42 ARTIFICIAL INTELLIGENCE

# Python Bootcamp - Module 00

## Basic stuff - The Eleven Commandments

*Summary: The goal of this module is to get you started with Python*

# Chapter I

# Common Instructions

- The version of Python recommended to use is 3.7, you can check the version of Python with the following command: `python -V`

- The norm: during this bootcamp, it is recommended to follow the PEP 8 standards, though it is not mandatory. You can install pycodestyle which is a tool to check your Python code.

- 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.

- If you are a student from 42, you can access our Discord server on 42 student's associations portal and ask your questions to your peers in the dedicated Bootcamp channel.

- You can learn more about 42 Artificial Intelligence by visiting our website.

- If you find any issue or mistake in the subject please create an issue on 42AI repository on Github.

- We encourage you to create test programs for your project even though this work **won't have to be submitted and won't be graded**. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.

- We are constantly looking to improve these bootcamps, and your feedbacks are essential for us to do so !
You can tell us more about your experience with this module by filling this form. Thank you in advance and good luck for this bootcamp !

# Contents

# Chapter II

# Exercise 00

| | Exercise : 00 | |
|---|---|---|
| | $PATH | |
| Turn-in directory : *ex00/* | | |
| Files to turn in : `answers.txt, requirements.txt` | | |
| Forbidden functions : `None` | | |

*The first thing you need to do is to install Python.*

Most modern Unix-based systems have a `python` interpreter installed by default, but its version might be lower/higher than the one used for these modules. It is also possible that the default `python` command uses a version 2.x (for legacy reasons). This is obviously very confusing for a new developper.

```
$> python -V
$> python3 -V
```

To deal with those versions issues we will use `conda`. This program allows you to manage your Python packages and several working environments.

*Note: the actual requirement is to use a Python 3.7.X version. You are free to use a different program/utilities to achieve this goal. At your own risk.*

# Conda manual installation

*Go to the next section for an automated installation.*

We recommend the following path for your `conda` folder.

```
$> MYPATH="/goinfre/$USER/miniconda3"
```

## 1. Download & Install conda

```
# For MAC
$> curl -LO "https://repo.anaconda.com/miniconda/Miniconda3-latest-MacOSX-x86_64.sh"
$> sh Miniconda3-latest-MacOSX-x86_64.sh -b -p $MYPATH

# For Linux
$> curl -LO "https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh"
$> sh Miniconda3-latest-Linux-x86_64.sh -b -p $MYPATH
```

## 2. Initial configuration of conda

```
# For zsh
$> $MYPATH/bin/conda init zsh
$> $MYPATH/bin/conda config --set auto_activate_base false
$> source ~/.zshrc

# For bash
$> $MYPATH/bin/conda init bash
$> $MYPATH/bin/conda config --set auto_activate_base false
$> source ~/.bash_profile
```

## 3. Create a dedicated environment for 42AI !

```
$> conda create --name 42AI-$USER python=3.7 jupyter pandas pycodestyle numpy
```

## 4. Check your 42AI Python environment

```
$> conda info --envs
$> conda activate 42AI-$USER
$> which python
$> python -V
$> python -c "print('Hello World!')"
```

## Help !

- **I have lost my miniconda3 folder !** Repeat step 1 and 3.

- **I have lost my home directory !** Repeat step 2.

