

# The Python Language (Part 1)

Copyright © 2022 by  
Robert M. Dondero, Ph.D.  
Princeton University

# Objectives

- . We will cover:
  - A subset of Python...
  - That is appropriate for COS 333...
  - Through example programs

2

## Objectives

We will cover:

A subset of Python...

That is appropriate for COS 333...

Through example programs

Note:

Comprehensive coverage is impossible!

Please supplement with reading

# Agenda

- **Overview**
- Setup
- Simplest program
- Building and running
- Functions
- Standard library
- Interactive Python

# Overview

- **Who:** Guido Van Rossum
- **When:** ~1990
- **Why:** Scripting lang  
for C/Unix pgmmers



4

## Overview

Who: Guido Van Rossum

A fan of Monty Python's Flying Circus

Python documentation often uses examples derived from Monty Python's Flying Circus

Dead parrots

Spam

When: ~1990

A contemporary of Java!

Why: Scripting language for C/Unix programmers

Scripting language

Need not declare variables before using them

Interpreted

Designed such that C programmers are comfortable with its features

# Overview

- Characteristics:
  - Weakly typed
  - Rich standard library
  - Expressive
  - Subsettable

“Python is the most powerful language you can still read.”

-- Paul Dubois

5

## Overview

### Characteristics

#### Weakly typed

- Objects have types, object refs do not
- Object refs are not declared

#### Rich standard library

- Often poorly documented
- Often uses inconsistent programming conventions
- The Python standard library is a mess

#### Expressive

- Can accomplish much work with little code

#### Subsettable

- Consider “Hello world” program in Java vs. in Python!

# Overview

- Why study Python?
  - It's elegant
  - It's popular
  - It can illustrate much of the course's material
  - You are new to it?
- We'll use Python 3.8

6

## Overview

### Why study Python?

#### Elegant

Python libraries are messy, but language itself is clean.

#### Popular

And is gaining in popularity

Can illustrate much of the course's material

But not client-side web pgmming

But not Android pgmming

You are new to it?

### We'll use Python 3.8

Python 2.x will not work

No longer supported as of 1/1/20

Python 3.6, 3.7, 3.9, or 3.10 probably will work, but no guarantees

# Agenda

- Overview
- **Setup**
- Simple programs
- Building and running
- Functions
- Standard library
- Interactive Python

## Setup

- How to setup a Python environment on **courselab...**
- See *A COS 333 Computing Environment* (from first lecture) for:
  - More detailed instructions
  - Instructions to setup a Python environment on **your** computer
- The general idea...



# Setup

- **Step 1:** Create a virtual environment

```
$ mkdir ~/.virtualenvs  
$ python3.8 -m venv ~/.virtualenvs/cos333
```

9

## Setup

Step 2: Create a virtual environment

Issue these commands

[see slide]

# Setup

- **Step 1: Create a virtual environment (cont.)**

```
Home directory
  .virtualenvs
    cos333
      bin
        Contains a "python" sym link
        to a compiler/interpreter
      lib
        Contains resources (packages,
        modules) that python should use
```

10

## Setup

Step 2: Create a virtual environment

Those commands create a directory structure

[see slide]

Don't install resources into the directory used by the compiler/interpreter bundled with the OS  
Doing so could affect the behavior of programs bundled with the OS

# Setup

## . **Step 2: Activate your virtual environment**

```
$ source ~/.virtualenvs/cos333/bin/activate
```

Shortcut (Bash alias):

```
$ activate333
```

11

### Setup

Step 3: Activate your virtual environment

[see slide]

Adds your bin directory to front of `PATH` env var

Thereafter `python` command runs the compiler/interpreter in your bin directory

Specified compiler/interpreter uses your lib directory to find resources

Defines a shell alias `deactivate`

Make sure you know about the shortcut

# Setup

- **Step 3:** Install resources, as required, into your virtual environment

```
$ python -m pip install --upgrade pip
$ python -m pip install pylint
$ python -m pip install coverage
$ python -m pip install PyQt5
$ python -m pip install flask
```

Installs resources from *PyPi*

12

## Setup

Step 4: Install resources, as required, into your virtual environment

[see slide]

Uses *pip* (Python package installer) program

Installs packages from *PyPi* (Python package index) into your lib directory

# Setup

- **Step 4:** Deactivate your virtual environment

```
$ deactivate
```

13

## Setup

Deactivate your virtual environment

[see slide]

On Unix-like systems

Runs deactivate shell alias

Restores PATH env var to its default

# Setup

- **Thereafter:**

```
$ activate333
```

```
(Use Python as desired)
```

```
$ deactivate
```

# Agenda

- . Overview
- . Setup
- . **Simple programs**
- . Building and running
- . Functions
- . Standard library
- . Interactive Python

# Simple Programs

- See **hello1.py**
  - The job:
    - Write “hello, world” to stdout

```
$ python hello1.py
hello, world
$
```

16

## Simple Programs

[see slide]

Notes:

Overall structure (minimal!)

