Python for Probability

CS 109 SPRING 2020

Slides by Julie Wang

Spring 2020

Contents

Overview and Installing Python

- What is Python
- Installation + Running Python

Modules

- What is a module
- Installing a module
- Important Modules for this class

Python Language Basics

- Syntax
- Functions

Setting up for PSET #1

Overview and Installing Python

SSSS

What is Python

Python

- High level, interpreted, and general-purpose dynamic programming language.

PRO’s:

- free, readable, interpreted, object oriented, extensible.

CON’s:

- Slower execution due to interpretation
- dynamic typing can lead to runtime errors
- Clunkier support for arrays (compared to MATLAB or Julia)

A Brief Timeline

- Python 2.7 released 2009. Will be supported until 2020 (uh oh)
- Python 3.0 released 2008, broke backwards compatibility with 2.x
- Python 3.7 (2018) and Python 3.8 (2019, latest version)

We will be using Python 3.7+

Installation

Follow the instructions on the CS 109 page. Two options:

1. Local
   a. Install/Locate terminal program – easiest way to install and run python
   b. Install Python 3.7 interpreter (program that knows how to read python files)
   c. Install an Integrated Development Environment, aka IDE. (program to make editing and development easy)
2. Remote with REPL.it
   a. PRO's: no need to set up on your machine
   b. CON's: Need internet connection

How to run in both environments

- `python3 hello.py`

Modules

DEFINITION AND USE

What is a module?

Certain functions are always defined, `split()`, `len(<list>)`. But to access more functions, must import their module

- `import <module name>` is equivalent to `#include <name>.h` in C and C++

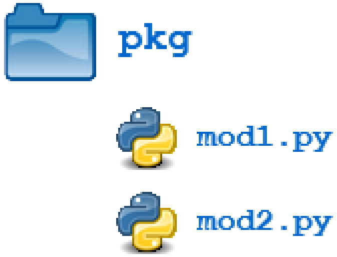
What is a module?

- A file containing functions, definitions, and/or executable statements.
- The module name is just the file name with .py removed.
- To import:

```
# Method 1 (Recommended)
import module1 as mod1
mod1.some_func()

# Method 2
import module1
module1.some_func()

# Method 3
from module1 import *
some_func()
```



What is a package?

- A directory structure containing modules. Can be nested.

Installing Modules

Python comes with a library of standard modules. `math`, `random`, and `os` are some common ones. But not all modules come preinstalled, so you must install them onto your computer using `pip`.

`pip`

- The Python package management tool
- To get a package, must first install it with `pip`, then will have access to its modules using `import`
- For example, to install numpy, scipy, and matplotlib packages, run this in your terminal:
  - `pip3 install numpy scipy matplotlib`

Want more? Read the docs

Important Modules

For this class, we will make use of scipy, numpy, and matplotlib.

Scipy library

- Fundamental library for scientific computing
- Collection of numerical algorithms and domain-specific toolboxes, including signal processing, optimization, statistics, etc

Numpy

- Multidimensional arrays
- Fundamental package for numerical computation. It defines the numerical array and matrix types and basic operations on them.

Matplotlib

- Plotting library

All are distributed by scipy.org.

Python Language Basics

LET'S SEE SOME STUFF

Python Language Basics

Numbers

```
# python has 2 numeric types: int and float
3        # 3 (int)
3.0      # 3.0 (float)
5 / 2    # 2.5 (float division)
5 // 2   # 2 (integer division)
5 % 2    # 1 (integer modulus)
5 ** 2   # 25 (exponentiation)
```

Strings

```
# Both ‘’ and “” create string literals
# no char in Python

A = “Hello”
B = ‘world’

# String concatenation done with +
C = A + “ “ + B + ‘!” # Hello world!
```

Adapted from Sonja’s Fall 2019 CS 109 Slides

Python Language Basics

Comments

```
# This is a single line python comment

“””
This is a multiline
Python comment.
“””
```

Variables

```
    my_number = 0
    my_string = “hi”
    my_other_string = ‘hello’
```

- No semicolons
- Dynamically-typed. i.e. not declared with explicit type

Adapted from Sonja’s Fall 2019 CS 109 Slides

Python Language Basics

Lists

```
# Lists are the most basic type of data storage
a = [10, 20, 30]
b = [0] * 4                    # creates [0, 0, 0, 0]
c = [x for x in range(11, 14)] # creates [11, 12, 13]

# Manipulation
a[0]      # returns 10
a[2]      # returns 30
a[3]      # ERROR! List index out of range
len(a)    # Returns 3
a + [40]  # appends 40 to a to get: [10, 20, 30, 40]
a[-1]     # returns 30
```

