Discussion 7

OOP and String Representation

Antonio Kam

anto [at] berkeley [dot] edu

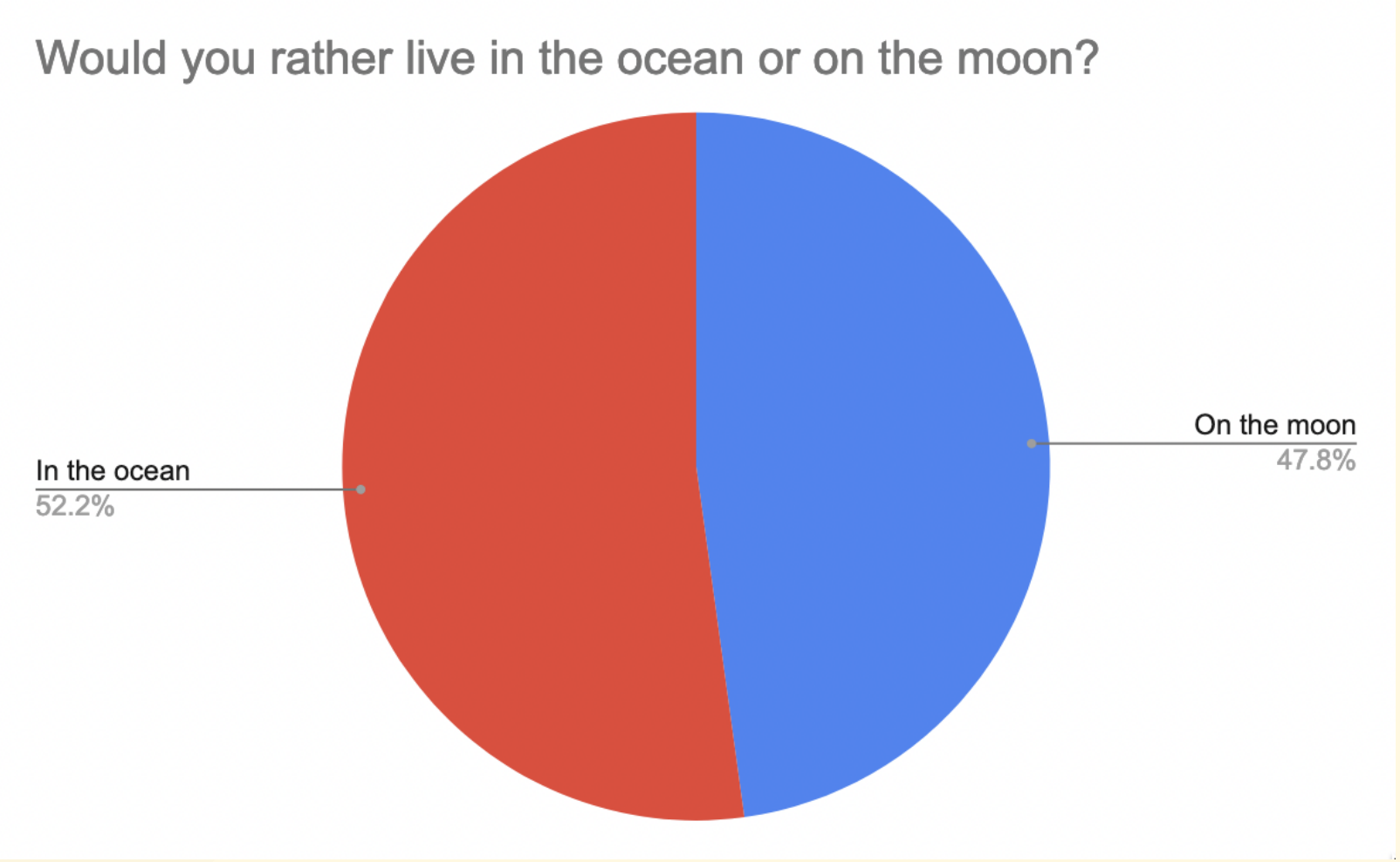
All slides can be found on

`teaching.rouxl.es`

Announcements

- HW 06 Released!
  - OOP/Linked Lists
  - Linked Lists are super important!
- Ants Phase 1 due today
  - Please finish it, the checkpoint is there to keep you on track and is super important
- Discussion Review
  - Will only cover problems that students ask to go over! If you want more review on questions, or want to go faster, discussion review is a great place to go to!
  - Can sign up for this on Ed.

Results from last section



Notes from last section

- OOP/Objects: when would you use OOP over another programming paradigm
  - Good question! OOP is very useful if there's a hierarchical structure to what you want to do (for example, video games tend to have this structure with enemies; you probably saw this with Ants, where implementing new ants took far less effort than implementing the 'first' ant)
- More on super and inheritance
  - This discussion!
- "Maybe ask if there are people working on a specific question midway through lab so that people can work together"
  - Good idea - I'll try something like this next lab
- I really enjoy the mini lectures!
  - 😳

Notes from last section

- I SUMMON POT OF GREED, THIS ALLOWS ME TO DRAW 3 CARDS
  - 🥳 thanks for this comment
  - Also would highly recommend doing the optional questions on the last lab - they're pretty useful for understanding more about how OOP works
- Recursion Practice
  - You'll get... quite a lot on this soon 👀
  - The rest of the course from here is pretty much recursion until the end!

