CS50's Introduction to Programming with Python

OpenCourseWare

Donate (https://cs50.harvard.edu/donate)

David J. Malan (https://cs.harvard.edu/malan/)
malan@harvard.edu
(https://www.facebook.com/dmalan) (https://github.com/dmalan)
(https://www.instagram.com/davidjmalan/) (https://www.linkedin.com/in/malan/)
(https://orcid.org/0000-0001-5338-2522) (https://www.quora.com/profile/David-J-Malan) (https://www.reddit.com/user/davidjmalan)
(https://www.tiktok.com/@davidjmalan) (https://twitter.com/davidjmalan)

Cookie Jar

Source: Sesame Street

Suppose that you'd like to implement a cookie jar (https://en.wikipedia.org/wiki/Cookie_jar) in which to store cookies. In a file called `jar.py`, implement a `class` called `Jar` with these methods:

- `__init__` should initialize a cookie jar with the given `capacity`, which represents the

maximum number of cookies that can fit in the cookie jar. If `capacity` is not a non-negative `int`, though, `__init__` should instead raise a `ValueError`.

- `__str__` should return a `str` with n 🍪, where n is the number of cookies in the cookie jar. For instance, if there are 3 cookies in the cookie jar, then `str` should return "🍪🍪🍪"
- `deposit` should add `n` cookies to the cookie jar. If adding that many would exceed the cookie jar's capacity, though, `deposit` should instead raise a `ValueError`.
- `withdraw` should remove `n` cookies from the cookie jar. Nom nom nom. If there aren't that many cookies in the cookie jar, though, `withdraw` should instead raise a `ValueError`.
- `capacity` should return the cookie jar's capacity.
- `size` should return the number of cookies actually in the cookie jar.

Structure your `class` per the below. You may not alter these methods' parameters, but you may add your own methods.

```
class Jar:
    def __init__(self, capacity=12):
        ...


    def __str__(self):
        ...

    def deposit(self, n):
        ...

    def withdraw(self, n):
        ...

    @property
    def capacity(self):
        ...

    @property
    def size(self):
        ...
```

Either before or after you implement `jar.py`, additionally implement, in a file called `test_jar.py`, **four or more** functions that collectively test your implementation of `Jar` thoroughly, each of whose names should begin with `test_` so that you can execute your tests with:

```
pytest test_jar.py
```

Note that it's not as easy to test instance methods as it is to test functions alone, since instance methods sometimes manipulate the same "state" (i.e., instance variables). To test one method (e.g., `withdraw`), then, you might need to call another method first (e.g., `deposit`). But the method you call first might itself not be correct!

And so programmers sometimes mock (https://en.wikipedia.org/wiki/Mock_object) (i.e., simulate) state when testing methods, as with Python's own mock object library (https://docs.python.org/3/library/unittest.mock.html), so that you can call just the one method but modify the underlying state first, without calling the other method to do so.

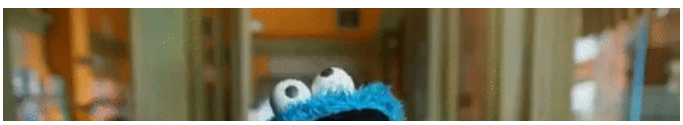
For simplicity, though, no need to mock any state. Implement your tests as you normally would!

▼ **Hints**

```
from jar import Jar


def test_init():
    ...


def test_str():
    jar = Jar()
    assert str(jar) == ""
    jar.deposit(1)
    assert str(jar) == "🍪"
    jar.deposit(11)
    assert str(jar) == "🍪🍪🍪🍪🍪🍪🍪🍪🍪🍪🍪🍪"


def test_deposit():
    ...


def test_withdraw():
    ...
```

Demo

You're welcome, but not required, to implement a `main` function, so this is all we can demo!



Source: Sesame Street

Before You Begin

Log into code.cs50.io (https://code.cs50.io/), click on your terminal window, and execute `cd` by itself. You should find that your terminal window's prompt resembles the below:

```
$
```

Next execute

```
mkdir jar
```

to make a folder called `jar` in your codespace.

Then execute

```
cd jar
```

to change directories into that folder. You should now see your terminal prompt as `jar/ $`. You can now execute

```
code jar.py
```

to make a file called `jar.py` where you'll write your program. You can also execute

```
code test_jar.py
```

to create a file called `test_jar.py` where you can write tests for your program.

How to Test

Here's how to test your code manually:

- Open your `test_jar.py` file and import your `Jar` class with `from jar import Jar`. Create

a function called `test_init`, wherein you create a new instance of `Jar` with `jar = Jar()`. `assert` that this jar has the capacity it should, then run your tests with `pytest test_jar.py`.

- Add another function to your `test_jar.py` file called `test_str`. In `test_str`, create a new instance of your `Jar` class and `deposit` a few cookies. `assert` that `str(jar)` prints out as many cookies as have been `deposit`ed, then run your tests with `pytest test_jar.py`.
- Add another function to your `test_jar.py` file called `test_deposit`. In `test_deposit`, create a new instance of your `Jar` class and `deposit` a few cookies. `assert` that the jar's `size` attribute is as large as the number of cookies that have been `deposit`ed. Also `assert` that, if you deposit more than the jar's `capacity`, `deposit` should raise a `ValueError`. Run your tests with `pytest test_jar.py`.
- Add another function to your `test_jar.py` file called `test_withdraw`. In `test_withdraw`, create a new instance of your `Jar` class and first `deposit` a few cookies. `assert` that `withdraw`ing from the jar leaves the appropriate number of cookies in the jar's `size` attribute. Also `assert` that, if you withdraw more than the jar's `size`, `withdraw` should raise a `ValueError`. Run your tests with `pytest test_jar.py`.

You can execute the below to check your code using `check50`, a program that CS50 will use to test your code when you submit. But be sure to test it yourself as well!

```
check50 cs50/problems/2022/python/jar
```

Green smilies mean your program has passed a test! Red frownies will indicate your program output something unexpected. Visit the URL that `check50` outputs to see the input `check50` handed to your program, what output it expected, and what output your program actually gave.

How to Submit

In your terminal, execute the below to submit your work.

```
submit50 cs50/problems/2022/python/jar
```