Adapted from Sonja’s Fall 2019 CS 109 Slides

Python Language Basics

Math

```
+      # Addition
-      # Subtraction
*      # Multiplication
/      # Division
%      # Modulo
**     # Exponentiation
```

One line Math

```
# makes a into 4
a = 2
a += 2
#makes a back into 2
a -= 2
# Multiplies by -300
a *= -300
```

Adapted from Sonja’s Fall 2019 CS 109 Slides

Python Language Basics

Boolean Operators

```
not       # Instead of ! or ~ or ^ or …
or        # Instead of || or |
and       # Instead of &&
==        # Equals operator
True      # True, note capitalization
False     # False
```

Bitwise Operators

```
x << y    # Return x shifted to the left y places. Fill 0
x >> y    # Return x shifted to the right y places.
x & y     # Bitwise and
x | y     # Bitwise or
~ x       # Bitwise complement
x ^ y     # Bitwise xor
```

Adapted from Sonja's Fall 2019 CS 109 Slides

Python Language Basics

Printing

```
#method 1
print(“a number: {}”.format(42))
print(“a float: {:3.2f}”.format(2.345))

#method 2
print(“a number: ” + str(42))
# this will not work
# print(“a number: ” + 42)
```

Adapted from Sonja’s Fall 2019 CS 109 Slides

Python Language Basics

if/elif/else

```
#To make if, else if, and else:
a = 4
if a < 0:
    print(“a is negative”)
     print(‘ this will also execute’)
elif a >= 0:
    print(“a is positive”)
else:
    print(“a is something else”)
```

Adapted from Sonja’s Fall 2019 CS 109 Slides

Python Language Basics

for loops

```
#Prints: 2 3 4
for a in range(2, 5):
   print(“{} “.format(a))

#Prints: 0 1 2 3 4
for a in range(4);
    print(“{} “.format(a))

#Prints: 12 15 17
for b in [12, 15, 17]
   print(“{} “.format(b))
```

while loops

```
#Stays in loop until x is >= 100
x = 0
while (x < 100):
   x += 1
```

Adapted from Sonja’s Fall 2019 CS 109 Slides

Python Language Basics

functions

```
#this how to define a function
def foo():
    print(“hello world”)

#and to call it in the same file
foo()

# to make another function with arguments and return:
def bar(arg1, arg2):
    result = arg1 + arg2
    return result
```

Adapted from Sonja’s Fall 2019 CS 109 Slides

Setting up for PSET #1

HIT THE GROUND RUNNING

PSET 1 Coding Problem

- Consider a game, which uses a generator that produces independent random integers between 1 and 100, inclusive.
- The game starts with a sum S = 0. The first player adds random numbers from the generator to S until S > 100, at which point they record their last random number x.
- The second player continues by adding random numbers from the generator to S until S > 200, at which point they record their last random number y.
- The player with the highest number wins; e.g., if y > x, the second player wins.
- Is this game fair? Write a program to simulate 100,000 games. Based on your simulations, what is the probability estimate that the second player wins? You should report this value to three decimal places.
- For extra credit, calculate the exact probability (without sampling).

Workflow

1. Make sure you have Python Installed
   a. in a command line, type in : “`python3`”, and you should get a “`>>>`” prompt
2. Download the starter code
3. Write your code
4. Make sure to use `np.random.randint` to generate random numbers
   a. Example Usage:

```
result = np.random.randint(4, 8, 100)
```

   Will generate 100 random integers between 4 (inclusive) and 8 (exclusive) in an array called result.

   In other words, result has 100 entries, all of which are either 4, 5, 6, or 7.
5. Test your code by running it in the command line (demo)
6. Upload to gradescope under “HW1 Programming”, an autograder will run and evaluate your work.

Common Mistakes

1.Notice the strict inequality in the looping condition (until S > 100, and S > 200)

2.The function np.random.randint(low, high) is INCLUSIVE of low and EXCLUSIVE of high. Hence, we should have low=1 and high=101.

3.Player 2 wins if and only if y > x, not when y >= x.

4.Player 2 resumes adding from Player 1's sum. Player 2 does NOT start over at 0.