

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. (10 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) Instance attributes defined in a class's `__init__` require the `self` prefix.
 - (b) All user-defined classes in Python are mutable.
 - (c) All data members/attributes of every class are private by default.
 - (d) Every class must have an explicit `__str__` method.
 - (e) All attributes of a class in Python can be accessed and modified from outside the class.
 - (f) Modifying an attribute of one instance of a class affects the same attribute in all other instances of that class.
 - (g) When creating a class object, it is necessary to explicitly provide values for all instance attributes.
 - (h) You can change the class of an object after it has been created in Python.
 - (i) In Python, if a class defines attributes in its `__init__` method, all instances of that class will have those attributes.
 - (j) The name of the class is used when defining a new instance of the class.

a	b	c	d	e	f	g	h	i	j

Questions 2-7 are multiple choice. Each counts 2 points. **Write the letter of the BEST answer in the box on page 3. Please write your answer in UPPERCASE. Each problem has a single answer.**

2. What best describes why you shouldn't use `print()` inside the `__str__` method?
 - 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.
 - C. `print()` can only be used within the constructor, not within methods.
 - D. Using `print()` in `__str__` is not an error but it makes the code less efficient and slower to execute.
3. Why are getter or setter methods useful?
 - A. Getter and setter methods are required to define a class.
 - 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.
 - D. Getter and setter methods provide a way to access and modify attributes of a class.
4. 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. What is the difference between functions and methods?
- A. Functions and methods are interchangeable terms in Python.
 - B. Functions can only take one parameter, while methods can take multiple parameters.
 - C. Functions can be defined independently, while methods are always associated with a class or object.
 - D. Functions return values, while methods perform actions without returning any value.
6. To be able to write `print(X)` for a class object `X` (and get a useful result) requires this be defined for the class?
- A. `__init__`
 - B. `__str__`
 - C. `__print__`
 - D. `__add__`
 - E. `__eq__`
7. What will be the output of the following Python code?

```
class tester:
    def __init__(self, id):
        self.id = str(id)
        id="224"
```

```
temp = tester(12)
print(temp.id)
```

- A. 12
- B. 224
- C. None
- D. it's an error

2	3	4	5	6	7

The following questions (2 points each) ask 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 in the provided box. Questions may reference one of the class definitions below: Cow, Doll and smileyFace.

```
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

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 + ")")

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

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

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

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

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

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

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

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

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

14. `Doll.dreamhouse()`

15. `cheery = smileyFace()`
`print(cheery)`