Inheritance

In [ ]:

```
'''
Develop a new class "MyList" that behaves as below:

>>> mylst = MyList()
>>> mylst.append(2)
>>> mylst.append(3)
>>> mylst.append(5)
>>> mylst.append(7)
>>> len(mylst)
4
>>> mylst.index(5)
2
>>> mylst.choice()
7
>>> mylst.choice()
3
>>> mylst.choice()
3
>>> mylst.choice()
5
>>> mylst.choice()
3
'''
```

In [1]:

```
import random
lst = [1, 2, 3, 4, 5]
```

In [2]:

```
random.choice(lst)
```

Out[2]: 5

In [3]:

```
random.choice(lst)
```

Out[3]: 4

In [4]:

```
random.choice(lst)
```

Out[4]: 4

In [5]:

```
random.choice(lst)
```

Out[5]: 4

In [6]:

```
class MyList:
    def __init__ (self):
        self.lst = []
    def append(self, item):
        self.lst.append(item)
    def __len__ (self):
        return len(self.lst)
```

In [7]:

```
mylst = MyList()
```

In [8]:

```
mylst.append(1)
```

In [9]:

```
mylst.append(2)
```

In [10]:

```
len(mylst)
```

Out[10]:

```
2
```

In [11]:

```
lst = list([2, 3])
lst
```

Out[11]:

```
[2, 3]
```

In [ ]:

```
mylst2 = MyList([2, 3])
```

In [13]:

```
class MyList:
    def __init__ (self, initial = []):
        self.lst = initial
    def append(self, item):
        self.lst.append(item)
    def __len__ (self):
        return len(self.lst)
```

In [14]:

```
mylst3 = MyList()
mylst3.append(1)
mylst3.append(2)
len(mylst3)
```

Out[14]:

```
2
```

In [15]:

```
mylst4 = MyList([1, 2])
mylst4.append(3)
len(mylst4)
```

Out[15]:

```
3
```

In [16]:

```
class MyList:
    def __init__ (self, initial = []):
        self.lst = initial # Without initiation, the list does not exist
    def append(self, item):
        self.lst.append(item)
    def __len__ (self):
        return len(self.lst)

    def choice(self):
        import random
        return random.choice(self.lst)
```

In [17]:

```
mylst5 = MyList([1, 2, 3, 4, 5])
mylst5.append(6)
```

In [18]:

```
len(mylst5)
```

Out[18]:

```
6
```

In [19]:

```
mylst5.choice()
```

Out[19]:

```
2
```

In [20]:

```
mylst5.choice()
```

Out[20]:

```
5
```

In [21]:

```
mylst5.choice()
```

Out[21]:

```
4
```

In [22]:

```
mylst5.choice()
```

Out[22]:

```
3
```

In [60]:

```
mylst5.index(1)
```

```
2
```

In [24]:

```
class MyList(list):
    def choice(self):
        import random
        return random.choice(self)
```

In [25]:

```
mylst6 = MyList()
mylst6.append(2)
mylst6.append(3)
mylst6.append(5)
mylst6.append(7)
```

In [26]:

```
len(mylst6)
```

```
Out[26]: 4
```

```
In [27]: mylst6.index(2)
```

```
Out[27]: 0
```

```
In [28]: mylst6.choice()
```

```
Out[28]: 2
```

```
In [29]: mylst6.choice()
```

```
Out[29]: 3
```

```
In [30]: mylst6.choice()
```

```
Out[30]: 2
```

```
In [31]: mylst6.choice()
```

```
Out[31]: 7
```

```
In [32]: dir(MyList)
```

```
Out[32]: ['__add__',
          '__class__',
          '__contains__',
          '__delattr__',
          '__delitem__',
          '__dict__',
          '__dir__',
          '__doc__',
          '__eq__',
          '__format__',
          '__ge__',
          '__getattribute__',
          '__getitem__',
          '__gt__',
          '__hash__',
          '__iadd__',
          '__imul__',
          '__init__',
          '__init_subclass__',
          '__iter__',
          '__le__',
          '__len__',
          '__lt__',
          '__module__',
          '__mul__',
          '__ne__',
          '__new__',
          '__reduce__',
          '__reduce_ex__',
          '__repr__',
          '__reversed__',
          '__rmul__',
          '__setattr__',
          '__setitem__',
          '__sizeof__',
          '__str__',
          '__subclasshook__',
          '__weakref__',
          'append',
          'choice',
          'clear',
          'copy',
          'count',
          'extend',
          'index',
          'insert',
          'pop',
          'remove',
          'reverse',
          'sort']
```

In [33]:

```
dir(mylst6)
```

Out[33]:

```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__delitem__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__gt__',
 '__hash__',
 '__iadd__',
 '__imul__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__module__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__reversed__',
 '__rmul__',
 '__setattr__',
 '__setitem__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 '__weakref__',
 'append',
 'choice',
 'clear',
 'copy',
 'count',
 'extend',
 'index',
 'insert',
 'pop',
 'remove',
 'reverse',
 'sort']
```