# Conda automated installation

Copy the following script into a file, launch it and follow the instructions. It downloads and installs miniconda in a `/goinfre` subfolder and creates a `python` environment in `conda`.

```bash
#!/bin/bash

function which_dl {
    # If operating system name contains Darwnin: MacOS. Else Linux
    if uname -s | grep -iqF Darwin; then
        echo "Miniconda3-latest-MacOSX-x86_64.sh"
    else
        echo "Miniconda3-latest-Linux-x86_64.sh"
    fi
}

function which_shell {
    # if $SHELL contains zsh, zsh. Else Bash
    if echo $SHELL | grep -iqF zsh; then
        echo "zsh"
    else
        echo "bash"
    fi
}

function when_conda_exist {
    # check and install 42AI environement
    printf "Checking 42AI-$USER environment: "
    if conda info --envs | grep -iqF 42AI-$USER; then
        printf "\e[33mDONE\e[0m\n"
    else
        printf "\e[31mKO\e[0m\n"
        printf "\e[33mCreating 42AI environnment:\e[0m\n"
        conda update -n base -c defaults conda -y
        conda create --name 42AI-$USER python=3.7 jupyter numpy pandas pycodestyle -y
    fi
}

function set_conda {
    MINICONDA_PATH="/goinfre/$USER/miniconda3"
    CONDA=$MINICONDA_PATH"/bin/conda"
    PYTHON_PATH=$(which python)
    REQUIREMENTS="jupyter numpy pandas pycodestyle"
    SCRIPT=$(which_dl)
    MY_SHELL=$(which_shell)
    DL_LINK="https://repo.anaconda.com/miniconda/"$SCRIPT
    DL_LOCATION="/tmp/"

    printf "Checking conda: "
    TEST=$(conda -h 2>/dev/null)
    if [ $? == 0 ] ; then
        printf "\e[32mOK\e[0m\n"
        when_conda_exist
        return
    fi
    printf "\e[31mKO\e[0m\n"
    if [ ! -f $DL_LOCATION$SCRIPT ]; then
        printf "\e[33mDonwloading installer:\e[0m\n"
        cd $DL_LOCATION
        curl -LO $DL_LINK
        cd -
    fi
    printf "\e[33mInstalling conda:\e[0m\n"
    sh $DL_LOCATION$SCRIPT -b -p $MINICONDA_PATH
    printf "\e[33mConda initial setup:\e[0m\n"
    $CONDA init $MY_SHELL
    $CONDA config --set auto_activate_base false

    printf "\e[33mCreating 42AI-$USER environnment:\e[0m\n"
    $CONDA update -n base -c defaults conda -y
    $CONDA create --name 42AI-$USER python=3.7 jupyter numpy pandas pycodestyle -y
```

```
    printf "\e[33mLaunch the following command or restart your shell:\e[0m\n"
    if [ $MY_SHELL == "zsh" ]; then
        printf "\tsource ~/.zshrc\n"
    else
        printf "\tsource ~/.bash_profile\n"
    fi
}

set_conda
```

Don't forget to check your 42AI Python environment !

```
conda info --envs
conda activate 42AI-$USER
which python
python -V
python -c "print('Hello World!')"
```

# (Finally) getting started

Now that your setup is ready to run, here are a few questions that need to be solved using `python`, `pip` or `conda`. Save your answers in a file `answers.txt` (one answer per line and per question), and check them with your peers.

Find the commands to:

- Output a list of installed packages and their versions.

- Show the package metadata of `numpy`.

- Remove the package `numpy`.

- (Re)install the package `numpy`.

- Freeze your `python` packages and their versions in a `requirements.txt` file you have to turn-in.

# Chapter III

# Exercise 01

| | Exercise : 01 |
|---|---|
| | Rev Alpha |
| Turn-in directory : *ex01/* | |
| Files to turn in : `exec.py` | |
| Forbidden functions : `None` | |

Make a program that takes a string as argument, reverses it, swaps its letters case and prints the result.

- If more than one argument is provided, merge them into a single string with each argument separated by a single space character.

- If no argument is provided, do nothing or print an usage.

## Examples

```
$> python3 exec.py 'Hello World!' | cat -e
!DLROw OLLEh$
$>
$> python3 exec.py 'Hello' 'my Friend' | cat -e
DNEIRf YM OLLEh$
$>
$> python3 exec.py
$>
```

# Chapter IV

# Exercise 02

| ![logo] | Exercise : 02 |
|---|---|
| | The Odd, the Even and the Zero |
| Turn-in directory : *ex02/* | |
| Files to turn in : `whois.py` | |
| Forbidden functions : `None` | |

Make a program that takes a number as argument, checks whether it is odd, even or zero, and prints the result.

- If more than one argument is provided or if the argument is not an integer, print an error message.

- If no argument is provided, do nothing or print an usage.

## Examples

```
$> python3 whois.py 12
I'm Even.
$>
$> python3 whois.py 3
I'm Odd.
$>
$> python3 whois.py
$>
$> python3 whois.py 0
I'm Zero.
$>
$> python3 whois.py Hello
AssertionError: argument is not an integer
$>
$> python3 whois.py 12 3
AssertionError: more than one argument is provided
$>
```

> No bonus point to be gained from a complex error management system.
> Keep it simple.

