Python (v3.6)

Notes open for creative commons use by Kyle Miskell at https://kylemiskell.com and kmiskell@protonmail.com

Learning Resources
Python Official Docs: https://docs.python.org/3/tutorial/index.html
as basis, reading in
Primary: *"Python Crash Course"* (2019)
in same topic/subject order as official docs

Intro

About Python
-Interpreted scripting language with OOP capabilities, excellent for automation, with large collection of micro-frameworks and extensions

-Syntax style is very minimalist, high-level language, with variety of built in data types

-Modular, enable easy re-usability, and easy import of existing collections

-named after *Monty Python's Flying Circus*

Theory
-Beautiful code is better than ugly code
-Simple is better than complex
-Complex is better than complicated (many components better than high level of difficulty)
-Readabilty counts
-Do it the obvious way that would make most sense to programming standards, designs, etc.
-Now is better than tomorrow, which is often never. Always keep learning.

Setup
-python3 and python 2 come installed by default on most linux

-Will want to install pip package manager: *sudo apt-get install -y python3-pip*
-Call pip with: *pip3 install package_name*

-Extras: *sudo apt-get install build-essential libssl-dev libffi-dev python-dev*

Virtual Environments
-isolated python runspace from rest of OS, ensuring each project has own set of dependencies, etc., useful work working with diff versions, third party packaes, etc.
-*pip* installed packages in 1 env are not installed in others

-Install v-environment packages: *sudo apt-get install -y python3-venv*

-Envs are directory based, with a couple scripts added to dir by *python3-venv* to set up env

-To create env, from folder want project in, run: *python3 -m venv my_env_name*
-Will generate bin, include, etc. dirs & files, similar to what *create-react-app* does with *node_modules*

-Enter env: *source my_env/bin/activate*
-Exit env: *deactivate*

Python Interpretor
-Enter interpreter terminal via *python3* command
-Allows interactive editing and execution of code real-time

Basic Script
-folder/file naming convention: *my_script.py*

-Run from command line via: *python my_script.py*

Variables
-syntax: *name = value*
-note no type, no *$* to signify var, etc.

-naming convention: underscore, letters, nums only. Lowercase. *my_var*

-in interactive terminal, last printed expression result is store in temp _ variable:

```
>>> 3 * 3
9
>>> 5 + _
14
```

Comments

```
some_code     #comment

#multi-line
#comment

#do actually include meaningful comments, to help with quick comprehension for others reading code
```

Multiple Assignment
-Can set multi values to multi vars at once via common separation and order basis:

```
a, b, c = 1, 2, 3
print(a)        #1
print(c)        #3
```

Constants
-No constants in python, but can <u>denote</u> one by giving var name in UPPERCASE

```
DOB = '10/11/1986'          #100% mutable, but don't
```

Tracebacks
-If interpretation fails due to error, interpreter throws traceback error, showing line and reason for halt

Numbers

-by default, integers go to *int*, decimals to *float*, with support for other nums, like *Fraction*

-mixed float/int operand equations & divisions always return a float: *3 / 3 = 1.0*
-discard decimal and return int division operator: *5 // 2 = 2*
-power operator: *5 ** 2 = 25*

-Can assign equations to variable, which holds result: *var = 3 * 3 #var == 9*

-can use underscores in long nums as pseudo-comma to make more readable: *20_000_000*

Strings
-Can wrap in ‘ or “, and wrap ‘’ in “”, etc., but cannot wrap pairs of same in pairs of same
-use \ to escape single quote, or wrap in pair of opposite type: *“kyle’s notes”*

Multi-Line Strings
-Can do multi-line, spacing preserved string literals by wrapping in ““ “ triple pair:

```
““ “/                    #backslash followed by no char in cuts out newline
String
        with spacing          kept
”””
```

Concat
-Concatenation: +
-Repeat: *3 * ‘ee’   #eeeeee*

-Auto-concatenation when strings next to each other w/ no operator in between:

```
long_string = '(this is a very long string '
                'it is all one string')          #enclosing ( ) needed for multi-line
```

Accessing Chars
-Can <u>access chars in string</u> by calling *[num]* on string var: *my_string[2]   #zero oriented, 3rd char*
-Can <u>pull chars from end of string</u> moving back by using negatives: *my_string[-1]   #last char*
-Out of bounds char index access attempt throws error
-Strings are immutable, so cannot change char via index access

f-strings
-Can reference vars or expression returns from within strings, JavaScript template style *f-strings* via:

```
f“{some_var} is {some_expression()}”      #f before, brackets inside
```

String Slicing
-Can slice substring via: *my_string[0: 2]*

-Slice is inclusive start, exclusive end: *my_string[2: 5]   #returns chars at index 234*

-If no num before or after *:* then slices to *end:start*   *my_string[:5]   #start to exclusive 5th char*
-Out of bounds char num just slices to start/end for out of bounds num

String Formatting
-*my_string.upper()* - all uppercase
-*my_string.lower()* - all lowercase

-*my_string.title()* - uppercase each word *#New York City*

-*my_string.strip()* - removing trailing whitespace on both sides of string
-*my_string.rstrip()* and *mystring.lstrip()* - remove right or left trailing whitespace only

- *lower()* and *strip()* nice for data normalization before storing user string input data
-whitespace is checked in comparisons of strings

Special Chars
\t – tab
\n – newline
\*'* - escape any type of unpaired quote