Complete Python Bootcamp: Go from zero to hero in Python 3

-Jose Portilla-

Python is a high-level programing language which is use

Hi!

Here is some useful info for you to get started:

1.Please watch the course overview lecture, lots of useful info there! Notebooks can be found here: https://github.com/Pierian-Data/Complete-Python-3-Bootcamp

2. Our QA forums can be found by going to your course dashboard (top right button when viewing a lecture)

3. Our student chat channel is here: https://discord.gg/TztE6B8

4. Video guide to the chat room: https://www.youtube.com/watch?v=bkH89OJ001M

4. Our notebooks can be found as resources in the Course Overview lecture.

5. Info on certification: https://support.udemy.com/hc/en-us/articles/229603868-Certificate-of-Completion

6. The best place to ask questions is in the QA forums, not the messaging system, please keep in mind I may just ask you to repost your question in the QA forums if you message me here. (Its just easier for me to manage that way.)

Thank you so much,

Jose

(This is an automated message, but I do see replies)

Contents

Explanation of notice

Color for data:

1/ Homework

2/ [Practical example]:

3/ **Importance concept**

I/ Course Overview

1/ Course introduction

- Mr. Jose recommend to gear up to 1.5x or 2x if we can understand English.
- We can download the lesson by download the Udemy app.
- It's huge advantage to use **Q&A Forums**. If any concern arises, first, put them on Google, second option is **StackOverflow** (Important to developer). Q&A is the ultimate savior to us. –In my opinion, Q&A, StackOverflow is also a great source of inspiration and creativity as people put anything arises in their head onto this.
- The **notebook** was created to help us define error. By reconcile/double check the code to notebook or maybe copy code from notebook and watch It run may help us to see the error (Which may arise from our/ Jose's typo).
- When submit Q to QA Forum, put detail of what I've tried, a screenshot of error or code and as much information as possible to get the answer as quickly.
- Any issues related to platform level, email support@udemy.com.
- Further notes:
  - Pay attention to **Chat channel on discord**
  - **FAQ** at the end of every lecture is good also (Difference from Q&A Forum)

2/ Course Curriculum *(Chương trình giáo dục)* Overview:

(Homework: This part is a short cut of what Mr. Jose would tech me. It like a summary of the course. So despite of the time consuming it'd take, you have to grab explanation for everything list out in this part. Even though we did not learn it. Do this, and have a test to confirm that remember it. Use block testing. 3 time, 3 continues days.

1/ Note down explanation of all concept

2/ Create question

3/ Review all.

This will help me save tons of effort later.)

Sumary: This course contains 20 section.

- **Section 1**: Overview
    - Course overview, curriculum Overview
    - Python 2 versus Python 3
    - How to approach this course (FAQs) >>> to maximize what I will get
- **Section 2**: Python setup
    - Python installation
    - Environment Selection
    - Jupyter Notebooks (*Jupyter is a notebook system use in this course*)
    - Additional Learning Resources
    - Git and Github Overview
- **Section 3**-Object and Data structure Basics: *This a live code section. There're quizzes to ensure understanding.*
    - **Numbers** –Number, including integer and float
    - **Strings**- text data
    - **Lists**: is a collection which is ordered and changeable. Allows duplicate members >>square brackets. []
    - **Dictionaries**: is a collection which is unordered, changeable and indexed. No duplicate members. curly brackets {}
    - **Tuples**: is a collection which is ordered and unchangeable. Allows duplicate members. << round brackets ()
    - **Files**
    - **Sets**: is a collection which is unordered and unindexed. No duplicate members. <<curly bracket {}
    - **Booleans.**
  - Finally, there'll be a test at the end.
- **Section 4**-Comparison Operators:
    - Basic Operators
    - Chained Comparison Operators
    - Quizzes
- **Section 5-** Python statement:
    - If, elis and else
    - For loops
    - While loops
    - Rangse()

        - List Comprehensions
        - **Assessment test**
- **Section 6** – Methods and Functions:
        - Methods
        - Funtions
        - Lambda expressions
        - Nested Statements
        - Scopes
        - Homework assignment
- **Section 7-** First Miles stone with a real game project
- **Section 8** -Object Oriented Programming (OOP).
        - Objects
        - Classes
        - Methods
        - Inheritance
        - Special Methods
    - **Homework!**
- **Section 9-** Errors and Exception handling
        - Errors
        - Exception
        - Try
        - Except
        - Finally
    - **Test!**
- **Section 10-** second mile stone – a more conplex game
        -
- **Section 11:** Module and package
        - Creatiing Modules
        - Installing Modules
        - Exploring the Python Ecosystem
- **Section 12: Built in Functions**
        - Map
        - Reduce
        - Filter
        - Zip
        - Enumerate
        - All and any

        - Complex
    - **Assessment test!**
- **Section 13:** Decorator in Python
        - 3 part Series of Homework assignment
- **Section 14:** Python Generators
        - Interation Vs Generation
        - Creating Generators
    - **Homeworks**
- **Finall Capstone Project**

Advanced Bonus Contents: The new lecture with new content is added to this course regularly. The below are section added later than the original

- **Section 15-**Advance Python Modules
- **Section 16-** Advance Python Objects and Data structures
- **sSection 17-** Bonus Material- Introduction to GUIs
- **Section 18** – Bonus special offers
- **Section 19** – Appendix: Older Python 2 material
- **Section 20** – Bonus coupons for other courses.

3/ Python 2 versus Python 3:

- The legendary Python 2 is still being used in many company. But it will stop being update with security in 2020.
- The course is focus on Python 3, which is assumed to be the future Python.

4/ FAQ

- **First:** Remember to use QA Forum
- **Second:** There are slides for lectures. Download them and print
- **Third:** Note book is read by some app name Anaconda Navigator. Which will be introduced in section 2. (*Later figured: Anaconda is the most popular Python/ R data Science Platform*).

II/ Python setup

1/ Command line: It's somehow different btw per Windown and per MacOS, Linux

a) For Windown:

- In Windown, we use "Command Prompt" to execute this.

- Command Prompt can be accessed by type cmd or Command Prompt in search bar (Windows + S)
- First basic commands:
  - **Cd:** Current directory
  - **Dir**: List everything in the directory
  - **Cd [directory name]**: Move to a directory (In this cd stand for **change directories**)
  - **Cls**: Clear the screen
  - **Cd ..** : Jump back to previous (directory)
- We can use key **tab** to point in files/folder in the current directory (?) to autocomplete a line of text (?)

b) MacOS or Linux:

- In MacOS and Linux, we use "Terminal"
- Commands:
  - **Pwd** (Present working Directory)**:** Current directory
  - **Ls** (List files): List everything in the directory
  - **Cd [directory name]**: Move to a directory
  - **Clear**: Clear the screen
  - **Cd ..** : Jump back to previous (directory)

2/ Installing Python:

- Guido Van Rossum in 1991.
- This language is quick to learn, less code, easier Syntax, huge amount of additional open-source library.
- Jose use Anaconda as the Python Platform for this course.
- **Syntax**: Cú pháp

(?) What does he mean when he distinguishes running python code by **script as a command line** from running python as **Jupyter Notebook**?

