An Introduction To Interactive Programing In Python (Part 2)

by Joe Warren, Scott Rixner, John Greiner, Stephen Wong

Quiz 6a – Classes *Skanda S Bharadwaj*

Question 1

Every class definition should include an *initializer* method. What is the name of the initializer method?

Refer to the first object-oriented programming video.

Note: While you can get away with not having an initializer method, doing so almost always implies using techniques beyond the scope of this course or bad program design. So, beginners should always define an initializer method.

Your Answer	Score	Explanation
The same as the name of the class		
`init`		
`_init_` (1 underscore on each side)		

✓ `__init__` (2 underscores on each side)	Correct	10.00
Total		10.00 / 10.00

Question 2

In Python, what is the main difference between a *function* and a *method*?

Your Answer		Score	Explanati
Methods have a parameter named `self`, while functions do not.			
✓ Functions are defined outside of classes, while methods are defined inside of and part of classes.	Correct	10.00	
There is no difference. They are interchangeable terms.			
Methods are defined in built-in library modules, while functions are defined in your own code.			

Total	10.00 / 10.00

Question 3

As an example class, consider the following code from one of the videos:

```
class Character:
    def __init__(self, name, initial_health):
        self.name = name
        self.health = initial_health
        self.inventory = []

    def __str__(self):
        s  = "Name: " + self.name
        s += " Health: " + str(self.health)
        s += " Inventory: " + str(self.inventory)
        return s

    def grab(self, item):
        self.inventory.append(item)
```

```
    def get_health(self):
        return self.health
```

What does the `self` parameter represent?

Your Answer		Score	Explanation
The method that is being defined			
Whatever happens to be passed to it.			
The `Character` class			
✓ An object (instance) of the `Character` class	Correct	10.00	
Total		10.00 / 10.00	

Question 4

Assume you have the following class and method definition, parts of which have been omitted.

```
class My_Class:
    …

    def my_method(self, value1, value2):
        """Assumes its inputs are two values and does something."""
        …

my_object = My_Class()
```

The last line defines the variable `my_object` as an object of `My_Class` class. Which of the following is proper syntax for using the method on this object?

Your Answer	Score	Explanation
`my_method(my_object, 1, 2)`		
`My_Class.my_object.my_method(1, 2)`		
`my_method(My_Class, 1, 2)`		

`My_Class.my_method(my_object, 1, 2)`		
✓ `my_object.my_method(1, 2)`	Correct	10.00
Total		10.00 / 10.00

Question 5

We want to have balls that move around. Which of the following designs represents encapsulation best?

Your Answer		Score	Explanation

class Ball:
  def __init__(self, c, r):
  self.center = c
  self.radius = r
 	
  def get_position(self):
  return self.center
 	
  def set_position(self, new_positi
on):
  self.center = new_position
	Inorrect	0.00	Each ball's data is encapsulated within an object. But, the `move` function breaks the encapsulation by being defined on t outside of the class, but looking inside.

```
# balls : A list of Ball objects
balls = …

def move(ball, move_vector):
    """Changes the position of the gi
ven Ball object by adding the given
vector."""
    position = ball.get_position()
    position[0] += move_vector[0]
    position[1] += move_vector[1]
    ball.set_position(position)
```

```
class Ball:
    def __init__(self, c, r):
        self.center = c
        self.radius = r


# balls : A list of Ball objects
balls = …

def move(ball, move_vector):
    """Changes the position of the gi
ven Ball object by adding the given
vector."""
    ball.center[0] += move_vector[0]
```

```
    ball.center[1] += move_vector[1]
```

```
# centers : A list of points, the balls'
center points
centers = …
# radii : A list of numbers, the balls'
radii
radii = …

def move(ball_number, move_vect
or):
    """Changes the position of the nu
mbered ball by adding the given ve
ctor."""
    centers[ball_number][0] += mov
e_vector[0]
    centers[ball_number][1] += mov
e_vector[1]
```

- ✓ class Ball:

```
    def __init__(self, c, r):
        self.center = c
        self.radius = r

    def move(self, move_vector):
```

```
        """Changes the position of the
ball by the given vector."""
        self.center[0] += move_vector
[0]
        self.center[1] += move_vector
[1]


# balls : A list of Ball objects
balls = …
```

Total	0.00 / 10.00

Question Explanation

Note that Python always allows you to break encapsulation, as in the versions where `move` is defined outside the class, but looks at the data inside. Some other languages (like Java or C++) provide the ability to prohibit such behavior.

