankaj
20
```

In [65]:
```
p1.__dict__
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_10708/3223873474.py in <module>
----> 1 p1.__dict__

AttributeError: 'Person' object has no attribute '__dict__'
```

In [66]:
```
p1.state = "Bihar"
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
C:\Users\PANKAJ~1\AppData\Local\Temp/ipykernel_10708/3759678514.py in <module>
----> 1 p1.state = "Bihar"

AttributeError: 'Person' object has no attribute 'state'
```

In [67]:
```
p1.__sizeof__() # 48
```

Out[67]:
```
48
```

In [6]:
```

```