Quiz 04 Review Session

Content on Quiz 2

- OOP
  - Objects
  - Classes
  - Methods
  - Magic methods
- Recursive Structures (linked lists)

Disclaimer: We haven’t seen the quiz; this review session covers the main topics in the unit.

Classes & Objects

Classes are the blueprints
Objects are instances of a class

Class & Object examples:
University: unc_chapel_hill
Chairs: this_chair
Node: my_node

Attributes and Methods

Attributes are variables that belong to a class
Methods are functions that belong to a class

All objects get the same attributes and methods defined in the class

Methods

Methods are functions you can build into a class, that any object of that class can access

Magic Methods:

- __init__: constructor/initializer of an object
  - called for <Class>()
- __str__: string representation of the object
  - called for str(<object>) or print(<object>)

Recursive Structures

Nodes:

- have a value (e.g. int) and a next (either Node or None)

- likely given an implementation for a Node, and have to answer some questions (similar to the practice quiz)

- You should be able to identify base cases v. recursive cases for recursive structures/functions.

Common errors

RecursionError – Occurs when a recursive function has no base case, creating infinite recursion.

Need an **object** (instance of a class) in order to call a method or access an attribute

Objects are stored in the **heap**

~~Point.x~~

~~Point.y~~

~~Point.dist_from_origin()~~

~~x~~

~~y~~

~~dist_from_origin()~~

You need an **instance** of the class (**an object**)

pt.x

pt.y

pt.dist_from_origin()

```
0 class Point:
1     x: float
2     y: float
3
4     def __init__(self, x: float, y: float):
5         self.x = x
6         self.y = y
7
8     def dist_from_origin(self) -> float:
9         return (self.x**2 + self.y**2) ** 0.5
10
11    def translate_x(self, dx: float) -> None:
12        self.x += dx
13
14    def translate_y(self, dy: float) -> None:
15        self.y += dy
16
17 pt: Point = Point(2.0, 1.0)
```

Basic Diagram Example

```python
class Movie:
    title: str
    watches: int

    def __init__(self, title: str, watches: int):
        self.title = title
        self.watches = watches

    def rewatch(self) -> None:
        self.watches += 1

flow: Movie = Movie("Flow", 1)
dune: Movie = Movie("Dune", 3)
dune.rewatch()
```

```python
class Movie:
    title: str
    watches: int

    def __init__(self, title: str, watches: int):
        self.title = title
        self.watches = watches

    def rewatch(self) -> None:
        self.watches += 1

flow: Movie = Movie("Flow", 1)
dune: Movie = Movie("Dune", 3)
dune.rewatch()
```

Output

Stack

Globals

Movie | id:0

flow | id:1

dune | id:2

Movie#__init__

RA | 12

RV | id:1

self | id:1

title | "Flow"

watches | 1

Movie #__init__

RA | 13

RV | id:2

self | id:2

title | "Dune"

watches | 3

Movie # rewatch

RA | 14

RV | None

self | id:2

Heap

id:0 | class lines 1-10

id:1

Movie	
title	"Flow"
watches	1

id:2

Movie	
title	"Dune"
watches	~~3~~ 4

Challenge Problem!

```python
class Library:
    books: dict[str, bool]

    def __init__(self, books: list[str]):
        """Given a list of books,
            adds the book to the library as available (True)"""
        self.books = {}
        for book in books:
            self.books[book] = True

    def checkout_books(self, requested: list[str], borrower: str) -> int:
        """Allows one borrower to request a list of books,
            returns the number of books left in library."""
        for book in requested:
            if book in self.books:
                self.books[book] = False
                print(f"{borrower} has checked out {book}.")
            else:
                print(f"{book} is not available.")
        return len(self.books)

books: list[str] = ["It", "Q", "1984", "Boy"]
my_lib: Library = Library(books)
my_lib.checkout_books(["It", "Me", "1984"], "Sophie")
```

```python
class Library:
    books: dict[str, bool]

    def __init__(self, books: list[str]):
        """Given a list of books,
            adds the book to the library as available (True)"""
        self.books = {}
        for book in books:
            self.books[book] = True

    def checkout_books(self, requested: list[str], borrower: str) -> int:
        """Allows one borrower to request a list of books,
            returns the number of books left in library."""
        for book in requested:
            if book in self.books:
                self.books[book] = False
                print(f"{borrower} has checked out {book}.")
            else:
                print(f"{book} is not available.")
        return len(self.books)

books: list[str] = ["It", "Q", "1984", "Boy"]
my_lib: Library = Library(books)
my_lib.checkout_books(["It", "Me", "1984"], "Sophie")
```

Output

Sophie has checked out It.

Me is not available.

Sophie has checked out 1984.

