

Class design

Oliver W. Layton

CS151: Computational Thinking: Visual Media

Lecture 30, Spring 2021

Monday April 26

Reminder: Project 8 Scene (Task 3)

- Learning goal: Control drawing system using interpreted characters.
- Only draw parts of your scene using strings interpreted by TurtleInterpreter (e.g. F+F+F+F+ to draw a square).
- Do not use turtle commands directly (forward, right, back, goto, etc), you should only use TurtleInterpreter methods.

Class design: instance variables

What are good instance variables to have?

- Classes are nouns (things). What essential properties define what it means to be **that thing**?

How are instance variables filled in with values?

Two ways instance variables:

1. Initialization data
2. Derived data

Initialization data

Data that you set when you create the object:

```
newStudent = Student(stuff)
```

Think about what you'd like to specify or what you'd find useful when creating a new object. What parameters/information would you feel the urge to pass in?

- Examples: Name, year, dorm, ...

Derived data

Data you calculate based on initialization variables or set at a later point in the program because you don't know the values at the time you create the object.

- Examples: grades, GPA

Define all instance variables in constructor

It is best practice to always define **all** instance variables used throughout the class in the constructor. Even if they are not parameters!

```
class Student():  
    def __init__(self, name, year=2022):  
        # Initialization data  
        self.name = name  
        self.year = year  
  
        # Derived data. Not known yet, but used in class.  
        # Given default values  
        self.grades = []  
        self.gpa = 0.0
```

Calling methods from inside class (object)

We call methods from inside the same class by prefixing `self.` to method name (like accessing instance variables).

```
class Student():  
    def __init__(self, name):  
        self.name = name  
  
    def getName(self):  
        return self.name  
  
    def printName(self):  
        print(self.getName()) # Calling getName method
```

Why the `self` . prefix

Allows us to call method outside the class with same name

```
def getName():  
    return 'Oliver'  
  
class Student():  
    def __init__(self, name):  
        self.name = name  
  
    def getName(self):  
        return self.name  
student = Student('Jane')  
print(student.getName()) # Jane  
print(getName())        # Oliver
```

Let's do the class worksheet

`class_worksheet_elevator_template.py`

Inheritance

Oliver W. Layton

CS151: Computational Thinking: Visual Media

Lecture 30, Spring 2021

Monday April 26

Inheritance: Parent and child classes

Technique to **share code** and **reduce code redundancy** when designing a class highly related to one you already have.

- **Parent class:** Class that you have that is a "broad thing".
- **Child class:** Class that you are designing that is a "more specific thing".

Child class inherits methods from parent class

Child class and objects you make out of it get a **copy** of all the **methods** in the parent (as if you copy-pasted the parent class methods into the child class).

Called **inheriting** parent's methods.

- Example: `lecture30_veggie_inherits.py`
- Method actually stored in the parent class, but the child class and its object have a connection ("arrow") to it.
- Methods defined in child class, but not in parent class, only useable in child class (not parent).
- Let's draw the symbol table for `lecture30_veggie_inherits.py`.