CS 124 Python Tutorial

January 14, 2019

Krishna Patel

kpatel7@stanford.edu
www.tinyurl.com/cs124python
Running Python
Interactive Interpreter

(cs124-env) kpatel$ python3
Python 3.5.2 (default, Nov 12 2018, 13:43:14)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.

>>> You can write Python code right here!

Based on a slide by Sam Redmond
Running Python Scripts

(cs124-env) kpatel$ python3 my_script.py
<output from the script>

(cs124-env) kpatel$ python3 hello.py

What is your name? Krishna
Hey Krishna, I'm Python!

Supply the filename of the Python script to run after the `python3` command

Based on a slide by Sam Redmond
Python Basics
Comments

# Single line comments start with a '#'


Multiline comments can be written between three "s and are often used as function and module comments.


Based on a slide by Sam Redmond
Numbers and Math

Python has two numeric types: **int** and **float**

- 3  # => 3  (int)
- 3.0 # => 3.0 (float)
- 1 + 1 # => 2
- 8 - 1 # => 7
- 10 * 2 # => 20
- 5 / 2 # => 2.5
- 9 / 3 # => 3.0
- 7 / 1.4 # => 5.0

- 7 // 3 # => 2  (integer division)
- 7 % 3 # => 1  (integer modulus)
- 2 ** 4 # => 16  (exponentiation)

Based on a slide by Sam Redmond
Strings

# Unicode by default

greeting = 'Hello'
group = "wørld"

# Concatenate
combo = greeting + ' ' + group + "!

combo == "Hello wørld!"  # => True

No char in Python!
Both ' and " make string literals

Based on a slide by Sam Redmond
Booleans

True  # => True
False # => False

not True # => False
True and False # => False
True or False # => True (short-circuits)

1 == 1 # => True
2 * 3 == 5 # => False
1 != 1 # => False
2 * 3 != 5 # => True

1 < 10 # => True
2 >= 0 # => True
1 < 2 < 3 # => True (1 < 2 and 2 < 3)
1 < 2 >= 3 # => False (1 < 2 and 2 >= 3)

bool is a subtype of int, where True == 1 and False == 0

Based on a slide by Sam Redmond
None

Similar to `null` in other languages

```python
not None # => True
bool(None) # => False
```

None has an inherent value of "False"
If Statements

No parentheses needed

Colon

No curly braces!

if the_world_is_flat:
    print("Don't fall off!")

Use 1 tab or 4 spaces for indentation — be consistent

Based on a slide by Sam Redmond
Exercise 1
elif and else

if name == "juraafsky":
    print("dan is lecturing")
elif name == "kpatel":
    print("krishna is lecturing")
else:
    print("someone else is lecturing")

Python has no switch statement, opting for if/elif/else chains

Based on a slide by Sam Redmond
Variables

```python
my_integer = 10  # => create an integer
my_string = 'hello!'  # => create a string

my_integer = my_string  # => set my_integer to my_string
print(my_integer)  # => prints 'hello!'
```

There are no types in Python!
For Loops

Strings, lists, etc.

for item in iterable:
    process(item)

No loop counter!

Based on a slide by Sam Redmond
Looping over Collections

# Loop over words in a list.
for color in ["red", "green", "blue", "yellow"]:  
    print(color)

# => "red"
# => "green"
# => "blue"
# => "yellow"

Based on a slide by Sam Redmond
Looping with an Index

```
for idx in range(3):
    print(idx)

# => 0
# => 1
# => 2

for idx in range(3, 6):
    print(idx)

# => 3
# => 4
# => 5
```
Exercise 2
While Loops

```python
while condition:
    # do something
```
break and continue

```
i = 0
while True:
    print(i)
    if i == 2:
        break
    i += 1

# => 0
# => 1
# => 2
```

```
i = 0
while i <= 3:
    i += 1
    if i == 1:
        continue
    print(i)

# => 2
# => 3
# => 4
```
Writing Functions

The `def` keyword defines a function.

```
def fn_name(param1, param2):
    ...
    return value_1, value_2
```

Parameters have no explicit types.

- `return` is optional.
- If either `return` or its value are omitted, implicitly returns `None`.
- You can return multiple values.

Based on a slide by Sam Redmond
Data Structures
easy_as = [1, 2, 3]
Inspecting List Elements

```
empty = []
also_empty = list()
letters = ['a', 'b', 'c', 'd']
numbers = [2, 3, 5, 7, 11]

# Access elements at a particular index
numbers[0]  # => 2
numbers[-1] # => 11

# You can also slice lists – the same rules apply
letters[:3]  # => ['a', 'b', 'c']
numbers[2:-1] # => [5, 7]
```

Based on a slide by Sam Redmond
Exercises 3, 4
Useful List Functions

# Add to end of list
lst.append(2)  # => [2]
lst.append(50) # => [2, 50]

# Insert at position
lst.insert(0, 'hi')  # => ['hi', 2, 50]

# Remove from position
lst.pop(1)  # => ['hi', 50]

Find more functions here:
https://docs.python.org/3/tutorial/datastructures.html
Length

numbers = [2, 3, 5, 7, 11]
hello = “hi!”