Stack

Library | id:0

books | id:1

my_lib | id:2

Library # __init__

RA | 23

RV | id:2

self | id:2

books | id:1

book | ~~"It"~~ ~~"Q"~~ ~~"1984"~~ "Boy"

Library # checkout_books

RA | 24

RV | 4

self | id:2

requested | id:4

borrower | "Sophie"

book | ~~"It"~~ ~~"Me"~~ "1984"

Heap

id:0 | class lines 1-20

id:1 | list[str]

0	~~"It"~~
1	~~"Q"~~
2	~~"1984"~~
3	"Boy"

id:2 | Library

books	id:3

id:3 | dict[str, bool]

~~"It"~~	~~True~~	False
~~"Q"~~	True	
~~"1984"~~	~~True~~	False
"Boy"	True	

id:4 | list[str]

0	~~"It"~~
1	~~"Me"~~
2	"1984"

Class Writing Tips

- attributes:
  - should be declared after class declaration, but initialized in constructor
  - whenever you're accessing the **attributes** of a class – self.<attribute>
- **self** should be the first parameter in every method
- __init__: set each attribute (self.<attribute>) equal to a parameter

```
class Example:
    a: int
    b: str

    def __init__(self, a: int, b: str):
        self.a = a
        self.b = b

    def increment(self) -> None:
        self.a += 1
```

Class Writing Example

- class name is `Car`, and it has two attributes: `mileage`, an `int`, and `gas_percent`, an `int`
- The `initializer` has one parameter to initialize the mileage. On initialization, `gas_percent` should always be set to 100.
- The `Car` class has a method called `drive` that adds a specified number of miles to the `mileage` attribute. Additionally, the gas percentage should decrease by 20 each time the car drives.
- The `Car` class has another method called `check_gas` that has the following behavior:
  - When `gas_percent` >= 50, "All good!" will print.
  - When `gas_percent` is between 10 and 50, "Consider refueling soon!" will print.
  - When `gas_percent` <= 10, "Gas low!" will print.

```
>>> my_car = Car(10000)
>>> my_car.drive(20000)
>>> print(my_car.mileage)
30000
>>> my_car.check_gas()
All good!
```

Class Writing Example

class name is `Car`, and it has two attributes: `mileage`, an `int`, and `gas_percent`, an `int`

```
1   class Car:
2       mileage: int
3       gas_percent: int
```

Class Writing Example

The `initializer` has one parameter to initialize the mileage. On initialization, `gas_percent` should always be set to 100.

```
5       def __init__(self, mileage: int):
6           self.mileage = mileage
7           self.gas_percent = 100
```

Class Writing Example

The `Car` class has a method called `drive` that adds a specified number of miles to the `mileage` attribute. Additionally, the gas percentage should decrease by 20 each time the car drives.

```
 9        def drive(self, miles: int) -> None:
10            self.mileage += miles
11            self.gas_percent -= 20
```

Class Writing Example

The `Car` class has another method called `check_gas` that has the following behavior:

- When `gas_percent` >= 50, "All good!" will print.
- When `gas_percent` is between 10 and 50, "Consider refueling soon!" will print.
- When `gas_percent` <= 10, "Gas low!" will print.

```
13      def check_gas(self) -> None:
14          if self.gas_percent >= 50:
15              print("All good!")
16          elif self.gas_percent <= 10:
17              print("Gas low!")
18          else:
19              print("Consider refueling soon!")
```

```python
class Car:
    mileage: int
    gas_percent: int

    def __init__(self, mileage: int):
        self.mileage = mileage
        self.gas_percent = 100

    def drive(self, miles: int) -> None:
        self.mileage += miles
        self.gas_percent -= 20

    def check_gas(self) -> None:
        if self.gas_percent >= 50:
            print("All good!")
        elif self.gas_percent <= 10:
            print("Gas low!")
        else:
            print("Consider refueling soon!")
```

Extra Practice (similar to practice quiz #3)

Given the previous definition of the Car class:

1. Write a line of code to create an explicitly typed instance of the Car class called my_car, with initial mileage of 20000.
2. Write a magic method so that when print(my_car) is called, the following will be outputted:

   mileage: 20000, gas tank: 100%
3. Write a line of code to increase the mileage of my_car to 40000, using a method call.
4. Write a line of code to change the value of my_car's gas_percent to 30.

```python
class Car:
    mileage: int
    gas_percent: int

    def __init__(self, mileage: int):…

    def drive(self, miles: int) -> None:…

    def check_gas(self) -> None:…

    # 2
    def __str__(self) -> str:
        return f"mileage: {self.mileage}, gas tank: {self.gas_percent}%"
# 1
my_car: Car = Car(mileage=20000)
# 3
my_car.drive(20000)
# 4
my_car.gas_percent = 30
```

Questions?

Other Resources!

- Practice quiz on the course site with answers and explanations
  - We would recommend trying the problems out on your own, then checking your answers
- Office Hours
  - Tomorrow and Wednesday 11 - 5 in SN008