Introduction to Python

Python & Statistics Bootcamp

NaLette M. Brodnax

The Institute for Quantitative Social Science
Harvard University

June 4, 2018

Set up

- Install Python 3.6 (Anaconda) from https://www.anaconda.com/download
- Access materials at https://nmbrodnax.github.io/python-stats

My goals

- Demystify programming
- Introduce useful features
- Provide opportunities to practice

What is Python?

Python is a general purpose programming language. It is easy to learn, highly readable, powerful, and flexible.

What is Python?

Python is a general purpose programming language. It is easy to learn, highly readable, powerful, and flexible.

Applications

- Data collection
- Data wrangling
- Analysis
- Visualization
- Automation

Introduction

Getting set up

Programming Basics

Command line interface

MacOS X or Linux $\longrightarrow$ **Terminal, Bash**
Windows $\longrightarrow$ **Putty, Powershell**

Features

- Interact with computer's operating system
- Manage Python installation
- Access Python interpreter
- Execute commands

Gathering the tools

Interpreter →Output

Text Editor + Interpreter →Output

Command Line + Text Editor + Interpreter →Output

↓

Integrated Development Environment (IDE) →Output

Launch the Spyder IDE!

Jupyter notebooks

Web application that mixes prose with chunks of executable code

- Useful for exploration and documentation
- Can be configured for multiple programming languages

Try Python

Use the Python interpreter:

```
print("Hello, world.")
```

Use the text editor:

```
print("Hello, world.")
```

Save this as a new script called `hello.py`

Programming language features

1. Data types
2. Conditionals
3. Loops
4. Functions and methods
5. Modules and packages

Data types

Categories for storing different kinds of information in memory

- Examples include integers, floats, and strings
- Form the basis of language syntax and grammar
- Help to allocate computing resources efficiently

Operating on data types

Assignment

assignment	=	`movie = 'Rogue One'`
		`print(movie)`

add and assign	+=	`i = 1`
		`i += 1`
		`print(i)`

Operating on data types

Assignment

assignment	=	`movie = 'Rogue One'`
		`print(movie)`

add and assign	+=	`i = 1`
		`i += 1`
		`print(i)`

String

concatenate	+	`print('A' + 'B')`
repeat	*	`print('me'*3)`

Operating on data types

Assignment

assignment	=	`movie = 'Rogue One'`
		`print(movie)`

add and assign	+=	`i = 1`
		`i += 1`
		`print(i)`

String

concatenate	+	`print('A' + 'B')`
repeat	*	`print('me'*3)`

Comparison

equal/not equal	== !=	`print('a' == 'a')`
		`print('a' == 1)`
		`print(5 != 25/5)`

greater/less greater/less/equal	> < >= <=	

Data types: sequences

string – ordered sequence of characters

```
mystring = 'happy'
```

Data types: sequences

string – ordered sequence of characters

```
mystring = 'happy'
```

list – ordered sequence of items

```
mylist = ['Leia', 'Rey', 'Maz']
```

Data types: sequences

string – ordered sequence of characters

```
mystring = 'happy'
```

list – ordered sequence of items

```
mylist = ['Leia', 'Rey', 'Maz']
```

dictionary – unordered sequence of key-value pairs

```
mydict = {'name': 'Kylo', 'side': 'dark'}
```

Referencing sequences

With an ordered sequences, such as a string or list, reference by **index number**, **starting with zero**

```
mystring = 'happy'
print(mystring[0])
print(mystring[2:4])

mylist = ['Leia', 'Rey', 'Maz']
print(mylist[-1])
```

Referencing sequences

With an ordered sequences, such as a string or list, reference by **index number**, **starting with zero**

```
mystring = 'happy'
print(mystring[0])
print(mystring[2:4])

mylist = ['Leia', 'Rey', 'Maz']
print(mylist[-1])
```

With a dictionary, reference by **key**

```
mydict = {'name': 'Kylo', 'side': 'dark'}
print(mydict['name'])
```

Conditionals

Control structures that allow decision making

```
name = 'Grace Hopper'

if len(name) < 20:
    print('Yes')
else:
    print('No')
```

Conditionals

Control structures that allow decision making

```
name = 'Grace Hopper'

if len(name) < 20:
    print('Yes')
else:
    print('No')
```

Four-space **indentation** tells Python what to execute if the condition is true

Loops

Control structures that allow repeated behavior

- **for** – repeats commands for a finite number of iterations
- **while** – evaluates a conditional statement and repeats commands while the condition is true

Loops

for loop

```
i = 0
for letter in name:
    if letter in ['a', 'e', 'i', 'o', 'u']:
        i = i + 1
print(name + ' has ' + str(i) + ' vowels.')
```

Loops

for loop

```
i = 0
for letter in name:
    if letter in ['a', 'e', 'i', 'o', 'u']:
        i = i + 1
print(name + ' has ' + str(i) + ' vowels.')
```

while loop

```
i = 0
vowel_count = 0
while i < len(name):
    if name[i] in ['a', 'e', 'i', 'o', 'u']:
        vowel_count = vowel_count + 1
    i = i + 1
print(name + ' has ' + str(vowel_count) + ' vowels.')
```

Loops

for loop

```python
i = 0
for letter in name:
    if letter in ['a', 'e', 'i', 'o', 'u']:
        i = i + 1
print(name + ' has ' + str(i) + ' vowels.')
```

while loop

```python
i = 0
vowel_count = 0
while i < len(name):
    if name[i] in ['a', 'e', 'i', 'o', 'u']:
        vowel_count = vowel_count + 1
    i = i + 1
print(name + ' has ' + str(vowel_count) + ' vowels.')
```

Q: Why do we use the `str()` function in each loop?

Functions and methods

function – named block of code that can accept any number of arguments

```
my_string = 'aBcDe'
print(my_string)
```

Functions and methods

function – named block of code that can accept any number of arguments

```
my_string = 'aBcDe'
print(my_string)
```

method – a function with a built-in argument for the object being acted on

```
print(my_string.lower())
```

Functions and methods

function – named block of code that can accept any number of arguments

```
my_string = 'aBcDe'
print(my_string)
```

method – a function with a built-in argument for the object being acted on

```
print(my_string.lower())
```

user-defined functions

```
def say_hello(name_string):
    print('Hello, ' + str(name_string) + '!')
    return None

say_hello('NaLette')
```

Modules

File containing Python definitions and statements and ending in `.py`

Module	Description
`datetime`	basic date and time types
`csv`	reading from and writing to CSV files
`re`	regular expression operations
`os`	miscellaneous operating system tools
`random`	pseudo-random number generation

Packages

Type of module that has a folder of submodules and tools to manage them

Package	Description
`numpy`	array processing and advanced math
`pandas`	high-performance data structures
`scipy`	algorithms and mathematical tools
`scikit-learn`	data mining and analysis
`matplotlib`	publication-quality figures

Questions?