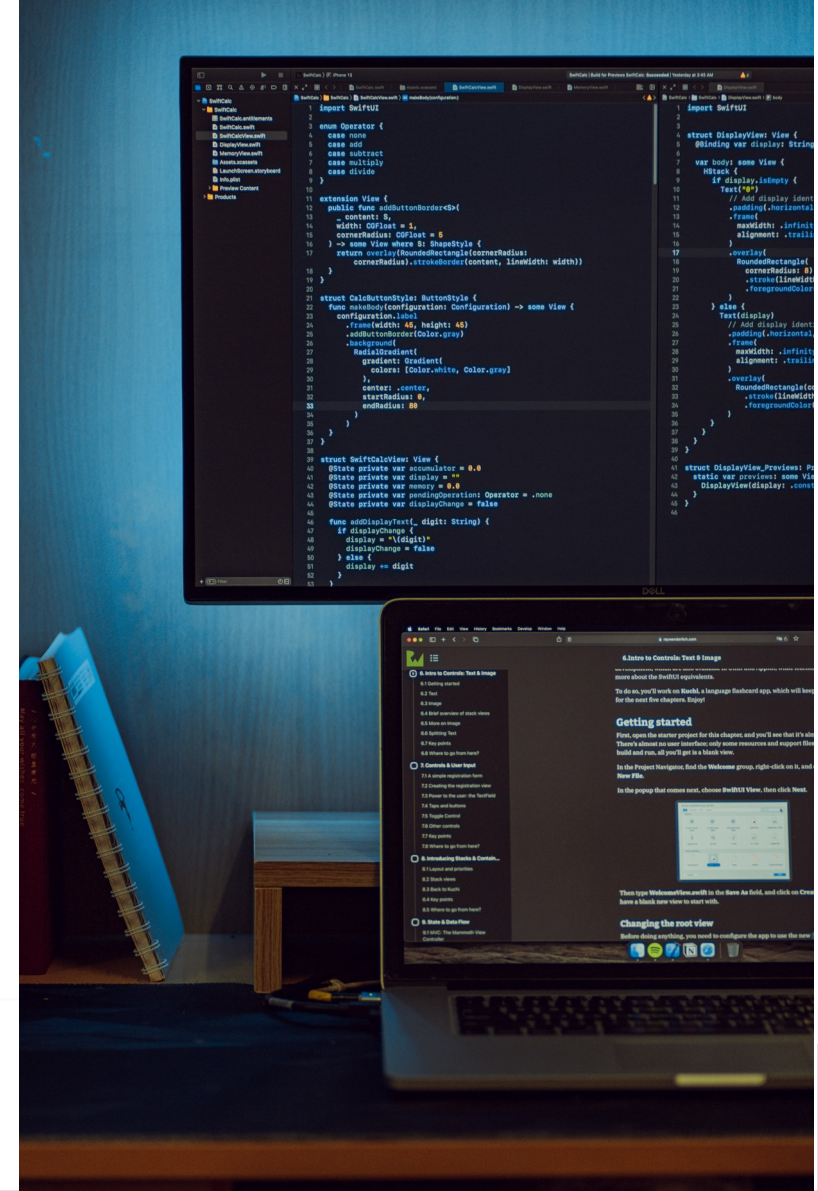
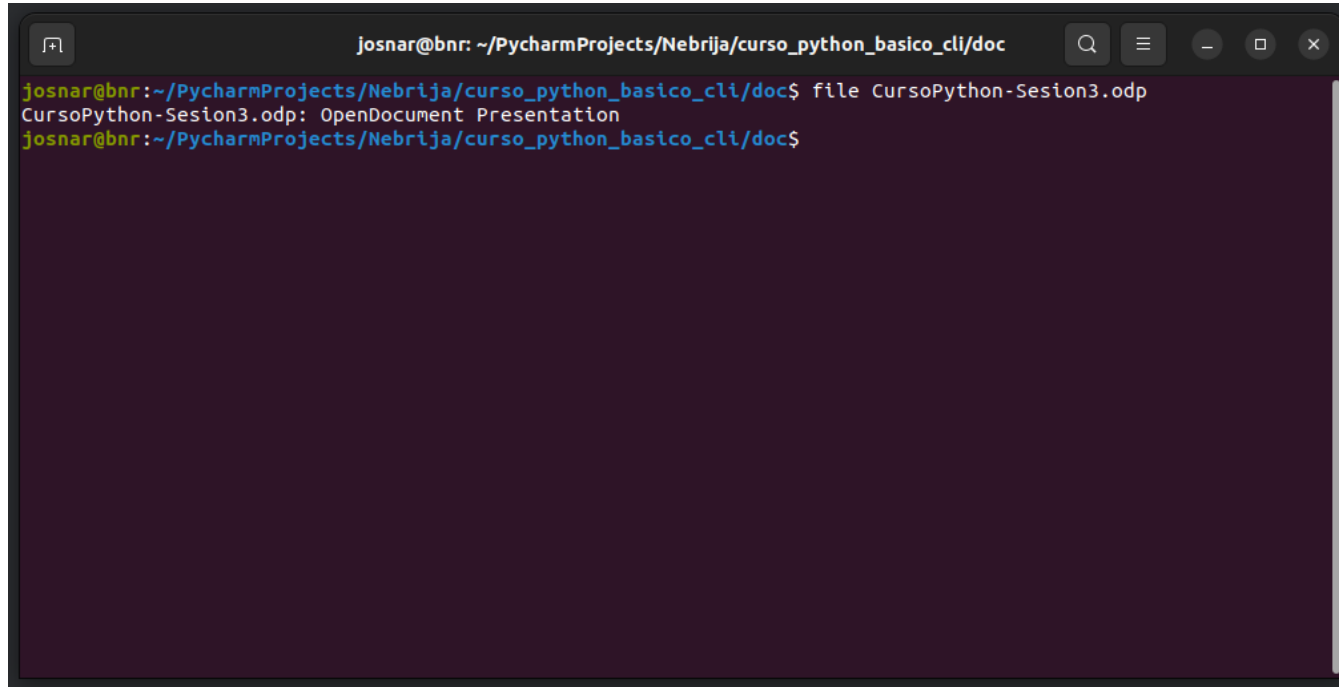