Overriding Superclass Methods

In [ ]:

```
'''
Implement a new class "Bird" that inherits from superclass "Animal".
We want the speak() method to behave differently.

>>> snoopy = Animal()
>>> snoopy.setSpecies('dog')
>>> snoopy.setLanguage('bark')
>>> snoopy.speak()
I am a dog and I bark.
>>> tweety = Bird()
>>> tweety.setSpecies('canary')
>>> tweety.setLanguage('tweet')
>>> tweety.speak()
tweet! tweet! tweet!
'''
```

In [34]:

```
class Animal:
    def setSpecies(self, species):
        self.spec = species

    def setLanguage(self, language):
        self.lang = language

    def speak(self):
        print('I am a {} and I {}.'.format(self.spec, self.lang))
```

In [35]:

```
class Bird:
    def setSpecies(self, species):
        self.spec = species

    def setLanguage(self, language):
        self.lang = language

    def speak(self):
        print('{}! {}! {}!'.format(self.lang, self.lang, self.lang))
```

In [36]:

```
tweety = Bird()
tweety.setSpecies('canary')
tweety.setLanguage('tweet')
tweety.speak()
```

```
tweet! tweet! tweet!
```

In [37]:

```python
class Bird(Animal):
    def speak(self):
        print('{}! {}! {}!'.format(self.lang, self.lang, self.lang))
```

In [38]:

```python
tweety = Bird()
tweety.setSpecies('canary')
tweety.setLanguage('tweet')
tweety.speak()
```

```
tweet! tweet! tweet!
```

Practice

In [ ]:

```python
'''
Implement a class "Person" that supports these methods:

__init__(): A constructor that takes as input a person's name (as a string) and birth y
age(): Returns the age of the person
name(): Returns the name of the person

Use the function localtime() from the Standard Library module time to compute the age.

The implementation of the class should behave as shown in the next code:

>>> p1 = Person('Blake', 2000)
>>> p1.age()
41
>>> p1.name()
'Blake'
'''
```

In [39]:

```python
import time
time.localtime()
```

Out[39]:

```
time.struct_time(tm_year=2023, tm_mon=12, tm_mday=21, tm_hour=13, tm_min=9, tm_sec=56, t
m_wday=3, tm_yday=355, tm_isdst=0)
```

In [40]:

```python
time.localtime().tm_year
```

Out[40]:

```
2023
```

In [41]:

```python
type(time.localtime().tm_year)
```

Out[41]:

```
int
```

In [42]:

```python
class Person:
    def __init__(self, name, year):
        self.p_n = name
        self.p_y = year
    def age(self):
        import time
        return time.localtime().tm_year - self.p_y
    def name(self):
        return self.p_n
```

In [43]:

```
p1 = Person('Blake', 2000)
```

In [44]:

```
p1.age()
```

Out[44]:

```
23
```

In [45]:

```
p1.name()
```

Out[45]:

```
'Blake'
```

In [ ]:

```
'''
Implement two subclasses of class "Person".

(1) The class "Instructor" supports methods:

__init__(): Constructor that takes the person's degree in addition to name and birth yea
degree(): Returns the degree of the instructor

(2) The class "Student", also a subclass of "Person", supports:

__init__(): Constructor that takes the person's major in addition to name and birth yea
major(): Returns the major of the student

The implementation of the three classes should behave as shown in the next code:

>>> x = Instructor('Blake', 2000, 'Masters')
>>> x.age()
21
>>> x.degree()
'PhD'
>>> y = Student('Jones', 1996, 'Business Administration')
>>> y.age()
25
>>> y.major()
'Business Administration'
'''
```

In [46]:

```
class Instructor(Person):
    def __init__(self, name, year, degree):
        self.i_n = name
        self.i_y = year
        self.i_d = degree
    def degree(self):
        return self.i_d

class Student(Person):
    def __init__(self, name, year, major):
        self.s_n = name
        self.s_y = year
        self.s_m = major
    def major(self):
        return self.s_m
```

In [47]:

```
x = Instructor('Blake', 2000, 'Masters')
```

In [61]:

```
x.age()
```

```
23
```

In [49]:

```
x.degree()
```

Out[49]:

```
'Masters'
```

In [50]:

```
y = Student('Jones', 1996, 'Business Administration')
```

In [62]:

```
y.age()
```

```
27
```

In [52]:

```
y.major()
```

Out[52]:

```
'Business Administration'
```

In [53]:

```
class Instructor(Person):
    def __init__(self, name, year, degree):
        self.p_n = name
        self.p_y = year
        self.p_d = degree
    def degree(self):
        return self.p_d

class Student(Person):
    def __init__(self, name, year, major):
        self.p_n = name
        self.p_y = year
        self.p_m = major
    def major(self):
        return self.p_m
```

In [54]:

```
x = Instructor('Blake', 2000, 'Masters')
```

In [55]:

```
x.age()
```

Out[55]:

```
23
```

```
In [56]:  x.degree()
```

```
Out[56]:  'Masters'
```

```
In [57]:  y = Student('Jones', 1996, 'Business Administration')
```

```
In [58]:  y.age()
```

```
Out[58]:  27
```

```
In [59]:  y.major()
```

```
Out[59]:  'Business Administration'
```