Python & ML - Módulo 00

Conceptos básicos - Once mandamientos

Resumen: El objetivo del módulo es iniciarse en el lenguaje Python.

Capítulo I

Instrucciones generales

- La versión de Python que se recomienda utilizar es la 3.7, puedes comprobar la versión de Python con el siguiente comando: `python -V`
- La norma: durante esta piscina, se recomienda seguir los estandares PEP 8, aunque no es obligatorio. Puedes instalar pycodestyle que es una herramienta para comprobar tu código Python.
- La función `eval` nunca está permitida.
- Los ejercicios están ordenados del más fácil al más difícil.
- Tus ejercicios van a ser evaluados por otras personas, así que asegúrate de que los nombres de tus variables y funciones sean apropiados y corteses.
- Tu manual es internet.
- Te animamos a crear programas de prueba para tu proyecto, aunque este trabajo **no tendrá que ser presentado y no será calificado**. Te dará la oportunidad de poner a prueba fácilmente tu trabajo y el de tus compañeros/as. Estos tests te serán especialmente útiles durante tu evaluación. De hecho, durante la evaluación, eres libre de utilizar tus pruebas y/o las pruebas del compañero/a al que estás evaluando.

Índice general

I.	**Instrucciones generales**	**1**
II.	**Ejercicio 00**	**3**
III.	**Ejercicio 01**	**8**
IV.	**Ejercicio 02**	**9**
V.	**Ejercicio 03**	**11**
VI.	**Ejercicio 04**	**13**
VII.	**Ejercicio 05**	**15**
VIII.	**Ejercicio 06**	**18**
IX.	**Ejercicio 07**	**21**
X.	**Ejercicio 08**	**22**
XI.	**Ejercicio 09**	**24**
XII.	**Ejercicio 10**	**26**

Capítulo II

Ejercicio 00

	Ejercicio : 00
	$PATH
Directorio de entrega : *ex*00/	
Archivos a entregar : `answers.txt, requirements.txt`	
Funciones prohibidas : `None`	

Lo primero que tienes que hacer es instalar Python.

La mayoría de los sistemas modernos basados en Unix tienen un intérprete de `python` instalado por defecto, pero su versión puede ser inferior/superior a la utilizada para estos módulos. También es posible que el comando `python` por defecto utilice una versión 2.x (por razones heredadas). Obviamente, esto es muy confuso para los nuevos desarrolladores.

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

Para lidiar con esos problemas de versión, usaremos `conda`. Este programa te permite gestionar tus paquetes Python y diferentes entornos de trabajo.

Nota: el verdadero requisito es utilizar una versión de Python 3.7.X. Eres libre de utilizar un programa o utilidades diferentes para lograr este objetivo. Por tu cuenta y riesgo.

Instalación manual de conda

Ve a la siguiente sección para una instalación automatizada.

Recomendamos la siguiente ruta para tu carpeta `conda`.

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

1. Descarga e instala 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. Configuración inicial de 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. ¡Crea un entorno para 42AI!

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

4. Comprueba tu entorno Python 42AI

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

¡Ayuda!

- **¡He perdido mi carpeta miniconda3!** Repite los pasos 1 y 3.
- **¡He perdido mi directorio personal!** Repite el paso 2.

Instalación automatizada de Conda

Copia el siguiente script en un archivo, ejecútalo y sigue las instrucciones. Descarga miniconda, lo instala en una subcarpeta `/goinfre` y crea un entorno `python` en `conda`.

```
#!/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
```

¡No olvides comprobar tu entorno Python 42AI!

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

(Finalmente) Empezemos!

Ahora que tu configuración está lista para funcionar, hay algunas preguntas que necesitan ser resueltas usando `python`, `pip` o `conda`. Guarda tus respuestas en un archivo `answers.txt` (una respuesta por línea y por pregunta), y revísalas con tus compañeros/as.

Encuentra los comandos para:

- Mostrar una lista de los paquetes instalados y sus versiones.
- Mostrar los metadatos del paquete `numpy`.
- Elimina el paquete `numpy`.
- (Re)instala el paquete `numpy`.
- Congela tus paquetes `python` y sus versiones en un archivo `requirements.txt` que tienes que entregar.

Capítulo III

Ejercicio 01

	Ejercicio : 01
	Rev Alpha
Directorio de entrega : $ex01/$	
Archivos a entregar : `exec.py`	
Funciones prohibidas : `None`	

Haz un programa que tome un string como argumento, lo invierta, cambie sus letras mayúsculas a minúsculas y viceversa, e imprima el resultado.

- Si se proporciona más de un argumento, combínalos en un solo string con cada argumento separado por un carácter de espacio.
- Si no se proporciona ningún argumento, el programa no hará nada o imprimirá un mensaje de utilización del comando.

Ejemplos

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

Capítulo IV

Ejercicio 02

	Ejercicio : 02
	The Odd, the Even and the Zero
Directorio de entrega : $ex02/$	
Archivos a entregar : `whois.py`	
Funciones prohibidas : `None`	

Haz un programa que tome un número como argumento, compruebe si es par, impar o cero e imprima el resultado.