Program consists of statements

Generally, each statement is on a distinct line

`#!/usr/bin/env python`

She-bang line

Meaningful on Unix-like systems

Meaningless (but harmless) on MS Windows systems

Commands `execvp()` to...

Use the `python` program

Search the `PATH` env var to find it

Comments (`#...`)

`print()` function

String literal (`'...'` or `"..."`)

No semicolons!!!



# Simple Programs

- See **hello2.py**
  - The job:
    - Same as hello1.py

```
$ python hello2.py
hello, world
$
```

17

## Simple Programs

[see slide]

Notes:

Function definitions and calls

```
__name__ == '__main__'
```

True if this module is the “main” module

True if this module is being interpreted directly from the command-line

False if this module is being interpreted via an import statement into another module  
(via an import stmt)

Indentation matters!!!

Function comments are important, but...

I'll omit from lecture example programs to save space/paper

# Simple Programs

- Generalizations:
  - Program consists of statements
  - Indentation indicates nesting
  - Try to do computation only within function defs

18

## Simple Programs

### Generalizations

Program consists of statements

Some of which are function defs

Indentation indicates nesting

Throwback to the early days of programming

Try to do computation only within function defs

File more reasonably can be included into others

Avoid some anomalies

I'll use the style of hello2.py consistently throughout the course

# Agenda

- Overview
- Setup
- Simple programs
- **Building and running**
- Functions
- Standard library
- Interactive Python

## Building and Running

- Initially...
  - Activate your cos333 virtual environment

# Building and Running

Procedure 1 (Unix/Windows):

```
$ python hello2.py
```

Procedure 2 (Unix):

```
$ chmod 700 hello2.py  
$ ./hello2.py
```

Procedure 3 (Windows):

```
C:\>hello2.py
```

21

## Building and Running

Procedure 1 (Unix/Windows)

```
$ python hello2.py
```

Works with or without `#!` line

Procedure 2 (Unix)

```
$ chmod 700 hello2.py
```

See *Unix File and Directory Permissions* doc  
700 => 111000000  
User has read, write, execute permissions  
Members of user's group have no permissions  
Others have no permissions

```
$ ./hello2.py
```

Works only with `#!` line

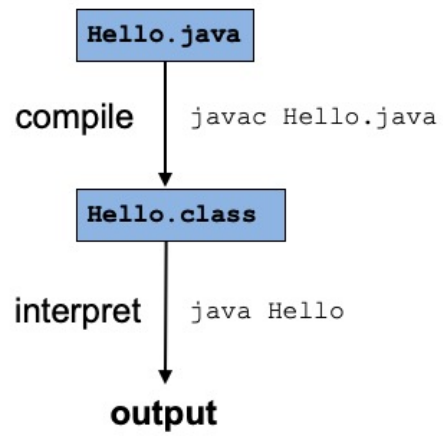
Procedure 3 (Windows)

```
C:\>hello2.py
```

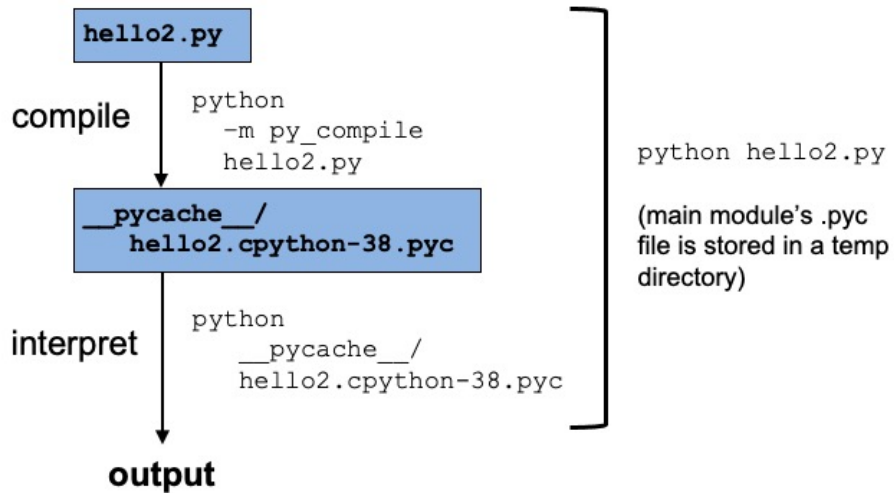
Windows is sensitive to filename suffix  
Works with or without `#!` line (which MS Windows ignores)

Throughout the course I'll stick with procedure 1 as much as possible

## Aside: Building and Running with Java



# Building and Running



23

## Building and Running

[see slide]

You won't see a `__pycache__` directory or `.pyc` files until you start importing Python modules that you compose

