CMSC201
Computer Science I for Majors

Lecture 26 – Python and emacs Fun

All materials copyright UMBC and Dr. Katherine Gibson unless otherwise noted

Today's Objectives

- Review the important points of classes
  - Classes will be on the exam
- Learn some cool Python stuff
  - Importing libraries to do tasks for you
  - (Pseudo) random numbers
- Emacs shortcuts!

Review of Classes

In-Class Exercise

- Labelling the parts of a class!

- Partial list of answers:
  - Class name
  - Constructor
  - Method
  - Attribute
  - Object the method is called on
  - Keyword to create class

Built-In Functions

- Classes have two important built-in functions
  - Have double underscores on either side of name

`__init__`

- Constructor for the class
- Initializes and creates attributes

`__str__`

- Creates the string representation of the object
- Used when we call **`print()`** with an instance

Abstraction and Encapsulation

- All programming languages provide some form of ***abstraction***
  - Hide the details of implementation from the user
  - All the user needs to know is the name and basics
- ***Encapsulation*** is a form of information hiding and abstraction used in classes
  - Data and functions that act on that data are located in the same place (inside a class)

The `self` Variable

- The **`self`** variable is how we refer to the ***current instance*** of the class
  - In **`__init__`**, **`self`** refers to the object that is currently being created
  - In other methods, **`self`** refers to the instance the method was called on

```
def speak(self):
    print("\"" + str(self.species) + " noise\"")
```

Inheritance

- ***Inheritance*** is when one class is based upon another class (child inherits from parent)
- The child class ***inherits*** most or all of its features from the parent class it is based on
  - Inherits both methods and attributes
- Child class can ***extend*** and ***override*** the methods from the parent class
  - What do each of these mean?

Python Fun!

Importing Modules

- A ***module*** is a Python file that contains function definitions and other statements

- To import modules, use this command:

```
import moduleName
```

- This imports the entire module of that name
  - Every single thing in the file is now available
  - This includes functions, data types, constants, etc.

Calendar Module Example

```
import calendar
exCal = calendar.TextCalendar()
printCal = exCal.formatmonth(2017, 5)
print(printCal)
```

```
      May 2017
Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

`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 ("**`moduleName.`**")

- To access a function called **`function`**:

```
moduleName.function()
```

“Random” Numbers

Random Numbers

- Random numbers are useful for many things
  - Like what?
  - Cryptography
  - Games of chance
  - Procedural generation
    - Minecraft levels, snowflakes in Frozen
- Random numbers generated by computers can only be ***pseudo*** random

Pseudo Randomness

- "Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin." – *John von Neumann*

- Pseudorandom appears to be random, but isn't
  - Mathematically generated, so it can't be
  - Called a Random Number Generator (RNG)

Seeding for Randomness

- The RNG isn't truly random
  - The computer uses a "seed" in an attempt to be as random as possible
- By default, the seed is the system time
  - Changes every time the program is run
- We can set our own seed
  - Use the **`random.seed()`** function

Seeding for Randomness

- Same seed means same "random" numbers
  - Good for testing, allow identical runs

```
random.seed(7)
random.seed("hello")
```

- 7 always gives .32, .15, .65, .07
- "hello" always gives .35, .66, .54, .13

How Seeds Work

- "Resets" the random number generator each time it is seeded

- Should only seed once per program
- Seeding and calling gives the same number

```
>>> random.seed(3)
>>> random.random() 0.2379646270918913 7
>>> random.seed(3)
>>> random.random() 0.2379646270918913 7
```

Generating Random Floats

- **`random.random()`**
- Returns a random float from 0.0 up to (but not including) 1.0

```
>>> random.seed(201)
>>> random.random()     0.06710225875940379
>>> random.random()     0.3255995543326774
>>> random.random()     0.0036753697681032316
>>> random.random()     0.28279809896785435
```

Generating Random Integers

- **`random.randrange()`**
- Works the same as normal **`range()`**
  - Start, stop, and step

```
>>> random.seed("dog")
>>> random.randrange(2, 21, 4)   14
>>> random.randrange(2, 21, 4)   6
>>> random.randrange(2, 21, 4)   10
>>> random.randrange(2, 21, 4)   10
>>> random.randrange(6)          5
>>> random.randrange(6)          4
```

Generating Random Options

- **`random.choice()`**
- Takes in a list, returns one of the options at random

```
>>> dogs = ["Yorkie", "Xolo", "Westie",
"Vizsla"]
>>> random.seed("yay, summer!")
>>> random.choice(dogs)      'Yorkie'
>>> random.choice(dogs)      'Xolo'
>>> random.choice(dogs)      'Yorkie'
>>> random.choice(dogs)      'Westie'
```

GL and emacs Shortcuts

Announcements

- Final is Friday, May 19th from 6 to 8 PM
  - Start studying now!
  - Review worksheet won't come out until Saturday
- Final exam locations:
  - Gibson (2, 3, 4, 5, 15, 16, 18) in ENGR 027
  - Wilson (9, 10, 11, 12, 20, 21, 22, 23) in MEYR 030
- Project 3 due on May 12th @ 8:59:59 PM

What do you want to learn?

Common Tasks

- Moving around the file
- Copying, cutting, and pasting
- Searching and replacing
- Advanced commands
  - (Un)comment region
- “Meta” (escape)
- GL vs bash