Worksheet 1 - Solution

Miscellaneous Exercises

Given the following class definition, answer the following questions. It is VERY IMPORTANT to try to answer the questions without running the code through the interpreter. Run the code only to verify your answers.

```
class Person:
  def __init__(self, name="Messi"):
    self.name = name
```

Problem 1:
Does the following code result in an error? if yes, why? and if no, what is the output?

```
p = Person()
print(p.name)
```

Solution:
No, the code is perfectly valid. Even though the __init__ method expects a name argument. There is a default value for that argument.

Problem 2:
What is the output of the following code?

```
p = Person()
p.name = "CR7"
print(p.name)
```

Solution:
The output is CR7

Problem 3:
What is the output of the following code?

```
p = Person("CR7")
```

Continue to the next page...

```
print(Person().name)
```

Solution:
The output is “Messi”
If you look inside the print function, we are constructing a new person instance with the default name attribute and then printing the name attribute. Since “Messi” is the default name attribute, then that’s why it gets printed.

Problem 4:
For the following Python expressions, answer whether they evaluate to True or False.

```
p1 = Person(), p2 = Person(); p1 is p2 # True or False?
p1 = Person("CR7"); p2 = Person("CR7"); id(p1) != id(p2) # True or False?
id(Person) == id(Person()) # True or False?
```

Solution:
1- False because the first Person() constructs an object, and the second one constructs a totally different object. They live in different locations in the memory.
2- True because of the same reason mentioned above. Even though the two objects p1 and p2 have the same attribute value. They are completely two separate objects.
3- False because `id(Person)` returns the location in the memory where the ***class*** `Person` lives, whereas `id(Person())` creates an instance of that class and returns the id of the instance that is created.

Capstone Project: MaxHandWins

Problem 5:
As an exercise, try to brainstorm all the different objects involved in MaxHandWins. There is not one single solution to this and there is usually more than one way to design your classes.

Solution:
Here is an example of how you’d decompose the problem into the various objects involved:

- Player
- Deck
- PlayingCard
- Game

Problem 6:
Create a class `PlayingCard` that represents a playing card object. A playing card can be defined by two attributes:

➢ **`rank`:** This is the number that you see on the playing card.

- **`suit`:** This is one of four categories that represent the suit, or the shape that appears on the playing card. The four categories are <**SPADES**, **HEARTS**, **DIAMONDS**, **CLUBS**>

Solution:

```
from enum import Enum
class Suit(Enum):
    SPADES = 4
    HEARTS = 3
    DIAMONDS = 2
    CLUBS = 1


class PlayingCard:
    def __init__(self, rank, suit):
        self.rank = rank
        self.suit = suit
```

Note: The integer values that I chose for the Suit enums are arbitrary, you can assign any integer values you desire for now. However, these values and how you assign them will be important in the future. For now, anything is fine.

Problem 7:

One of the objects in MaxHandWins is the "player". Define a class `Player` that has the following attributes:

- **`name`:** a string that represents the name of the player.
- **`hand`:** a list that represents the set of cards that this player has.

The `hand` attribute **MUST** be initialized to an empty list.

Solution:

```
class Player:
    def __init__(self, name):
        self.name = name
        self.hand = []
```

Problem 8:

Create a class that represents a Deck of Playing Cards. Let's name this class `Deck`. For the sake of this problem and throughout the project, you can assume that we have a single 52-card deck. Class `Deck` has one attribute `cards` that is a list of `PlayingCard`s. At initialization, this list MUST be initialized with all 52 playing cards.

Solution:

```
class Deck:
    def __init__(self):
        self.cards = []
        for i in range(13):
            self.cards.append(PlayingCard(i+2, Suit.SPADES))
            self.cards.append(PlayingCard(i+2, Suit.DIAMONDS))
            self.cards.append(PlayingCard(i+2, Suit.HEARTS))
            self.cards.append(PlayingCard(i+2, Suit.CLUBS))
```

As you can see in each iteration, we append four playing cards with the same rank but 4 different suits. After 13 iterations, the attribute cards will be successfully initialized.

Also note that we start with PlayingCard(i+2,...) because the rank of the smallest card is 2.