Assignment_19

1. Make a class called Thing with no contents and print it. Then, create an object called example from this class and also print it. Are the printed values the same or different?

In [1]:

```
class Thing:
    pass
```

In [2]:

```
print(Thing)
```

```
<class '__main__.Thing'>
```

In [3]:

```
example = Thing()
```

In [4]:

```
print(example)
```

```
<__main__.Thing object at 0x00000230FB6E58B0>
```

2. Create a new class called Thing2 and add the value 'abc' to the letters class attribute. Letters should be printed.

In [5]:

```
class Thing2:
    letters = 'abc'
```

In [6]:

```
print(Thing2.letters)
```

```
abc
```

3. Make yet another class called, of course, Thing3. This time, assign the value 'xyz' to an instance (object) attribute called letters. Print letters. Do you need to make an object from the class to do this?

In [7]:

```
class Thing3:
```

```
    def __init__(self):
        self.letters = 'xyz'
```

In [8]:

```
print(Thing3.letters)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-8-6f5d5916809a> in <module>
----> 1 print(Thing3.letters)

AttributeError: type object 'Thing3' has no attribute 'letters'
```

In [9]:

```
something = Thing3()
```

In [10]:

```
print(something.letters)
```

```
xyz
```

4. Create an Element class with the instance attributes name, symbol, and number. Create a class object with the values 'Hydrogen,' 'H,' and 1.

In [11]:

```
class Element:
    def __init__(self, name, symbol, number):
        self.name = name
        self.symbol = symbol
        self.number = number
```

In [12]:

```
hydrogen = Element('Hydrogen', 'H', 1)
```

5. Make a dictionary with these keys and values: 'name': 'Hydrogen', 'symbol': 'H', 'number': 1. Then, create an object called hydrogen from class Element using this dictionary.

In [13]:

```
el_dict = {'name': 'Hydrogen', 'symbol': 'H', 'number': 1}
```

In [14]:

```
hydrogen = Element(el_dict['name'], el_dict['symbol'], el_dict['number'])
```
In [15]:
```
hydrogen.name
```
Out[15]:
```
'Hydrogen'
```
In [16]:
```
hydrogen = Element(**el_dict)
```
In [17]:
```
hydrogen.name
```
Out[17]:
```
'Hydrogen'
```

6. For the Element class, define a method called dump() that prints the values of the object's attributes (name, symbol, and number). Create the hydrogen object from this new definition and use dump() to print its attributes.

In [18]:
```
class Element:
    def __init__(self, name, symbol, number):
        self.name = name
        self.symbol = symbol
        self.number = number
    def dump(self):
        print('name=%s, symbol=%s, number=%s' %
            (self.name, self.symbol, self.number))
```
In [19]:
```
hydrogen = Element(**el_dict)
```
In [20]:
```
hydrogen.dump()
name=Hydrogen, symbol=H, number=1
```

7. Call print(hydrogen). In the definition of Element, change the name of method dump to str, create a new hydrogen object, and call print(hydrogen) again.

In [21]:

```
print(hydrogen)
```

```
<__main__.Element object at 0x00000230FB7C90A0>
```

In [22]:

```
class Element:
    def __init__(self, name, symbol, number):
        self.name = name
        self.symbol = symbol
        self.number = number
    def __str__(self):
        return ('name=%s, symbol=%s, number=%s' %
            (self.name, self.symbol, self.number))
```

In [23]:

```
hydrogen = Element(**el_dict)
```

In [24]:

```
print(hydrogen)
```

```
name=Hydrogen, symbol=H, number=1
```

8. Modify Element to make the attributes name, symbol, and number private. Define a getter property for each to return its value.

In [25]:

```
class Element:
    def __init__(self, name, symbol, number):
        self.__name = name
        self.__symbol = symbol
        self.__number = number
    @property
    def name(self):
        return self.__name
    @property
    def symbol(self):
        return self.__symbol
    @property
    def number(self):
```

```
        return self.__number
```

In [26]:

```
hydrogen = Element('Hydrogen', 'H', 1)
```

In [27]:

```
hydrogen.name
```

Out[27]:

```
'Hydrogen'
```

In [28]:

```
hydrogen.symbol
```

Out[28]:

```
'H'
```

In [29]:

```
hydrogen.number
```

Out[29]:

```
1
```

9. Define three classes: Bear, Rabbit, and Octothorpe. For each, define only one method: eats(). This should return 'berries' (Bear), 'clover' (Rabbit), or 'campers' (Octothorpe). Create one object from each and print what it eats.

In [30]:

```
class Bear:
      def eats(self):
         return 'berries'
class Rabbit:
    def eats(self):
      return 'clover'
class Octothorpe:
    def eats(self):
        return 'campers'
```

In [31]:

```
b = Bear()
```

In [32]:

```
r = Rabbit()
```
In [33]:
```
o = Octothorpe()
```
In [34]:
```
print(b.eats())
```
```
berries
```
In [35]:
```
print(r.eats())
```
```
clover
```
In [36]:
```
print(o.eats())
```
```
campers
```

10. Define these classes: Laser, Claw, and SmartPhone. Each has only one method: does(). This returns 'disintegrate' (Laser), 'crush' (Claw), or 'ring' (SmartPhone). Then, define the class Robot that has one instance (object) of each of these. Define a does() method for the Robot that prints what its component objects do.

In [4]:
```
class Laser:
    def does(self):
            return 'disintegrate'
class Claw:
    def does(self):
            return 'crush'

class SmartPhone:
    def does(self):
            return 'ring'

class Robot:
    def __init__(self):
            self.laser = Laser()
            self.claw = Claw()
```

```
        self.smartphone = SmartPhone()
    def does(self):
        return '''I have many attachments:
        My laser, to %s.
        My claw, to %s.
        My smartphone, to %s.''' % (
        self.laser.does(),
        self.claw.does(),
        self.smartphone.does() )
```

In [5]:

```
robbie = Robot()
```

In [6]:

```
print(robbie.does())
```

```
I have many attachments:
        My laser, to disintegrate.
        My claw, to crush.
        My smartphone, to ring.
```