Overloading Constructor

> Overloading, in the context of programming, refers to the ability of a function or an operator to behave in different ways depending on the parameters that are passed to the function, or the operands that the operator acts on.

In [1]:
```
lst = [1, 2, 3]
lst
```

Out[1]: [1, 2, 3]

In [2]:
```
lst_1 = list()
lst_1.append(1)
lst_1.append(2)
lst_1.append(3)
lst
```

Out[2]: [1, 2, 3]

In [3]:
```
class Point:
    def __init__(self, xcoord = 0, ycoord = 0):
        self.x = xcoord
        self.y = ycoord
    def setx(self, xcoord):
        self.x = xcoord
    def sety(self, ycoord):
        self.y = ycoord
    def get(self):
        return (self.x, self.y)
```

In [4]:
```
p = Point(10, 20)
p.get()
```

Out[4]: (10, 20)

In [5]:
```
p_1 = Point(100, 200)
p_1.get()
```

Out[5]: (100, 200)

In [6]:
```
p_2 = Point()
p_2.get()
```

Out[6]: (0, 0)

In [7]:
```
lst1 = []
```

In [8]:
```
lst2 = [1, 2, 3]
```

In [ ]:
```
'''
Modify the class "Animal" so it supports a two, one, or no input argument constructor.

>>> snoopy = Animal('dog', 'bark')
>>> snoopy.speak()
I am a(an) dog and I bark.
>>> tweety = Animal('canary')
>>> tweety.speak()
I am a(an) canary and I make sounds.
>>> animal = Animal()
>>> animal.speak()
I am a(an) animal and I make sounds.
'''

'''
class Animal:
    def setSpecies(self, sp):
        self.spec = sp
    def setLanguage(self, lg):
        self.lang = lg
    def speak(self):
        print('I am a(an) {} and I {}.'.format(self.spec, self.lang))
'''
```

In [9]:
```
class Animal:
    def setSpecies(self, sp):
        self.spec = sp
    def setLanguage(self, lg):
        self.lang = lg
    def speak(self):
        print('I am a(an) {} and I {}.'.format(self.spec, self.lang))
```

In [10]:
```
snoopy = Animal()
```

In [11]:
```
snoopy.setSpecies('dog')
```

In [12]:
```
snoopy.setLanguage('bark')
```

In [13]:
```
snoopy.speak()
```

```
I am a(an) dog and I bark.
```

In [ ]:
```
snoopy = Animal('dog', 'bark')
```

> __init__ is a special Python method that is automatically called when memory is allocated for a new object.

> The sole purpose of __init__ is to initialize the values of instance members for the new object.

In [15]:
```
class Animal:
    def __init__(self, sp, lg):
        self.spec = sp
        self.lang = lg
    def setSpecies(self, sp):
        self.spec = sp
    def setLanguage(self, lg):
        self.lang = lg
    def speak(self):
        print('I am a(an) {} and I {}.'.format(self.spec, self.lang))
```

In [16]:
```
snoopy = Animal('dog', 'bark')
```

In [17]:
```
snoopy.speak()
```

```
I am a(an) dog and I bark.
```

In [ ]:
```
snoopy = Animal()
```

In [19]:
```
class Animal:
    def __init__(self, sp = 'animal', lg = 'make sounds'):
        self.spec = sp
        self.lang = lg
    def setSpecies(self, sp):
        self.spec = sp
    def setLanguage(self, lg):
        self.lang = lg
    def speak(self):
        print('I am a(an) {} and I {}.'.format(self.spec, self.lang))
```

