Introduction to Python

Charles Daniels

September 17, 2020

Introduction

Who Am I?

- Charles Daniels
- PhD CS student
- HeRC research group
- B.S.E in Computer Engineering from USC
- TA for 313, 317, 611

Why Learn Python?

- Easy to write
  - Much less boilerplate than Java
  - No need for manual memory management like C/C++
- Popular
  - Widely used in scientific computing and industry
- Huge, mature library ecosystem
  - Numpy/Scipy
  - Matplotlib
  - PIL
  - many, many more

How Do I Use Python?

I use Python all the time in my work as a graduate student...

- Data processing scripts
  - Convert from one format to another
  - Summarize or gather statistics
- Create figures
- Automate other programs
- Create prototypes for systems to be lowered to C/hardware later

Python Essentials

- Interpreted, not compiled
- Duck-typed
- Object-oriented
- Garbage-collected

The Basics

Want to Follow Along?

- Just open a terminal and run `python3`
- Or use repl.it
- Or use `ipython3` if you feel fancy

Syntax - Variables

Python syntax is different from C and Java...

```
# assign a new variable x
x = 7
y = 3
# print x^y
print("{} to the power of {}={}".format(x, y, x**y))
```

Output:
7 to the power of 3=343

Syntax – Loops

```python
x = 2
while x > 0:
  print("x is ", x)
  x -= 1

for y in range(0, 3):
  print("y is ", y)
```

Output:

```
x is 2
x is 1
y is 0
y is 1
y is 2
```

Syntax – Functions

Defining a function. . .

```python
def doubleit(x):
  return x * 2

# here, message has a default value
def sayit(x, message="value is: "):
  print(message, x)

print("doubleit(3)=", doubleit(3))
sayit(5)
sayit(5, "different message!")
```

Output:

```
doubleit(3)= 6
value is: 5
different message! 5
```

Syntax – Classes 1

Defining a class. . .

```
class Dog:
  # __init__ is the constructor, the first argument
  # doesn't *have* to be "this", this is just a
  # convention ("self" is also popular)
  #
  # __init__ is defined like any other function, this
  # time we use default values
  def __init__(this, fleas=5, greeting="bark"):
    this.fleas = fleas
    this.greeting = greeting

  def bark(this):
    print(this.greeting)
```

Syntax – Classes 2

Using our class. . .

```
fido = Dog()
# single quotes are also allowed for strings
spot = Dog(3, 'woof')
doge = Dog(greeting="wow, such class, very types")

# create a list with the dogs in it
dogs = [fido, spot, doge]

# loop over it
for dog in dogs:
  dog.bark()
```

Output:

```
bark
woof
wow, such class, very types
```

Imports

Some functions, such as sin() are in *modules* which we must import before we can use them. sin() lives in the `math` module.

```
import math

print("pi = ", math.pi)
print("sin(1.5*pi) = ", math.sin(1.5*math.pi))

# we can also import specific items from a module
from math import sin
print("sin(2.5*pi) = ", sin(2.5*math.pi))
```

Output:

```
pi = 3.141592653589793
sin(1.5*pi) = -1.0
sin(2.5*pi) = 1.0
```

Duck Typing

```
class Duck:
  def quack(this):
    print("Quack quack!")

class Goose:
  def quack(this):
    print("Hong honk!")

duck = Duck()
goose = Goose()
for bird in [duck, goose]:
  bird.quack()
```

Output:
Quack quack!
Hong honk!