Question 6

A common feature in many object-oriented languages is method *overloading*. In this quiz question, you will learn by example what overloading is and whether or not Python supports it.

Turn the following English description into code.

- Start a class definition. We'll call the class `Overload`.

- Define an `__init__` method. Along with the standard `self`, it has one parameter. The method does nothing useful for this example — use the Python do-nothing statement `pass` for the body.
- Define a **second** `__init__` method. Along with `self`, it has two parameters. This method also does nothing useful.

Outside of the class, we want to create two `Overload` objects. If Python supports overloading, you will be able to create an `Overload` object with one argument, and create another `Overload` object with two arguments. Does Python support overloading?

Your Answer		Score	Explanation
✓ No	Correct	20.00	
Yes			
Total		20.00 / 20.00	

Question Explanation

The second definition of `__init__` replaces the first. They can't both be used. So, Python does not support overloading, i.e., having multiple definitions of the same method.

Instead, Python supports very flexible function and method definitions. While we haven't illustrated it for you previously, we could have accomplished the same idea as above with a single method definition.

```
class Overload:
```

```
    def __init__(self, one, two=0):
        """Example of method that takes one required argument and one optional argument."""
        pass


Overload(1)        # Implicitly, we leave the second argument as its default value, 0.
Overload(1,2)
```

While this toy example doesn't do anything useful, it doesn't have any errors.

Question 7

First, complete the following class definition:

```
class BankAccount:
    def __init__(self, initial_balance):
        """Creates an account with the given balance."""
        …
    def deposit(self, amount):
        """Deposits the amount into the account."""
        …
    def withdraw(self, amount):
        """
        Withdraws the amount from the account.  Each withdrawal resulting in a
```

```
        negative balance also deducts a penalty fee of 5 dollars from the balance.
        """
        …
    def get_balance(self):
        """Returns the current balance in the account."""
        …
    def get_fees(self):
        """Returns the total fees ever deducted from the account."""
        …
```

The `deposit` and `withdraw` methods each change the account balance. The `withdraw` method also deducts a fee of 5 dollars from the balance if the withdrawal (before any fees) results in a negative balance. Since we also have the method `get_fees`, you will need to have a variable to keep track of the fees paid.

Here's one possible test of the class. It should print the values 10 and 5, respectively, since the withdrawal incurs a fee of 5 dollars.

```
my_account = BankAccount(10)
my_account.withdraw(15)
my_account.deposit(20)
print my_account.get_balance(), my_account.get_fees()
```

Copy-and-paste the following much longer test. What two numbers are printed at the end? Enter the two numbers, separated only by spaces.

```
my_account = BankAccount(10)
my_account.withdraw(5)
my_account.deposit(10)
my_account.withdraw(5)
my_account.withdraw(15)
my_account.deposit(20)
my_account.withdraw(5)
my_account.deposit(10)
my_account.deposit(20)
my_account.withdraw(15)
my_account.deposit(30)
my_account.withdraw(10)
my_account.withdraw(15)
my_account.deposit(10)
my_account.withdraw(50)
my_account.deposit(30)
my_account.withdraw(15)
my_account.deposit(10)
```

```
my_account.withdraw(5)
my_account.deposit(20)
my_account.withdraw(15)
my_account.deposit(10)
my_account.deposit(30)
my_account.withdraw(25)
my_account.withdraw(5)
my_account.deposit(10)
my_account.withdraw(15)
my_account.deposit(10)
my_account.withdraw(10)
my_account.withdraw(15)
my_account.deposit(10)
my_account.deposit(30)
my_account.withdraw(25)
my_account.withdraw(10)
my_account.deposit(20)
my_account.deposit(10)
my_account.withdraw(5)
my_account.withdraw(15)
```

```
my_account.deposit(10)
my_account.withdraw(5)
my_account.withdraw(15)
my_account.deposit(10)
my_account.withdraw(5)
print my_account.get_balance(), my_account.get_fees()
```