3/Running Python code:

- There're several ways to run Python code.
- There's a various option of option we can chose for development environment (Which we are going to type code into) which are:
  - **Text Editors**: We have to sacrifice some specific Python functions (?) that will not have already been added to this. But there will always be some plugins/add-ons can help U complete what u are doing (As Python is so popular)

        - General editors for any text files
        - Work with a variety of file types
        - Can be customized with plugins and add-ons
        - Most are not designed with Python only (Can open Java, py, and so on…)
            - **Most popular: Subline text** (ultimate free license) **and Atom (**Open source from Github)
    - **IDEs** (Integrated Development Environment):
        - Developed environment designed specifically for Python.
        - Can proceed larger programs (Larger in size)
        - This contain a **community editions** (free) and a **pro version** (have to pay)
            - Most popular: **PyCharm** and **Spyder** (Spyder might be familiar to MATLAB user).
    - **Notebook Environments:**
        - Great for learning
        - See input and output next to each other. Everything is so clear. Unlike **IDEs** and **Text Edittor**, where code are presented as larger code file and will be run by commands.
        - Code in Notebook are present in cells. So we can separate our code in to difference part and run them separately.
        - Support in-line markdown notes, visualizations, video and a lot more.
        - And because of the multi usage as above, Noteboook despite of having an extension of .py. The Notebook file end up by .ipynb (Stand for I Python Notebook) (We cannot open this by double click on the file)
            - Most popular: **Jupyter notebook.**

[Practical example]:

1/ Create a .py script and run the file at your command line:

- First, create a line of script in Sublime Text and save it under Python format. Note: Sublime only show it's use when it has the file format (By save it as or by open an existed files).
- Open this by command prompt by move to the files' directory and type its name. The file is run now.
- Type "quit" to quit Command Prompt Python interpreter

- We also can create and run Python code in Command prompt by type Python to code ground. But it's just to know as this only allow simple coding and there're lack of convenience.

2/ Then with a Jupyter notebook:

- Open Jupyter notebook by opening Anaconda navigator
- It run on Web platform. Remember to check Keyboard short cut and use "mark down" to leave note to help you understand well code.

(?):
1/ How to chose another disk (Not C:) >>> Open Command prompt, Change directory to destination Directory and then run Jupyter notebook , also in command prompt.

Ex: Way 1: First, run Path/to/my/directory, second, run "Jupyter notebook" (Check again, I think this is not applicable)

Way 2: Change the cd to destination directory, run "Jupyter notebook"

2/ What is the difference btw Cell and new line? Is Cell like a small program?

4/ Download course contents and introduce to Git and Github

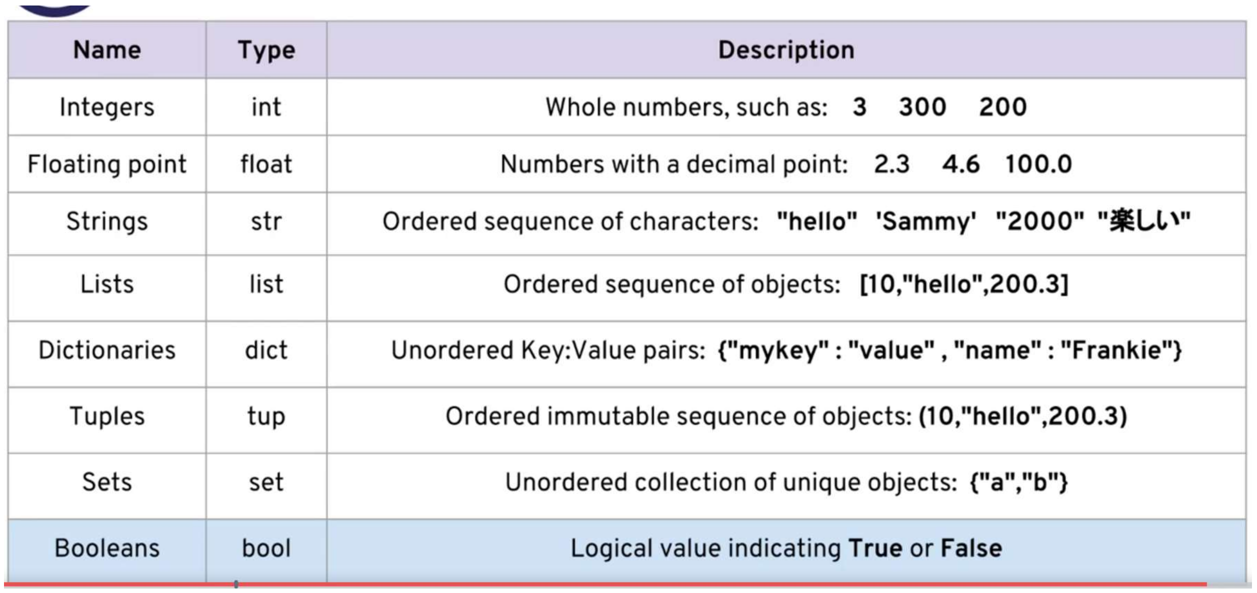
- So, GIT and Github:

    - GIT: Is a Version control system. A tool to help develop program, code, software, etc.… It's help control version of a software, like access to previous version, see change and who change. The principle of GIT is to create a mirror to a previous version if it is unchanged. If any change happened, it saves the new file to the current version. (The other system usually save the change of the file-not new file).
    - Github: A Platform for Repository (Nhà kho- software warehouse)- a open Platform for developing open source code (Mã nguồn mở, sound familiar?). There are a lot of functions integrated in Github. We can also pay and use Pro-version of Github if we want to keep our open source code away from being shared.

III/ Python Object and Data structure basis:

1/ Introduce to Python Data types:

- There are 3 data types, 4 data structure and 1 logical value. (Remaining: Object- I think it might be each of the number/string below as “3”, “Sammy”)
-



Name	Type	Description
Integers	int	Whole numbers, such as: **3 300 200**
Floating point	float	Numbers with a decimal point: **2.3 4.6 100.0**
Strings	str	Ordered sequence of characters: **"hello" 'Sammy' "2000" "楽しい"**
Lists	list	Ordered sequence of objects: **[10,"hello",200.3]**
Dictionaries	dict	Unordered Key:Value pairs: **{"mykey" : "value" , "name" : "Frankie"}**
Tuples	tup	Ordered immutable sequence of objects: **(10,"hello",200.3)**
Sets	set	Unordered collection of unique objects: **{"a","b"}**
Booleans	bool	Logical value indicating **True** or **False**

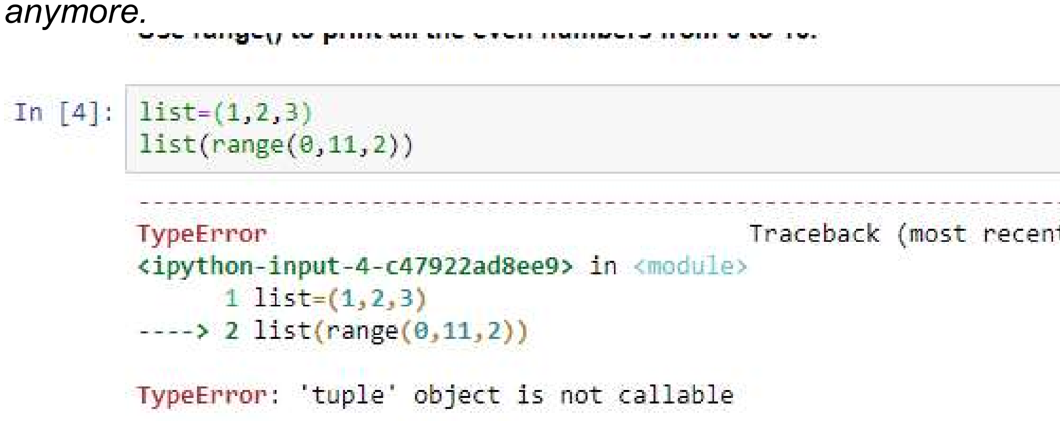
2/ Numbers:

- Concept:
  - Integer(int): Whole number
  - Floating Point(float): Number with a decimal
  - Variable and assigning values to them.
- Basis math:
  - Basis calculate
  - Modulo or “Mod” operator: “X % Y”. It will show the remainer of the first number (After perform devision)
  - Power 2, power 3, power 4 mean **2, **3, **4 or (Mũ 2, mũ 3, mũ 4)
  - We can use parentheses: “(Calculating operation)” the same way as usual math.

3/ Variable Assignment:

- Variable can be used to store even function or class or method. Like every thing in Python are object (OOP). Even when you delete the original. You still can call it by the alternative variable you assigned it to. (IF existed)
- Rules:
    - Name cannot start by number
    - Name cannot contain space, use "_" instead
    - We cannot use the symbols: ….etc (Check online if needed, or else just don't use any symbols =)))).
    - We should use lowercase only. Uppercase is recommended when you are a professional who want to create a global Python Variable which is used widely.
    - Avoid use names that might make misstatement like **list, strings, number, set**, … etc.
        - **Note:** In 20-April, I accidently found that this can cause the below error:

          *In this block of code, I want to call the function **list()** to create a **list** but Python thought that I call an element in the tuple "list".*
          Python interpret list as a variable name but a function anymore.