Input – File

```
# open example.txt for reading, the "with" will
# cause the file to be closed automatically when we
# reach the end of the "with" block, so we don't
# have to call f.close()
with open("example.txt", "r") as f:
  lineno = 0
  for line in f:
    print("line", lineno, "is", line)
    lineno += 1
```

Output:

```
line 0 is line 1
line 1 is line two
line 2 is this is the third line
```

Input – Standard In

This example shows how to loop over all the lines of standard input. . .

```
import sys

lineno = 0
for line in sys.stdin:
  print("line", lineno, "is", line)
  lineno += 1
```

Output – File

```
with open("output.txt", "w") as f:
  for i in range(5):
    f.write("line #{}\n".format(i))
with open("output.txt", "r") as f:
  for line in f:
    print(line)
```

Output:

```
line #0

line #1

line #2

line #3

line #4
```

Getting Fancy

List Comprehensions (Map)

For a list L, apply a function f to each item, creating a new list L' such that $L'[i] = f(L[i]) \forall i$.

```
numbers = [1, 2, 3, 4]
squared = [x*x for x in numbers]
print("squared=", squared)

# convert a string to a list of it's ASCII codes
s = "Hello!"
print("characters=", [ord(c) for c in s])
```

Output:

```
squared= [1, 4, 9, 16]
characters= [72, 101, 108, 108, 111, 33]
```

Setting up to Plot

Code taken from matplotlib.org.

```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np


# Data for plotting
t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2 * np.pi * t)
```

Plot the Data

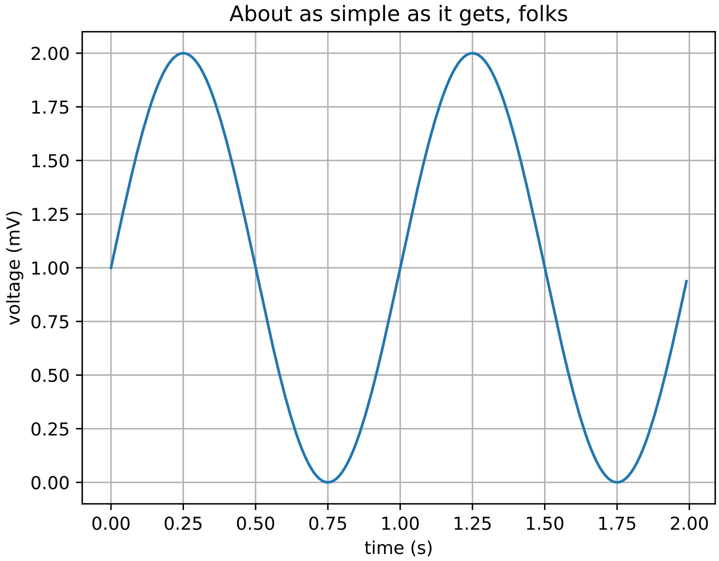
```python
fig, ax = plt.subplots()

ax.plot(t, s)
ax.set(xlabel='time (s)', ylabel='voltage (mV)',
       title='About as simple as it gets, folks')
ax.grid()

# un-comment to save out to a file
# fig.savefig("test.png")

# un-comment to show GUI plot window
# plt.show()
```

The Result. . .


About as simple as it gets, folks
voltage (mV)
time (s)
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Attributes Aren't Pre-Declared

Remember our `class Dog` from earlier? This technique is great for annotating objects you didn't instantiate (but be careful to avoid name collisions)

```
fido = Dog()
fido.name = "Fido"
fido.bark()
print(fido.name)
```

Output:
bark
Fido

What Next?

Libraries

- Numerical computing
  - NumPy
  - SciPy
- Plots
  - matplotlib
- GUIs
  - tkinter
    - shameless plug
- Argument Parsing
  - argarse

How to Install Them

- `pip` will let you install Python modules from the internet.
  - Official docs on python.org
- Search packages: `pip3 search searchterm`
- Install a package `pip3 install --user packagename`
  - Don't install globally with `sudo pip install` unless you know what you are doing.
- Find packages on pypi.org.
  - Also try awesome-python.com

Questions?

End.

Thanks

- This slideshow was written using `pandoc` with caiofcm/filter_pandoc_run_py used to execute in-line Python code and embed the output.
- Thanks to Josh for copyediting.