Quiz 04 - Practice

COMP 110: Introduction to Programming
SS1 2025

June 10, 2025

Name: ______________________________

9-digit PID: ______________________________

Question 1: Multiple Choice Answer the following questions about concepts covered in class.

1.1. All instances of a class have the same attribute *values*.
- ◯ `True`
- ◯ `False`

1.2. An object's attribute values *cannot* be accessed from outside the class.
- ◯ `True`
- ◯ `False`

1.3. What is the difference between a class and an object?
- ◯ A class is a collection of objects
- ◯ A class is a blueprint; an object is a specific instance of that blueprint
- ◯ They are the same in Python
- ◯ An object can contain classes, but not the other way around

1.4. Because class definitions have attributes, local variables are not allowed inside method definitions.
- ◯ `True`
- ◯ `False`

1.5. What does it mean to "instantiate" a class?
- ◯ Define the class
- ◯ Import a module
- ◯ Create an object from a class
- ◯ Define attributes

1.6. What is the purpose of the `__str__` magic method in Python?
- ◯ To convert an object to a `str` data type.
- ◯ To define how an object should be represented as a string when using `str(<object>)` or `print(<object>)`.
- ◯ To print a string's location ("address") in a computer's memory.
- ◯ To prevent an error from occurring when printing an object.

1.7. The initializer (also called a constructor) of a class is only called once in a program, no matter how many objects of that class are constructed.
- ◯ `True`
- ◯ `False`

1.8. The first parameter of any method is ______ and it is given a reference to the object the method was called on.
- ◯ `me`
- ◯ `self`
- ◯ `init`
- ◯ `this`

1.9. An instance of a class is stored in the:
- ◯ stack
- ◯ heap
- ◯ output

1.10. Why is the type of the `next` attribute in a Node class typically defined as `Node | None`?
- ◯ It ensures the `next` attribute always has a valid Node instance.
- ◯ It allows the `next` attribute to represent the end of a linked list by being assigned `None`.
- ◯ Python requires all attributes to be initialized to None by default.
- ◯ It tells the computer to raise an error if the `next` attribute is `None`.

1.11. What happens if a recursive function does not have a base case?
- ◯ The program compiles but never runs.
- ◯ The function stops automatically after 1,000,000 iterations.
- ◯ The function converts to an iterative process.
- ◯ The function enters infinite recursion and raises a RecursionError.

Question 2: Identifying Elements of a Python Class Consider the following class definition.

```
 1  class Point:
 2    x: float
 3    y: float
 4
 5    def __init__(self, x: float, y: float):
 6      self.x = x
 7      self.y = y
 8
 9    def flip(self) -> None:
10      temp: float = self.x
11      self.x = self.y
12      self.y = temp
13
14    def shift_y(self, dy: float) -> None:
15      self.y += dy
16
17    def diff(self) -> float:
18      return self.x - self.y
```

Bubble in all lines on which any of the concepts below are found. Bubble `N/A` if the concept is not in the code listing.

2.1. Initializer/Constructor Declaration
☐ 1 ☐ 2 ☐ 5 ☐ 9 ☐ 11

2.2. Attribute Declaration
☐ 2 ☐ 3 ☐ 6 ☐ 7 ☐ 10

2.3. Attribute Initialization
☐ 2 ☐ 3 ☐ 6 ☐ 7 ☐ 10

2.4. Method Declaration
☐ 1 ☐ 9 ☐ 10 ☐ 14 ☐ 17

2.5. Local Variable Declaration
☐ 2 ☐ 3 ☐ 6 ☐ 7 ☐ 10

2.6. Instantiation
☐ 1 ☐ 5 ☐ 9 ☐ 10 ☐ `N/A`

Question 3: Using Classes Given the code listing above, use the `Point` class in the next questions.

3.1. Write a line of code to create an *explicitly typed* instance of the `Point` class called `my_point` with an `x` of 3.7 and `y` of 2.3.

3.2. Write a magic method that would cause `print(my_point)` to print `(3.7, 2.3)`, or the attribute values for any other Point object. In other words, the literal values 3.7 and 2.3 should not be written anywhere in your method definition; instead, use the attribute names to access their values. Assume this method would be added inside the Point class (no need to rewrite any of the class).

