Python Illustrated

Python Basic Information

A general purpose programming language
- can be used for many different things

A high level programming language
- abstracts away from machine code
- needs to run through a python interpreter

Here are the two repos that will be used as a reference in this cheat sheet: https://github.com/iamshaunjp/python-3-playlist
https://github.com/Richard-Burd/python-3-sandbox

This is the shell your python code is being passed through in order to execute

This is the actual code being executed

Everything in Python is considered to be an object, and objects have attributes and functions; when we talk about these functions with respect to these objects, we call them methods - thus, objects can have attributes and methods.

Numbers

Python has two types of numbers; integers and floats. You can find the data type by using the built in `type()` method

Integers
Floats (or floating point numbers)

BIDMAS stacking order

Brackets
Indicies
Multiplication
Addition
Subtraction

"Strings" & [Lists]

You can escape a character with the backslash

We start with 0 when counting string elements from the front and -1 when counting string elements from the back

slicing

We can start with 0 and count up to (but not including) the 4 when grabbing a sequence of elements in a string; we can also count from the back

upper()

add (concatenate) & multiply strings

Python has special methods for strings

you can split a string into a list (in python, arrays are called "lists") at a specified character

.split()

Length of a string or list can be found with `len()`

.append()

find the index of a list item

concatenate elements onto a list

add at the end of a list

remove the last element in a list

.pop()

remove a selected item from a list

remove a selected index from a list

find nested list element

del([])

[List] Comprehension

Given a list of numbers we want to double; there are two different methods for doing this shown below

Another example below squares all even numbers from 1 to 10

The kind of comprehension method can be used (mutatis mutandis) on dictionaries as well

[List] & (Tuple) Manipulation

Tuples are used for coordinates, colors, rectangles, & other mathy stuff; they are similar to lists

Lists & tuples can both contain mixed data types and nested elements; the `slice()` method can be used on both of them.

Tuples are immutable and lists are mutable; in example: tuples cannot be appended but lists can be appended

Decorators

Decorators are used to modify the behavior of a function without modifying the function itself; they are use extensively in web frameworks like Django.

Decorators are basically wrapper functions; they are defined with an @ symbol & the decorator function name must match the decorator itself

This code runs before the function

This code runs after the function

Conditionals

User inputs are always accepted as strings (e.g. "2" unless they are type-casted with `int()` to become integers

if : elif : else :

Python uses colons and needs the following line to be indented

Python uses the `and` keyword instead of && like in Ruby or JavaScript

if

starts with index 1 and goes up to, but not including, index 3

for in :

after `moe`, break out of the loop and ignore the other elements

break

this will continue on with the iteration after doing the stuff above

this will give us only even numbers

while

without these specific indentations, the Python code will not work

continue

File Importing

To import relative from the top level directory, use dot notation like this

To import from beyond the top level package, you will need this block of code

Reference: https://github.com/Richard-Burd/python-3-sandbox

Dictionaries

Python dictionaries are similar to Ruby hashes or JavaScript objects

see if key exists in the dictionary with `in`

find the value of a specified key

in

grabbing the keys in a dictionary will give a value like this

list

either the dictionary keys or values can be returned as a list (array) data type with the `list` keyword

get the number of times a given key is found in the dictionary with the `count` method

This is an alternative syntax for declaring a dictionary, when called, it uses "=" but uses "=" in the declaration

This takes the user input from the console and sends it to the dictionary

continue

this keeps the code active using the while loop when the user enters in "y"

items

This function cycles through the dictionary

for...in

This function call starts the program in the console

{Dictionary} Comprehension

Reference: https://www.datacamp.com/community/tutorials/python-dictionary-comprehension

Dictionary comprehension is a powerful concept and can be used to substitute for loops and lambda functions. However, not all for loops can be written as a dictionary comprehension but all dictionary comprehension can be written with a for loop.

