Carson Sklare

2.

```
class BankAccount():
    def __init__(self, inital = 0):
        if inital < 0:
            raise ValueError('Illegal Balance')
        self._value = inital


    def withdraw(self, ammount):
        if ammount > self._value:
            raise ValueError('Overdraft')
        self._value -= ammount

    def deposit(self, ammount):
        if ammount < 0:
            raise ValueError('Negative deposit')
        self._value += ammount

    def balance(self):
        return self._value
```

Test Run:

```
>>> x = BankAccount(-700)
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    x = BankAccount(-700)
  File "C:/Users/foshi/Desktop/Files/Programs/BankAccount.py", line 4, in __init__
    raise ValueError('Illegal Balance')
ValueError: Illegal Balance
>>> x = BankAccount(700)
>>> x.withdraw(30)
>>> x.balance()
670
>>> x.deposit(3.55)
>>> x.balance()
673.55
>>> x.withdraw(700)
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    x.withdraw(700)
  File "C:/Users/foshi/Desktop/Files/Programs/BankAccount.py", line 10, in withdraw
    raise ValueError('Overdraft')
ValueError: Overdraft
>>> x.balance()
673.55
>>> x.deposit(-300)
Traceback (most recent call last):
  File "<pyshell#22>", line 1, in <module>
    x.deposit(-300)
  File "C:/Users/foshi/Desktop/Files/Programs/BankAccount.py", line 15, in deposit
    raise ValueError('Negative deposit')
ValueError: Negative deposit
>>> x.balance()
673.55
>>>
```

3.

```
class Interval():

    def __init__(self, a, b):
        'make new interval'
        if a > b:
            self._a = a
            self._b = a
        else:
            self._a = a
            self._b = b

    def __repr__(self):
        return 'Interval({},{})'.format(self._a, self._b)

    def is_empty(self):
        'checks if interval is empty'
        return self._a == self._b

    def __str__(self):
        'prints interval'
        return '({},{})'.format(self._a,self._b)

    def __eq__(self,other):
        'sees if two intervals both empty'
        return self.is_empty() == other.is_empty()

    def __contains__(self,n):
        'sees if a number is inside the interval'
        return self._a < n < self._b
```

```
def __and__(self,other):
    'intersection of two intervals'
    if self.__contains__(other._a):
        a = other._a
    elif other.__contains__(self._a):
        a = self._a

    if self.__contains__(other._b):
        b = other._b
    elif other.__contains__(self._b):
        b = self._b
    else:
        if self._a >= other._a:
            a = self._a
            b = self._a
        else:
            a = other._a
            b = other._a
    newInter = Interval(a,b)
    return newInter
```

```
class Interval():

    def __init__(self, a, b):
        'make new interval'
        if a > b:
            self._a = a
            self._b = a
        else:
            self._a = a
            self._b = b

    def __repr__(self):
        return 'Interval({},{})'.format(self._a, self._b)

    def is_empty(self):
        'checks if interval is empty'
        return self._a == self._b

    def __str__(self):
        'prints interval'
        return '({},{})'.format(self._a,self._b)

    def __eq__(self,other):
        'sees if two intervals both empty'
        return self.is_empty() == other.is_empty()

    def __contains__(self,n):
        'sees if a number is inside the interval'
        return self._a < n < self._b

    def __and__(self,other):
        'intersection of two intervals'
        if self.__contains__(other._a):
            a = other._a
        elif other.__contains__(self._a):
            a = self._a

        if self.__contains__(other._b):
            b = other._b
        elif other.__contains__(self._b):
            b = self._b
        else:
            if self._a >= other._a:
                a = self._a
                b = self._a
            else:
                a = other._a
                b = other._a
        newInter = Interval(a,b)
        return newInter
```

Test Runs:

```
>>> i1 = Interval(-3.5,10.2)
>>> i2 = Interval(7,30.5)
>>> i3 = Interval(20,25)
>>> 3.2 in i1
True
>>> -3.5 in i1
False
>>> i1
Interval(-3.5,10.2)
>>> i1 & i2
Interval(7,10.2)
>>> i1 & i3
Interval(20,20)
>>> i1 & i3 == Interval(10,10)
True
>>> i1 & i3 == Interval(10,13)
False
>>> i3 & i1
Interval(20,20)
>>> i3 & i2
Interval(20,25)
>>> i2 & i3
Interval(20,25)
```

4.

```
from html.parser import HTMLParser
from urllib.request import urlopen
class UnclosedParser(HTMLParser):

    def __init__(self):
        HTMLParser.__init__(self)
        self._opentags = []

    def handle_data(self, data):
        pass

    def handle_starttag(self, tag,attrs):
        self._opentags.append(tag)

    def handle_endtag(self,tag):
        if tag != self._opentags[len(self._opentags) -1]:
            print('Found unclosed {} tag'.format(self._opentags[len(self._opentags) -1]))
            self._opentags.remove(self._opentags[len(self._opentags)-1])
            foundtag = False
            while not(foundtag):
                if tag != self._opentags[len(self._opentags) -1]:
                    print('Found unclosed {} tag'.format(self._opentags[len(self._opentags) -1]))
                    self._opentags.remove(self._opentags[len(self._opentags)-1])
                else:
                    foundtag = True
        if tag in self._opentags:
            self._opentags.remove(tag)
```

Test runs:

```
>>> html = '<html><head><title>My Page</title></head><body>Hello World<br>How are you <hr> Rest of Text. Lorem Ipsum</body></html>'
>>> up = UnclosedParser()
>>> up.feed(html)
Found unclosed hr tag
Found unclosed br tag
>>> html = urlopen('http://ovid.cs.depaul.edu/Teaching.htm').read().decode()
>>> up.feed(html)
Found unclosed meta tag
Found unclosed meta tag
Found unclosed meta tag
Found unclosed meta tag
Found unclosed img tag
Found unclosed td tag
Found unclosed hr tag
```