Answer for Question 7

You entered:

Your Answer		Score	Explanation
✓ -60	Correct	7.50	
✓ 75	Correct	7.50	
Total		15.00 / 15.00	

Question 8

We will again use the `BankAccount` class from the previous problem. You should be able to use the same definition for both problems.

15

Of course, a bank with only one account will go out of business, so we want our `BankAccount` class to work correctly with many accounts. Naturally, each bank account should have its own balance, with deposits and withdrawals going to the appropriate account. Similarly, the penalty fees for each account should be kept separate.

```
class BankAccount:
    def __init__(self, initial_balance):
        """Creates an account with the given balance."""
        …
    def deposit(self, amount):
        """Deposits the amount into the account."""
        …
    def withdraw(self, amount):
        """
        Withdraws the amount from the account.  Each withdrawal resulting in a
        negative balance also deducts a penalty fee of 5 dollars from the balance.
        """
        …
    def get_balance(self):
        """Returns the current balance in the account."""
        …
    def get_fees(self):
```

```
        """Returns the total fees ever deducted from the account."""

        …
```

Here's one possible test with multiple accounts. It should print the values 10, 5, 5, and 0.

```
account1 = BankAccount(10)

account1.withdraw(15)

account2 = BankAccount(15)

account2.deposit(10)

account1.deposit(20)

account2.withdraw(20)

print account1.get_balance(), account1.get_fees(), account2.get_balance(), account2.get_fees()
```

Copy-and-paste the following much longer test. What four numbers are printed at the end? Enter the four numbers, separated only by spaces.

```
account1 = BankAccount(20)

account1.deposit(10)

account2 = BankAccount(10)

account2.deposit(10)

account2.withdraw(50)

account1.withdraw(15)

account1.withdraw(10)
```

```
account2.deposit(30)
account2.withdraw(15)
account1.deposit(5)
account1.withdraw(10)
account2.withdraw(10)
account2.deposit(25)
account2.withdraw(15)
account1.deposit(10)
account1.withdraw(50)
account2.deposit(25)
account2.deposit(25)
account1.deposit(30)
account2.deposit(10)
account1.withdraw(15)
account2.withdraw(10)
account1.withdraw(10)
account2.deposit(15)
account2.deposit(10)
account2.withdraw(15)
account1.deposit(15)
```

```
account1.withdraw(20)
account2.withdraw(10)
account2.deposit(5)
account2.withdraw(10)
account1.deposit(10)
account1.deposit(20)
account2.withdraw(10)
account2.deposit(5)
account1.withdraw(15)
account1.withdraw(20)
account1.deposit(5)
account2.deposit(10)
account2.deposit(15)
account2.deposit(20)
account1.withdraw(15)
account2.deposit(10)
account1.deposit(25)
account1.deposit(15)
account1.deposit(10)
account1.withdraw(10)
```

```
account1.deposit(10)
account2.deposit(20)
account2.withdraw(15)
account1.withdraw(20)
account1.deposit(5)
account1.deposit(10)
account2.withdraw(20)
print account1.get_balance(), account1.get_fees(), account2.get_balance(), account2.get_fees()
```

Answer for Question 8

You entered:

Your Answer		Score	Explanation
✓ -55	Correct	3.75	
✓ 45	Correct	3.75	
✓ 45	Correct	3.75	
✓ 20	Correct	3.75	

20

Total	15.00 / 15.00