The Rest of the Semester

April 19, 2021

Administrative Notes

Test 2 grades - we'll go over that

Project 2 is due in one week

- We'll spend some time on that

Then we'll talk about last week's lab & classes

Then we'll start talking about importing modules and using Jupyter Notebooks

- They're key to Project 3
- Project 3 assigned 4/26; due 5/12

The rest of the semester (tentative; as of today)

Monday, 4/19 - today's lecture; Exam 2; Project 2; classes; intro to importing code & Jupyter Notebooks (for project 3)

Wednesday, 4/21 - binary and hexadecimal numbering systems

Monday, 4/26 - Project 2 due; Project 3 assigned; more on importing code; Jupyter Notebooks; graphics

Wednesday, 4/28 - Sorting/Searching/algorithm analysis & complexity

Monday, 5/3 - Homework 6 assigned; algorithm analysis and more on Project 3

Wednesday, 5/5 - More on project 3; graphics

Monday, 5/10 - Homework 6 due; Project 3; course wrap up; higher-level Python topics

Wednesday, 5/12 - Project 3 due; course wrap-up; review for Final

Monday, 5/17, 3:30 - 5:30 - Final Exam - on Blackboard; like the other two exams except longer

Classes/Lab 10 stuff

Classes

“Classes”, “objects”, “object-oriented programming” - buzzwords, but what are they really?

Abstractions that make programming easier

Help programmers think about ‘objects’ with properties to which they can relate

- A “car” has properties like engine type; speed; fuel consumption; number of seats; …
- It can also have properties like “who’s the driver; who are the passengers”

We want to implement “objects” to make programming easier

Done slightly differently in each programming language

Python is somewhat unique in that it is not inherently an object-oriented language

- Classes are not a fundamental part of Python
- You can do a lot of programming in Python without ever getting into classes/objects - and most people do!!

But since this is a first programming course, students need to be introduced to the topic

Classes in Python

First you have to define a class. Use the reserved word class, followed by the name of the class - any valid Python variable name. In CMSC 201, we use UpperCamelCase for class names, to make it apparent to the reader

```
class MyClass:
    """A simple example class"""
    i = 12345


    def f(self):
        return 'hello world'
```

Class instantiation

To create an object that is an instance of a class, use an assignment statement

```
x = MyClass()
```

But that produces an empty object, which is not really useful. So Python defines a “constructor method” that lets you create a new object with properties that you want

```
class Car:
   def __init__(self):
       self.make = 'Toyota'
       self.model = 'Camry'
       self.vin = '123412341234'
       self.license_plate = '3AB1234'
```

Two underscore characters

From lab 10

Class “car”

(from the previous slide)

That code will let you define an object that is of class ‘car’ and has the properties that you want.

But that means every car you create is a Toyota Camry with *that* VIN and *that* license plate - which is likely not what you want

So you can define the class using a skeleton

Skeleton:

```
class Car:
   def __init__(self, make, model, vin, license_plate):
       self.make =  make
       self.model = model
       self.vin =  vin       self.license_plate = license_plate
```

Now you can create a car:

It looks like a function call

my_car = Car.(‘Toyota’, ‘Corolla’, ‘12345’, ‘GoDogs’)

The arguments get mapped to the variables in the class

your_car = Car.(‘Honda’, ‘CRV’, ‘98765’, ‘GoTerps’)

Accessing those variables

```
my_car.make

your_car.model

my_car.vin
```

Methods

Functions can be defined inside of classes to operate on the variables and values internal to the class - these are called “methods” - you’ve seen that word before!!!

```
class Passenger:
   def __init__(self, name):
       self.name = name
```

```
class Car:
   def __init__(self, make, model, vin, license_plate
       self.make = make
       self.model = model
       self.vin = vin
       self.license_plate = license_plate
       self.passengers = []
```

A method to add a passenger to a c

```
def add_passenger(self, passenger):
   self.passengers.append(passenger)
```

Add a passenger to my_car

```
eric = Passenger('Eric')
my_car.add_passenger(eric)
```

Available Code

One of the big advantages of using Python is that there is a ton of code that exists “in the wild” that is available for your use.

- Math library - https://docs.python.org/3/library/math.html - math functions
- Numpy - https://numpy.org/ - arrays; linear algebra; FFT; random numbers; …
- Pandas - https://pandas.pydata.org/ - data analysis/data science
- Pillow - https://pypi.org/project/Pillow/ - image manipulation
- Matplotlib - https://matplotlib.org/index.html - graphics and plots
- Ggplot - http://ggplot.yhathq.com/ - more plots

See https://www.ubuntupit.com/best-python-libraries-and-packages-for-beginners/ for more

Importing Code

You can import any of these existing libraries into your Python program

You can also import any code you’ve previously written yourself (or that your professor or classmate has written)

If the package has already been installed, just use the import statement:

```
  import math  #imports the existing math library; you can now use any of the
               #functions in that library
  import hailstone #imports the hailstone.py program from last week and lets you
                   #use the “flight” function
```

There are multiple formats

Basic:

```
import math
```

Means you must include the module name in each function call:

```
x = math.sin(0)  # if you just said “sin” the Python interpreter wouldn’t know what you meant
```

Rename:

```
import math as m
```

Now you can call the functions with a simpler name: x = m.sin(o)

Import formats

Only import the functions you need:

```
from math import sin
```

Then you don't have to refer to the module - x = sin(0) # valid

Or import all functions with the * wildcard character:

```
from math import *
```

Now you can refer to all the math functions without referring to the module

Just make sure you don't have another function with the same name!!!!

An example using the pandas module

You can read in and process a .csv file with a single statement:

First, install pandas

Then in your program:

```
import pandas as pd
```

Now read in the file:

```
df =  pd.read_csv(“Spring_2022.csv”)
```

And we can easily drop those pain-in-the-neck rows of commas:

```
df.dropna()
```

Importing your own code

We’ll write a program called “dates.py” that includes a function called “convert_date.”

Then we’ll write a separate program that imports “dates” and calls that function.

The separate program is:

```
import dates
if __name__ == “__main__”:
        print(dates.convert_date(‘07042020’))
```

```
from dates import *
if __name__ == “__main__”:
    print(convert_date(‘07042020’))
```

That’s it - really!!!

‘The rules’ for importing your own code:

1. The name of the module MUST end in ‘.py’
2. The module must be in the Python path so that it can be found
   a. Be in the same directory as the calling program
   b. Be in a directory that Python always searches for programs
   c. Include the entire path to the file in the import statement

The Jupyter Notebook

https://jupyter.org/

Formerly the “iPython Notebook”

- A format for producing a document including Markdown language, executable code, and code results
- An ordered series of cells. You specify the format of each cell
- “A Jupyter Notebook can be converted to a number of open standard output formats (HTML, presentation slides, LaTeX, PDF, ReStructuredText, Markdown, Python) “