- Si se proporciona más de un argumento o si el argumento no es un entero, imprime un mensaje de error.
- Si no se proporciona ningún argumento, el programa no hará nada o imprimirá un mensaje de utilización del comando.

Ejemplos

```
$> 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 are provided
$>
```

No se gana ningún punto extra con un sistema complejo de gestión de errores. Hazlo sencillo.

Capítulo V

Ejercicio 03

	Ejercicio : 03
	Fichero funcional
Directorio de entrega : $ex03/$	
Archivos a entregar : `count.py`	
Funciones prohibidas : `None`	

Parte 1. text_analyzer

Crea una función llamada `text_analyzer` que tome un único argumento de string y cuente la cantidad de caracteres en mayúsculas, minúsculas, signos de puntuación y espacios.

- Si es `None` o no se proporciona nada, se pide al usuario que introduzca un string.
- Si el argumento no es un string, imprimirá un mensaje de error.
- Esta función debe tener un `docstring` que explique su comportamiento.

Prueba tu función con la consola de `python`

Ejemplos

```
$> 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 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 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 8 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.
```

Parte 2. ___name___==___main___

En el apartado anterior, escribiste una función que se puede utilizar en la consola o en otro archivo cuando se importa. Sin que cambie su comportamiento, actualiza tu archivo para que también pueda ejecutarse como un programa independiente.

- Si se proporciona más de un argumento al programa, deberá imprimir un mensaje de error.
- En caso contrario, utilizará la función `text_analyzer`.

Ejemplos

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

Capítulo VI

Ejercicio 04

	Ejercicio : 04
	Básico
Directorio de entrega : $ex04/$	
Archivos a entregar : `operations.py`	
Funciones prohibidas : `None`	

Escribe un programa que tome dos enteros A y B como argumentos e imprima el resultado de las siguientes operaciones:

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

- Si se proporcionan más o menos de dos argumentos o si alguno de los argumentos no es un número entero, imprime un mensaje de error.
- Si no se proporciona ningún argumento, el programa no hará nada o imprimirá un mensaje de utilización del comando.
- Si una operación es imposible, imprime un mensaje de error en lugar de un resultado numérico.

Ejemplos

```
$> 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 se gana ningún punto extra por la coma decimal o la notación científica. Hazlo sencillo.

Capítulo VII

Ejercicio 05

	Ejercicio : 05
El formato adecuado	
Directorio de entrega : $ex05/$	
Archivos a entregar : `kata00.py, kata01.py, kata02.py, kata03.py, kata04.py`	
Funciones prohibidas : `None`	

Familiaricémonos con el útil concepto de **formateo de strings** a través de una serie de katas.

Cada ejercicio te proporcionará una variable de `kata`. Esta variable puede modificarse hasta cierto punto: tu programa debe reaccionar en consecuencia.

kata00

La variable `kata` es siempre una tupla y solo puede rellenarse con números enteros.

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

Escribe un programa que muestre el contenido de esta variable según el formato que se muestra a continuación:

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

kata01

La variable `kata` es siempre un diccionario y solo puede rellenarse con strings.

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

Escribe un programa que muestre el contenido de esta variable según el formato que se muestra a continuación:

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

kata02

La variable `kata` es siempre una tupla que contiene 5 enteros no negativos. El primer número entero contiene hasta 4 dígitos, el resto hasta 2 dígitos.

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

Escribe un programa que muestre el contenido de esta variable según el formato que se muestra a continuación:

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

kata03

La variable `kata` es siempre un string cuya longitud no es superior a 42.

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

Escribe un programa que muestre el contenido de esta variable según el formato que se muestra a continuación:

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

kata04

La variable `kata` es siempre una tupla que contiene, en el siguiente orden:

- 2 números enteros no negativos de hasta 2 cifras
- 1 decimal
- 1 entero
- 1 decimal

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

Escribe un programa que muestre el contenido de esta variable según el formato que se muestra a continuación:

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

Capítulo VIII

Ejercicio 06

	Ejercicio : 06
Una receta	
Directorio de entrega : $ex06/$	
Archivos a entregar : `recipe.py`	
Funciones prohibidas : `None`	

Parte 1: Diccionarios anidados

Crea un diccionario llamado `cookbook`. Utilizarás este `cookbook` para guardar recetas.

Una receta es un **dictionary** que almacena (al menos) 3 parejas clave-valor:

- "ingredients": una **lista de strings** que representa la lista de ingredientes
- "meal": un **string** que representa el tipo de comida
- "prep_time": Un **entero no negativo** que representa un tiempo en minutos

En el `cookbook`, la **key** de una receta es el nombre de la receta.
Inicializa tu recetario `cookbook` con 3 recetas:

- Los ingredientes del bocadillo son *jamón*, *pan*, *queso* y *tomate*. Es un *almuerzo* y requiere 10 minutos de preparación.
- Los ingredientes de la tarta son *harina*, *azúcar* y *huevos*. Es un *postre* y requiere de 60 minutos de preparación.
- Los ingredientes de la ensalada son *aguacate*, *rúcula*, *tomates* y *espinacas*. Es un *almuerzo* y requiere de 15 minutos de preparación.

Parte 2: Una serie de funciones útiles

Crea una serie de funciones útiles para manejar tu `cookbook`:

1. Una función que imprima todos los nombres de las recetas.

2. Una función que tome un nombre de receta e imprima sus detalles.

3. Una función que tome un nombre de receta y lo borre.

4. Una función que añada una receta a partir de la entrada del usuario. Necesitarás un nombre, una lista de ingredientes, un tipo de comida y un tiempo de preparación.

Ejemplo de entrada

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

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

Parte 3: ¡Un ejecutable de línea de comandos!

Crea un programa que utilice tu `cookbook` y tus funciones.

El programa pedirá al usuario que elija entre imprimir el contenido del recetario, imprimir una receta, añadir una receta, borrar una receta o salir del recetario.

Tu programa continuará preguntando hasta que el usuario decida abandonarlo. El programa no puede bloquearse si se introduce un valor incorrecto: debes gestionar el error y pedir otra entrada.