## Command Line Interface



# ¿Qué es una CLI?

Es una interfaz que permite a un usuario interactuar con un intérprete de texto (las típicas consolas en Linux).

A terminal window with a dark background and light-colored text. The window title bar shows the user 'josnar@bnr' and the current directory '~/PycharmProjects/Nebrija/curso\_python\_basico\_cli/doc'. The terminal content shows a command 'file CursoPython-Sesion3.odp' being executed, followed by the output 'CursoPython-Sesion3.odp: OpenDocument Presentation'. The prompt 'josnar@bnr:~/PycharmProjects/Nebrija/curso\_python\_basico\_cli/doc\$' is visible at the end of the line.

```
josnar@bnr: ~/PycharmProjects/Nebrija/curso_python_basico_cli/doc
josnar@bnr:~/PycharmProjects/Nebrija/curso_python_basico_cli/doc$ file CursoPython-Sesion3.odp
CursoPython-Sesion3.odp: OpenDocument Presentation
josnar@bnr:~/PycharmProjects/Nebrija/curso_python_basico_cli/doc$
```



# ¿De qué se compone una CLI?

- Comando → Por ejemplo el comando *git*
- Argumentos → De 0 a un número indeterminado de argumentos
- Documentación → Un comando debe tener una documentación asociada

```
Josnar@bnr: ~/PycharmProjects/Nebrija/curso_python_basico_
josnar@bnr:~/PycharmProjects/Nebrija/curso_python_basico_cli/doc$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   CursoPython-Sesion2.odp

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted:    CursoPython-Sesion2.odp

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  ../.idea/
  ../lock.CursoPython-Sesion3.odp#
  CursoPython-Sesion3.odp
```

```
Josnar@bnr: ~/PycharmProjects/Nebrija/curso_python_basico_
GIT-STATUS(1)                                Git Manual

NAME
    git-status - Show the working tree status

SYNOPSIS
    git status [<options>...] [--] [<pathspec>...]

DESCRIPTION
    Displays paths that have differences between the index file and the cu
    have differences between the working tree and the index file, and path
    are not tracked by Git (and are not ignored by gitignore(5)). The firs
    by running git commit; the second and third are what you could commit
    running git commit.

OPTIONS
    -s, --short
        Give the output in the short-format.
```



# ¿De qué se compone una CLI?

- Comando → Por ejemplo el comando *git*
- Argumentos → De 0 a un número indeterminado de argumentos
- Documentación → Un comando debe tener una documentación asociada

```
Josnar@bnr: ~/PycharmProjects/Nebrija/curso_python_basico_
josnar@bnr:~/PycharmProjects/Nebrija/curso_python_basico_cli/doc$ git status
On branch main
Your branch is up to date with 'origin/main'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    new file:   CursoPython-Sesion2.odp

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    deleted:    CursoPython-Sesion2.odp

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  ../.idea/
  ../lock.CursoPython-Sesion3.odp#
  CursoPython-Sesion3.odp
```

```
Josnar@bnr: ~/PycharmProjects/Nebrija/curso_python_basico_
GIT-STATUS(1)                                Git Manual

NAME
    git-status - Show the working tree status

SYNOPSIS
    git status [<options>...] [--] [<pathspec>...]

DESCRIPTION
    Displays paths that have differences between the index file and the cu
    have differences between the working tree and the index file, and path
    are not tracked by Git (and are not ignored by gitignore(5)). The first
    by running git commit; the second and third are what you could commit
    running git commit.

OPTIONS
    -s, --short
        Give the output in the short-format.
```



# CLI en Python → `sys.argv`

**sys** es un módulo de python que gestiona un montón de variables que utiliza el intérprete. No es necesario instalar ningún paquete y está built-in.