We'll study the import statement soon

## Building and Running

- Finally...
  - Deactivate your cos333 virtual environment



## Aside: python -m

```
$ python somefile
```

python **executes** *somefile*, which resides in your working directory

```
$ python -m somefile
```

python **executes** *somefile*, which resides in your virtual environment's `lib` directory (More precisely, uses Python's `sys.path` to find *somefile*)

25

**Aside: python -m**

[see slide]

In the light of that information, the commands that you issued to install resources make more sense

# Agenda

- Overview
- Setup
- Simple programs
- Building and running
- **Functions**
- Standard library
- Interactive Python

# Functions

- See **sub.py**
  - The job:
    - Perform some subtractions, and write the results to stdout

```
$ python sub.py
3
5
3
$
```

27

## Functions

[see slide]

Notes:

- A program can define multiple functions
- Weakly typed
  - Parameter types are not declared
- Call is by value
- Fancy function features:
  - Default parameter values
  - Named arguments

# Agenda

- Overview
- Setup
- Simple programs
- Building and running
- Functions
- **Standard library**
- Interactive Python

# Standard Library

- See **squareroot1.py**
  - The job:
    - Write the square root of 2 to stdout
    - Uses `import` statement

```
$ python squareroot1.py
1.4142135623730951
$
```

29

## Standard library

[see slide]

Notes:

```
import math
```

Commands interpreter to interpret the contents of `math.py` (or `math.pyc`)

Makes all names defined in the `math` module accessible within this module

This module can use those names via `math.` prefix

## Standard Library

- See **squareroot2.py**
  - The job:
    - Same as squareroot1.py
    - Uses `from` statement

```
$ python squareroot2.py
1.4142135623730951
$
```

30

### Standard library

[see slide]

Notes:

```
from math import sqrt
```

Imports `sqrt` from the `math` module into this module

Can use `sqrt` as if it were defined in this module

Good: No need to specify module name prefixes

Bad: Potential name conflicts

New def overwrites old def

# Agenda

- Overview
- Setup
- Simple programs
- Building and running
- Functions
- Standard library
- **Interactive Python**

# Interactive Python

```
$ python
Python 3.8.10 (v3.8.10:3d8993a744, May  3 2021,
08:55:58)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for
more information.
>>> print(1 + 2)
3
>>> 1 + 2
3
>>> quit()
$
```

32

## Interactive Python

[see slide]

Issuing a `python` command with no command-line argument runs python in interactive mode

Python writes a `>>>` prompt

Can call `print()`

It writes to your terminal

Can enter an expression

Python evaluates it, and writes the value to your terminal

Calling `quit()` exits python



# Interactive Python

Interactive Python is valuable for **learning**

```
$ python
Python 3.8.10 (v3.8.10:3d8993a744, May  3 2021,
08:55:58)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for
more information.
>>> import math
>>> math.sqrt(2)
1.4142135623730951
>>> quit()
$
```

33

## Interactive Python

Interactive Python is valuable for *learning*

Example: Can learn about the `math.sqrt()` function

[see slide]

Need not compose a program to call a function

Can call it interactively

(Similar to Java with DrJava or IntelliJ)

# Interactive Python

Interactive Python is valuable for **learning**

```
$ python
Python 3.8.10 (v3.8.10:3d8993a744, May  3 2021,
08:55:58)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for
more information.
>>> from math import sqrt
>>> sqrt(2)
1.4142135623730951
>>> quit()
$
```

34

## Interactive Python

Using from instead of import

# Interactive Python

Interactive Python is valuable for **testing**

```
$ python
Python 3.8.10 (v3.8.10:3d8993a744, May  3 2021,
08:55:58)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for
more information.
>>> import hello1
hello, world
>>> quit()
$
```

35

## Interactive Python

Interactive Python is valuable for *testing*

[see slide]

Need not compose a program to execute your code

Can execute it interactively

(Similar to Java with DrJava or IntelliJ)

Global code is executed upon import

# Interactive Python

Interactive Python is valuable for **testing**

```
$ python
Python 3.8.10 (v3.8.10:3d8993a744, May 3 2021,
08:55:58)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for
more information.
>>> import hello2
>>> hello2.main()
hello, world
>>> quit()
$
```

36

## Interactive Python

[see slide]

It's best to do all computation in function definitions!

Doing so makes your code easier to test

# Interactive Python

Interactive Python is valuable for **testing**

```
$ python
Python 3.8.10 (v3.8.10:3d8993a744, May  3 2021,
08:55:58)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for
more information.
>>> from hello2 import main
>>> main()
hello, world
>>> quit()
$
```

37

## Interactive Python

[see slide]

Using from instead of import

## Summary

- We have covered these aspects of Python:
  - Overview
  - Setup
  - Simple programs
  - Building and running
  - Functions
  - Standard library
  - Interactive Python