Object-Oriented Programming I

CSCI 1030U - Intro to Computer Science
@IntroCS

Randy J. Fortier
@randy_fortier

OntarioTech
UNIVERSITY

Outline

- Motivation
    - What are classes?
    - What are objects?
    - Objects and classes in real life
    - Member variables and functions
- Writing object-oriented programs
    - Python syntax

Classes and Objects

OntarioTech UNIVERSITY

What are Classes?

- A class is an abstract concept
  - e.g. The general concept of a laptop
- A class describes the general properties and behaviours of that concept
  - Properties: Data elements related to the concept
  - Behaviours: Code (functions) related to the concept
- Analogy: A variable declaration (describes name and type, but not value)

What are Classes?

- A class is a type
  - You get to customize that type to the needs of your program
  - A class will define variables that all of its instances will have
  - A class will define functionality that all of its instances will have

What are Objects?

- An object is an instance of some class
  - e.g. Your (specific) laptop
- An object/instance will have specific values for its properties
  - e.g. This laptop has 32GB RAM and a 3.6GHz CPU
- A class can be used to create many instances

Object-Oriented Programs

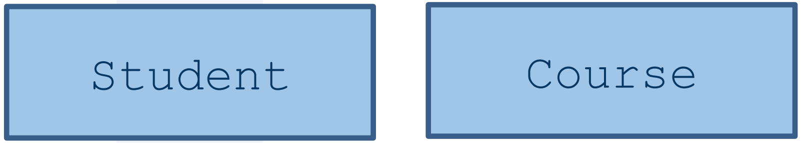
- An object-oriented program is merely a system of interacting objects
  - One object will initiate that interaction
  - Interactions are essentially calling functions
  - An object-oriented program can create instances of classes at any time

Benefits of Object-Orientation

- Benefits of object-orientation:
  - A class is just a larger unit of modularity, so its benefits are similar to functions
    - Except that now we can have behaviour and data, together (related to the same concept) in one place (encapsulation)
    - Single responsibility principle (each class -> one concept)
  - The real world is object-oriented
    - I'm an object performing a behaviour (instruct) on you (another object) now
    - An auto mechanic is an object performing a behaviour (repair) on another object (your car)

Documenting Classes

- One way to document classes is using the Unified Modelling Language (UML):



Student

Course

Class Members

- Classes are made up of two parts:
  - Data
    - Called instance variables or member variables
    - These are just variables, which could hold instances of other classes or merely simple types, attached to a class
  - Behaviour
    - Called member functions or methods
    - These are just functions, attached to a class

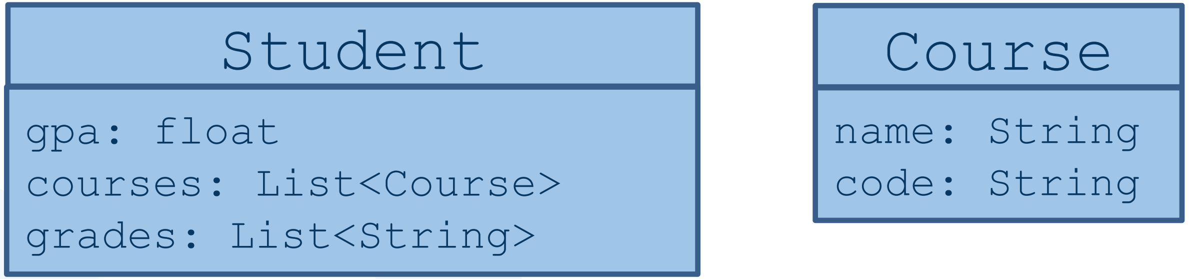
Instance Variables

- Variables associated with instances of a class
- *Scope*: Instance variables exist as long as the object exists
  - Thus, the scope of instance variables is between global and local variables
- *Visibility*: In most programming languages, you can make the variable visible, or invisible, to the outside of the class
  - Private: Visible only to methods in that class
    - Information hiding
  - Public: Visible inside and out
  - Protected: Visible only to methods in that class or a descendant class

Accessing Class Members

- Instance variables are attached to an instance
  - If you have many instances of a class, each has its own instance variable
  - An instance variable lives as long as the object to which it is attached
- Many programming languages use the dot (.) operator (which means member):
  - `some_object.instance_var_name = 7`