Use range() to print all the even numbers from 0 to 10.

```
In [4]: list=(1,2,3)
        list(range(0,11,2))

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent
<ipython-input-4-c47922ad8ee9> in <module>
      1 list=(1,2,3)
----> 2 list(range(0,11,2))

TypeError: 'tuple' object is not callable
```

    - Python use **dynamic typing** (run-time) which is different from **statically-typed** (Compile-time. Which mean we can change value and type of variable. Not like the most other language which do not allow to change variable type.
    - Anyway, it's easy to have an error due to this flexible data type. So always remember to check data type by function **Type()**.

4/ Strings:

a/ Basic concept

- Strings are actually sequence of characters.

- We can use single quote or double quote or combination of those. Like "This's the example" . So that we don't need to escapes the single quote in this sentence.
- As refered, string are actually sequences. Therefore, we can:
    - **Indexing**: We use square bracket to grab the single character from the strings. Like String[Index-number]. Ex: a[1]. We can also use reverse indexing: a[-1]. The first character is automatically set to index of zero "0"
    - **Slicing**: Is to grab a subsection of multiple characters – a slice of a strings. Syntax: [start:stop:step].
        - a = 'abcdefg'
        - a[1:] = 'c'
        - a[:1] = 'a' >>> so in Python, the **start** is exactly at the index we put, but **stop** is up to but **not including** character at the index we input.
        - a[0:2] ='ab'
        - a[::2] = 'aceg'
        - a[0:4:2]='ace'
        - a[::-1] = 'gfedcba' >>> this is reversing the string\
    - As we cannot access to specific item in string, we have to use split/indexing/slicing to split the string into different strings and uppercase the target related strings (which contain the need_to_be_uppercase item only)
- The popular escape character in Python: \n >>> enter to a new line and \t >>> tab.
- Len() function: len('I am') = 4
- Lines with hashtag at first are comments (?) We can use this in python or it just in notebook/example.
- Python does not support item assignment for string (it does for some other data structure-explaining later. *If you want to do so, you have to split the original string into smaller without the items needed to be changed and concatenate the data with new items* (Concatenate: "+")
- Keep in mind that Python only allow string operation w string and number operation with number. Which mean 1 +3 =4 but "1' + '3' = '13'
- Have an assignment x = 'Hello world'. Type x. then hit Tab and you will see all the attributes and method of Python. Remember the syntax – Ex: x.upper**().**
- X.split('1') mean to split the string without the item '1'.

b/ Printing Formating with string:

.**format()** method and **f-strings** (formatted string literals)

- Overall:
  - Use ^ < > to set the aligment., . - = ,… to chose what in the blank. Example as below
    - print('{0:=<8} | {1:-^8} | {2:.>8}'.format('Left','Center','Right'))
    - print('{0:=<8} | {1:-^8} | {2:.>8}'.format(11,22,33))
    - Left==== | -Center- | ...Right
    - 11====== | ---22--- | 33
- **.format():**
  - 'String here {} then also {}'.format('something1','something2')
  - print('{} {} {}').format('a','b','c','d')
  - print('{a} {b} {c} {d}'.format(a='string1', b='string 2', c='string 3', d='string 4')).
  - Float formatting: "{value:width.precision f}"
    - Width: Minimum width of the number
    - Precision: count of number after the decimal point) *Note*: But whatever the width is, numbers in the left of decimal point will all be showed.
    - If we have f after precision, the precision will equal how many number from decimal point (to the right of it) are showed.
    - If we don't, the precision will equal how many number from the beginning of the float (from the left to the right) are showed.
  - .format is available from 2.6 Python
- **F-string:**
  - Syntax: f'string {string 1} {string 2}'.f
  - This is m uch more shorter and readable.
  - This syntax is also much similar to other language.
  - F-string is available from Python 3.6
  - Syntax for float: result: {value:{width}.{precision}*'f'*} .
    - We can ignore and do not use the inside curved bracket, the result is the same as in .format(). But in case we don't directly use number but use variable, that's when curved bracket is compulsory.
  - Use !r to set the value to representative, not string. Reprentation mean including the quotation marks. Detail diff between !r and !s will be taught later in the course.

5/Lists in Python:

- **Lists** are ordered sequences that can hold a variety of object types.

- **Lists** use keys (Can be any type of data), [] brankets and commas.
- **List** is just like string which support indexing and slicing.
- The noticeable different btw lists and strings is that **strings are immutable** with **lists are not**, list's item are reasignable.
- Regular method:
    - *Append* method: add to end of the list
    - *Pop* method": remove the end from the list.
        - We can assign some variable to the pop item
        - Type the index position into between the parentless to pop out a specific item.
    - *Sort* and *reverse:* Notice that sort/.reverse is just a method/operation. So strings.sort()/lists.sort() have no value, we cannot assign any variable to this.

In Python, thers is a "None" value.

6/ Dictionaries in Python:

- **Dictionaries** ar unordered mapping for stored objects.
- Dictionaries use a **key-value paring** instead of using ordered sequences.
- This allow user quickly grab objects without need knowing index location.
- **Syntax**: {'key 1':'value 1','key 2':'value 2',etc}**;**
- **Between** dictionaries **and** list:
    - Dictionaries: Objects retrieved by key name. >> cannot be sorted/index-ed/slice-ed
    - Lists: Being retrieved by location, ordered sequences can be sorted/indexed/sliced.
- Unlike list, we don't need to use append to add new key to Dictionary. We can just recall as if there was a key there and pretend as if we are reassigning it to a new value. For example: d['Non exist key']='New value'
- Method:
    - the_dictionary.value()
    - the_dictionary.key()
- *Dictionary do not have order but later on this course we will learn about ordereddict, which would be use to keep the order. ?*

7/ Tuples:

- **Tuples** are very similar to **list**. The most different is the **immutability**. Once an element is inside a Tuple, it cannot be change.
- Tuples use **parentless** () Tuples = (1,2,3).
- Mean that Tuples are ordered and can be sorted or indexing or slicing.

- *Tuples is useful later on when we have already been an expert. In a large and complex thing, the immutability is a great solution to ensure that no element was changed by accident.*

8/ Sets:

- **Sets** are unordered unique element. Which mean there's only one representative for the same value.
- We create set by assign a variable to a set: a = set()

9/ Boolean:

- **Booleans** are operator that allow us to convey True or false statement.
- Later on there will be lesson on flow and logic. Those and Booleans are very important. (When we have enough knowledge to make his usefull to us).
- Remember that True/False/None start by a capital character. (Maybe this related to the Worldwide_well_knowed_variable refered in the string lecture)

10/ I/O With basic files in Python:

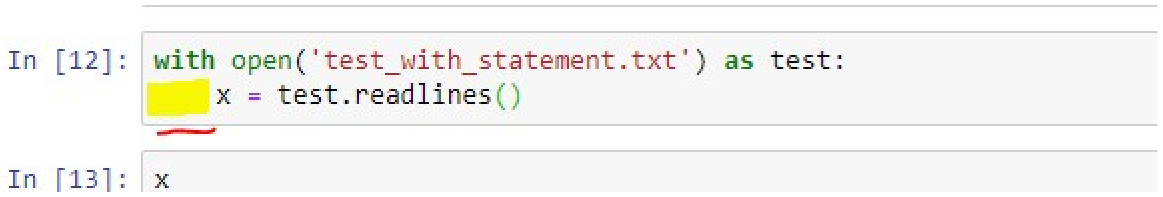
a/ I/O: Input and output

- Finally he refer to st call file path here. Bravooooooo!!!!! << Nope finally this not what I expected ☹
- Later, he will show how to call a file from anywhere.
  - We use method:
    - %%writefile: a magic functionality which only works in Jupyter notebook
    - .open()
    - .read(): Keep in mind that Python work as if there's a cursor scan through from the beginning to the end of the files.
    - .seek(): The cursor won't return to the first position after finish its job. Then we have to relocation it by this. The value should be inputed between the parentless is 0
    - .readlines(): This function return the file as a list. Each line will be a separate element of the list.
    - .close()
- Be notice that when you open a file, you have to close it to avoid errors. For instant, if you don't close it in Python, you cannot delete it anywhere, an error 'This action cannot be completed because the file is opened in Python" errors.

