HW3

Reading: Chapter 8

Problems:

Use the usual naming scheme. Make sure you include the name(s) of any collaborators, and run the doctest on your solutions.

1. Implement a container class `Stat` that stores a sequence of numbers and provides statistical information about the numbers. It supports an overloaded constructor that initializes the container either by supplying a list or by giving no arguments (which creates an empty sequence). The class also includes the methods necessary to provide the following behaviors:

```
>>> s = Stat()
>>> s.add(2.5)
>>> s.add(4.7)
>>> s.add(78.2)
>>> s
Stat([2.5, 4.7, 78.2])
>>> len(s)
3
>>> s.min()
2.5
>>> s.max()
78.2
>>> s.sum()
85.4
>>> s.mean()
28.46666666666667
>>> s.clear()
>>> s
Stat([])
```

   If a Stat is empty, several (but not all) methods raise errors. Note that you won't literally see "…". You will instead see more information on the error.

```
>>> s = Stat()
>>>
>>> len(s)
0
>>> s.min()
Traceback (most recent call last):
...
EmptyStatError: empty Stat does not have a min
```

```
>>> s.max()
Traceback (most recent call last):
...
EmptyStatError: empty Stat does not have a max
>>> s.mean()
Traceback (most recent call last):
...
EmptyStatError: empty Stat does not have a mean
>>> s.sum()
0
```

2. Implement a class `intlist` which a list that stores only integers. You MUST subclass `list`. Please note the following:

    - constructor – can be passed a list of `int`s, or, by default constructs an empty `intlist`.
    - `append, insert,extend` – can also be used to add `ints` to an intlist. If you don't know how extend works for a list, look it up. All should raise errors if a non-int is added.
    - `__setitem__` - can be used for item assignment using an index. Raises error if non-int is used.
    - `odds()` – write a method `odds()` which returns an `intlist` consisting of the odd `int`'s. They should not be removed from the original.
    - `evens()` – same as `odds()`, but for even `ints`
    - `NotIntError` – also write an `Exception` class `NotIntError` that subclasses `Exception`.
    - `NotIntError` – a `NonIntError` should be raised when client code attempts to place something other than an `int` in an `intlist`. This can happen in three ways (all shown in code below):
        - o `append`
        - o `insert`
        - o The constructor – when passed a `list` that contains something other an int
        - o Note: you can check whether `item` is an `int` by evaluating the expression `type(item)==int`

    Your goal is to get the following behavior:

```
>>> il = intlist()
>>> il
intlist([])
>>> il = intlist([1,2,3])
>>> il
intlist([1, 2, 3])
>>> il.append( 5 )
>>> il
intlist([1, 2, 3, 5])
>>> il.insert(1,99)
>>> il
intlist([1, 99, 2, 3, 5])
>>> il.extend( [22,44,66] )
>>> il
intlist([1, 99, 2, 3, 5, 22, 44, 66])
>>> odds = il.odds()
>>> odds
intlist([1, 99, 3, 5])
>>> evens = il.evens()
>>> evens
intlist([2, 22, 44, 66])
>>> il
intlist([1, 99, 2, 3, 5, 22, 44, 66])
>>> il[2] = -12   # calls __setitem__
>>> il
intlist([1, 99, -12, 3, 5, 22, 44, 66])
>>> il[4]   # calls __getitem__
5
```

Trying to put anything except for an int into an intlist will always raise an `NotIntError`. Note that there 5 different ways this could be attempted:

```
>>> il.append(33.4)
Traceback (most recent call last):
...
NotIntError: 33.4 not an int
>>> il.insert(2,True)
Traceback (most recent call last):
...
NotIntError: True not an int
>>> il = intlist([2,3,4,"apple"])
Traceback (most recent call last):
...
NotIntError: apple not an int
>>> il.extend( [2,3,'hello'])
Traceback (most recent call last):
...
NotIntError: hello not an int
>>> il[2] = 22.3
Traceback (most recent call last):
...
NotIntError: 22.3 not an int
```