Instance Variables in UML



Student

gpa: float
courses: List<Course>
grades: List<String>

Course

name: String
code: String

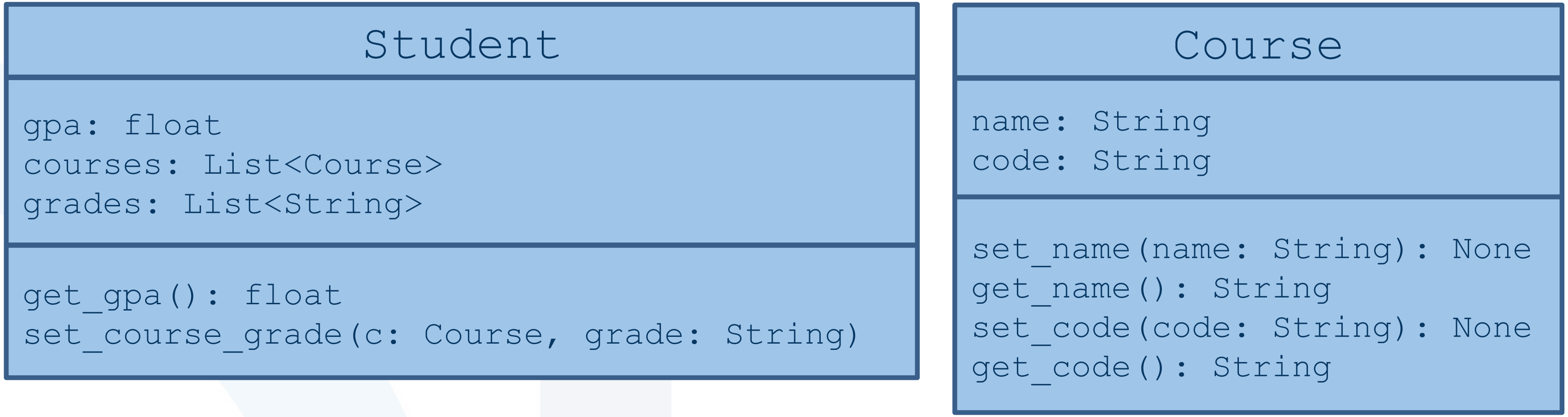
Methods

- Functions associated with instances of a class
  - Methods have access to the instance variables of that instance
- *Visibility*: Like instance variables, methods can have different levels of visibility
  - Private: Can only be called from inside the class
  - Public: Can be called from inside or outside the class
  - Protected: Can be called from inside the class or one of its descendants

Calling Methods

- Unlike calling a function, a method is invoked on an object
  - Many programming languages use the following syntax:
    - `some_object.method_name(arg1, arg2)`
    - In this example, you could also consider `some_object` to be an implicit argument to the method

Methods in UML



```
Student

gpa: float
courses: List<Course>
grades: List<String>

get_gpa(): float
set_course_grade(c: Course, grade: String)
```

```
Course

name: String
code: String

set_name(name: String): None
get_name(): String
set_code(code: String): None
get_code(): String
```

Classes in Python

OntarioTech UNIVERSITY

Writing Classes in Python

```python
class Student:
    def get_average(self):
        return self.average

    def set_course_grade(self, course, grade):
        self.courses.append(course)
        self.grades.append(grade)
        sum = 0
        for g in self.grades:
            sum += g
        self.average = sum / len(self.grades)
```

Writing Classes in Python

```
liza = Student()
intro_to_cs = Course()
intro_to_cs.set_name("Introduction to Computer Science")
liza.set_course_grade(intro_to_cs, 71.25)
print("Liza's Average: ", liza.get_gpa())
```

Constructors

- A constructor is a special method used to initialize the instance variables of a class
  - The constructor is poorly named; it does not construct anything
  - A constructor is called immediately after an instance is created
- In Python, constructors are functions called `__init__`, that do not return any value

Instance Variables

- Instance variables allow each instance of a class to have the same variable, each with their own value
- Instance variables can be public, private, or protected

```
class Student:
   def __init__(self, fname, lname, sid):
        self.first_name = fname # public
        self._last_name = lname # protected
        self.__sid = sid        # private
```

Converting to String

- In Python, `__str__`, is a special method used to determine the string representation of an object
  - This will be the return value when passing one of your objects to the `str()` function
  - The function will create a new string which contains all of the data that you want to be displayed
  - This function is used when you call print

```
course = Course()
print(course)
```

Example

```
class Course:
   def __init__(self, code, name):    # constructor
      self.code = code
      self.name = name

   def get_code(self):                # accessor
      return self.code

   def set_code(self, code):          # mutator
      self.code = code
```

Example

```
def get_name(self):                # accessor
   return self.name

def set_name(self, name):          # mutator
   self.name = name

def __str__(self):                 # string repr.
   return self.code + ' (' + self.name + ')'
```

Calling Constructors

```
intro_to_cs = Course("CSCI 1030U", "Intro to CS")
intro_to_bio = Course("BIOL 1020U", "Intro to Biology")
```

Coding Exercise 06a.1

- Write a class, `Dog` (or `Cat`, if you prefer), that represents a pet dog/cat, and some code to test it
- Instance variables:
  - Name
  - Mass
- Methods:
  - Constructor
  - A string converter (`__str__`), which returns a string representation
  - Less than operator (`__lt__`), which compares by mass

Coding Exercise 06a.2

- Write a class, `Square_Generator`, that implements an iterator
  - The iterator will take a `start_num` and `end_num`, similar to `range()`

e.g.

```
list(Square_Generator(5, 10)) → [25, 36, 49, 64, 81]
```

Hacker's Corner: Data Classes

- If you want data but no functionality, you can use data classes
  - Functionally similar to dictionaries
  - Data classes are similar to `struct` in C++

```
from dataclasses import dataclass

@dataclass
class Student:
    sid: str
    first_name: str
    last_name: str

priya = Student('100000001', 'Priya', 'Agarwal')
```

Wrap-up

- Motivation
  - What are classes?
  - What are objects?
  - Objects and classes in real life
  - Member variables and functions
- Writing object-oriented programs
  - Python syntax

Coming Up

- Inheritance
- Method resolution
- Multiple inheritance