COEN 5830, Fall 2024
Introduction to Robotics

Lecture 5
Object-Oriented Programming 1

Leopold Beuken (leopold.Beuken@Colorado.edu)
Tuesday, 9/10/2024

Administrative

- Robotics Seminar this afternoon. Prof. Chahat Singh, 3:30pm, CASE E220

Minimal Perception: Towards the Future of Tiny Autonomous Robots

- Remember to sign up for **Piazza** page!

https://piazza.com/colorado/fall2024/coen5830

- HW1 Posted on Canvas, due **9/17** at **11:59pm**
  - Please answer all questions in a **single** .py file

Objects in Python

- Any variable is handled internally as an **object**
- Values are actually stored in memory is a **reference to an object**
- The **object** contains the **actual value** of the variable
- This is different to other languages that can have variables directly store information as primitive types

```
a = 3
```

Methods for objects

- Data stored in an object can be accessed and modified through **methods**
- A **method** is a function which **operates** on a **specific object** it is attached to. This is a function related to a specific type of object.

```
tout, yout, xout, u, dt = get_data(A,B)
```

Function call

```
my_list = [1,2,3]
my_list.append(4)
```

Method call

Creating Objects

- Some **objects** have special pre-defined syntax to create them:

```
my_string = "vedder"
my_list = [1,2,3]
my_dictionary = {"one": 1, "two:": 2}
```

- Most **objects** require a special initialization function called a **constructor**:

```
from fractions import Fraction
half = Fraction(1,2)
third = Fraction(1,3)
another = Fraction(3,11)
```

“Fraction()” - constructor

A class is the blueprint of an object

- A class contains the structure and functionalities of any object which represents it. A class tells you which **attributes** are associated with an object and what **methods** can be used on that object.

- An **object is the actual instance** of the class blueprint.

- A class **defines** the variables, when an object is created those variables are **assigned values**

- A single class can be used to generate multiple **independent** objects

Accessing object methods vs attributes

- Both **methods** and **attributes** associated with an object instance can be accessed, but are **accessed in slightly different ways**

```
my_num.numerator
```

Accessing attributes

limit_denominator(*max_denominator=1000000*)

Finds and returns the closest `Fraction` to `self` that has denominator at most max_denominator. This method is useful for finding rational approximations to a given floating-point number:

```
my_num.limit_denominator()
```

Accessing methods

You can define your own classes!

- A single class definition should represent a single whole, the contents of which should be linked together in some way.
- A class is defined using the **class** keyword
- **YOU** define which **attributes** and **methods** are associated with your class.
- It is **always** a good idea to define a **constructor** for your class. __init__()
  - This generates an easy way for a client to create an object from your class
  - A constructor can be thought of as special type of method for your class that only gets called once for every object creation

Why OOP?

We often want to group related data together in some data structure:

```python
# LIST

name = "In Search of Lost Typing"
author = "Marcel Pythons"
year = 1992

# Combine these in a tuple
book = (name, author, year)

# Print the name of the book
print(book[0])
```

```python
# TUPLE

name = "In Search of Lost Typing"
author = "Marcel Pythons"
year = 1992

# Combine these in a tuple
book = (name, author, year)

# Print the name of the book
print(book[0])
```

```python
# DICTIONARY

name = "In Search of Lost Typing"
author = "Marcel Pythons"
year = 1992

# Combine these in a dictionary
book = {"name": name, "author": author, "year": year}

# Print the name of the book
print(book["name"])
```

Classes (and related objects) allow us to do this and **add much more functionality through custom methods**