# find the length
len(numbers)    # => 5
len(hello)      # => 3
Counting vs. Indexing

Index

0   1   2   3

letters = ['a', 'b', 'c', 'd']

len(letters)  # => 4
Sets

empty = set()
letters = ['a', 'b', 'a']
letters_set = set(letters)  # => set('a', 'b')

# Add to a set
letters_set.add('c')       # => set('a', 'b', 'c')
letters_set.add('b')       # => set('a', 'b', 'c')

# Membership
'b' in letters_set         # => True
'c' in letters             # => False
empty = {}
also_empty = dict()
d = {'one': 1, 'two': 2, 'three': 3}

# Getting

d['one']    # => 1

d['four']   # => Raises KeyError

# Setting

d['one'] = 3    # => {'one': 3, 'two': 2, 'three': 3}
d['six'] = 4    # => {'one': 3, 'two': 2, 'three': 3, 'six': 4}

# Membership (looks at keys)

'one' in d    # => True
2 in d        # => False

Keys must be unique
Exercise 5
**Tuples**

```
letters = ['a', 'b', 'c', 'd']  # tuples are immutable

# Creating Tuples
letter_tup = tuple(letters)  # => ('a', 'b', 'c', 'd')
num_tup = (1, 2)

# Indexing
letter_tup[0]  # => 'a'

# Cannot change contents once created
letter_tup[0] = 4  # => Raises Error

# Membership
'a' in letter_tup  # => True
```
Numpy
import numpy as np
Arrays

# Array filled with zeros
zeros = np.zeros(4)  # => array([0., 0., 0., 0.])

# Array filled with ones
ones = np.ones((2, 1))  # => array([[1.],
                                  [1.]]

# Python List to Numpy Array
arr = np.array([1, 3, 4])  # => array([1., 3., 4.])
np.ones(((1, 2, 3))
# array([[[[1., 1., 1.],
          [1., 1., 1.]]]])
Arrays

\[
\text{ones} = \text{np.ones}((2, 1)) \quad \# \Rightarrow \text{array}([[1.], [1.]])
\]

\[
\text{arr} = \text{np.array}([1, 3, 4]) \quad \# \Rightarrow \text{array}([1., 3., 4.])
\]

\[
\text{cat} = \text{np.array}([3, 2]) \quad \# \Rightarrow \text{array}([3., 2.])
\]

# Useful Tricks
\[
\text{arr}[1] = 9 \quad \# \Rightarrow \text{array}([1., 9., 4.])
\]

\[
\text{len}(\text{ones}) \quad \# \Rightarrow 2
\]

\[
\text{ones.shape} \quad \# \Rightarrow (2, 1)
\]

\[
\text{ones.reshape}(2) \quad \# \Rightarrow \text{array}([[1., 1.]])
\]

\[
\text{np.concatenate}((\text{arr}, \text{cat})) \Rightarrow \text{array}([1, 3, 4, 3, 2])
\]
Useful Math

# Natural log
np.log(10)  # => 2.302

# Squareroot
np.sqrt(4)  # => 2

# Dot Product
a = np.array([1, 0])
b = np.array([2, 3])
a.dot(b)  # => 2

# L2 Norm / Euclidian Distance
np.linalg.norm(a)  # => 1.0

\[ \|x\|_2 = \left( \sum_{i=1}^{N} |x_i|^2 \right)^{1/2} = \sqrt{x_1^2 + x_2^2 + \ldots + x_N^2} \]
Broadcasting

range = np.arange(4)  # => array([0, 1, 2, 3])

# Vector and a Scalar
range * 2  # => array([0, 2, 4, 6])

# Vector and a Vector
range / range  # => array([nan, 1., 1., 1.])

# Vector and a Matrix
matrix = np.ones((3, 4))
range + matrix  # => array([[ 1.,  2.,  3.,  4.],
                         [ 1.,  2.,  3.,  4.],
                         [ 1.,  2.,  3.,  4.]])

When you think of broadcasting, think of element-wise operations
Exercise 6
Documentation

https://docs.scipy.org/doc/numpy/reference/routines.html

The best resource!
Regular Expressions
# Search for pattern match anywhere in string; return None if not found
m = re.search(r"(\w+) (\w+)", "Isaac Newton, Physicist")
m.group(0) # "Isaac Newton" — the entire match
m.group(1) # "Isaac" — first parenthesized subgroup
m.group(2) # "Newton" — second parenthesized subgroup

# Match pattern against start of string; return None if not found
m = re.match(r"(?P<fname>\w+) (?P<lname>\w+)", "Malcolm Reynolds")
m.group('fname') # => 'Malcolm'
m.group('lname') # => 'Reynolds'
re — Regular expression operations

# Substitute occurrences of one pattern with another
re.sub(r'@\w+\..*\.', '@stanford.edu', 'k@cal.com jurafsky@bears.com')
# => k@stanford.edu jurafsky@stanford.edu

pattern = re.compile(r'\[a-z]+[0-9]{3}\') # compile pattern for fast ops
match = re.search(pattern, '@@@abc123') # pattern is first argument
match.span() # (3, 9)
Exercise 7
Hello, World!
Variables and Types
Lists
Basic Operators
String Formatting
Basic String Operations
Conditions
Loops
Functions
Dictionaries

Modules and Packages
Numpy Arrays
Generators
List Comprehensions
Regular Expressions
Sets
Decorators

Unix Videos (follow along)
https://tinyurl.com/unix-videos
Videos 1, 4, 6-10, 14-19

Based on a slide by Sam Redmond
Sklearn
Documentation


The best resource!
Linear Regression

```python
from sklearn.linear_model import LinearRegression
x = np.array([1, 2, 3, 4])
y = np.array([4, 5, 6, 7])

# Fit your linear regression
reg = LinearRegression().fit(x, y)

# Predict
reg.predict(np.array([[1, 2]]))
```