

L-system branching and Class design

Oliver W. Layton

CS151: Computational Thinking: Visual Media

Lecture 28, Fall 2020

Friday October 30

Project 8 overview

- You will update `TurtleInterpreter` class to support drawing tree branches. Then you will draw some trees using L-systems.
- You will add symbols for drawing berries, leaves, and changing color (e.g. trees have different color leaves).
- You will make trees of different sizes (play with tree L-system iterations).
- You will make a scene with your trees!

L-systems that draw trees

Let's discuss how to add symbols to our L-system alphabet to handle drawing tree branches.

We will also discuss why we take char-by-char approach in L-system `replace` method, instead of using the string `replace` method.

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`