b/ With statement in Python:

- **With** statement help close the file when Python finish work with it.
- In the example, it's right after assigning contents to the variable 'x'.
    - **with open(**'the_file.txt'**) as** variable**:**

      (Intended block) contents = variable.read()

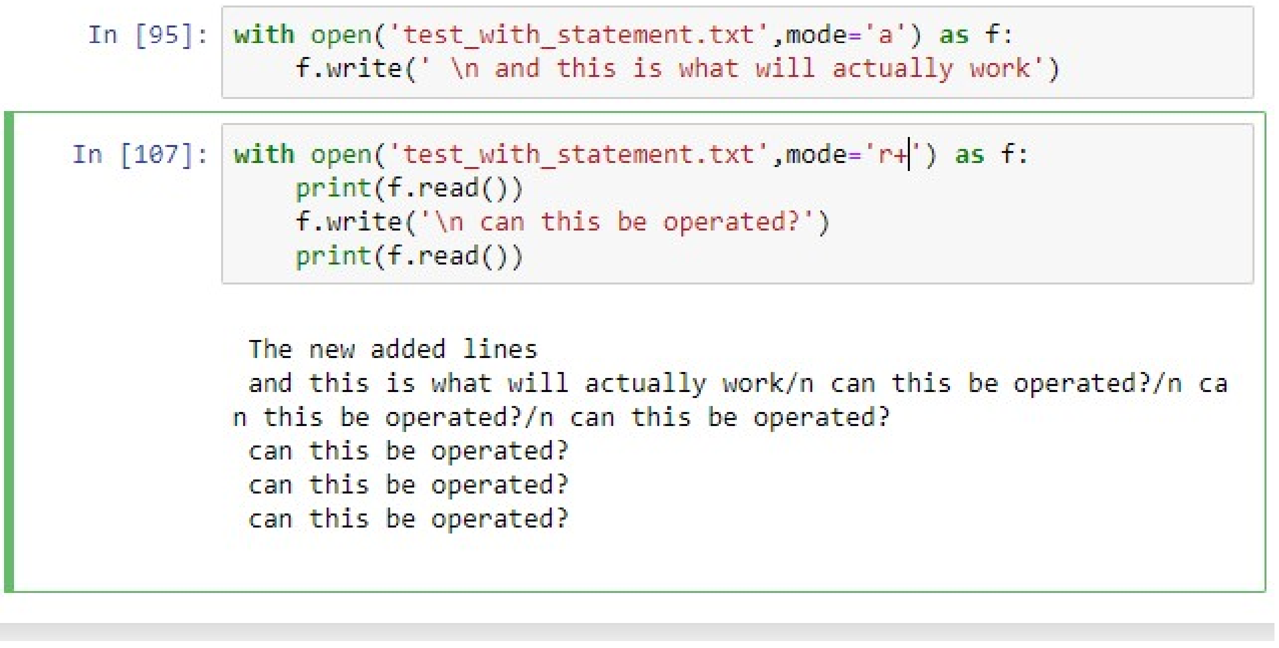


```
In [12]: with open('test_with_statement.txt') as test:
             x = test.readlines()

In [13]: x
```

(the yellow highlight is an intended block which will be discuss later in the course)

- **With** statement also support some mode as below:
    - 'r': read only
    - 'w': overwrite or create new file
    - 'a': append to existed file
    - 'r+': read and write file
    - 'w+: overwrite or create new file and read.

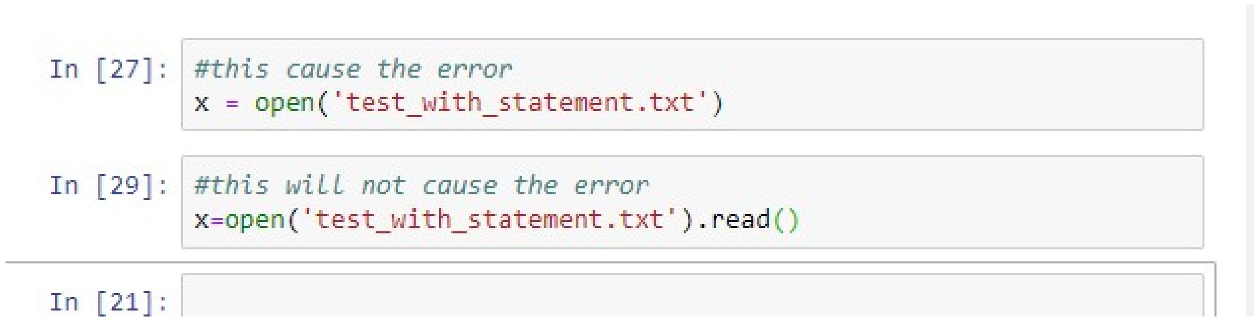


```
In [95]: with open('test_with_statement.txt',mode='a') as f:
             f.write(' \n and this is what will actually work')

In [107]: with open('test_with_statement.txt',mode='r+') as f:
              print(f.read())
              f.write('\n can this be operated?')
              print(f.read())

           The new added lines
           and this is what will actually work/n can this be operated?/n ca
          n this be operated?/n can this be operated?
           can this be operated?
           can this be operated?
           can this be operated?
```

To avoid keeping the file opening, we can also ***skip*** *the .open() method by directly assign the value to variable:*



```
In [27]: #this cause the error
         x = open('test_with_statement.txt')

In [29]: #this will not cause the error
         x=open('test_with_statement.txt').read()

In [21]:
```

https://www.guru99.com/reading-and-writing-files-in-python.html

*#Note on **With** statement and the solution I found: Those are **context manager** (Quản lý tài nguyên). Which is really important later as there's a limitation for number of opening file at the same time.*

#Note on file location:

- In windown we need to use duble backslashes, so that Python wouldn't treat the second backslashes as an escape character.

  (In MacOS. The file path use Forward slashes.)]

- To see what how the file path look, use 'pwd'.
-

III/*Resources for More Basic Practice:

Before you begin your assessment, I wanted to point out some helpful links for practice (don't worry about being able to do these exercises, I just want you to be aware of the links so you can visit them later, since we still haven't discussed functions, you won't be able to utilize a lot of these resources yet!):

1/Basic Practice:

http://codingbat.com/python

2/More Mathematical (and Harder) Practice:

https://projecteuler.net/archives

3/List of Practice Problems:

http://www.codeabbey.com/index/task_list

4/A SubReddit Devoted to Daily Practice Problems:

https://www.reddit.com/r/dailyprogrammer

5/A very tricky website with very few hints and touch problems (Not for beginners but still interesting)

http://www.pythonchallenge.com/

IV/ Python comparison operators:

- Comparison produce Boolean value which are True and False
- Some comparison: <, >, ==, '!=', <=, >=
- Some statement to chain comparison:
    - **and**
    - **or**
    - **not**

V/ Python Statement

- Note arisen during preparation:
    - **Statement:**
        - If, elif, else, and, or, not, with, break, continue, pass
        - **Continue**: I think without continue statement, the loop still continues. Let's check that. Or does the **continue** statement will return the right after the while line, which mean the while condition won't operate.
    - **If** statement accept anything that return the Boolean True value.
    - **Interate** and **interable**
    - **Shortcut** of mathematic value assignment.
    - **Loop.**
    - Check the infinity loop.
    - **Generator**
    - What if the :step: have a value < 1. Or a float ?. For range() function. << input of those must be integer.
    - I want to test which data structure will **enumerate**('abcde') create.
    - Crucial.
    - Notation (Ký hiệu)