# Chapter V

# Exercise 03

| ![logo] | Exercise : 03 |
| --- | --- |
| Functional file | |
| Turn-in directory : *ex03/* | |
| Files to turn in : `count.py` | |
| Forbidden functions : `None` | |

## Part 1. text_analyzer

Create a function called `text_analyzer` that takes a single string argument and displays the total number of printable characters, and respectively : the number of upper-case characters, lower-case characters, punctuation characters and spaces.

- If `None` or nothing is provided, the user is prompted to provide a string.

- If the argument is not a string, print an error message.

- This function must have a `docstring` explaning its behavior.

Test your function within the `python` console

## Examples

```
$> python3
>>> 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 printable character(s):
- 2 upper letter(s)
- 113 lower letter(s)
- 4 punctuation mark(s)
- 18 space(s)
>>> 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 printable character(s):
- 5 upper letter(s)
- 187 lower letter(s)
- 8 punctuation mark(s)
- 30 space(s)
>>> text_analyzer()
What is the text to analyze?
>> Hello World!
The text contains 12 printable character(s):
- 2 upper letter(s)
- 8 lower letter(s)
- 1 punctuation mark(s)
- 1 space(s)
>>> text_analyzer(42)
AssertionError: argument is not a string
>>> print(text_analyzer.__doc__)

    This function counts the number of upper characters, lower characters,
    punctuation and spaces in a given text.
```

> Python has a lot of very convenient built-in functions and methods.
> Do some research, you do not have to reivent the wheel !

# Part 2.  ___name___==___main___

In the previous part, you wrote a function that can be used in the console or in another file when imported. Without changing this behavior, update your file so it can also be launched as a standalone program.

- If more than one argument is provided to the program, print an error message.

- Otherwise, use the `text_analyzer` function.

## Examples

```
$> python3 count.py 'Hello World!'
The text contains 12 character(s):
- 2 upper letter(s)
- 8 lower letter(s)
- 1 punctuation mark(s)
- 1 space(s)
$> python3
>>> from count import text_analyzer
>>> text_analyzer("Hello World!")
The text contains 12 character(s):
- 2 upper letter(s)
- 8 lower letter(s)
- 1 punctuation mark(s)
- 1 space(s)
```

# Chapter VI

# Exercise 04

| ![logo] | Exercise : 04 |
|---------|---------------|
| | Elementary |
| Turn-in directory : *ex04/* | |
| Files to turn in : `operations.py` | |
| Forbidden functions : `None` | |

Write a program that takes two integers A and B as arguments and prints the result of the following operations:

```
Sum:        A+B
Difference: A-B
Product:    A*B
Quotient:   A/B
Remainder:  A%B
```

- If more or less than two arguments are provided or if one of the arguments is not an integer, print an error message.

- If no argument is provided, do nothing or print an usage.

- If an operation is impossible, print an error message instead of a numerical result.

## Examples

```
$> python3 operations.py 10 3
Sum:        13
Difference: 7
Product:    30
Quotient:   3.3333...
Remainder:  1
$>
$> python3 operations.py 42 10
Sum:        52
Difference: 32
Product:    420
Quotient:   4.2
Remainder:  2
$>
```

```
$> python3 operations.py 1 0
Sum:         1
Difference: 1
Product:     0
Quotient:   ERROR (division by zero)
Remainder: ERROR (modulo by zero)
$>
$> python3 operations.py
Usage: python operations.py <number1> <number2>
Example:
    python operations.py 10 3
$>
$> python3 operations.py 12 10 5
AssertionError: too many arguments
$>
$> python3 operations.py "one" "two"
AssertionError: only integers
$>
```

> No bonus point to be gained from handling decimal point or scientific notation.  Keep it simple.

# Chapter VII

# Exercise 05

| | |
|---|---|
| ![logo] | Exercise : 05 |
| | The right format |
| Turn-in directory : *ex05/* | |
| Files to turn in : `kata00.py, kata01.py, kata02.py, kata03.py, kata04.py` | |
| Forbidden functions : `None` | |

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

Each exercice will provide you with a `kata` variable. This variable can be modified to a certain extent: your program must react accordingly.

