

# Introduction to Python

Installation, the basic idea, and some exploration

Daniel Guest

November 16, 2016

Installation

Why Python?

Exploring Python Through the Terminal

- Variables

- Strings

- Lists

Practical Python

- Modules

- A relatable example

Thoughts and resources

# Installation

# Anaconda - the easy way

1. Check to see if you already have Python — just type "Python" your terminal
2. Download Anaconda for your operating system (use the Python 3.5 version)
  - ▶ Windows: [https://repo.continuum.io/archive/Anaconda3-4.1.1-Windows-x86\\_64.exe](https://repo.continuum.io/archive/Anaconda3-4.1.1-Windows-x86_64.exe) (64-bit) and <https://repo.continuum.io/archive/Anaconda3-4.1.1-Windows-x86.exe> (32-bit)
  - ▶ Mac OS: [https://repo.continuum.io/archive/Anaconda3-4.1.1-MacOSX-x86\\_64.pkg](https://repo.continuum.io/archive/Anaconda3-4.1.1-MacOSX-x86_64.pkg)
3. Install Anaconda
4. Anaconda is popular because it manages much of the difficult part of Python for you, and it includes most of the packages for scientific computing, data processing/visualization, and machine learning
5. Install any packages not included with Anaconda through the Anaconda GUI

## If you install the hard way...

- ▶ Install from `https://www.python.org/downloads/release/python-352/`
- ▶ Learn how to install packages from `https://packaging.python.org/installing/`
- ▶ You'll probably want...
  1. NumPy — MATLAB style matrix and signal processing
  2. Matplotlib — plotting ala ggplot2
  3. pandas — data frames similar to R's data frames
  4. SciPy — all sorts of handy stuff for scientific computing
  5. PyMC3 — Bayesian models in Python
  6. Theano — Fast computation of complex mathematical expressions (used in deep learning)
  7. PyBrain — "easy" neural networks
- ▶ Check out the "SciPy Stack" for a good group of useful packages.

# Why Python?

# The design of Python

- ▶ Python is *readable* — the syntax is clean and designed for human eyes
- ▶ Python is *interpreted* — you can test small segments of code easily
- ▶ Python is *extensible* and *modular* — if basic Python doesn't do something you want, it's easy to install packages that do
- ▶ Python is *easy* — much of Python's scientific computing code was designed by scientists for scientists
- ▶ Python is *fast* — although not every built-in Python method is fast, there are many ways to speed up Python if you need it
- ▶ Python is *expressive* — your code will often be shorter and clearer
- ▶ Python is *general purpose* — you can do anything, all in one language

# The design of Python

- ▶ Together, all of these characteristics mean that Python will make your life easier!
- ▶ Many people will tell you that Python is *natural* and *intuitive*, or that Python is the easiest way to express their thoughts in code
- ▶ As I recently saw someone on a forum write, "Python makes programming fun again"



# The Python community

- ▶ One key advantage of Python is its community
- ▶ The community is *large*, *diverse*, and *active*, so most areas of your interest are probably covered, and it's easy to get help
- ▶ Much of the community follows the PEP 8 style guide (<https://www.python.org/dev/peps/pep-0008/>), leading to consistency in how code is written across the community
- ▶ The vast majority of Python code is *open source*, so you'll never have to pay for anything

# Exploring Python Through the Terminal

# Starting Python

- ▶ If you have Python anywhere on your computer, you can simply run "python" in your terminal
- ▶ But, no one really uses the plain Python terminal! For this demo, we'll use IPython, an enhanced Python terminal
- ▶ To run IPython, just run "ipython" in your terminal or command prompt
- ▶ The IPython terminal works very similarly to the terminal in R or MATLAB
- ▶ We can talk about writing scripts and using integrated development environments (IDEs) in the future — for now, we'll focus on getting the basics down!
- ▶ We'll move on once everyone has IPython running...

# Variables

```
x = 5  
print(x)
```

```
## 5
```

```
x = 5  
x = x + 5  
print(x)
```

```
## 10
```

```
x = "Hello"  
y = " World!"  
print(x + y)
```

```
## Hello World!
```

# Variables

```
x = True  
y = False  
print(x and y)  
  
## False
```

```
x = True  
y = False  
print(x or y)  
  
## True
```

# Variables

```
x = 5
y = "words"
print(x + y)
```

```
## Traceback (most recent call last):
##   File "<string>", line 3, in <module>
## TypeError: unsupported operand type(s) for +: 'int' and
```

```
x = 5
y = "words"
print(str(x) + y)
```

```
## 5words
```

# Variables — notes

1. Values are assigned to a name by the `=` operator
2. Variables are *dynamic*, so they can store anything!
3. While powerful, dynamic variables can lead to errors if you don't keep track of what they contain!
4. String concatenation and boolean comparison is easy and readable
5. Convert non-strings to strings with `str()`
6. At least in terminal, just writing the name of a variable and passing it to the "print" function do the same thing

# Strings

```
x = "Hello, how are you?"  
print(x[0:5])  
print(x[0:5] + " " + x[0:5] + "?")  
  
## Hello  
## Hello Hello?
```

```
x = "Good day!"  
print(len(x))  
print(x.count("G"))  
  
## 9  
## 1
```