In [20]:
```
snoopy = Animal('dog', 'bark')
```

In [21]:
```
snoopy.speak()
```

```
I am a(an) dog and I bark.
```

In [22]:
```
snoopy = Animal('dog')
```

In [23]:
```
snoopy.speak()
```

```
I am a(an) dog and I make sounds.
```

In [24]:
```
snoopy = Animal()
```

In [25]:
```
snoopy.speak()
```

```
I am a(an) animal and I make sounds.
```

Overloading repr(), operator +, operator ==

In [26]:
```
s1 = 'he'
s2 = 'llo'

s1 + s2
```

Out[26]:
```
'hello'
```

In [27]:
```
s1.__add__(s2)
```

Out[27]:
```
'hello'
```

In [28]:
```
str.__add__(s1, s2)
```

Out[28]:
```
'hello'
```

In [29]:
```
s1 == s2
```

Out[29]:
```
False
```

In [30]:
```
s1.__eq__(s2)
```

Out[30]:
```
False
```

In [31]:
```
str.__eq__(s1, s2)
```

Out[31]:
```
False
```

In [32]:
```
s1 != s2
```

Out[32]:
```
True
```

In [33]:
```
s1.__ne__(s2)
```

Out[33]:
```
True
```

In [34]:
```
str.__ne__(s1, s2)
```

Out[34]:
```
True
```

In [35]:
```
repr(s1)
```

Out[35]:
```
"'he'"
```

In [36]:
```
s1.__repr__()
```

Out[36]:
```
"'he'"
```

In [37]:
```
len(s1)
```

Out[37]:
```
2
```

In [38]:
```
s1.__len__()
```

Out[38]:
```
2
```

In [39]:
```
len(s1 + s2)
```

Out[39]:
```
5
```

In [40]:
```
s1.__add__(s2).__len__()
```

Out[40]:
```
5
```

In [41]:
```
str.__len__(str.__add__(s1, s2))
```

Out[41]:
```
5
```

In [42]:
```
repr([1, 2, 3])
```

Out[42]:
```
'[1, 2, 3]'
```

In [43]:
```
[1, 2, 3].__repr__()
```

Out[43]:
```
'[1, 2, 3]'
```

In [ ]:
```
'''
>>> a = Point(3, 4)
>>> a
Point(3, 4)
'''
```

In [44]:
```
class Point:
    def __init__(self, xcoord = 0, ycoord = 0):
```

```
        self.x = xcoord
        self.y = ycoord
    def setx(self, xcoord):
        self.x = xcoord
    def sety(self, ycoord):
        self.y = ycoord
    def get(self):
        return (self.x, self.y)
    def move(self, dx, dy):
        self.x += dx
        self.y += dy
```

In [45]:
```
a = Point(3, 4)
```

In [46]:
```
a
```

Out[46]:
```
<__main__.Point at 0x2b7cc5f74f0>
```

In [47]:
```
a.get()
```

Out[47]:
```
(3, 4)
```

In [48]:
```
class Point:
    def __init__(self, xcoord = 0, ycoord = 0):
        self.x = xcoord
        self.y = ycoord
    def setx(self, xcoord):
        self.x = xcoord
    def sety(self, ycoord):
        self.y = ycoord
    def get(self):
        return (self.x, self.y)
    def move(self, dx, dy):
        self.x += dx
        self.y += dy
    def __repr__(self):
        return 'Point({}, {})'.format(self.x, self.y)
```

In [49]:
```
a = Point(3, 4)
a
```

Out[49]:
```
Point(3, 4)
```

In [50]:
```
a.__repr__()
```

Out[50]:
```
'Point(3, 4)'
```

In [51]:
```
Point.__repr__(a)
```

Out[51]:
```
'Point(3, 4)'
```

In [ ]:
```
'''
>>> a = Point(3, 4)
>>> b = Point(1, 2)
>>> a + b
Point(4, 6)
'''
```

In [52]:
```
class Point:
    def __init__(self, xcoord = 0, ycoord = 0):
        self.x = xcoord
        self.y = ycoord
    def setx(self, xcoord):
        self.x = xcoord
    def sety(self, ycoord):
        self.y = ycoord
    def get(self):
        return (self.x, self.y)
    def move(self, dx, dy):
        self.x += dx
        self.y += dy
    def __repr__(self):
        return 'Point({}, {})'.format(self.x, self.y)
```

In [53]:
```
a = Point(3, 4)
b = Point(1, 2)
```

In [54]:
```
a + b
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-54-bd58363a63fc> in <module>
----> 1 a + b

TypeError: unsupported operand type(s) for +: 'Point' and 'Point'
```

In [55]:
```
class Point:
    def __init__(self, xcoord = 0, ycoord = 0):
        self.x = xcoord
        self.y = ycoord
    def setx(self, xcoord):
        self.x = xcoord
    def sety(self, ycoord):
        self.y = ycoord
    def get(self):
        return (self.x, self.y)
    def move(self, dx, dy):
        self.x += dx
        self.y += dy
    def __repr__(self):
        return 'Point({}, {})'.format(self.x, self.y)
    def __add__(self, point):
        return Point(self.x + point.x, self.y + point.y)
```