1/ If, elif and else statements in Python:

- **Control flow.**

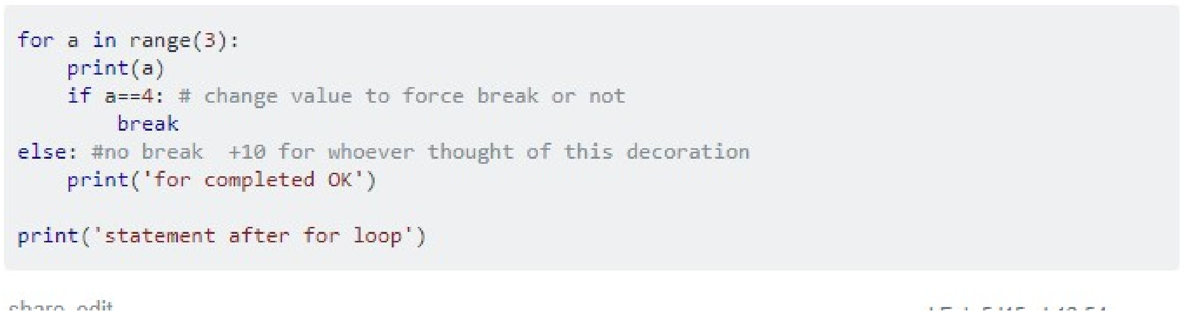
  - **if**
  - **elif**
  - **else**
- Control Flow syntax make use of **colon** and **indentation** (whitespace). These indentation system is crucial to Python and is what sets Python apart from other program language. With these indentation, Python code are easy to read and prototype (to be a prototype).

- **Syntax:**
  - **if** some_condition:
        #execute some code
  - **elif** some_other_condition
        #do something different
  - **else**
        #do something else.
  - These statement are line up to each other in notation lies.
  - **Note:** Python stop the loop right the after the first condition which have value of **True**

2/ For loops in Python

- **Interable-interate over** a sequences
- **Syntax:**
  - my_interable=[1,2,3]
    for item_name_this is a variable in **my**_interable:
        print(item_name_this_is_a_variable)
  - If we don't want to use any variable name, just use the underscore _.
- Dictionary:
  - **.items()**
  - **.keys()**
  - **.values()**
- We can also create a **list** by using for loop combine with .append()
- **Note:** we can nest multiple loop inside each other

- **Else** can also be use as *#no break.* Put it in the same level as for.



```
for a in range(3):
    print(a)
    if a==4: # change value to force break or not
        break
else: #no break  +10 for whoever thought of this decoration
    print('for completed OK')

print('statement after for loop')
```

share edit

3/ While loops:

- While loops continue to execute a block of code while some condition remains True Boolean value. (**Note**: **while** or for is an iterative loops, **if** is a statement to support **control flow** and execute once, not a loops.
    - **while** some_oolean_condition:
      #do something
      else:
      #do something else
- Useful statement:
    - **break**: This stop the loop.
    - **continue**: Go to the top of the closest enclosing loop. Which mean the remaining codes after continue statement are ignored.
    - **pass**: sometime, especially when we are building some function or method, we use for just to create a holder and no code need to be executed after the colon. Then, we use the statement pass to keep the whole cell continuing.
        - *>>> We do not use this much at this moment but keep this in mind as they are super useful and be use really frequently later. Especially when build functions and methods.*

4/ Useful operators:

- **range ()**: This help to generator number instead of save a huge list in memory.
    - range(10): generate number all the way up to but not including
    - range(3:10)
    - range(0:10:2) – For step
- **enumerate**(object): Result are paird values to key in
  range(0,len(object)).
- **zip** (*keys list*, values *list* , etc) **:** pair value in objects.

- **in**: to check if an item is in an object
  - for **dictionary:**
    - check on key
    - if want to check on value, use the method .values()
- **min** and **max** functions.
- **from** …. **Import** ….: to import method from external library
- **input**: this will transform any date type to string. We have to assign new type to it if needed.

5/ List comprehension:

- This's an alternative for using for loop + .append() to create **list**.
- **Syntax:** the_list = [x **for** x **in** the_existed_sequense]
- We can use **elif** and **else** in **list comprehension** but Jose recommend not to do this because it will become really hard to read latter. The ultimate purpose of code's looking is readable. Don't try to be one line guys and get your code ugly.

6/ Guessing game challenge:

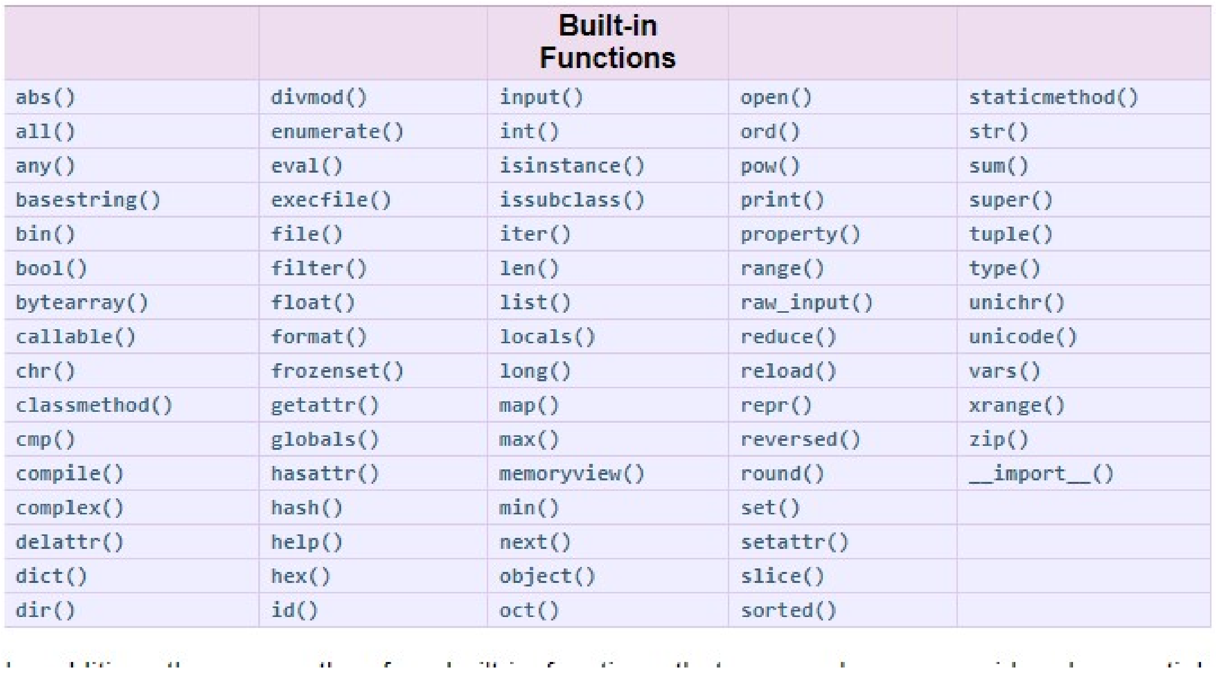
- **abs():** Absolute value

(Later section) In this game, I have been unable to check if the input was integer or not due to lack of knowledge of error and exception (I guess). Then when reach to the section 10. Please comeback and check this.