## kata00

The `kata` variable is always a tuple and can only be filled with integers.

```
# Put this at the top of your kata00.py file
kata = (19,42,21)
```

Write a program that displays this variable content according to the format shown below:

```
$> python3 kata00.py
The 3 numbers are: 19, 42, 21
$>
```

## kata01

The `kata` variable is always a dictionary and can only be filled with strings.

```
# Put this at the top of your kata01.py file
kata = {
    'Python': 'Guido van Rossum',
    'Ruby': 'Yukihiro Matsumoto',
    'PHP': 'Rasmus Lerdorf',
    }
```

Write a program that displays this variable content according to the format shown below:

```
$> python3 kata01.py
Python was created by Guido van Rossum
Ruby was created by Yukihiro Matsumoto
PHP was created by Rasmus Lerdorf
$>
```

## kata02

The `kata` variable is always a tuple that contains 5 non-negative integers. The first integer contains up to 4 digits, the rest up to 2 digits.

```
# Put this at the top of your kata02.py file
kata = (2019, 9, 25, 3, 30)
```

Write a program that displays this variable content according to the format shown below:

```
$> python3 kata02.py | cat -e
09/25/2019 03:30$
$> python3 kata02.py | wc -c
17
$>
```

## kata03

The `kata` variable is always a string whose length is not higher than 42.

```
# Put this at the top of your kata03.py file
kata = "The right format"
```

Write a program that displays this variable content according to the format shown below:

```
$> python3 kata03.py | cat -e
------------------------The right format%
$> python3 kata03.py | wc -c
42
$>
```

## kata04

The `kata` variable is always a tuple that contains, in the following order:

- 2 non-negative integers containing up to 2 digits

- 1 decimal

- 1 integer

- 1 decimal

```
# Put this at the top of your kata04.py file
kata = (0, 4, 132.42222, 10000, 12345.67)
```

Write a program that displays this variable content according to the format shown below:

```
$> python3 kata04.py
module_00, ex_04 : 132.42, 1.00e+04, 1.23e+04
$> python3 kata04.py | cut -c 10,18
,:
```

# Chapter VIII

# Exercise 06

| | |
|---|---|
|  | Exercise : 06 |
| | A recipe |
| Turn-in directory : *ex06/* | |
| Files to turn in : `recipe.py` | |
| Forbidden functions : `None` | |

## Part 1: Nested Dictionaries

Create a dictionary called `cookbook`. You will use this `cookbook` to store recipes.

A recipe is a **dictionary** that stores (at least) 3 key-value pairs:

- "ingredients": a **list of strings** containing the list of ingredients

- "meal": a **string** representing the type of meal

- "prep_time": a **non-negative integer** representing a preparation time in minutes

In the `cookbook`, the **key** to a recipe is the recipe's name.
Initialize your `cookbook` with 3 recipes:

- The Sandwich's ingredients are *ham*, *bread*, *cheese* and *tomatoes*. It is a *lunch* and it takes 10 minutes of preparation.

- The Cake's ingredients are *flour*, *sugar* and *eggs*. It is a *dessert* and it takes 60 minutes of preparation.

- The Salad's ingredients are *avocado*, *arugula*, *tomatoes* and *spinach*. It is a *lunch* and it takes 15 minutes of preparation.

# Part 2: A Handful of Helpful Functions

Create a series of useful functions to handle your `cookbook`:

1. A function that prints all recipe names.

2. A function that takes a recipe name and prints its details.

3. A function that takes a recipe name and delete it.

4. A function that adds a recipe from user input. You will need a name, a list of ingredients, a meal type and a preparation time.

## Input example

```
>>> Enter a name:
chips
>>> Enter ingredients:
potatoes
oil
salt

>>> Enter a meal type:
lunch
>>> Enter a preparation time:
15
```

# Part 3: A command line executable !

Create a program that uses your `cookbook` and your functions.

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

Your program will continue to prompt the user until the user decides to quit it.

The program cannot crash if a wrong value is entered: you must handle the error and ask for another prompt.

```
$> python3 recipe.py
Welcome to the Python Cookbook !
List of available options:
   1: Add a recipe
   2: Delete a recipe
   3: Print a recipe
   4: Print the cookbook
   5: Quit

Please select an option:
>> 3

Please enter a recipe name to get its details:
```

```
>> cake

Recipe for cake:
   Ingredients list: ['flour', 'sugar', 'eggs']
   To be eaten for dessert.
   Takes 60 minutes of cooking.

Please select an option:
>> Hello

Sorry, this option does not exist.
List of available options:
   1: Add a recipe
   2: Delete a recipe
   3: Print a recipe
   4: Print the cookbook
   5: Quit

Please select an option:
>> 5

Cookbook closed. Goodbye !
$>
```

# Chapter IX

# Exercise 07

| ![logo] | Exercise : 07 |
| --- | --- |
| | Shorter, Faster ... Pythonic ! |
| Turn-in directory : *ex07/* | |
| Files to turn in : `filterwords.py` | |
| Forbidden functions : `filter` | |

Make a program that takes a string S and an integer N as argument and prints the list of words in S that contains more than N non-punctuation characters.