Double each value in the dictionary

Double each value in the dictionary only if the value is an even number

Identify odd and even entries

Similarly, dictionaries can be nested and thus their comprehensions can be nested as well

Classes, Modules, & Packages

Everything in Python is an object and all objects have a class type; each of those class types in turn have methods that can be called on them; `num_var` is based on the list class type therefore it inherits all methods for the list class

the init function

The init function can be hardwired like this but is usually built to accept instance variables

class name is capitalized & followed by ":"

This is where class attributes are defined

These are the instance attributes

This function defines the class attributes & it takes in a self property

static method

cls

class method

These are both instance methods, so they both must take in the self parameter

You don't need anything in this file; it's existence alone is enough to tell Python that the contents in this directory called space are a package of modules

This directory gets generated automatically when you create the folder above to setup a module

Static methods have no access to class (cls) or class instances (self)

space is the folder name, planet is the name of the class file; this import is enabled by the `__init__.py` file that is in the space directory

a class instance called 'naboo' is created above in classes.py

this is a module

Console Inputs & Strings

User inputs are always accepted as strings (e.g. "2") unless they are type-casted with `int()` to become integers

type casting

print()

int()

This will get printed out and prompt the user for an input

Up here we broke out of the string. But now let's look at some string formatting

.format()

f-strings

We could specify 2 or even 9 digits in the number (precision)

These are called F-Strings & they don't require ".format()"

Or we could add the little f for float to specify the deceimal number we want to see

Ranges

In Python, ranges generate a list of numbers for us that we can then use to iterate over in for loops

this will go up to but not including 5

this will start from and include 5 and go through but not include 10

this will start from and include 20 and go through but not include 300 in intervals of 80

len()

this `len()` method finds the length of the names list and cycles through the range for each element in that names list

this -1 is the last element in the list and the start position, this -1 is the position right before the start of the list because this value tells us the looping end point and it is an up-to-but-not-included value, finally, this -2 is the increment amount, and it is negative

Dictionary Iteration

code shown in the section below is available here: https://github.com/Richard-Burd/python-3-sandbox/blob/main/dictionary_iteration.py

for loops can be used to find values within nested dictionaries

and in

JavaScript & Ruby pipes (||) are replaced with a single pipe in Python

Maps

Maps are a way to take a list, apply some kind of function to each item within that list, and return a new list with the changes made by the function to each item in the list

random

we need the random module from the Python Standard Library for this example: https://docs.python.org/3/library/

list

the `list()` method takes a string and makes each character its own string in a new list like: ['r', 'o', 'n']

join

the `join()` method will take elements of a list and put them into the same string

map()

This doesn't quite work because it's mapping the result onto an unusable object; to make it readable, it must be typecasted into a list

Filters

The `filter` method is used to determine if a specified condition is true or false for each element in a given list; if true, the element remains in the filtered list, if not, it is dropped; The example here filters out bad grades

filter()

comprehension is still the shortest way to go

Collections - Deque

all code shown in this section is available here: https://github.com/Richard-Burd/python-3-sandbox/timfiles/collections_deque.py

Why use a deque over a traditional list? because it's faster in terms of adding items to the beginning and end of the list, but if you're going to want to randomly access elements within the container, then it's better to use a traditional list.

the Deque must be imported

appendleft

it takes each element of a string and makes it an element of a list-like data structure

pop

items can be added to the front or back of a deque

clear

this will destructively clear all contents of the deque

extend

the `extend()` method takes in anything iterable, such as a list or string, and puts it at the end of the deque

rotate

The `rotate()` method shifts the order of items as shown on the left

The `maxlen()` method limits the number of items in the deque

NOTE: you cannot reassign this maxlen value after it's initially declared

maxlen

you can only add to the deque with the `extend()` method, but that will still maintain the original maxlen of 5

{Sets} [Lists] & Sorting

all code shown in this section is available here: https://github.com/Richard-Burd/python-3-sandbox/timfiles/sets.py

Python lists are similar to JavaScript & Ruby arrays whereas Python sets are essentially Python dictionaries with only keys, and no values: Every element in a set must be immutable but the set itself is mutable

this will be a dictionary, not a set because that is what an empty {} defaults to in Python

sorted() method sorts numbers & strings

it will remove duplicates in a set

it will order items with capital letters first

set() will return a set and thus, remove duplicates in a list of strings, but it will not order the elements so the return value can change each time

https://stackoverflow.com/questions/2831212/python-sets-vs-lists/2831242

Lists are slightly faster than sets when you just want to iterate over the values. Sets, however, are significantly faster than lists if you want to check if an item is contained within it. They can only contain unique items though. It turns out tuples perform in almost exactly the same way as lists, except for their immutability.