VI/ Functions and Methods

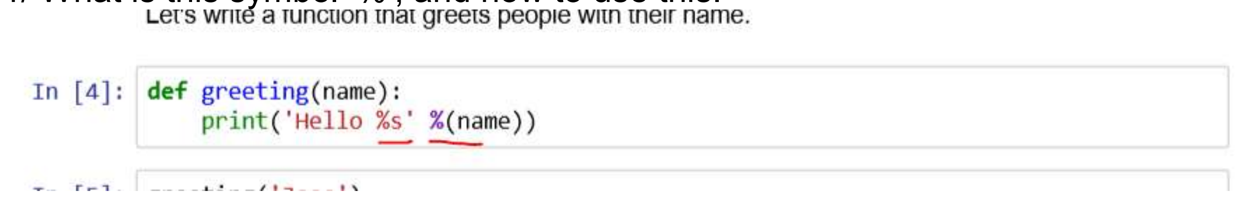
https://docs.python.org/2/library/functions.html

docstrings



		Built-in Functions		
abs()	divmod()	input()	open()	staticmethod()
all()	enumerate()	int()	ord()	str()
any()	eval()	isinstance()	pow()	sum()
basestring()	execfile()	issubclass()	print()	super()
bin()	file()	iter()	property()	tuple()
bool()	filter()	len()	range()	type()
bytearray()	float()	list()	raw_input()	unichr()
callable()	format()	locals()	reduce()	unicode()
chr()	frozenset()	long()	reload()	vars()
classmethod()	getattr()	map()	repr()	xrange()
cmp()	globals()	max()	reversed()	zip()
compile()	hasattr()	memoryview()	round()	__import__()
complex()	hash()	min()	set()	
delattr()	help()	next()	setattr()	
dict()	hex()	object()	slice()	
dir()	id()	oct()	sorted()	

*Note while preparing the lectures:
1/ What is this symbol '%', and how to use this.



Let's write a function that greets people with their name.

```
In [4]: def greeting(name):
            print('Hello %s' %(name))
```

2/ **def** statement
3/ **return** statement. Return make the result appear on the output. But would it still process the work and store the result in memory as other function (which doesn't return result on screen but we still can call those? << those are method, bro (3rd –May)
return allow function to return a result that **can be stored as a variable** later.
return the function will shut down as soon as it **returns** something. **So no need a break** statement.
4/ Write a function that check if a number is a prime number or not.
5/ We can use **global()** and **local()** to check if a variable is local or global.
6/ ***args**: This is asbitrary. Any word will do so long as it's preceded by an asterisk (*).
7/ ****kwargs**: Same as above
8/ **args** must go before **kwargs**

1/ Methods:

- use tab to see all the method supporting to object. Use shift tab or help(object.method) to see help.

THERE ARE DOCS.PYTHON.ORG/3 which provise very good documentation. Especially for advance uses. Library and language

2/ Function

- This is important which is a huge leap in program ability.
- Syntax:
  - **Def** name_of_the_function(name):

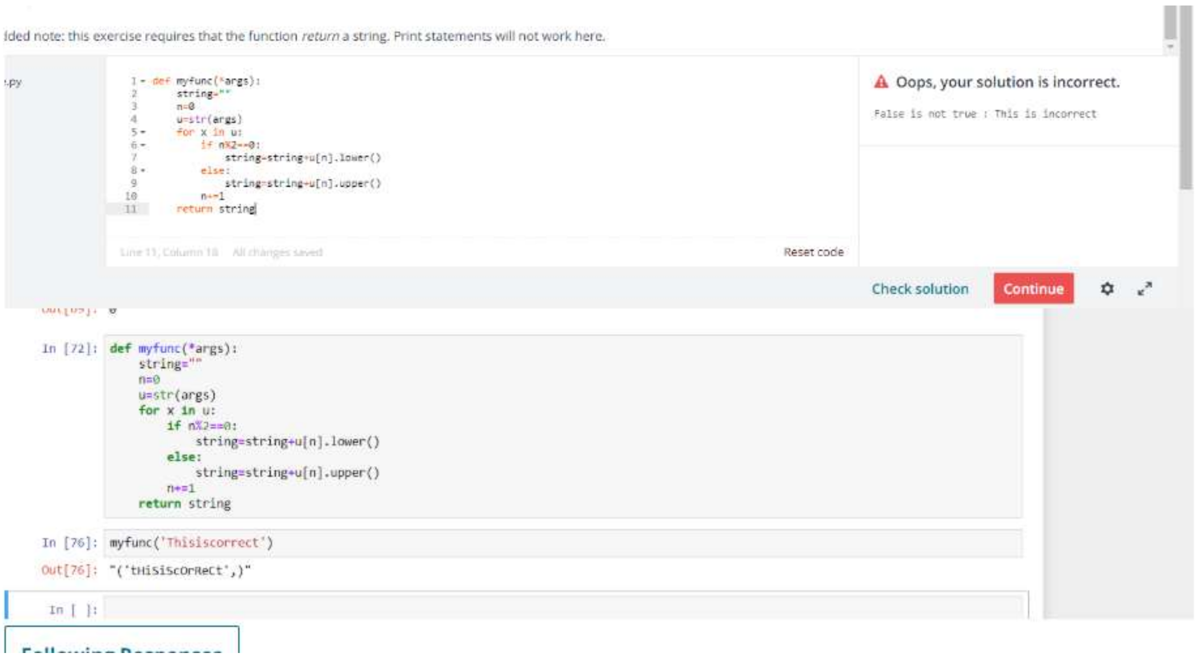
    ‘’’ or “””
    ‘Docstring: Information abt the Function
    Input:
    Output:
    ‘’’ or “””
    print(‘Hello’+name)
  - >>>>**Name_of_the_function(**‘Jose’)
    >>>>Hello Jose.
- **Docstring:** To help other people to understand what is going on
- At the end of docstring, use **return** would not only show the result but also allow us to assign the result to a variable.
- Type the name of function without
  - **Def** say_hello(name=**’NAME’**): (*Name is a default value we assign to the variable just in case no value is type in to the function)*
    return ‘Hello’+ name
  - Or we can do: **def** function_name(**a1,a2,a3,…**) to allow this function to have multiple input

3/ *args and **kwarg:

- ***args** and ****kwargs** help to shorten syntax for building function: **Def** my_function(*args,**kwargs).
- Syntax for **kwargs** in building function is a bit different from use in dictionary: we say **key1=’abc’, key2=’bcd’,key3=’def’.** Instead of use colon, we use the equal sign
- It’s not necessary to use exactly “args” and “kwargs”.
  - As long as we use the asterisk, it’s ok.

    - But but this is not recommended as it get the code become harder for the others to read.
- ***args** and ****kwargs** can be use together as combine.
    - All value of **args** will be in a **tuple** (use indexing) and **kwarg** is in a dict (use key name to call)
    - But the position must be match to syntax we build by **def**.

- Avoid using *args if thers's only 1 input. As it recognizes the input as an tupple. See the example below. It was a such miss understanding.



lded note: this exercise requires that the function *return* a string. Print statements will not work here.

```
def myfunc(*args):
    string=""
    n=0
    u=str(args)
    for x in u:
        if n%2==0:
            string=string+u[n].lower()
        else:
            string=string+u[n].upper()
        n+=1
    return string
```

Oops, your solution is incorrect.

```
False is not true : This is incorrect
```

```
In [72]: def myfunc(*args):
             string=""
             n=0
             u=str(args)
             for x in u:
                 if n%2==0:
                     string=string+u[n].lower()
                 else:
                     string=string+u[n].upper()
                 n+=1
             return string

In [76]: myfunc('Thisiscorrect')
Out[76]: "('tHiSiScOrReCt',)"

In [ ]:
```

4/ Practical Exercise:

- 'Strings'.**split()**
- ''.**join(a,b,c)**

- **for else** syntax :

5/ Lambda expression, Map and filter:

- **map** function:
  - To map a function to an iterable object and compute the function on each iterable.
  - **syntax: map**(*function_name_without_parentless,iterable object***)**
  - return **the result** of the function
  - We can **print** value in map type directly.
- **filter** function:
  - To filter item that return **True** when pass in the function.
  - **Syntax: filter(***function_name_without_parentless,iterable object***)**.
  - Return items in iterable object.
- **lambda expressions:** #anonymous *#one of the most usefull tools (and for beginner, the most confusing).*
  - **Syntax: lambda** *variable: the return value*
  - **def** *function_name()***: return** *value_you_want*

  - We can also assign a function name to **lambda** expression:
    - square = **lambda** num: num**2
    - and call it as square(2), result = 4. But this is not conventional.

