Exercise: int set

Exercise: int set

0.0/5.0 points (graded)

ESTIMATED TIME TO COMPLETE: 10 minutes

Consider the following code from the last lecture video:

```
class intSet(object):
    """An intSet is a set of integers
    The value is represented by a list of ints, self.vals.
    Each int in the set occurs in self.vals exactly once."""

    def __init__(self):
        """Create an empty set of integers"""
        self.vals = []

    def insert(self, e):
        """Assumes e is an integer and inserts e into self"""
        if not e in self.vals:
            self.vals.append(e)

    def member(self, e):
        """Assumes e is an integer
           Returns True if e is in self, and False otherwise"""
        return e in self.vals

    def remove(self, e):
        """Assumes e is an integer and removes e from self
           Raises ValueError if e is not in self"""
        try:
            self.vals.remove(e)
        except:
            raise ValueError(str(e) + ' not found')

    def __str__(self):
        """Returns a string representation of self"""
        self.vals.sort()
        return '{' + ','.join([str(e) for e in self.vals]) + '}'
```

Your task is to define the following two methods for the `intSet` class:

1. Define an `intersect` method that returns a new `intSet` containing elements that appear in both sets. In other words,

   ```
   s1.intersect(s2)
   ```

   would return a new `intSet` of integers that appear in both `s1` and `s2`. Think carefully - what should happen if `s1` and `s2` have no elements in common?

2. Add the appropriate method(s) so that `len(s)` returns the number of elements in

Sorry we could not search the store for annotations

Hint: look through the Python docs to figure out what you'll need to solve this problem.

```
17            Returns True if e is in self, and False otherwise
18        return e in self.vals
19
20    def remove(self, e):
21        """Assumes e is an integer and removes e from self
22           Raises ValueError if e is not in self"""
23        try:
24            self.vals.remove(e)
25        except:
26            raise ValueError(str(e) + ' not found')
27
28    def __str__(self):
29        """Returns a string representation of self"""
30        self.vals.sort()
31        return '{' + ','.join([str(e) for e in self.vals]) + '}'
32
```

Press ESC then TAB or click outside of the code editor to exit

Unanswered

Sorry we could not search the store for annotations

```
class intSet(object):
    """An intSet is a set of integers
    The value is represented by a list of ints, self.vals.
    Each int in the set occurs in self.vals exactly once."""

    def __init__(self):
        """Create an empty set of integers"""
        self.vals = []

    def insert(self, e):
        """Assumes e is an integer and inserts e into self"""
        if not e in self.vals:
            self.vals.append(e)

    def member(self, e):
        """Assumes e is an integer
           Returns True if e is in self, and False otherwise"""
        return e in self.vals

    def remove(self, e):
        """Assumes e is an integer and removes e from self
           Raises ValueError if e is not in self"""
        try:
            self.vals.remove(e)
        except:
            raise ValueError(str(e) + ' not found')

    def intersect(self, other):
        """Assumes other is an intSet
           Returns a new intSet containing elements that appear in both sets."""
        # Initialize a new intSet
        commonValueSet = intSet()
        # Go through the values in this set
        for val in self.vals:
            # Check if each value is a member of the other set
            if other.member(val):
                commonValueSet.insert(val)
        return commonValueSet

    def __str__(self):
        """Returns a string representation of self"""
        self.vals.sort()
        return '{' + ','.join([str(e) for e in self.vals]) + '}'

    def __len__(self):
```

```
        This method is called by the `len` built-in function."""
    return len(self.vals)
```

Submit

Answers are displayed within the problem

Exercise: int set

Topic: Lecture 9 / Exercise: int set

Hide Discussion

Add a Post

All Posts

[staff] I find this problem very confusing, please help !!!

question posted 3 months ago by -Mayster-

I've managed to do this exercise but I'm still very confused with concept of `__iter__,__next__,__len__`. I'll try to explain my confusion briefly.