It is possible to do comprehension on sets

Sets are mutable

however, elements of a set cannot be mutable objects

NOTE: this order will change each time the code is ran because there is no true order to a set, unlike a list

keys values

here we are typecasting the ages into a set using the `set()` method so that we do not have duplicate ages down here

Try & Accept

all code shown in this section is available here: https://github.com/Richard-Burd/python-3-sandbox/timfiles/try_&_accept.py

Try & accept lets you run code that actually would crash if it is falsey; Here I want my string to only be numbers:

Here the text is typecasted into an integer

The `except` block will run if the `try` block of code is either falsey, or crashes

Functions & Variable Scope

Python uses colons to start a function body

The function body must be indented or Python will not compile

To override default values, specify the variable that will have the default overridden

Here we are passing a function into a function (as a variable)

variables can be redefined in a lower scope but still retain their original value in the higher scope; the global value can be called with the global keyword in a lower scope

global

Advanced Overview Features

Most of the code shown in this section is available here: https://github.com/Richard-Burd/python-3-sandbox/timfiles/expert.py

Python is compiled into bytecode before it is interpreted. Compilers take high-level code and translate it into a lower-level. An interpreter takes some code and performs the operations in the code. This is unique to Python because it is a compiled language, here we have a class with an undefined 'bark' method:

This is unique to Python because it a compiled language, here we have a class with an undefined 'bark' method that has not yet been defined; if I ran the code at this point there will be no errors. In other languages, the compiler would detect this error and tell you to define what 'bark' is, but here, this bit of code is executed at runtime instead of compile time. All the compiler does for us is translate the Python into bytecode, and it does not always check to see if the code is actually valid. Thus, the error above is said to be 'only caught at runtime' and not at compile time.

Let's look at another example in which the compiler doesn't care if the code is valid or not, so long as you have valid syntax

We can define a class inside a function because that is how Python works; we can nest classes as deep as we want

This returns the class Cat and not an instance of the class; the class is being created and stored in memory

Here we're calling the class method

This `cls` variable is actually a class, so it's another name for Cat

the name of this instance of the Cat class is "Timmy"

Functions can be put inside of a for loop

This will run each time the loop runs

This will only run once on the final item in the range, but it is aware of the existence of `show()` inside a deeper scope it is not a part of, but `show()` must be declared on a line ABOVE where it is being called or it will not run and will generate an 'is not defined' error

all code shown in this tiny subsection is available here: https://github.com/Richard-Burd/python-3-sandbox/timfiles/nested_function_calls.py

A function inside of a function must be accessed by nesting the function calls like this

inspect

Everything in Python is an object so each thing has its own unique memory address

getsource

The inspect module can show us some pretty cool things because of the fact that all of our Python objects are objects; here we use the `getsource()` method to get the sourcecode of a specific method, function, class, or other object

Zip Function

Reference: https://www.w3schools.com/python/ref_func_zip.asp

all code shown in this section is available here: https://github.com/Richard-Burd/python-3-sandbox/zip.py

The `zip()` function returns a zip object, which is an iterator of tuples where the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together etc. If the passed iterators have different lengths, the iterator with the least items decides the length of the new iterator.

use the `tuple()` function to display a readable version of the result

or use the `dict()` function instead, but you can only run one or the other in the same script on the same object

Collections

Standard Python Containers
1.) list
2.) set
3.) dictionary (dict)
4.) tuple - this one is immutable

Collections Module Containers
1.) counter
2.) deque
3.) namedTuple
4.) OrderedDict
5.) defaultDict

The Python data types above must be imported via their commonsnake notation, ready to be used as shown on the left

Collections - Counter

all code shown in this section is available here: https://github.com/Richard-Burd/python-3-sandbox/timfiles/collections_counter.py