6/ Nested Statements and Scope: **NEVER OVERWRITE BUILT-IN FUNCTION NAME** and **AVOID USING GLOBAL KEY WORD** if not really necessary, it's **DANGEROUS!.**

- **LEGB** rural format:
  - **L** – stand for local. In **def** or **lambda**
  - **E** – stand for enclosing functions?
  - **G** – stand for global (module)
  - **B** - stand for built-in (Python) ?
- We can simply understand that Python have different warehouse to store variable value.
  - Global ware house, local warehouse, Python warehouse.
  - Use the keyword **global + variables name** in the next line of the **def** line, to tell Python that you want everything happen inside the function will affect the variable at global warehouse. Not just in local.

VII/ First Milestone – Tic tac toe game:

- There're so many weakness in my codes:
    - I can put a # value to first item of the list so that later on it can be much more easier to call the item with the matching position in this game
    - My function are actually just some kind of put code in for a shorter view. They right now haven't had much specs as a function. My function should have:
        - 1/ Clear input
        - 2/ Clear output that can easily be understand from the function name.
        - 3/ I didn't know that function can have multiple objects/values in **return**
    - All the manual code should be typed outside of the function.

VIII/ Object Oriented Programming

1/ Overview:

- Object oriented programming (OOP) is a concept that in which, programming allows coder to create their own object with attribute and method.
- The method and attribute is call by the syntax: object_name**.method()** or object_name**.attribute**. Which is repeatable and organized.
- Those method act as function using information about the object to return results or change the current object.
- For **larger and larger script** of Python code, functions by themselves aren't enough for organization and repeatability. (*So true*). And repeated tesk created by **OOP** is **much more usable**.

- Syntax:
  class Name_of_class():

  #name of the class should be capitalize to distinguish them from function and variable.
  #code outside def is automatically run when we hit enter and create the ***class****.*

  def __init__ (**self**, param1,param2):

  #code in def __init__ is automatically run when we create an object under the ***class****.*
  self.param1 = param1

self.param2 = param2

def method_name(**self**):

#perform some action

print(**self**.param1)

- I **highly recommend** to use **comments**)start with #) and **docstring**. Which will show other programmer as well as your self when reuse the object class in the future.

2/ Attributes and class keyword

- Syntax:
  class Name_of_class():
  #name of the class should be capitalize to distinguish them from function and variable.

  Class object attribute
  Same to any instance under this class

  spiceies= 'mammal'

  def __init__ (**self**, param1,param2):

  # we can set default value by put **param1= 'default value'** here. We can reassign it anytime in code below.

  #the 1st and 3rd param1,param2 can be any name, they work as variable inside this attribute instanciation. The 2nd one is what matter, it is attribute's name. But we should use all 3 of them in the same, this make thing clearer and much more readable latter. ***We are assigning input to attributes. And we use those attribute later for coding in the class. Don't use the variable which represent for input.***

  self.param1 = param1
  self.param2 = param2

  def method_name(**self**):
  #perform some action

print(**self**.param1 **+** arg + Spic)

- This is somewhat difer from function which can call ‘name’ as an variable. But in method of object, because we use ‘name’ as an attribute, It’s can be called **only** by ‘**self.attribute’** which have reference to the instance.

- **Class object attribute**: Can be call directy as it’s consider as a global attribute(variable) in side the **class.**

- The arg is variable created for the method itself, so it can be use as variable in function

- Avoid use global variable. Just use attribute self.x in **class**. It will cause us confusing.

3/ Methods:

- Syntax:
  class Name_of_class():

  def __init__ (**self**, param1,param2):

  self.param1 = param1
  self.param2 = param2

  def method_name(**self,**arg):

  *#the “**self**” here is to let Python know that this is a method, not a regular function. So that here after, it interpret self method as method.*

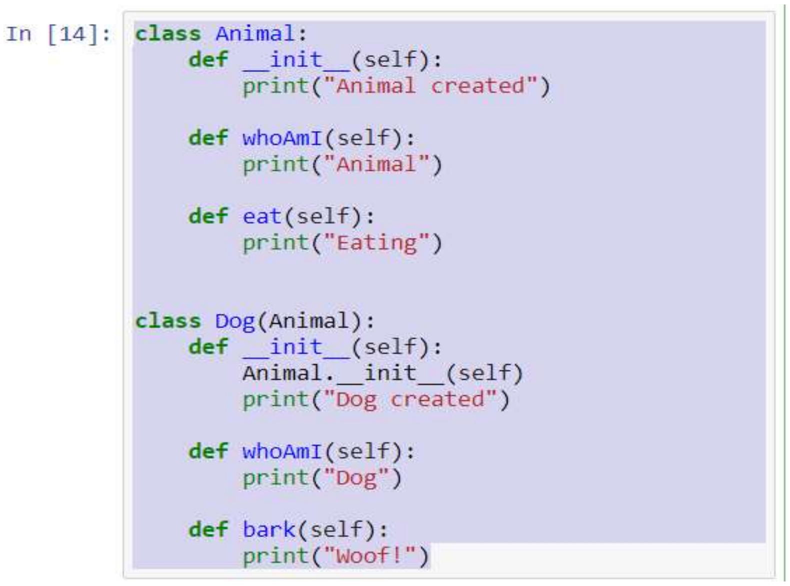
  #perform some action

  print(**self**.param1 **+** arg)

  #this is somewhat differs from function which can call ‘name’ as an variable. But in method of object, because we use ‘name’ as an attribute, It’s can be called only by ‘self.attribute’ which have reference to the instance.

  #the arg is variable created for the method itself, so it can be use as variable in function.

4/ Inheritance:



```
In [14]: class Animal:
             def __init__(self):
                 print("Animal created")

             def whoAmI(self):
                 print("Animal")

             def eat(self):
                 print("Eating")


         class Dog(Animal):
             def __init__(self):
                 Animal.__init__(self)
                 print("Dog created")

             def whoAmI(self):
                 print("Dog")

             def bark(self):
                 print("Woof!")
```

```
class Animal():
    def __init__(self):
        print('animal created')
    def eat(self):
        print("I eat what I eat")
class Dog(Animal):
    def __init__(self):
        Animal.__init__(self)
        print('Dog created')
        pass
    def eat(self):
        Animal.eat(self)
        print('I eat meat')
```

- Type Mother_class.method/attribute(**self**) in attribute/method builder to add Mother Classs's asset to Sub-Class. Sub Class will execute both original script from Mother class and new script (If available).

Polymorphism”

- **Won’t really need it until much later in my Python career ?**

6/ Special Method (Magic/Dunder method):

- any keyword or built-in method/function will do exactly what it does. But it will do some additional task if we tell them to by create a method in side by **def** + **__**method-name**__** (self):
- **del** is the key word to delete a variable from computer memory.

IX/ Modules and packages:

1/ Pip install and Pypi:

- Google search: “Python packages for What_ever_you_need.”
- (My opinion) This will be more usefull when I want to do something more particular, such as web developing or building API ?
- Pypi is a repository (kho lưu trữ?) for open-souce third-party Python packages- libraries.
- **pip install** is used in command line to install all those packages. This is a simple way to download packages directly from the Pypi repository
  - **pip install pack ages_name** <<< this can actually download packages directly from the internet.
  - in command promp, this code is run before run Python.
  - Then hit **from** library/packages_name **import** function_name()
- For **web development**, try google Python with **Django** or **Flask**. Those are also in Pypi pages.
- **I WILL REVIEW AGAIN AND COMPLETE THIS LATTER.** My head seem not receiving this right now :(
- This would cost me tons of time.

2/ Modules an Packages:

- **Modules** are scripts we call from other .py script. << files.
- **Packages** are collection of modules. <<< folders.

- There need to be **a postscript call __init__.py** in the folder. This lets Python know .py collection inside the folder is a collection of modules. Which is a package.
- A __pycache__ folder will be automatically created whenever we save a .py file.
- We can import function from modules, import module from package, from subpackage. For instance:
    - **From module**: from mymodule import myfunc
    - **From package:** from package import module
    - **From subpackage:** from package.subpackage import function
- Python doesn't recognize file/folder/package/module which isn't in the **same directory** with current program. Therefore all module and package must be put in the same folder. Deeper package and module need link to be call.:

  #This is to import function directly to our program
    - **from** main_package.subpackage.module **import** function_name

      #or class_name

      #Remember, the function name without parentless

      function_name()

      #This is to import the whole module. Line 2 is way to run a function from imported module.
    - **from** main_package.subpackage **import** module

      module.function_name()

      #or class_name()

- All script which is in **level 0 identation** will be automatically ran whenever being imported.

