

UNIDAD III

Fundamentos de Python



Lenguaje de programación Python

Algunas características importantes:

- Es un lenguaje **interpretado**. El intérprete puede usarse para leer scripts en texto así como interactivamente.
- Es un lenguaje **multiplataforma** (Linux, Mac, Windows, etc..).
- Es **modular**: los códigos desarrollados se pueden reutilizar en otros programas de Python.
- Tiene una **gran colección de módulos** (librerías) estándar disponibles.
- Es también una **calculadora** de escritorio práctica.
- Es un lenguaje de muy **alto nivel**, siguiendo la **orientación a objetos**.

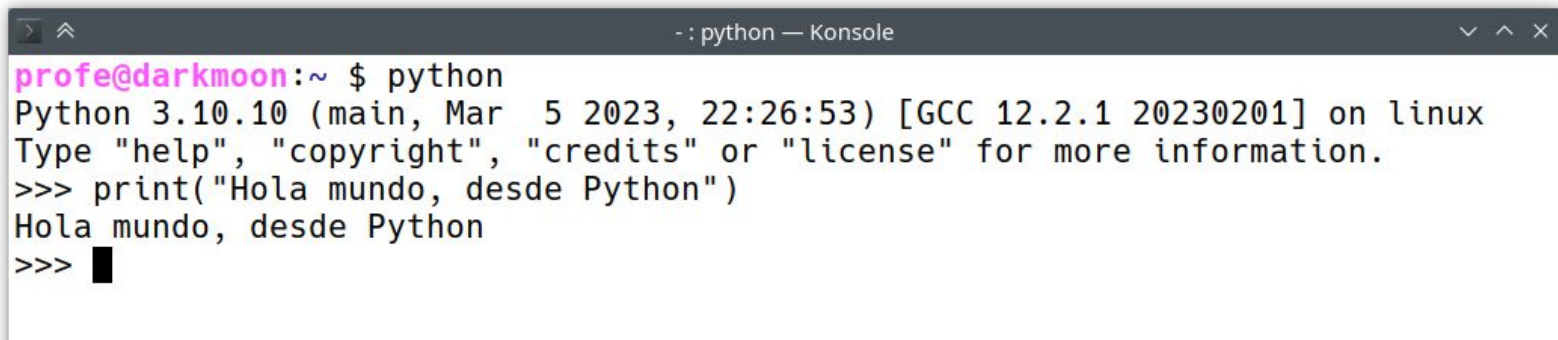
Intérprete

A diferencia de Bash que es parte del sistema, Python es un software que debe ser instalado para funcionar.

- La instalación de Python puede ser **parte del sistema operativo** o puede instalarse de forma **aislada en un directorio** (ya sea manualmente o como parte de un software), incluyendo todos sus módulos.
- Los paquetes/módulos/librerías de una instalación de Python se pueden administrar con el software **pip**.
- Existe una amplia variedad de posibles instalaciones y **Entornos de Desarrollo Integrados (IDEs)** disponibles para programar en Python.
- Actualmente el estándar es Python 3, aunque su versión 2 (2.6-2.7) sigue siendo utilizada en muchos sistemas y desarrollos.

Intérprete (modo interactivo)

En su forma más simple, el intérprete se puede iniciar desde la terminal:

A screenshot of a terminal window titled ': python — Konsole'. The prompt is 'profe@darkmoon:~ \$'. The user has entered 'python', which has started the Python 3.10.10 interpreter. The interpreter shows the version and build information: 'Python 3.10.10 (main, Mar 5 2023, 22:26:53) [GCC 12.2.1 20230201] on linux'. It then prompts the user to type 'help', 'copyright', 'credits', or 'license' for more information. The user has entered '>>> print("Hola mundo, desde Python")', and the interpreter has printed 'Hola mundo, desde Python'. The prompt '>>>' is followed by a black cursor block.