I've came up with four solutions while trying to catch the idea. Versions 1÷3 are working fine, 4'th fails:

Version 1 (both iter & next are defined):

```
def __iter__(self):
    return self.__next__()

def __next__(self):
    for i in self.vals:
        yield i
```

Version 2 (iter is defined, next is not defined):

```
def __iter__(self):
    for i in self.vals:
        yield i
```

Version 3 (iter is defined, next is not defined):

Sorry we could not search the store for annotations

Version 4 (both iter & next are defined):

```
def __iter__(self):
    return self

def __next__(self):
    for i in self.vals:
        yield i
```

Questions:

1. Which version 1÷3 is the best and which one is worst and why ?
2. What is the solution suggested by Professor / Staff (hopefully with comments) ?
3. Why do we need this `__next__` method if all works well without it ?
4. What is `__len__` for ? All working solutions do their task of intersecting even if this method is not defined ?
5. Version 4 enters an infinite loop when I want to iterate an instance of `intSet()` with `for loop`. Console prints infinite sequence of two repeatable lines:

   ```
   <generator object intSet.__next__ at 0x000000A4F0FF28E0>
   <generator object intSet.__next__ at 0x000000A4F0FF2938>
   <generator object intSet.__next__ at 0x000000A4F0FF28E0>
   <generator object intSet.__next__ at 0x000000A4F0FF2938>
   ```

   I don't understand why ?

This post is visible to everyone.

Add a Response

4 responses

kiwitrader (Community TA)
3 months ago

OK. I'll have a go. Not staff though.

I'm not sure I really like any of them. Test them on my suite ...

```
ii = intSet()
for i in range(7): ii.insert(i)
print(ii)
print(next(ii))
print(next(ii))
print()
for i in ii: print(i, end=', ')
print()
ii = intSet()
for i in range(7): ii.insert(i)
for i in ii: print(i, end=', ')
```

which should return

```
{0,1,2,3,4,5,6}
0
1

2, 3, 4, 5, 6,
0, 1, 2, 3, 4, 5, 6,
```

If I was turning this into a generator I'd go with something like this (but, yes, I like the versions with both next & iter best):

```
    def __iter__(self):
        return self

    def __next__(self):
        try:
            return self.vals.pop(0)
        except IndexError:
            raise StopIteration
```

A little bit of an explanation of these methods at the bottom.

next & iter are involved with creating iterators & generators so you don't need them unless you want to have an object that provides that functionality.

dunder len provides a method for the interpreter when someone calls len(yourobjectsname)

Re the infinite loop. I'm not sure what it is but I suspect your return self results in the return of multiple loops (recursively on themselves). Notice that my version yields one value at a time, popping it from the internal list (vals) which makes the list one shorter so that when it finishes it throws a StopIteration error which tells the interpreter to STOP LOOPING NOW!!!!

Thanks for Your response. I tried Your code and `return self.vals.pop(0)` used in `__next__` changes your intSet() instance irreversibly. Let's say `a` is non empty instance of `Int(set)`. After one `for elm in a: ...` loop we mutate `a` to empty `intSet()`.

p.s. I've always considered CTA's as a part of Staff since You are adding so much value with Your contributions :)

posted 3 months ago by **-Mayster-**

Your conclusion is right ... but that's how generators work ... one shot & then you have to create a new instance.

But I also found it unsatisfactory in this case which is why i did the other one below.

Some people mean MIT staff when they say that so I ignore their posts - I suspected you might be more flexible.

posted 3 months ago by **kiwitrader** (Community TA)

Sorry for my comment ... I've read Your next answer and see that You spot it.

posted 3 months ago by **-Mayster-**

No. It was a good comment. Shows that you're seeing what's happening.

posted 3 months ago by **kiwitrader** (Community TA)

Add a comment

kiwitrader (Community TA)
3 months ago

The first version has the downside of behaving like a generator and removing each item generated.

Here's a version that uses indexing instead so that the object is preserved. Because I didn't want to add `self.__i` to the init I test for its non-existence. The index is deleted when you get to the end but it could have just been set to zero.

