Bootcamp Python

Day00 - Basic stuff - Eleven Commandments

The goal of the day is to get started with the Python language.

Notions of the day

Basic setup, variables, types, functions, ...

General rules

- The version of Python to use is 3.7, you can check the version of Python with the following command: **`python -V`**
- The norm: during this bootcamp you will follow the Pep8 standards https://www.python.org/dev/peps/pep-0008/
- The function eval is never allowed.
- The exercises are ordered from the easiest to the hardest.
- Your exercises are going to be evaluated by someone else, so make sure that your variable names and function names are appropriate and civil.
- Your manual is the internet.
- You can also ask questions in the dedicated channel in the 42 AI Slack: 42-ai.slack.com.
- If you find any issue or mistakes in the subject please create an issue on our dedicated repository on Github: https://github.com/42-AI/bootcamp_python/issues.

Helper

How do you install and link Python in the $PATH? That's the first exercise!

Ensure that you have the right Python version.

```
$> which python
/goinfre/miniconda/bin/python
$> python -V
Python 3.7.*
$> which pip
/goinfre/miniconda/bin/pip
```

Exercise 00 - $PATH.

Exercise 01 - Rev Alpha.

Exercise 02 - The Odd, the Even and the Zero.

Exercise 03 - Functional file.

Exercise 04 - Elementary.

Exercise 05 - The right format.
Exercise 06 - A recipe.
Exercise 07 - Shorter, faster, pythonest.
Exercise 08 - S.O.S.
Exercise 09 - Secret number.
Exercise 10 - Loading bar !

Exercise 00 - $PATH

Turnin directory :	ex00
Files to turn in :	installer.sh
Forbidden function :	None
Remarks :	n/a

We noticed that the students had trouble with the setup of their Python path. You are thus going to learn how to master the setup of the Python path!

In order to make our life easier, let's create a script to check if the correct Python version is installed, and install it if this is not the case. To install Python, you are going to use the 'miniconda' installer :
https://docs.conda.io/projects/conda/en/latest/user-guide/install/macos.html.

Python is a binary that you can find in **`/usr/local/bin`** for example.
When you enter **`python`** in your terminal, your OS is going to look in your $PATH var in the */bin folder for a python binary.
The Python interpreter that will be chosen is generally the one which is found first when searching in the **`bin`** directories.

When you are installing Python with miniconda, you are actually going to be adding a new **`bin`** to your system with a version of Python inside. This Python folder will have to be installed in the **`/goinfre`** to save space on your session.

To use this newly installed Python, you have 2 options:
- use the absolute path **`mypath/mydir/bin/python`**.
- add the bin dir to your shell's PATH variable via **`export PATH=mypath/mydir/bin:$PATH`**. (To verify if this second option has worked, you can use the command **`which python`**.)

```
$> ./installer.sh install-python
Python has been installed.
$> ./installer.sh install-python
Python is already installed, do you want to reinstall it ?
[yes|no]> yes
Python has been removed.
Python has been installed.
$> python -V
Python 3.7.*
$> ./installer.sh install-python
Python is already installed, do you want to reinstall it ?
[yes|no]> no
exit.
```

WARNING: Do not install python using brew on 42's computers, it will overload your session.

Exercise 01 - Rev Alpha

Turnin directory :	ex01
Files to turn in :	exec.py
Forbidden function :	None
Remarks :	n/a

You will have to make a program that reverses the order of a string and the case of its words. If we have more than one argument we have to merge them into a single string and sperate each arg by a ' ' (space char).

```
$> python exec.py "Hello World!" | cat -e
!DLROw OLLEh$
$> python exec.py "Hello" "my Friend" | cat -e
DNEIRf YM OLLEh$
$> python exec.py
$>
```

Exercise 02 - The Odd, the Even and the Zero.