In [56]:
```
a = Point(3, 4)
b = Point(1, 2)
```

In [57]:
```
a + b
```

Out[57]:
```
Point(4, 6)
```

In [58]:
```
a.__add__(b)
```

Out[58]:
```
Point(4, 6)
```

In [59]:
```
Point.__add__(a, b)
```

Out[59]:
```
Point(4, 6)
```

In [ ]:
```
'''
>>> appts = Queue()
>>> len(appts)
0
'''
```

In [60]:
```
class Queue:
    def __init__(self):
        self.q = []
    def isEmpty(self):
        return (len(self.q) == 0)
    def enqueue(self, item):
        return self.q.append(item)
    def dequeue(self):
        return self.q.pop(0)
    def check(self):
        return self.q
```

In [61]:
```
appts = Queue()
```

In [62]:
```
appts.enqueue('Blake')
```

In [63]:
```
appts.enqueue('Jeff')
```

In [64]:
```
appts.enqueue('Grace')
```

In [65]:
```
appts.check()
```

Out[65]:
```
['Blake', 'Jeff', 'Grace']
```

In [66]:
```
appts.dequeue()
```

Out[66]:
```
'Blake'
```

In [67]:
```
appts.check()
```

Out[67]:
```
['Jeff', 'Grace']
```

In [68]:
```
appts.isEmpty()
```

Out[68]:
```
False
```

In [69]:
```
len(appts)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-69-4b7d4f3655a4> in <module>
----> 1 len(appts)

TypeError: object of type 'Queue' has no len()
```

In [70]:
```
class Queue:
    def __init__(self):
        self.q = []
    def isEmpty(self):
        return (len(self.q) == 0)
    def enqueue(self, item):
        return self.q.append(item)
    def dequeue(self):
        return self.q.pop(0)
    def check(self):
        return self.q
    def __len__(self): # len() has to be added in
        return len(self.q)
```

In [71]:
```
appts = Queue()
appts.enqueue('Blake')
appts.enqueue('Jeff')
appts.enqueue('Grace')
appts.check()
```

Out[71]:
```
['Blake', 'Jeff', 'Grace']
```

In [72]:
```
len(appts)
```

Out[72]:
```
3
```

In [ ]:
```
'''
>>> a = Point(3, 5)
>>> b = Point(3, 5)
>>> a == b
True
>>> a == a
True
'''
```

In [74]:
```
class Point:
    def __init__(self, xcoord = 0, ycoord = 0):
        self.x = xcoord
        self.y = ycoord
    def setx(self, xcoord):
        self.x = xcoord
    def sety(self, ycoord):
        self.y = ycoord
    def get(self):
        return (self.x, self.y)
    def move(self, dx, dy):
        self.x += dx
        self.y += dy
    def __repr__(self):
        return 'Point({}, {})'.format(self.x, self.y)
    def __add__(self, point):
        return Point(self.x + point.x, self.y + point.y)
```

In [75]:
```
a = Point(3, 5)
b = Point(3, 5)
```

In [76]:
```
a == b
```

Out[76]:
```
False
```

In [77]:
```
a == a
```

Out[77]:
```
True
```

In [78]:
```
class Point:
    def __init__(self, xcoord = 0, ycoord = 0):
        self.x = xcoord
        self.y = ycoord
    def setx(self, xcoord):
        self.x = xcoord
    def sety(self, ycoord):
        self.y = ycoord
    def get(self):
        return (self.x, self.y)
    def move(self, dx, dy):
        self.x += dx
        self.y += dy
    def __repr__(self):
```

```
        return 'Point({}, {})'.format(self.x, self.y)
    def __add__(self, point):
        return Point(self.x + point.x, self.y + point.y)
    def __eq__(self, point):
        return self.x == point.x and self.y == point.y
```

In [79]:
```
a = Point(3, 5)
b = Point(3, 5)
```

In [80]:
```
a == b
```

Out[80]: True

In [81]:
```
a == a
```

Out[81]: True

In [82]:
```
a + b
```

Out[82]: Point(6, 10)

In [83]:
```
c = a + b
```

In [84]:
```
c
```

Out[84]: Point(6, 10)

In [85]:
```
d = Point(6, 10)
```

In [86]:
```
c == d
```

Out[86]: True