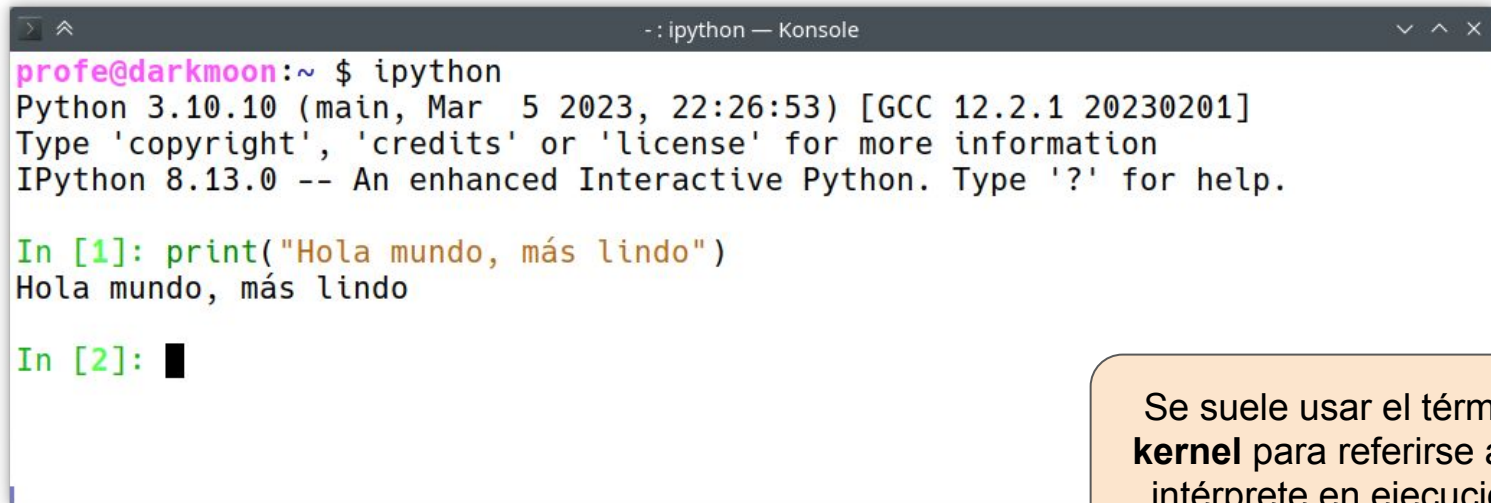
```
profe@darkmoon:~ $ python
Python 3.10.10 (main, Mar 5 2023, 22:26:53) [GCC 12.2.1 20230201] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hola mundo, desde Python")
Hola mundo, desde Python
>>> █
```

El intérprete se comporta muy similar a Bash:

- reconoce los saltos de línea como indicación de término de una instrucción,
- intenta autocompletar cuando se pulsa doble TAB, y
- tiene variables (y objetos) predefinidos al iniciarse.

Intérprete interactivo ++

Una versión **visualmente** más amigable del intérprete de Python es **iPython**. Este software se puede instalar a través de pip.



```
profe@darkmoon:~ $ ipython
Python 3.10.10 (main, Mar  5 2023, 22:26:53) [GCC 12.2.1 20230201]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.13.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: print("Hola mundo, más lindo")
Hola mundo, más lindo

In [2]: █
```

Se suele usar el término **kernel** para referirse a un intérprete en ejecución.

Scripts en Python

El intérprete puede ejecutar directamente archivos de texto con código en Python:

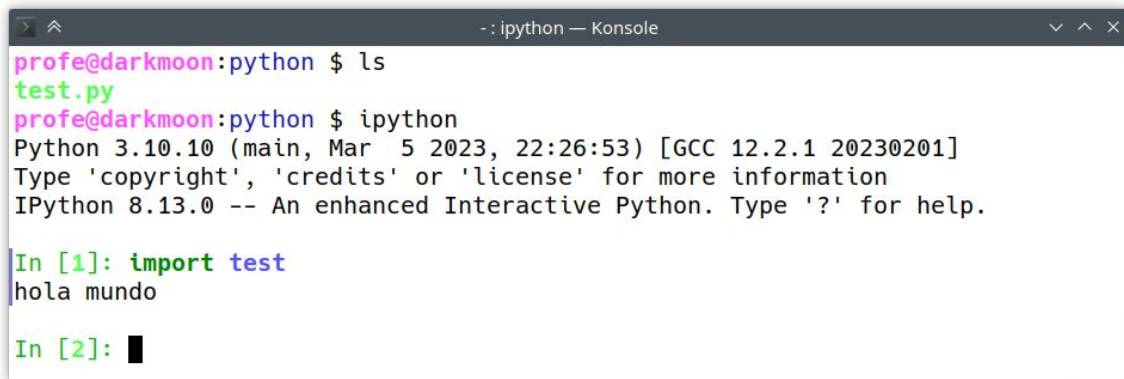
```
profe@darkmoon:python $ echo "print('hola mundo')" > test.py
profe@darkmoon:python $ python test.py
hola mundo
profe@darkmoon:python $
```

Estos archivos también pueden ser ejecutados directamente si se indica el intérprete en su primera línea con `#!/usr/bin/env python`:

```
profe@darkmoon:python $ sed -i 's/^/#!/usr/bin/env python\n/' test.py
profe@darkmoon:python $ chmod +x test.py
profe@darkmoon:python $ ./test.py
hola mundo
profe@darkmoon:python $
```

Scripts en Python

- Los scripts de python tienen la **ventaja** de ser directamente **reutilizables** por otros scripts, de la misma manera en que son cargados los módulos estándar del lenguaje: con la instrucción **import**.