¿Qué es **sys.argv**? **argv** es una variable del módulo **sys** donde se almacenan los **argumentos** del comando. Tal cual: **Una lista**

```
import sys

if __name__ == "__main__":
    print(sys.argv)
```

```
$ python3 print_args.py arg1 arg2 blabla 1
['print_args.py', 'arg1', 'arg2', 'blabla', '1']
$
```



# Nuestra primera CLI

Vamos a construir una CLI que espera los siguientes argumentos:

- 1) Subcomando. El primer argumento indicará al programa qué acción queremos ejecutar. De momento, nuestra primer acción será:  
**obtener\_pilotos**
- 2) Este subcomando **obtener\_pilotos** requiere a su vez un argumento que será el año. Por ejemplo: **2022**

De esta forma nuestro programa tendrá una primera CLI así:

```
base_de_datos_f1 obtener_pilotos 2022
```



# ¿Errores en la CLI?

¿Qué pasa si no metemos exactamente lo que pedimos en la anterior diapositiva?

```
josnar@bnr:~/PycharmProjects/Nebrija/curso_python_basico_cli/sys_argv$ python3 base_datos_f1_cli_1.py obtener_pilotos 2022
El subcomando recibido es: base_datos_f1_cli_1.py
El primer argumento recibido es: obtener_pilotos
josnar@bnr:~/PycharmProjects/Nebrija/curso_python_basico_cli/sys_argv$ python3 base_datos_f1_cli_1.py
El subcomando recibido es: base_datos_f1_cli_1.py
Traceback (most recent call last):
  File "/home/josnar/PycharmProjects/Nebrija/curso_python_basico_cli/sys_argv/base_datos_f1_cli_1.py", line 6, in <module>
    primer_argumento = sys.argv[1]
IndexError: list index out of range
```

Estamos accediendo de forma explícita a una posición de la lista que no existe (porque no le hemos pasado ese argumento) → **IndexError**



# ¿Tipos en la CLI?

Todo lo que entra en `sys.argv` es un string, de forma que manualmente tendremos que transformar ese string en el tipo que nos interesa.

Lo mismo, obviamente, si el argumento que esperamos es una lista

```
anyo_int = int(sys.argv[1])
```

```
lista_elementos_int_pero_de_verdad = [int(arg) for arg in sys.argv[2].split(",")]
```





# DRY

Para evitar tener que controlar de forma explícita que los argumentos se han pasado correctamente o tener que ajustar el tipo de cada argumento, de nuevo explícitamente, disponemos de unos módulos ya programados que realizan estas funciones, permitiéndonos generar una interfaz de línea de comandos completa.

- Con control de errores automático
- Con tipos explícitos en cada argumento
- Que ofrece ayuda y simplifica la generación de documentación



# Argparse

Es un módulo que nos facilita la creación de CLIs.

Incluido en la Python standard library, no necesita ser instalado.

Hay múltiples proyectos con objetivos similares.

**<https://docs.python.org/3/library/argparse.html>**



# ArgumentParser

El parseador se construye alrededor de una instancia de **argparse.ArgumentParser()**

1)

```
parser = argparse.ArgumentParser(  
    prog="Programa de prueba",  
    description="Esto es una CLI de ejemplo",  
)
```

2)

```
parser.add_argument("subcomando")  
parser.add_argument("año")
```

3)

```
args = parser.parse_args()
```



# Args posicionales / Args opcionales

Hasta ahora hemos visto en los ejemplos argumentos posiciones, que son aquellos argumentos que van colocados siempre en la misma posición y no tienen ninguna etiqueta identificativa.

Los argumentos opcionales se componen de dos partes: la propia opción y su valor. Es decir, es necesario indicar la opción en el comando.

```
parser = argparse.ArgumentParser(  
    prog="Optional vs Positional",  
    description="Esto es una CLI de ejemplo",  
)  
  
parser.add_argument("subcomando")  
parser.add_argument("--verbosity", help="Indica el nivel de verbosidad")  
args = parser.parse_args()
```



# Tipo de los argumentos

Ya vimos que todo lo que se pasa por línea de comandos es un string, pero podemos indicar a nuestra CLI cuál es el tipo que debería tener cada argumento.

```
parser = argparse.ArgumentParser(  
    prog="Programa de prueba",  
    description="Esto es una CLI de ejemplo",  
)  
  
parser.add_argument("subcomando")  
parser.add_argument("año", type=int)  
  
args = parser.parse_args()  
print(type(args.año))
```

<https://docs.python.org/3/library/argparse.html#type>



# Otras características

- **Listas?** → `add_argument('-a', nargs="+", type=int)`
- Argumentos **obligatorios** no posicionales →  
`add_argument('--obligatorio', type=int, required=True)`
- **Valor por defecto** → `add_argument('--por_defecto', type=int, default=1)`
- Y un montón de otras características integradas en el módulo