3/ __name__ and "__main__":

- **If** __name__ == "__main__": to know whether a module is being used as an imported or is being run directly in the module file.

  #__name__ is a built-in variable

X/ Errors and Exception Handling:

1/ Errors and Exception Handling:

- **try**: Block of code to be attempt
- **except:** Block of code which will be execute in case there is an error in **try** block.
    - But except go with an type of error. Which is similar to if statement
    - If except go alone, the code below it run with any kind of errors.
- **else:** This will be executed if no errors was found in **except**.
  finally: A final block of code which finally be execute, regardless of errors.
- **break** and **continue** will not stop the **try/exception/else/finally** from being excute. Anything which is effected is after the error/exception handling block.

2/ Pylint Overview:

#When we work with large team, it's really important to test whether the old code still run well after teammate have change/update to the project.

Here Jose focus on two testing tools:

- **pytlint:** this library looks at our code and reports back possible issues.\
    - First just import the library pylint by **pip install pylint.**
    - Then to run the assessment: pylint your_file_name.py
    - pylint is much more usual when come to teamwork/ large project
- **unittest:** this is an built-in library will allow to test our own program and check we are getting desired outputs.
- **unittest:**
    - First, import the library unittest.
    - Then import the module/package/function or what ever you want
    - Start by create a new class which is sub class of unittest.TestCase: Class name (unittest.TestCase):
    - Create function :
        - Get some variable to assign for result of imported function.
        - Compare result to expected result by self.assertEqual(result, expected_result)
    - If __name__=="__main__":

      Unittest.main() *#this will excute all necessary checking we put in.*

Python Enhancement Proposal << remember r to check out this

XI/ Second Mile stone – Black Jack game:

- First try, after more than 9 hours keep trying. I give up. This is so hard.
- Lesson from this try:
    - 1/ I overuse too much on function and method. I put too much variable/ attribute outside of the function/method in to it.
        - This made thing really confusing when come to handling errors.
        - Also, this can be even worse when we want to apply those function/method to other scenario. And what is method, function's purpose ? **To be reusable**. Not just to replace a block of code once.
    - 2/ I didn't know that module random (or library nhỉ) have a shuffle function.
    - 3/ Jose cut some complicated rules of the game. Maybe I should too.
- I will try build this game again. And next time:
    - 1/ I will build function and method exactly what it should be built for.
    - 2/ Use shuffle.
    - 3/ I will do it once with the simple rules. But I will do it once again with full rules. It's challenging.

XII/ Python decorators:

1/ Overview: (Just take a look, no serious thing).

- This is an advance Python concept.
- Functionality of a function **–** extra code in a function.
- The concept **decorators**
    - Quick add new functionality to old function.
    - Add / Delete 1 line after using which is **@some_decorators**
        - **@some_decorators**

```
@some_decorators
def simpe_func():
        #Do simple stuff return something.
```

- function created inside a function can only be executed inside that function.
- We can not only nested a function, return a function but also **pass** a function # put in argument position.
-

2/ Main content:

- So decorator is a concept that we can use function as an object (and we should know too, everything in Python is interpret as an object). What ever can be done to an object can be done to an function.:
    - Function can be used as
        - A variable
        - An argument
        - Can be returned in a function's result.
- An instance:

```
def new_decorator(nested_function):

        def wrap_func():

                print("Some extra code, before the nested_function")

                nested_function()

                print("Some extra code, after the nested_function")

        return wrap_func

def decorator_func():

        print("I want to be decorated!")
```

- >>> run code : u= new_decorator(decorator_func).
- Result:

```
                Some extra code, before the nested_function

                I want to be decorated!

                Some extra code, after the nested_function
```

- Instead of traditional way, we can also use **@**. @new_decorator, Write the function_need_to_be_decorated below. And Python will automatically add it to position of args in new_decorator

- Notice that I have assign variable name u to the return function **wrap_func**. Then if we execute **u()** the same result as above will be return.

  # Jose said this is used usually in web development, web dev framework as Flask or Django. Use decorator when rendering website or point to another website?

XIII/ Generators:

- New concept:
  - State suspension. #what is the different? As I tested, value returned from a function is also interable.

    # As I see, normal return can only return value once. Mean if we have an iterable object, we have to return them all at once.

    #But with **yield** statement, we can return each iterating item one by one. I return a value and wait until the next value is called for.
- Iterable: object that can be interate in for loop. And will create iterator by the function iter().
- Iterator: Object that can be pass in next() built-in function. Such object remember where it is during the iteration.
- Use **yield** statement.

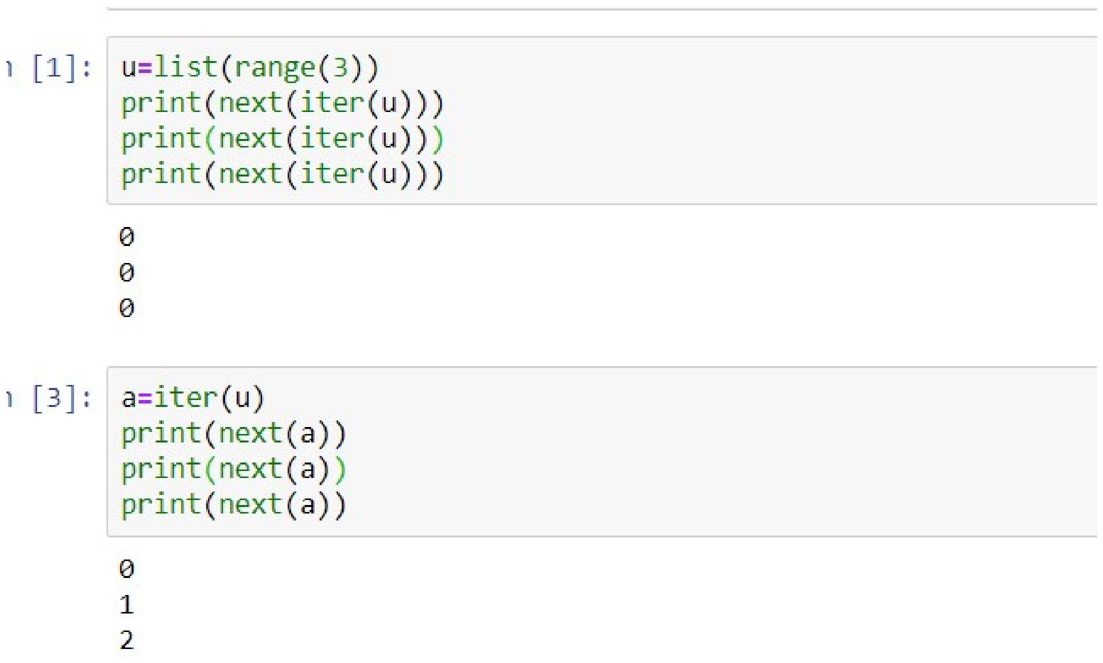
# In the below example, I have a significant misunderstood. After testing, I have a result that iter(u) or range() aren't an iteration.

# But it's turn out they aren't. =)). The result are iterable but aren't iteration or generator

# There're 2 type of object can be use be next: generator (result of a yield function) and iterator (result of iter()).

# One more: iter(something) cannot be iterated directly as each time iter(something) appear. Python receive this as a function call and iter() was operated, and a new iteration was created.

#So we have to assign a variable to iter(something). Then each time we call the result (u). None was created. And it'll continue from where it left off.



```
[1]: u=list(range(3))
     print(next(iter(u)))
     print(next(iter(u)))
     print(next(iter(u)))
```

```
0
0
0
```

```
[3]: a=iter(u)
     print(next(a))
     print(next(a))
     print(next(a))
```

```
0
1
2
```