This module implements specialized container datatypes providing alternatives to Python's general purpose built-in containers

the Counter must be imported

a string as a variable will return a letter count

a list will return a dictionary showing how many times each item occurs in the dictionary

a dictionary will return a sorted dictionary

keys can be non-string variables

but must be called with a string

a key that doesn't exist will return 0

listing elements will return a list of items

Counter keys can be different data types

most_common

Counter has a `most_common` method for returning items by number of most common items

you can subtract keys that are strings but not ones that are integers like these two

Counter keys can be updated like this or like this

the Counter can be cleared of all its contents

When you subtract elements on a counter, it will not show values of 0 or negative values

Here we say that b is "intersecting" with a and this gives the lowest common values for the items in the counter, in this case, a value of 3 (in Counter a above) and a value of 10 (in Counter b above) so since 3 is the smallest, that is the intersect value

intersection & union of Counters

The opposite of intersecting is called "union" (x & y) which is shown here: These are the maximum values between the two Counters a and b above (x | y)

Reading Files

Python allows you to open up files and read them then do something with those files

this opens an external file at: python-3-sandbox/lessons/ipsum.txt

this iterates through each line of the file & reads each one

the `rstrip()` method removes empty space between lines

this starts at the 6th character in the file and returns it in list brackets

this starts from the `seek()` method starting point above (6) and reads 5 characters of the file

the file should always be closed to prevent any performance penalties

this checks to see if the ">" character is in the line or not

using with `open()` as: is generally better than using the seperate `open()` & `close()` statements shown above; the file remains open while code beneath is indented and closes when the code indentation ends

this finds all instances of the ">" character in the dna_sequence.txt file and filters them out of the printed list in the console

Lambdas

Lambda expressions (or lambda functions) are similar to anonymous functions in JavaScript; they are suitable for situations in which you're only gonna call the function once.

This is a standard function that will square a number with a map function typecasted into a list

lambda

The lambda will automatically return the result to the right of the ":" without a `return` statement

You could pass in multiple arguments into a lambda like this

Collections - NamedTuple

all code shown in this section is available here: https://github.com/Richard-Burd/python-3-sandbox/timfiles/collections_namedtuple.py

The main difference between a regular tuple and a named tuple is that with a named tuple you can access things by element and it's a lot nicer to read in your program

the namedtuple must be imported

subclass constructor

instantiator

a namedtuple subclass name must be a capitalized string and the item names are declared in the second string with each string seperated by a space

the items can be put in a list like this

the items can also be stored in a dictionary

_asdict()

Named tuples allow for the use of dot notation, but regular tuples do not

with Named tuples, items can be found by index as well

_replace()

values can be converted to a dictionary with the `_asdict()` method

_fields()

We can print out the keys with the `_fields()` method

we can replace the value of a specified key, with the `_replace()` method, but this is not destructive, in other words, we need to assign a new value to the operation because we cannot change the original namedtuple in this way

_make()

The `_make()` method can be used to create a new instance of the Data class and this new instance will have the specified values passed into the `_make()` method: NOTE: the values are now strings, not integers as in the original declaration above, because we can change the data type

Queue Basics

all code shown in this section is available here: https://github.com/Richard-Burd/python-3-sandbox/misc_tutorials/Python_Threading_Series/queue_basics.py

There are 3 types of queues: FIFO, LIFO, and Priority; FIFO is the default type where no other type is specified.

First In First Out (FIFO)

The `.get()` method returns the next item in the queue to be retrieved; in this case, it is 5 because the 5 was the first in, and is therefore the first out - the 5 is next in line and thus, the `.get()` method is destructive

The `.empty()` method returns a boolean value

get empty

A while loop can return all existing values, we use `empty()` instead of `get()` because the latter will cause our computer to lock up when there are no more items to actually "get()" in the queue - to solve that problem we can use an advanced method called threading or the `empty()` method as shown

LifoQueue

This is the last item put in the queue, and so it is the first item to go out when the `get()` method is called on the queue

PriorityQueue

If the queue items are simply integers, then the `get()` method will return the integers from smallest to largest