# Strings

```
x = "Goodbye!"  
print(x.find("!"))
```

```
## 7
```

```
x = "C:\\Documents\\lab\\python\\something_old.txt"  
x = x.replace("old", "new")  
x = x.replace(".txt", ".py")  
print(x)
```

```
## C:\Documents\lab\python\something_new.py
```

# Strings — notes

1. Strings are very powerful and easy to manipulate in Python
2. You can concatenate strings using the `+` operator
3. You can index and slice strings in various ways — we'll see more of this later
4. There are many built-in functions to find and count characters and words inside of strings, or to replace parts of strings (we only scratched the surface here — string manipulation is much easier in Python than R or MATLAB)
5. You can check the length of a string with the `len()` — this is used for most objects in Python
6. Strings, and all other objects in Python, are indexed with `[index]` and not `(index)`, which makes code more readable

## Lists (or, where Python starts to get cool)

```
x = [1, 2, 3, 4, 5] # Spaces optional!
print(x)
print(x[2])
print(x[0])

## [1, 2, 3, 4, 5]
## 3
## 1
```

```
x = [1, 4, 9, 16, 25]
print(x[-1])
print(x[-3:])
print(x[-4:-2])

## 25
## [9, 16, 25]
## [4, 9]
```

# Lists

```
x = ["Austin", "Houston", "Dallas"]
print(x)
print(len(x))
x.insert(1, "San Antonio")
print(x)
print(len(x))

## ['Austin', 'Houston', 'Dallas']
## 3
## ['Austin', 'San Antonio', 'Houston', 'Dallas']
## 4
```

# Lists

```
x = ["Mary", "Sue", "Jane", "Sally"]  
print(x)  
x.sort()  
print(x)
```

```
## ['Mary', 'Sue', 'Jane', 'Sally']  
## ['Jane', 'Mary', 'Sally', 'Sue']
```

```
x = [1, "One", [1, "One"], "1"]  
print(x[0])  
print(x[3])  
print(x[2][1])
```

```
## 1  
## 1  
## One
```

# Lists — notes

1. Lists can store anything, even other lists!
2. As with strings, we index with `[index]`
3. Use `"dir(x)"` when `x` is a list to see all the methods available to manipulate lists

# Practical Python

# Importing Modules

Python, unlike MATLAB, has a small standard library. Although Python's standard library is very powerful, sometimes we need specific tools to get the job done. Python does this by allowing the user to import modules. We'll see how this is done below...

```
x = [1, 4, 9, 16, 25]  
sqrt(x[1])
```

```
## Traceback (most recent call last):  
##   File "<string>", line 2, in <module>  
## NameError: name 'sqrt' is not defined
```



# Importing Modules

```
x = [1, 4, 9, 16, 25]
import math # Method 1
print(math)
print(math.sqrt(x[1]))

## <module 'math' (built-in)>
## 2.0
```

```
x = [1, 4, 9, 16, 25]
import math as m # Method 2
print(m.sqrt(x[4]))

## 5.0
```

# Importing Modules

```
x = [1, 4, 9, 16, 25]
from math import sqrt # Method 3
print(sqrt(x[3]))

## 4.0
```

# Modules — notes

1. Modules are bundles of related code
2. You have to explicitly import modules to use them by using the *import* keyword
3. There are multiple ways to import modules depending on your needs
4. This might seem tedious at first, but it (a) allows for functions to share names, if they come from different modules, (b) makes it easy to import your own functions across different projects, and (c) makes larger projects more organized than in other languages

# A relatable example — the idea

Suppose we want to...

1. Import a .wav file of a recorded vowel
2. Calculate the spectrum of the vowel
3. Plot the spectrum of the vowel and add labels

# A relatable example — the code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile

location = "/home/daniel/Documents/lab/vowgenface/vowels/unswapped/kabrii01.wav"

[fs, vowel] = wavfile.read(location)
n = len(vowel)
time_axis = np.arange(n)/fs
vowel = vowel/np.max(np.abs(vowel))

plt.plot(time_axis, vowel)
plt.ylabel("Amplitude")
plt.xlabel("Time")
plt.show()

log_spectrum = 20*np.log10(np.fft.rfft(vowel))
dt = n/fs
freq_axis = (np.arange(n)/dt)[0:len(log_spectrum)]

plt.plot(freq_axis, log_spectrum)
plt.ylabel("Amplitude")
plt.xlabel("Frequency")
plt.show()
```

## A relatable example — notes

1. Usually, modules are imported at the top of the code
2. NumPy and PyPlot are designed to be familiar to MATLAB users (easy transition)

# Thoughts and resources

## Some thoughts...

1. I love Python, but even I don't use Python for everything — I still prefer R+dplyr for data manipulation and R+ggplot2 for data visualization
2. That said, Python is catching up in every area that it's behind, and is ahead in many areas
3. Python is much better than MATLAB/R for machine learning and advanced Bayesian modeling, for example
4. Python enables you to focus on just one language, rather than being split between multiple languages
5. Python is useful at every skill level and scale — it's easy for a beginner, while still being powerful for an expert



## Some resources...

1. Type `"help(function)"` into the Python terminal to get built-in help!
2. <https://www.python.org/about/gettingstarted/> — great documentation!
3. <https://wiki.python.org/moin/BeginnersGuide/Programmers> — lots of tutorials, videos, guides, example code, etc.
4. <https://docs.python.org/3/> — formal documentation, for the technically adventurous