A screenshot of a terminal window titled "- : ipython — Konsole". The terminal shows a user named 'profe' at a machine named 'darkmoon'. The user runs 'python \$ ls', which outputs 'test.py'. Then, the user runs 'python \$ ipython', which shows the IPython version (8.13.0) and the Python version (3.10.10). The user then enters 'In [1]: import test' and the output is 'hola mundo'. The prompt 'In [2]:' is shown with a cursor.

```
profe@darkmoon:python $ ls
test.py
profe@darkmoon:python $ ipython
Python 3.10.10 (main, Mar  5 2023, 22:26:53) [GCC 12.2.1 20230201]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.13.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import test
hola mundo

In [2]:
```

- Estos pueden ser desarrollados en cualquier editor de texto (vim, emacs, nano, etc.)

Entornos de Desarrollo Integrado (IDEs)

Son programas que habilitan herramientas gráficas especializadas en el desarrollo de software (Python), incluyendo: edición de software, ejecución de intérprete, debugging, organización de proyectos, etc.

Algunos ejemplos gratuitos multiplataforma:

- **PyCharm**

<https://www.jetbrains.com/pycharm/>



- **PyDev (Eclipse)**

<https://www.eclipse.org/pydev/>



- **IDLE**

Se incluye con Python



- **VScode**

<https://code.visualstudio.com>



- **Spyder**



SPYDER

The Scientific Python Development Environment

<https://www.spyder-ide.org/>

Otras herramientas para desarrollo de Python

- **Jupyter** (notebook y lab)

<https://jupyter.org/>



Utiliza una rica interfaz web para la escritura y ejecución de código, utilizando su propia estructura de ejecución y tipos de archivos. Está enfocada hacia el análisis de datos.

También se puede utilizar online como parte de las herramientas de Google:

<https://colab.research.google.com/>

- **Anaconda**

<https://www.anaconda.com/>



Es una herramienta para administrar instalaciones autocontenidas de Python, que incluye colecciones de módulos ya instalados y entornos de desarrollo de Jupyter. Está enfocada hacia el análisis de datos.

Para el curso...

- Las **tareas** y los **programas avanzados** los realizaremos en **formato script**, de modo que puedan ser leídos tanto por el intérprete como por cualquier interfaz de desarrollo.
La IDE (o editor de texto) a utilizar queda a su elección.
- Ocasionalmente **también utilizaremos Jupyter Notebooks** para desarrollar códigos sin enfoque en reutilización, pruebas, ejemplos u otros donde convenga tener una mejor visualización.
- Regularmente utilizaremos una consola con python ejecutándose de forma **constante** para probar en tiempo real nuestras ideas en desarrollo, explorar los objetos, sus métodos y consultar la documentación.

P: Verificar que tenga un intérprete de Python funcionando en su computadora y alguna herramienta de desarrollo disponible.

Verificar el funcionamiento de jupyter de forma local, y en Google Colab.

– Actividad 1 –

A partir de acá, realizaremos actividades de programación en la clase con un intérprete en Python para profundizar los conceptos y estudiar los fundamentos del lenguaje.

- Explorar en Python las operaciones matemáticas que puede calcular el intérprete.
- Verificar las variables tanto **numéricas** como **strings** que se pueden crear en Python.
- Verificar qué operaciones se pueden hacer con ellas.

La clave para dominar Python:

¿Qué son los objetos?

Analicemos la siguiente declaración en Python:

```
>>> var = 6.8
```

Para crear esta variable, el intérprete debe:

- Definir el **tipo de dato** que almacenará la variable (entero, flotante, string, etc..). Esto define cómo se comportará la variable dentro del programa.
- Reservarle un lugar en la memoria y almacenar la información en él. Esto crea una **instancia** del tipo de dato correspondiente.
- Almacenar el nombre de la variable entre los nombres disponibles durante la ejecución. Esto crea una **referencia** a la instancia.

La clave para dominar Python:

¿Qué son los objetos?

