CS303E Week 7 Worksheet: Objects and Classes

Name: _____ EID: _____

Read the questions carefully, and answer each question in the space provided. Use scratch paper to do your work and then copy your answers neatly and legibly onto the test paper. Only answers recorded on the test paper will be graded.

1. (11 points: 1 point each) The following are true/false questions. **Write either T or F in the boxes at the bottom of page 1.** If there's any counterexample, it's false.

   (a) Python allows you to create a class without any attributes or methods.

   (b) Variables defined in a class's `__init__` require the `"self"` prefix to be recognized as an instance attribute.

   (c) All user-defined classes in Python are mutable.

   (d) In Python, if a class's `__str__` method is not defined, you can directly print an object of that class.

   (e) In Python, it is possible to create an instance of a user-defined class without explicitly defining a constructor (`__init__`) method.

   (f) All attributes of a class in Python can be accessed and modified from outside the class.

   (g) Modifying an attribute of one instance of a class affects the same attribute in all other instances of that class.

   (h) When creating a class object, it is necessary to explicitly provide values for all instance attributes. Not true! Remember default parameters? Example: homework 7, the Student class had a default parameter (0) for the exam grades, so we could make a Student by only providing the name.

   (i) In Python class method definitions, `"self"` is a reference to a class instance, and using 'self' allows the method to access the attributes of that particular instance.

   (j) You can change the class of an object after it has been created in Python.

   (k) In Python, if a class defines attributes in its `__init__` method, all instances of that class will have those attributes. I guess technically not true, if you're doing error checking in your __init__ and returning early/before the attributes are created, but for the intent of our class, yes, this is true. =)

a	b	c	d	e	f	g	h	i	j	k
T	T	T	T	T	F	F	F	T	F	.T

Page total: _____/11

Questions 2-7 are multiple choice. Each counts 2 points. **Write the letter of the BEST answer in the box on the next page. Please write your answer in UPPERCASE. Each problem has a single answer.**

2. Why does the `__str__` method in Python need to return strings instead of using `print()` directly?

   2A). print() does rely on __str__, but when you use print within __str__, it doesn't automatically trigger another call to __str__. The print statement simply outputs the string representation returned by __str__ to the console or wherever the output is directed.

   A. Using `print()` within `__str__` would cause an infinite loop, because print implicitly calls `__str__`.

   B. The `__str__` method is used for the creation of the string to print, not for the printing itself.

   Yes! __str__ is used by print() to know what to display. So, if nothing (None) is returned by __str__, Python will be confused!

   C. `print()` can only be used within the constructor, not in methods.

   clearly not true! in HW7, getAverage (a method besides the constructor), we had a print()

   D. Using `print()` in `__str__` is not erroneous but it makes the code less efficient and slower to execute.

   Not true! We need to return a string in a class's __str__, otherwise the code will crash if we try printing an instance of the class.

3. Why are getter or setter methods useful?

   2B) Not true! A setter method takes in a parameter and overwrites an attribute's value with the new value. But for whatever reason, you might make a class with a method that isn't necessarily a setter, but still modifies class attributes. EX: a Playlist class that has a method letting you append something to (not overwrite) a song list

   A. Getter and setter methods are required to define a class. clearly not true!

   B. For an attribute to be changed after it is initialized, a setter method is required.

   C. Getter and setter methods help us understand the structure of a class. clearly not true! we can try reading the code. LOL

   D. Getter and setter methods provide a way to access and modify private attributes of a class.

   Yes! Without getters/setters, it's difficult to access and change private class attributes when needed.

4. Which of the following is true about classes?

   4A) False! Because when you define functions outside of a class, they are just regular functions, not bound to any particular class or object instance. So there is no "self" instance for them to refer to.

   A. `"self"` can be used outside of a class to refer to specific instances.

   B. Defining only the `__gt__` (greater than) and `__ge__` (greater than or equal to) comparisons methods in a class handles all comparisons.

   4B) True! If __gt__ and __ge__ are defined, but the counterparts __lt__ and __le__ are not defined, Python will automatically provide fallbacks for the missing methods by reversing the existing comparisons.

   C. If the `__eq__` method for a class is not explicitly defined, then comparing two instances of this class by doing `X == Y` will cause an error.

   4C) Not true! when __eq__ is not defined for a class, then when we compare two instances, X == Y will be the same as X is Y

   D. Using `type()` on any user-defined class will always return back 'Object'.

   4D) Not true! When we make a new class, we make a new type. Also, object is already a class in Python — it's the 'root' or 'parent' class of everything!

5. When might the result of `X is Y` be different from `X == Y`?

   A. When comparing two lists that have the same elements.

   B. When comparing two strings with the same content stored in different variables.

   C. When comparing two integers having the same value.

   D. All of the above.

   E. Never; they are equivalent expressions.

   F. B and C

   5) Because strings and integers are immutable, when X and Y are both strings (or both integers) the expressions 'X is Y' and 'X == Y' will be the same. What's going on behind the scenes: when we do something like X = "howdy" and Y = "howdy" , because strings (and ints) are immutable, Python will only make a certain string (or int) once, for optimization. And it will reuse it for any variable that happens to take on the same value. So because X and Y have the same value, and refer to the same object, 'X is Y' and 'X == Y' will both be True. But if X and Y were lists, then Python will create distinct/new objects for X and Y, because lists are mutable and can be changed. So even if two lists have the exact same items, they will be distinct objects in memory. So, 'X is Y' will be False, while 'X == Y' will be True.

6. What is the difference between functions and methods?

   Not true — otherwise, we'd just refer to everything as a function. But we don't!

   A. Functions and methods are interchangeable terms in Python.

   B. Functions can only take one parameter, while methods can take multiple parameters. We've seen examples of functions that require multiple parameters, such as max() in the math library.