Turnin directory :	ex02
Files to turn in :	whois.py
Forbidden function :	None
Remarks :	n/a

You will have to make a program that checks if a number is odd, even or zero.
The program will accept only one parameter, an integer.

```
$> python whois.py 12
I'm Even.
$> python whois.py 3
I'm Odd.
$> python whois.py
$> python whois.py 0
I'm Zero.
$> python whois.py Hello
ERROR
$> python whois.py 12 3
ERROR
```

Exercise 03 - Functional file.

Turnin directory :	ex03
Files to turn in :	count.py
Forbidden function :	None
Remarks :	n/a

Create a function called **`text_analyzer`** that displays the sums of upper characters, lower characters, punctuation characters and spaces in a given text.

`text_analyzer` will take one parameter: the text to analyze. You should handle the case where the text is empty (maybe use a default value). If there is no text passed to the function, the user is prompted to give one.

Test it in the Python console:

```
$> python
>>> from count import text_analyzer
>>> text_analyzer("Python 2.0, released 2000, introduced
features like List comprehensions and a garbage collection
system capable of collecting reference cycles.")
The text contains 143 characters:
- 2 upper letters
- 113 lower letters
- 4 punctuation marks
- 18 spaces
>>> text_analyzer("Python is an interpreted, high-level,
general-purpose programming language. Created by Guido van
Rossum and first released in 1991, Python's design philosophy
emphasizes code readability with its notable use of significant
whitespace.")
The text contains 234 characters:
- 5 upper letters
- 187 lower letters
- 8 punctuation marks
- 30 spaces
>>> text_analyzer()
What is the text to analyse?
>> Python is an interpreted, high-level, general-purpose
programming language. Created by Guido van Rossum and first
released in 1991, Python's design philosophy emphasizes code
readability with its notable use of significant whitespace.
The text contains 234 characters:
- 5 upper letters
- 187 lower letters
- 8 punctuation marks
```

```
- 30 spaces
```

You're free to write your docstring and format it the way you want.

```
>>> print(text_analyzer.__doc__)
    This function counts the number of upper characters, lower characters,
    punctuation and spaces in a given text.
```

Exercise 04 - Elementary.

Turnin directory :	ex04
Files to turn in :	operations.py
Forbidden function :	None
Remarks :	n/a

You will have to make a program that prints the results of the four elementary mathematical operations of arithmetic (addition, subtraction, multiplication, division) and the modulo operation. This should be accomplished by writing a function that takes 2 numbers as parameters and returns 5 values, as formatted in the console output below.

```
$> python operations.py 10 3
Sum:         13
Difference: 7
Product:     30
Quotient:    3.3333333333333335
Remainder: 1
$>
$> python operations.py 42 10
Sum:         52
Difference: 32
Product:     420
Quotient:    4.2
Remainder: 2
$>
$> python operations.py 1 0
Sum:         1
Difference: 1
Product:     0
Quotient:    ERROR (div by zero)
Remainder: ERROR (modulo by zero)
$>
$> python operations.py
Usage: python operations.py
Example:
    python operations.py 10 3
$>
$> python operations.py 12 10 5
InputError: too many arguments
Usage: python operations.py
Example:
    python operations.py 10 3
```

```
$>
$> python operations.py "one" "two"
InputError: only numbers
Usage: python operations.py
Example:
    python operations.py 10 3
$>
$> python operations.py "512" "63.1"
InputError: only numbers
Usage: python operations.py
Example:
    python operations.py 10 3
```

Exercise 05 - The right format.