If you need to assign a priority to a data type, let's say to a string, then you use a tuple for each use of the `put()` method wherein the first tuple item is an integer (representing the priority value) and the second tuple item is the string (or some other data type) - the first tuple item must always be an integer

Writing Files

When we open up a file, it is read-only by default unless the `open()` method takes in the second variable, "w" that permits writing to the file or an "a" that permits appending the file

the "w" method will open up the file, look at what is inside, and override whatever is inside with this string; that means this string will appear only once in that file and everything else in that file will be deleted. The "a" method on the other hand will only add text to the end of the file and leave the existing text content alone

write() writelines()

The `writelines()` method expects some kind of (Python) list that it will go through as print each element within it

Dunder Magic

all code shown in this section is available here: https://github.com/Richard-Burd/python-3-sandbox/timfiles/dunder_magic.py

By default we get the memory address location of this object because we have not told the object what to do when we print it

repr

To get meaningful information, we need to implement a 'Dunder' method AKA a 'magic' method

mul

call

The `__mul__` data model method will tell Python how to handle multiplication operations on an object that is of the data type Person, e.g. an instance of the Person class

len

raise Exception

Whenever this Person class is called as if it were a function, this `__call__` method will handle that call for the Person class

These are all data model methods

data model methods

The `__len__` data model method will tell Python what to do when the length (`len()`) is called on the Person class; in this example, the string "four" has 4 items

Queue

this just returns an object address

OK let's now let's set things up so that when we print the queue by calling the print method on the declaration, we get something meaningful to print in the console instead of this

add

sub

The `__add__` method will fire anytime something is added to the queue

The `__sub__` method will fire anytime something is subtracted from the queue

Without the `__add__` method above, this would generate this error:

TypeError: unsupported operand type(s) for +: 'Queue' and 'int'

In this case, it doesn't matter what value you put here, because this queue is (by default) a First-In-First-Out (FIFO) queue, and the first item ("a") will be gotten (`get()`) and then subtracted from the queue because the `__sub__` method is being called to perform a subtraction operation

Threading

all code shown in this section is available here: https://github.com/Richard-Burd/python-3-sandbox/misc_tutorials/corey_threading/vid_1.py

Diagrams are from this video here on Python threading by Corey Schafer: https://www.youtube.com/watch?v=IEEhzQoKtQU

Threading enables concurrent code execution & requires the importing of the threading module

Here we are running code synchronously

Here we are running code concurrently; the 2nd function fires after the 1st function fires, but the code is not ran simultaneously

The `join()` method will ensure that t1 and t2 finish running before moving onto the next lines where the finish time is calculated and this statement gets printed out

join

Downloading Files

We need this module to be imported in order to download

These receive input from the (CLI) user & store their answers as variables

Running these commands in the bash prompt will create the Airplane.jpg file inside the images folder

Create this folder and leave it empty; images will be download to it

Python 3 Tutorial for Beginners by The Net Ninja
https://www.youtube.com/playlist?list=PL4cUxeGkcC9idu6GZ8EU_5B6WpKTdYZbK

Python Programming Tutorials by Tech With Tim
https://www.youtube.com/playlist?list=PLzMcBGfZo4-mFu00qxl0a67RhjjZj3jXm

Intermediate Python Tutorials by Tech With Tim
https://www.youtube.com/playlist?list=PLzMcBGfZo4-nhWva-6OVh1yKWHBs4o_tv

Expert Python Tutorials by Tech With Tim
https://www.youtube.com/playlist?list=PLzMcBGfZo4-kwmIcMDdXSuy_wSqtU-xDP

Mastering Python by Tech With Tim
https://www.youtube.com/watch?v=p15xzjzR9j0

Python Threading Tutorial by Corey Schafer
https://www.youtube.com/watch?v=IEEhzQoKtQU

Python Tutorials : Threading Beginners by PyMoondra
https://www.youtube.com/watch?v=bnm5_GH04fM

github.com/Richard-Burd/python-3-sandbox

last updated @ 6:37pm on 04/May/2021 by Richard Burd rick.a.burd@gmail.com

Python Illustrated