CS 120: Intro to Computer Programming II

In-Class Activity - Advanced Topics - Hash Tables

Activity 1 - Turn in this one

I've told you that a `set` in Python uses a hash table, just like a dictionary does. Let's try it out, ourselves!

Write a class, `MySet`, which uses a dictionary as its own internal data structure. Start with an empty dictionary.

Any time that a user wants to add an element to the set, it's easy to do: create the mapping `val->True` inside the dictionary. Create an `add()` method to do this - and make sure it exactly matches the definition of the `add()` method of the standard `set` class.

Add a `__contains__(self,val)` method; this is the underlying method that is used when the `in` operator is used. Also add `__len__` and `__str__`.

Investigate what the `union()` method does - (either by doing `help(set.union)` inside Python, or by reading the documentation online). Write a `union()` method for your class. (Remember, it must return a **new** object, different than `self`.)

Solution:

```
class MySet:
    def __init__(self):
        self._data = {}

    def add(self, val):
        self._data[val] = True
    def __contains__(self, val):
        return val in self._data

    def __len__(self):
        return len(self._data)

    def __str__(self):
        tmp = []
        for v in self._data:
            tmp.append(repr(v))
        vals = ", ".join(tmp)
        return '{'+tmp+'}'
```

Activity 2 - Turn in this one

Now, let's test it. Write some simple test code, which creates a set, adds some values to it, prints it out, and confirms that the values are what you expect.

At first, write this using the original Python `set` class. But if you did it correctly, it should be possible to simply replace one line of code:

```
x = set()
```

with

```
x = MySet()
```

and it should work just fine!

Solution:

```
a = set()
assert len(a) == 0
set.add(10)
assert 10 in a
assert 20 not in a
set.add(20)
assert len(a) == 2
set.add(10)
assert len(a) == 2
set.add(-1)   == 3
print(a)                # can't assume what the order of the string is :(

b = set()
b.add(20)
b.add(30)
b.add(40)
print(b)

c = a.union(b)
assert type(c) == type(a)
assert c is not a
assert c is not b
assert len(c) == 5
assert 10 in c
assert 20 in c
assert 30 in c
assert 40 in c
assert -1 in c
assert 40 not in a
assert -1 not in b
```