Mid-Semester Feedback

- Sometimes there's very contrasting feedback
  - Some people want discussions to go faster
  - Some people want discussions to go slower
  - I unfortunately cannot accommodate for all of this
  - Discussion review sections!
  - Also, worksheets are designed to be longer than what can be covered
- I speak too fast
  - I knew this was an issue 😭 - I do tend to speak fast when I get excited, which is not a great habit
  - Feel free to scream at me if I'm speaking too fast - you can also ask me to go over something again, and I'm more than happy to do that for you

Mid-Semester Feedback

- Lab mini-lectures are too long
  - You're more than welcome to start doing the lab even while I'm talking
  - Even asking questions is fine: one of the AIs will get to you if it's a lab based question
- More group discussions/engagement
  - Please help me with this 🙄 - would like this to improve too!
  - I might try 'enforcing' something next discussion if there's not enough group discussion here
  - I promise group discussion is useful (Lost on the Moon!)
- Many of you said I have a lot of energy 😳
  - This is something that I don't have ever, unless I'm teaching, which I find super strange 🤔

Temperature Check 🌡️

- OOP
- Inheritance
- Class Methods
- Representation (`repr`, `str`)

Object-Oriented Programming

What is OOP?

- One way I like to think of OOP is as a sort of 'advanced' data abstraction
  - You would use OOP for similar things that you would use data abstractions for
  - Can make a `City` class (similar to the `City` data abstraction you messed around with during Lab)
- OOP also allows for inheritance (less repetition of code, more on this later)
- OOP also allows for mutation
  - Similar to list mutation (`.append`, `.extend`, etc.)
- You may have seen this if you've seen Java before (I didn't have any exposure to OOP when I took CS 61A for the first time)

OOP Terminology

- Class
  - A class is sort of a 'blueprint' for something. You can think of it as a template for creating an object
- Instance
  - An instance of a class is one object of that blueprint, or one physical object that you create based on your template
- Variables
  - Instance Variables: Variables unique to each instance
  - Class Variables: Variables shared between each instance in the same class
- Method
  - Function bound to a class

Functions vs Methods

- Methods need to take in `self` as an argument
  - This is very often *implicitly* passed in when the thing on the left side of the dot is an instance
  - `my_car.drive(100)` is the same as `Car.drive(my_car, 100)`
- `self` refers to the actual instance (rather than the class)
- Two ways of calling methods:
  - `Class.method(self, args)`
  - `instance.method(args)`

Q1 Mini-lecture

Worksheet!

Inheritance

```
class Dog():
    def __init__(self, name, owner):
        self.is_alive = True
        self.name = name
        self.owner = owner
    def eat(self, thing):
        print(self.name + " ate a " + str(thing) + "!")
    def talk(self):
        print(self.name + " says woof!")

class Cat():
    def __init__(self, name, owner, lives=9):
        self.is_alive = True
        self.name = name
        self.owner = owner
        self.lives = lives
    def eat(self, thing):
        print(self.name + " ate a " + str(thing) + "!")
    def talk(self):
        print(self.name + " says meow!")
```

Inheritance

Notice the redundancies in the code? One of the core foundations in this class is to not repeat yourself (DRY)

- Instead, you can use inheritance to solve this problem
- Syntax when creating a class is to put brackets around the class you want to inherit:

```
class Cat(Animal): # Cat inherits the Animal class - as in, all cats are animals
    ...
```

Inheritance

```
class Pet():
    def __init__(self, name, owner):
        self.is_alive = True    # It's alive!!!
        self.name = name
        self.owner = owner
    def eat(self, thing):
        print(self.name + " ate a " + str(thing) + "!")
    def talk(self):
        print(self.name)

class Dog(Pet): # Inherits all methods/variables from the Animal class
    def talk(self):
        print(self.name + ' says woof!')
```

Inheritance - `super()`

- Calling `super()` will refer to the class's *superclass*
- You can use the parent's method and then add on to that.

```
class Cat(Pet): # Inherits all methods/variables from the Animal class
    def __init__(self, name, owner, lives = 9):
        super().__init__(name, owner)
        # same as calling Pet.__init__(self, name, owner) from here
        self.lives = 9
    def talk(self):
        print(self.name + ' says meow!')
```

Worksheet!

Class Methods

```
class Dog(Pet):
    # With the previously defined methods not written out
    @classmethod
    def robo_factory(cls, owner):
        return cls("RoboDog", owner)
```

- Uses the `@classmethod` decorator
- Useful for when you want to create multiple instances of a class based on information you already have
- Also useful when you have a method for a class that may not necessarily be associated with an instance (e.g. `Path.cwd()`)
- For example, in the case above, if I want to create a bunch of dogs with a certain owner, I can use this Class Method

Attendance

`links.rouxl.es/disc`

Worksheet!

Representation

What is Representation?

- Python objects by default have really ugly names if you try to output them into the interpreter:
  - `<__main__.Cat object at 0x7fe611abff70>`
- Representation lets you define a better way to display this out in the console such that other people know what you're talking about

`repr` and `str`

- `repr`
  - Affects what is displayed when object is evaluated in terminal
  - 'Computer readable' (as in, you can use `eval` on something from a `__repr__` method, and it should not error if implemented correctly)
- `str`
  - Changes what is displayed when an object is printed
  - 'Human readable'
- If you directly output to a console, `__repr__` is used, but if you try to print, it will first try and find `__str__`, and if that doesn't exist, it will use `__repr__` instead.

Example

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

    def __repr__(self):
        return "Good " + self.name

    def __str__(self):
        return self.name + " says hello!"
```

- `rex = Pet("Rex")`
- `rex` -> Good Rex (notice how this doesn't have quotes? Python behaviour!)
- `print(rex)` -> Rex says hello!

Worksheet!

Mental Health Resources

- CAPS:
  - If you need to talk to a professional, please call CAPS at 510-642-9494.
- After Hours Assistance
  - For any assistance after hours, details on what to do can be found at this link

Anonymous Feedback Form

`links.rouxl.es/feedback`

Thanks for coming! 🥳

Please give me feedback on what to improve!