   C. Functions can be defined independently, while methods are always associated with a class or object. Yes!!! We don't require a class or object to call functions. But if we want to use a method, we either need an instance of a specific class, or the class itself.

   D. Functions return values, while methods perform actions without returning any value. Functions *and* methods can both return values! That's why we have getter methods in classes.

7. Which of the following scenarios benefit from creating a class in Python?

   A. Building a representation of a real-world object with various measurements and behaviors. Yes!! Ex: our Student class

   B. When you want to override or customize "magic methods" in Python, such as `__str__`. Yes!! If we wanted to, we could recreate already existing classes in Python, but customize their methods like __str__ so the class behaves slightly differently.

   C. Grouping together specific data and and multiple functions that act on that data. Yes!!!! This can help modularize our code, and organize things, because each instance would have its own data that it is keeping hold of and modifying.

   D. To create blocks of code that perform a specific task.

   E. All of the above

   F. Both A and B

   G. A, B, and C

We wouldn't need a class for this — we can just make a function.

2	3	4	5	6	7
B	D	B	A	C	G

Page total: _____/12

The following 8 questions require you to trace the behavior of some Python code and identify the output of that code. For each question, write the output for the code segment on the provided line.

```
class Cow:
    def __init__(self, name, milkPerHour):
        self.__name = name
        self.__milkPerHour = milkPerHour

    def makeMilk(self, hoursSpent):
        gallonsMilk = self.__milkPerHour * hoursSpent
        print(gallonsMilk)

    def getSpeed(self):
        return self.__milkPerHour

    def __lt__(self, other):
        return not other.getSpeed() < self.__milkPerHour

    def __eq__(self, other):
        return other.getSpeed() == self.__milkPerHour
```

8. (3 points)

```
moomoo = Cow("MooMoo", 20)
print(str(moomoo.getSpeed()) + moomoo.__name)
```

We're trying to directly access a private class attribute (.__name) here, so this will cause an error.

Error

9. (3 points)

```
moomoo = Cow("MooMoo", 20)
elsie = Cow("Elsie", 25)
print(moomoo == elsie, moomoo < elsie, moomoo > elsie)
```

False True False

The __eq__ method tells us two Cows are equal if their milkPerHour values are the same. moomoo and elsie have different milkPerHour values, so the first expression will be False. However, because moomoo has a milkPerHour of 20, and elsie has 25, moomoo < elsie will be True and moomoo > elsie will be False.

Page total: _____/6

10. (3 points)

```
moomoo = Cow("MooMoo", 20)
milks = moomoo.makeMilk(10)

if milks < 10:
    print("Not much milk...")
else:
    print("Tons of milk!")
```

Note that the makeMilk method doesn't return — it prints. So we spend 10 hours milking moomoo, who makes 20 milks per hour. We calculate 10 * 20, print the result (which is 200), and return None, which is stored in the variable 'milks'. But when we try comparing milk with 10, this will cause an error.

200 Error

```
class Doll:
    def __init__(self, name):
        self.name = name

    def __str__(self):
        return "Hi, " + self.name + "!"

    def dreamhouse():
        print("Come on, Barbie, let's go party!")

    def __add__(self, other):
        return Doll(self.name + " (and " + other.name  + ")")
```

Question 11:
We have two Dolls with the same name (Barbie). The string created when we convert each Doll will be equivalent, because __str__ only relies upon the self.name variable. Thus, the first expression will be True. But the second expression will be false because when __eq__ is not defined for a class, then == will test to see if two objects are the exact same. Because barbie and barbie 2 are two separate instances, barbie == barbie2 will be False.

11. (3 points)

```
barbie = Doll("Barbie")
barbie2 = Doll("Barbie")
print(str(barbie) == str(barbie2), barbie == barbie2)
```

True False

12. (3 points)

```
barbie = Doll("Barbie")
ken = Doll("Ken")

ship = barbie + ken
print(ship)
```

We have two Doll objects. According to the Doll __add__ method, when we add two Dolls, we get back a new Doll that has their names combined, in this form: dollname1 (and dollname2). So we can fill this in for Barbie and Ken, and we obtain "Barbie (and Ken)". When we print this Doll, print calls the __str__ method, which will return "Hi," in front of the Doll's name — "Hi, Barbie (and Ken)!"

Hi, Barbie (and Ken)!

Page total: _____/9

Tricky question! But note that the dreamhouse method does not have the 'self' parameter. This means that we cannot call the dreamhouse method on an instance of Doll, because an instance is 'self', but the dreamhouse method doesn't accept 'self'. So if we want to call this method, we need to do so in an alternate way.

Fun fact: barbie.dreamhouse() is the same as Doll.dreamhouse(barbie)

Maybe that makes it clearer why this causes an error? (because we're implicitly providing the 'self' parameter when the method doesn't allow for one)

13. (3 points)

```
barbie = Doll("Barbie") # this question and the next are
barbie.dreamhouse()     # only meant to get you thinking!
```

Error

14. (3 points)

```
Doll.dreamhouse()
```

Aha!!! This is how we can call the dreamhouse method. We can access it through the class. Because dreamhouse doesn't require an instance, we simply specify the class it is in, and we can call it directly. So, we call methods like dreamhouse class methods!

Come on, Barbie, let's go party!

```
class smileyFace:
    def __str__(self):
        return "=)"
```

15. (3 points)

```
cheery = smileyFace()
print(cheery)
```

This question is intended to show you kiddos that, even when __init__ is not defined for a class, we can still create an instance of that class. Python will simply use the default __init__, which requires no parameters and creates no class attributes.

=)

Page total: _____/9