Turnin directory :	ex05
Files to turn in :	kata00.py kata01.py kata02.py kata03.py kata04.py
Forbidden function :	None
Remarks :	n/a

Let's get familiar with the useful concept of **string formatting** through a kata series.

kata00

```
t = (19,42,21)
```

Including the tuple above in your file, write a program that dynamically builds up a formatted string like the following:

```
$> python kata00.py
The 3 numbers are: 19, 42, 21
```

kata01

```
languages = {
    'Python': 'Guido van Rossum',
    'Ruby': 'Yukihiro Matsumoto',
    'PHP': 'Rasmus Lerdorf',
    }
```

Using the `languages` dictionary above, a similar exercise:

```
$> python kata01.py
Python was created by Guido van Rossum
Ruby was created by Yukihiro Matsumoto
PHP was created by Rasmus Lerdorf
```

kata02

```
(3,30,2019,9,25)
```

Given the tuple above, whose values stand for: **(hour, minutes, year, month, day)**, write a program that displays it in the following format:

```
$> python kata02.py
09/25/2019 03:30
```

kata03

```
phrase = "The right format"
```

Write a program to display the string above right-aligned with '-' padding and a total length of 42 characters:

```
$> python kata03.py | cat -e
---------------------------The right format%
$> python kata03.py | wc -c
    42
```

kata04

```
( 0, 4, 132.42222, 10000, 12345.67)
```

Given the tuple above, return the following result:

```
$> python kata04.py
day_00, ex_04 : 132.42, 1.00e+04, 1.23e+04
```

Exercise 06 - A recipe.

Turnin directory :	ex06
Files to turn in :	recipe.py
Forbidden function :	None
Remarks :	n/a

It is time to discover Python dictionaries. Dictionaries are collections that contain mappings of unique keys to values.
Hint: check what is a nested dictionary in Python.

First, you will have to create a cookbook dictionary called **`cookbook`**.

`cookbook` will store 3 recipes:
- sandwich
- cake
- salad

Each recipe will store 3 values:
- ingredients: a **list** of ingredients
- meal: type of meal
- prep_time: preparation time in minutes

Sandwich's ingredients are **ham**, **bread**, **cheese** and **tomatoes**. It is a **lunch** and it takes **10** minutes of preparation.
Cake's ingredients are **flour**, **sugar** and **eggs**. It is a **dessert** and it takes **60** minutes of preparation.
Salad's ingredients are **avocado**, **arugula**, **tomatoes** and **spinach**. It is a **lunch** and it takes **15** minutes of preparation.

1. Get to know dictionaries. In the first place, try to print only the **`keys`** of the dictionary. Then only the **`values`**. And to conclude, all the **`items`**.

2. Write a function to print a recipe from **`cookbook`**. The function parameter will be: name of the recipe.

3. Write a function to delete a recipe from the dictionary. The function parameter will be: name of the recipe.

4. Write a function to add a new recipe to **`cookbook`** with its ingredients, its meal type and its preparation time. The function parameters will be: name of recipe, ingredients, meal and prep_time.

5. Write a function to print all recipe names from **`cookbook`**. Think about formatting the output.

6. Last but not least, make a program using the four functions you just created.

The program will prompt the user to make a choice between printing the cookbook, printing only one recipe, adding a recipe, deleting a recipe or quitting the cookbook.

It could look like the example below but feel free to organize it the way you want to:

```
$> python recipe.py
Please select an option by typing the corresponding number:
1: Add a recipe
2: Delete a recipe
3: Print a recipe
4: Print the cookbook
5: Quit
>> 3
Please enter the recipe's name to get its details:
>> cake
Recipe for cake:
Ingredients list: ['flour', 'sugar', 'eggs']
To be eaten for dessert.
Takes 60 minutes of cooking.
```

Your program must continue running until the user exits it (option 5):

```
$> python recipe.py
Please select an option by typing the corresponding number:
1: Add a recipe
2: Delete a recipe
3: Print a recipe
4: Print the cookbook
5: Quit
>> 5
Cookbook closed.
```

The program will also continue running if the user enters a wrong value.
It will prompt the user again until the value is correct:

```
$> python recipe.py
Please select an option by typing the corresponding number:
1: Add a recipe
2: Delete a recipe
3: Print a recipe
4: Print the cookbook
5: Quit
>> test
This option does not exist, please type the corresponding number.
To exit, enter 5.
>>
```

Exercise 07 - Shorter, faster, pythonest.