```
$> python3 recipe.py
Welcome to the Python Cookbook !
List of available option:
   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 option:
   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 !
$>
```

Capítulo IX

Ejercicio 07

	Ejercicio : 07
	Más corto, más rápido, más pythonico
Directorio de entrega :	$ex07/$
Archivos a entregar :	`filterwords.py`
Funciones prohibidas :	`filter`

Haz un programa que tome como argumento un string S y un número entero N e imprima la lista de palabras de S que contenga más de N caracteres que no sean de puntuación.

- Las palabras están separadas entre sí por espacios
- Los signos de puntuación deben eliminarse de la lista impresa: no forman parte de una palabra ni son separadores
- El programa debe contener al menos una expresión de **comprensión de lista**.

Si el número de argumentos es distinto de 2, o si el tipo de algún argumento es incorrecto, el programa imprime un mensaje de error.

Ejemplos

```
$> 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
```

Capítulo X

Ejercicio 08

	Ejercicio : 08
	S.O.S.
Directorio de entrega : $ex08/$	
Archivos a entregar : `sos.py`	
Funciones prohibidas : `None`	

Haz un programa que tome un string como argumento y lo codifique en código Morse.

- El programa admite espacios y caracteres alfanuméricos
- Un carácter alfanumérico se representa mediante puntos `.` y guiones `-`:
- Un carácter de espacio se representa mediante una barra `/`
- Los caracteres morse completos se separan con un solo espacio

Si se proporciona más de un argumento, combínalos en un solo string con cada argumento separado por un carácter de espacio.

Si no se proporciona ningún argumento, el programa no hará nada o imprimirá un mensaje de utilización del comando.

Ejemplos

```
$> 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

Capítulo XI

Ejercicio 09

	Ejercicio : 09
	Número secreto
Directorio de entrega : $ex09/$	
Archivos a entregar : `guess.py`	
Funciones prohibidas : `None`	

Tienes que hacer un programa que sea un juego de adivinanzas interactivo. Pedirá al usuario que adivine un número entre 1 y 99. El programa indicará al usuario si su entrada es demasiado alta o demasiado baja. El juego termina cuando el usuario averigua el número secreto o teclea `exit`. Importarás el módulo `random` con la función `randint` para obtener un número aleatorio. Tienes que contar el número de intentos e imprimir ese número cuando el usuario gane.

Ejemplos

```
$> 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!
```

Si el usuario descubre el número secreto en el primer intento, díselo. Si el número secreto es 42, haz una referencia a 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!
```

Otro ejemplo:

```
$> 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!
```

Capítulo XII

Ejercicio 10

	Ejercicio : 10
¡Barra de carga!	
Directorio de entrega : $ex10/$	
Archivos a entregar : `loading.py`	
Funciones prohibidas : `tqdm or any library for automatic loading bar`	

¡Estás a punto de descubrir el operador `yield`!
Así que vamos a crear una función llamada `ft_progress(lst)`.
La función mostrará el progreso de un bucle `for`.

Ejemplos

```
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
```

Te aconsejamos que eches un vistazo a la maravillosa biblioteca de tqdm, te resultará muy útil en muchas situaciones

Reconocimientos

Los módulos Python & ML son el resultado de un trabajo colectivo, al que queremos dar las gracias:

- Maxime Choulika (cmaxime),
- Pierre Peigné (ppeigne, pierre@42ai.fr),
- Matthieu David (mdavid, matthieu@42ai.fr),
- Quentin Feuillade–Montixi (qfeuilla, quentin@42ai.fr)

que supervisó la creación, la mejora y esta transcripción.

- 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)

por su inversión en la creación y desarrollo de estos módulos.

- Barthélémy Leveque (bleveque@student.42.fr)
- Remy Oster (roster@student.42.fr)
- Quentin Bragard (qbragard@student.42.fr)
- Marie Dufourq (madufour@student.42.fr)
- Adrien Vardon (advardon@student.42.fr)

que realizaron las pruebas beta de la primera versión de los módulos de aprendizaje automático.