Sorry we could not search the store for annotations

```
class intSet(object):
    """An intSet is a set of integers
    The value is represented by a list of ints, self.vals.
    Each int in the set occurs in self.vals exactly once."""

    def __init__(self):
        """Create an empty set of integers"""
        self.vals = []

    def insert(self, e):
        """Assumes e is an integer and inserts e into self"""
        if not e in self.vals:
            self.vals.append(e)

    def member(self, e):
        """Assumes e is an integer
           Returns True if e is in self, and False otherwise"""
        return e in self.vals

    def remove(self, e):
        """Assumes e is an integer and removes e from self
           Raises ValueError if e is not in self"""
        try:
            self.vals.remove(e)
        except:
            raise ValueError(str(e) + ' not found')

    def __iter__(self):
        return self

    def __next__(self):
        try:
            try:
                self.__i += 1
            except AttributeError:
                self.__i = 0
            return self.vals[self.__i]
            # return self.vals.pop(0)
        except IndexError:
            del self.__i
            raise StopIteration

    def __str__(self):
        """Returns a string representation of self"""
        self.vals.sort()
        return '{' + ','.join([str(e) for e in self.vals]) + '}'
```

```
for i in range(7): ii.insert(i)
print(ii)
print(next(ii))
print(next(ii))
print()
for i in ii: print(i, end=', ')
print('\ntry again')
print(ii)
for i in ii: print(i, end=', ')
print('\nrestart')
ii = intSet()
for i in range(7): ii.insert(i)
for i in ii: print(i, end=', ')

{0,1,2,3,4,5,6}
0
1

2, 3, 4, 5, 6,
try again
{0,1,2,3,4,5,6}
0, 1, 2, 3, 4, 5, 6,
restart
0, 1, 2, 3, 4, 5, 6,
```

Please clarify: `self.__i` is simply a 'intended to be private' index variable ? I'm new to this underscore naming thing ;) and it poses some ambiguity for me.

posted 3 months ago by **-Mayster-**

I probably shouldn't have done it but unfortunately I play with code when I'm answering questions. So in this case, I decided that maybe seeing I didn't want i my index variable to be part of the outside worlds view of my class I'd disguise it. The rules are simple

x says, use it as you want
x_ says that I have chosen to distinguish this object from the real x but you can use it
_x says, please don't use it, I'd prefer it was left alone
__x says, leave me the heck alone - I know you can get to it if you try hard but DONT !!!!

At least I'm pretty sure that's the hierachy

posted 3 months ago by **kiwitrader** (Community TA)

Sorry we could not search the store for annotations

I understand Your above example but still don't get why not doing this with simple:

```
def __iter__(self):
    return iter(self.vals)
```

It seems that this code compared to Your above alternative:

1. Is shorter.
2. Has the same functionality.
3. Is easier to understand.
4. You've also used it in Your example posted in separate discussion thread :)

posted 3 months ago by **-Mayster-**

It doesn't work with next() though. What would you be using for `__next__` in that case? You need to test your solution by running next 2 or 3 times to make sure it advances & then iterate over the rest of the list & see if it continues from that point.

posted 3 months ago by **kiwitrader** (Community TA)

Now i spotted the difference. If i use `for` loop i'm sending the whole iterator to it so `__next__` in this iterator is used instead of `__next__` in my class. When i use `next()` for looping over my class instance then i get the `TypeError: 'intSet' object is not an iterator`. I initially thought that `for` loop uses same `__next__` function as `next()` and I've checked only behavior of my class with `for`.

Does it mean that this `return iter(self.vals)` solution is a bit cheating ? I mean we don't iterate over our class but we expose iterator that iterates through a our `self.vals` variable. If this is the case then I suppose that it goes with aliasing so there is not actually any memory or speed downfall compared to Your last solution.

Great thanks for sharing Your insights :) !!!

posted 3 months ago by **-Mayster-**

Add a comment

-Mayster-
3 months ago

Well after our discussion and Your contribution to it, I'am now a bit smarter. Thx.

Sorry we could not search the store for annotations

```
def __iter__(self):
    return self

def __next__(self):
    for e in self.vals:
        yield e
    raise StopIteration
```

For me it looks like it is a prefect use case for it. It's almost like Yours with advantage that You don't have to take care of iterator by Yourself. It seems to be more pythonic ;) I'm playing with it for few hours by now but still didn't get close to catch it. It always enters an infinite loop. For my understanding of this code this should work as intended. I don't see any clues why it does not :(

Maybe You could shed some light on it kiwitrader ?

Add a comment

kiwitrader (Community TA)
3 months ago

I've never figured out how to use yield in a class without burying it in a function, I'll have a look. Here's a subclass of list that creates a self resetting list generator.

```
class List(list):
    def __init__(self, e):
        self.extend(e)
        self.idx = -1
    def __iter__(self):
        return self
    def __next__(self):
        try:
            self.idx += 1
            return self[self.idx]
        except IndexError:
            self.idx = 0
            raise StopIteration
```

Add a comment

Sorry we could not search the store for annotations

Showing all responses

Add a response:

Preview

Submit