Turnin directory :	ex07
Files to turn in :	filterwords.py
Forbidden function :	filter
Remarks :	n/a

Using list comprehensions, you will have to make a program that removes all the words in a string that are shorter than or equal to n letters, and returns the filtered list with no punctuation.
The program will accept only two parameters: a string, and an integer n.

```
$> python filterwords.py "Hello, my friend" 3
['Hello', 'friend']
$> python filterwords.py "A robot must protect its own existence as long as
such protection does not conflict with the First or Second Law" 6
['protect', 'existence', 'protection', 'conflict']
$> python filterwords.py Hello World
ERROR
$> python filterwords.py 300 3
ERROR
```

Exercise 08 - S.O.S.

Turnin directory :	ex08
Files to turn in :	sos.py
Forbidden function :	None
Remarks :	n/a

You will have to make a function which encodes strings into Morse code.
The input will accept all alphanumeric characters.

```
$> python sos.py "SOS"
... --- ...
$> python sos.py
$> python sos.py "HELLO / WORLD"
ERROR
$> python sos.py "96 BOULEVARD" "Bessiere"
----. -.... / -... --- ..- .-.. . ...- .- .-. -.. / -... . ... ... .. . .-.
.
```

@ref: https://morsecode.scphillips.com/morse.html

Exercise 09 - Secret number.

Turnin directory :	ex09
Files to turn in :	guess.py
Forbidden function :	None
Remarks :	n/a

You will have to make a program that will be an interactive guessing game. It will ask the user to guess a number between 1 and 99. The program will tell the user if their input is too high or too low. The game ends when the user finds out the secret number or types **`exit`**.

You will have to import the **`random`** module with the **`randint`** function to get a random number.
You have to count the number of trials and print that number when the user wins.

```
$> python guess.py
This is an interactive guessing game!
You have to enter a number between 1 and 99 to find out the secret number.
Type 'exit' to end the game.
Good luck!
What's your guess between 1 and 99?
>> 54
Too high!
What's your guess between 1 and 99?
>> 34
Too low!
What's your guess between 1 and 99?
>> 45
Too high!
What's your guess between 1 and 99?
>> A
That's not a number.
What's your guess between 1 and 99?
>> 43
Congratulations, you've got it!
You won in 5 attempts!
```

If the user discovers the secret number on the first try, tell them.
Bonus: if the secret number is 42, make a reference to Douglas Adams.

```
$> python guess.py
What's your guess between 1 and 99?
>> 42
The answer to the ultimate question of life, the universe and everything is
```

```
42.
Congratulations! You got it on your first try!
```

Other example:

```
$> python guess.py
This is an interactive guessing game!
You have to enter a number between 1 and 99 to find out the secret number.
Type 'exit' to end the game.
Good luck!
What's your guess between 1 and 99?
>> exit
Goodbye!
```

Exercise 10 - Loading bar !

Turnin directory :	ex10
Files to turn in :	loading.py
Forbidden function :	None
Remarks :	n/a

This is a bonus exercice! You are about to discover the **`yield`** operator!
So let's create a function called **`ft_progress(list)`**.

The function will display the progress of a **`for`** loop.

```
listy = range(1000)
ret = 0
for elem in ft_progress(listy):
    ret += (elem + 3) % 5
    sleep(0.01)
print()
print(ret)
```

```
$> python script.py
ETA: 8.67s [ 23%][=====>                ] 233/1000 | elapsed time 2.33s
...
2000
```

```
listy = range(3333)
ret = 0
for elem in ft_progress(listy):
    ret += elem
    sleep(0.005)
print()
print(ret)
```

```
$> python script.py
ETA: 14.67s [ 9%][=>                    ] 327/3333 | elapsed time 1.33s
...
5552778
```