CMSC201
Computer Science I for Majors
Lecture 16 – Classes and Modules

Prof. Katherine Gibson
Last Class We Covered

• Review of Functions
• Code Design
  – Readability
  – Adaptability
• Top-Down Design
• Modular Development
Any Questions from Last Time?
Today’s Objectives

• To reinforce what exactly it means to write “good quality” code
• To learn more about importing
• To better understand the usefulness of modules
• To learn what a class is, and its various parts
  – To cover vocabulary related to classes
  – To be able to create instances of a class
“Good Code”

• If you were to ask a dozen programmers what it means to write good code, you would get a different answer from each.

• What are some characteristics that we have discussed that help you write “good code?”
8 Characteristics of Good Code

1. Readability
   – As we previously discussed, writing code that is easy to understand what it is doing

2. Adaptability (or Extensibility)
   – Relates to how easy it is to change conditions or add features or functionality to the code

3. Efficiency
   – Clean code is fast code

From: http://www.codeexcellence.com/2012/05/8-must-have-characteristics-for-writing-quality-code/
8 Characteristics of Good Code

4. Maintainability
   – Write it for other people to read!

5. Well Structured
   – How well do the different parts of the code work together? Is there a clear flow to the program?

6. Reliability
   – Code is stable and causes little downtime
8 Characteristics of Good Code

7. Follows Standards
   – Code follows a set of guidelines, rules and regulations that are set by the organization

8. Regarded by Peers
   – Good programmers know good code
   – You know you are doing a good programming job when your peers have good things to say about your code and prefer to copy and paste from your programs
Importing and Modules
Reusing Code

• If we take the time to write a good function, we might want to reuse it later!

• It should have the characteristics of good code
  – Clear, efficient, well-commented, and reliable
  – Should be extensively tested to ensure that it performs exactly as we want it to
  – Reusing bad code causes problems in new places!
Modules

• A *module* is a Python file that contains definitions (of functions) and other statements
  – Named just like a regular Python file: `myModule.py`

• Modules allow us to easily reuse parts of our code that may be generally useful
  – Functions like `isPrime(num)` or `getValidInput(min, max)`
Importing Modules

• To use a module, we must first *import* it.

• There are three different ways of importing:
  
  ```python
  import somefile
  from somefile import *
  from somefile import className
  ```

• The difference is *what* gets imported from the file and *what name* refers to it after importing.
import

• In Lab 9, when we practiced using pdb (Python debugger), we used the import command

  import pdb

• This command imports the entire `pdb.py` file
  – Every single thing in the file is now available
  – This includes functions, classes, constants, etc.
import

• To use the things we’ve imported this way, we need to append the filename and a period to the front of its name (”myModule.”)

• To access a function called myFunction:

myModule.myFunction(34)

• To access a class method:

myModule.myClass.classMethod()
from someFile import *

• Again, everything in the file someFile.py gets imported (we gain access to it)
  – The star (*) means we import every single thing from someFile.py

• Be careful!
  – Using this import command can easily overwrite an existing function or variable
from someFile import *

• When we use this import, if we want to refer to anything, we can just use its name

• We no longer need to use “someFile.” in front of the things we want to access

  myFunction(34)
  myClass.classMethod()

• These things are now in the current namespace
from someFile import X

• Only the item X in someFile.py is imported
• After importing X, you can refer to it by using just its name (it’s in the current namespace)
• But again, be careful!
  – This would overwrite anything already defined in the current namespace that is also called X
from someFile import X

from myModule import MyClass

• We have imported this class and its methods
  MyClass.classMethod()

• But not the other things in myModule.py
  myFunction(34) (not imported)

• We can import multiple things using commas:
  from myModule import thing1, thing2
Where to Import From?

• Where does Python look for module files?
  – In the current directory
  – In a list of pre-defined directories

• The list of directories where Python will look for files to be imported is called \texttt{sys.path}
  – To add a directory to this list, append it

  \texttt{sys.path.append('}/my/new/path'\texttt{)}
The `sys.path` Variable

- The "**path**" variable is stored inside the "**sys**" module (the "system" module)
- We can see what it contains like so:

```python
>>> import sys
>>> sys.path
```

this means to look in the current directory
Object Oriented Programming: Defining Classes
Classes

• A *class* is a special data type which defines how to build a certain kind of object.
• The *class* also stores some data items that are shared by all the instances of this class.
• Classes are blueprints for something.
• *Instances* are objects that are created which follow the definition given inside of the class.
Classes

• In general, classes contain two things:

1. Attributes of an object (data members)
   • Usually variables describing the thing

2. Things that the object can do (methods)
   • Usually functions describing the action
Class Parts

• **Data member:** A class variable or instance variable that holds data associated with a class and its objects.

• **Method:** A special kind of function that is defined in a class definition.
Instances of a Class

• **Object:** A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods.
Class Description

• If a class describes a thing, we can think about it in terms of English
  – Object -> Noun
  – Attribute -> Adjective
  – Method (Function) -> Verb
Class to build dogs

Class Example

class Dog:
    def __init__(self, name):
        self.name = name
        self.tricks = []  # creates a new empty list for each dog

    def add_trick(self, trick):
        self.tricks.append(trick)

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.add_trick('roll over')
>>> e.add_trick('play dead')
>>> d.tricks
['roll over']
>>> e.tricks
['play dead']

From: https://docs.python.org/2/tutorial/classes.html
class Dog:
    def __init__(self, name):
        self.name = name
        self.tricks = []  # creates a new empty list for each dog

    def add_trick(self, trick):
        self.tricks.append(trick)

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.add_trick('roll over')
>>> e.add_trick('play dead')
>>> d.tricks
['roll over']
>>> e.tricks
['play dead']

From: https://docs.python.org/2/tutorial/classes.html
Defining a Class

• Instances are objects that are created which follow the definition given inside of the class
• Python doesn’t use separate class interface definitions as in some languages
• You just define the class and then use it
Everything an Object?

• Everything in Python is really an object.
  – We’ve seen hints of this already...
    “hello”.upper()
    list3.append(‘a’)
  – New object classes can easily be defined in addition to these built-in data-types.
• In fact, programming in Python is typically done in an object-oriented fashion.
Methods in Classes

• Define a *method* in a *class* by including *function* definitions within the scope of the class block
• There must be a special first argument *self* in *all* of method definitions which gets bound to the calling instance
• There is also usually a special method called *__init__* in most classes
• We’ll talk about both later...
class student:
    def __init__(self, n, a):
        self.full_name = n
        self.age = a
    def get_age(self):
        return self.age
Using Class Student

def main():
    a = student("John", 19)
    print(a.full_name)
    print(a.get_age())
main()

Create new student object \(a\) with name “John”, age 19

Print an attribute of the student

Call a method of student

Output:

bash-4.1$ python class_student.py
John
19
bash-4.1$
Any Other Questions?
Announcements

• Midterm Survey (on Blackboard)
  – Due by Friday, November 6th at 8:59:59 PM

• Project 1 is out
  – Due by Tuesday, November 17th at 8:59:59 PM
  – Do NOT procrastinate!

• Next Class: Objects Continued