3.3. Write a line of code to change the value of the `my_point` variable's `x` attribute to 2.0.

3.4. Write a line of code to cause the `my_point` variable's `y` attribute to increase by `1.0` using a *method call*.

3.5. Write a line of code to declare an *explicitly typed* variable named `x`. Initialize `x` to the result of calling the `diff` method on `my_point`.

Question 4: Traversing a Linked List Print the output of the function calls below. Write "Error" if code would result in an error.

```
from __future__ import annotations

class Node:
  value: int
  next: Node | None

  def __init__(self, value: int, next: Node | None):
    self.value = value
    self.next = next

  def __str__(self) -> str:
    rest: str
    if self.next is None:
      rest = "None"
    else:
      rest = str(self.next)
    return f"{self.value} -> {rest}"

sun: Node = Node(4, None)
moon: Node = Node(7, sun)
```

4.1. Print the output.

```
print(moon)
```

4.2. Print the output.

```
print(sun.value)
```

4.3. Print the output.

```
print(moon.next)
```

4.4. Print the output.

```
print(moon.next.next)
```

Question 5: Memory Diagram Trace a memory diagram of the code listing.

```
class Dog:
  name: str
  age: int

  def __init__(self, n: str, a:int):
    self.name = n
    self.age = a

  def speak(self) -> None:
    print(self.name + " says woof!")

  def birthday(self) -> int:
    self.age += 1
    return self.age

class Cat:
  name: str
  age: int

  def __init__(self, n: str, a:int):
    self.name = n
    self.age = a

  def speak(self) -> None:
    print(self.name + " says meow!")

  def birthday(self) -> int:
    self.age += 1
    return self.age

rory: Dog = Dog(n = "Rory", a = 4)
print(rory.birthday())
miso: Cat = Cat("Miso", 2)
miso.speak()
```

Output

Stack

Heap

Globals

Question 6: Memory Diagram Trace a memory diagram of the code listing.

```
class Concert:
  artist: str
  seats: dict[str, bool]

  def __init__(self, a: str, s: dict[str, bool]):
    self.artist = a
    self.seats = s

  def assign_seats(self, wanted_seats: list[str], name: str) -> None:
    for seat in wanted_seats:
      if seat in self.seats:
        available: bool = self.seats[seat]
        if available:
          print(f"{name} bought seat {seat} to see {self.artist}!")
          self.seats[seat] = False
        else:
          print(f"Seat {seat} is unavailable :(")

lenovo_seats: dict[str, bool] = {"K1": True, "K2": True, "K3": False}
show: Concert = Concert(a = "Travisty", s = lenovo_seats)
show.assign_seats(wanted_seats = ["K2", "K3"], name = "Kay")
```

Output

Stack

Heap

Globals

Question 7: Class Definition Writing Write a class definition with the following attributes and methods:

- The class name is `BankAccount`, and it has two attributes: `name`, a `str`, and `balance`, a `float`.
- The `initializer` (also called a `constructor`) has parameters to initialize the `name` and `balance` of an instance of `BankAccount`.
- The `BankAccount` class has a method called `deposit` that adds a specified amount into the `balance` attribute of the `BankAccount` object the method is called on.
- The `BankAccount` class has a method called `withdraw` that will subtract a specified amount from the `balance` attribute of the `BankAccount` object the method is called on *if the* `balance` *is at least the amount to withdraw.* If the balance IS at least the amount to withdraw, return the remaining balance after withdrawal. If the balance is NOT greater than the amount to withdraw, the code should print ``Insufficient funds'' and return a value of `-1.0`.
- Explicitly type variables, parameters, and return types.

The following REPL examples demonstrate expected behavior of an instance of your `BankAccount` class:

```
>>> my_account = BankAccount("Prati", 30.0)
>>> my_account.deposit(10.0)
>>> print(my_account.balance)
40.0
>>> print(my_account.withdraw(5.0))
35.0
>>> print(my_account.withdraw(1000.0))
Insufficient funds
-1.0
```

7.1. Write your class definition here:

Congratulations on prepping for your last COMP 110 quiz!