- Words are separated from each other by space characters

- Punctuation symbols must be removed from the printed list: they are neither part of a word nor a separator

- The program must contain at least one **list comprehension** expression.

If the number of argument is different from 2, or if the type of any argument is invalid, the program prints an error message.

## Examples

```
$> python3 filterwords.py 'Hello, my friend' 3
['Hello', 'friend']
$> python3 filterwords.py 'Hello, my friend' 10
[]
$>  python3 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']
$> python3 filterwords.py Hello World
ERROR
$> python3 filterwords.py 3 'Hello, my friend'
ERROR
$> python3 filterwords.py
ERROR
```

# Chapter X

# Exercise 08

| | Exercise : 08 |
|---|---|
| | S.O.S |
| Turn-in directory : *ex08/* | |
| Files to turn in : `sos.py` | |
| Forbidden functions : `None` | |

Make a program that takes a string as argument and encodes it into Morse code.

- The program supports space and alphanumeric characters

- An alphanumeric character is represented by dots `.` and dashes `-`

- A space character is represented by a slash `/`

- Complete morse characters are separated by a single space

If more than one argument is provided, merge them into a single string with each argument separated by a single space character.

If no argument is provided, do nothing or print an usage.

## Examples

```
$> python3 sos.py "SOS"
... --- ...
$> python3 sos.py
$> python3 sos.py "HELLO / WORLD"
ERROR

$> python3 sos.py "96 BOULEVARD" "Bessiere"
----. -.... / -... --- ..- .-.. . ...- .- .-. -.. / -... . ... ... .. . .-. .
```

https://morsecode.world/international/morse2.html

# Chapter XI

# Exercise 09

| ![logo] | Exercise : 09 |
|---|---|
| | Secret number |
| Turn-in directory : *ex09/* | |
| Files to turn in : `guess.py` | |
| Forbidden functions : `None` | |

You have to make a program that will be an interactive guessing game.

It will prompt 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 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.

## Examples

```
$> 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 !

If the secret number is 42, make a reference to Douglas Adams.

```
$> 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?
>> 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!
```

# Chapter XII

# Exercise 10

| ![logo] | Exercise : 10 | |
|---|---|---|
| | Loading bar! | |
| Turn-in directory : *ex10/* | | |
| Files to turn in : `loading.py` | | |
| Forbidden functions : `tqdm or any library for automatic loading bar` | | |

You are about to discover the `yield` operator!

So let's create a function called `ft_progress(lst)`.

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

> **i** We strongly advise you to take a look at the wonderful **tqdm** library, it will come in handy in many situations !

## Examples

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

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

# Contact

You can contact 42AI by email: contact@42ai.fr

Thank you for attending 42AI's Python Bootcamp module00 !

# Acknowledgements

The Python bootcamp is the result of a collective work, for which we would like to thank:

- Maxime Choulika (cmaxime),

- Pierre Peigné (ppeigne, pierre@42ai.fr),

- Matthieu David (mdavid, matthieu@42ai.fr),

- Quentin Feuillade–Montixi (qfeuilla, quentin@42ai.fr)

- Mathieu Perez (maperez, mathieu.perez@42ai.fr)

who supervised the creation, the enhancement of the bootcamp and this present transcription.

- Louis Develle (ldevelle, louis@42ai.fr)

- Augustin Lopez (aulopez)

- Luc Lenotre (llenotre)

- Owen Roberts (oroberts)

- Thomas Flahault (thflahau)

- Amric Trudel (amric@42ai.fr)

- Baptiste Lefeuvre (blefeuvr@student.42.fr)

- Mathilde Boivin (mboivin@student.42.fr)

- Tristan Duquesne (tduquesn@student.42.fr)

for your investment in the creation and development of these modules.

- All prior participants who took a moment to provide their feedbacks, and help us improve these bootcamps !