Los lenguajes con orientación a objetos generalizan el concepto “tipo de dato”, permitiendo la creación de estructuras complejas nuevas llamadas **clases**.

Las clases, así como los tipos de datos, fijan cómo se comportarán sus instancias dentro del programa. Para ello, **para cada clase puede contener:**

- **Atributos** (componentes)
- **Métodos** (funciones)
- Definición de uso de **operadores** (e.g: +, -, *, [], etc..)

Un **objeto** es una **instancia** de una **clase** definida.

Objetos: uso en Python

- **En Python todo es un objeto** (incluyendo los tipos de datos básicos).
- Todos los objetos tienen asignado un tipo, el cual se puede consultar con la función **type()**
- Con el **punto .** se puede acceder a los atributos o métodos disponibles de cualquier objeto.
- Con **help()** se puede consultar a la documentación de cualquier objeto definido.
- Con **dir()** se obtiene una lista con todas las referencias definidas en el objeto (atributos y métodos). En el modo interactivo, el punto seguido de doble TAB también entrega esta información.

Actividad 2: explorar y utilizar los métodos/atributos del objeto **string**.

Actividad 3

Explorar los siguientes tipos de datos compuestos, el uso de sus objetos y sus métodos en Python:

- Listas (incluyendo la instrucción `del`)
- Tuplas
- Sets (conjuntos)
- Diccionarios

¿Cómo leemos el stdin en Python?

Actividad 4

Revisar herramientas de control de flujo disponibles en Python. Estudiar cómo funcionan las instrucciones anidadas mediante indentación.

- Condicionales: `if`, `elif`, `else`
- Loops: `while`
- Loops: `for`
- La función generadora `range()`
- Las sentencias `break` y `continue`
- Funciones: `def`
- Visibilidad y alcance de variables en las instrucciones anteriores

Problemas de práctica

1. **Calculadora Básica:** Escribe un programa que pida al usuario dos números y luego muestre el resultado de la suma, resta, multiplicación y división de esos dos números. Recuerda manejar el caso en que el usuario intente dividir por cero.
2. **Lista de Compras:** Crea un diccionario para representar una lista de compras. Las llaves del diccionario serán los nombres de los productos y los valores serán las cantidades necesarias. El programa debe permitir al usuario agregar, eliminar productos y modificar cantidades, así como mostrar la lista completa.
3. **Eliminación de Duplicados:** Escribe una función que tome una lista y devuelva una nueva lista que contenga todos los elementos de la lista original, pero sin duplicados. Haz esto utilizando un conjunto (set). Por ejemplo, si la lista original es [1, 2, 2, 3, 3, 3, 4, 4, 4, 4], la nueva lista debería ser [1, 2, 3, 4].
4. **Contador de Palabras:** Escribe una función que cuente la cantidad de veces que cada palabra aparece en una cadena de texto. La función debe devolver un diccionario donde las llaves son las palabras y los valores son las cuentas. Por ejemplo, para la cadena "hola mundo hola", la función debería devolver {"hola": 2, "mundo": 1}.
5. **Fibonacci (el regreso):** Escribe una función que tome un número entero n y devuelva el n -ésimo número de Fibonacci.

Actividad 5

Profundizar en clases herramientas de I/O de Python: stdout, archivos y parámetros de entrada.

- Formateo de cadenas literales y con `.format()`.
- Parámetros de entrada en scripts
(estructuras de los scripts)
- Lectura/escritura de archivos.

Módulos de Python

Un módulo es un archivo con instrucciones y definiciones de Python, que pueden ser reutilizadas por otro script o kernel.

- Para crear un módulo, es necesario almacenar las instrucciones y/o definiciones en un script .py, e **importarlo** (instrucción `import`)
- Dentro de un módulo, el nombre del mismo (como cadena) queda disponible automáticamente en la variable `__name__`.

Actividad 6: Crear un módulo con los ejercicios resueltos en clase de la Unidad anterior, y cargarlo desde otro intérprete utilizando `import`, `from - import -`, incluyendo uso de `*` y `as`.

También se usa el término **paquete** para referirse a módulos definidos en múltiples archivos.

Módulos de Python

Un script en Python puede crearse con funcionalidad doble: que sea **utilizable como módulo** y también pueda **ejecutarse desde terminal**:

- Utilizando la variable `__name__` es posible condicionar instrucciones para cuando el script sea ejecutado.

Desarrollar scripts pensando en esta dualidad y con documentación adecuada, constituyen la forma más óptima de crear código **compartible** y **reutilizable**.

Actividad 7: Crear una nueva versión del módulo desarrollado anteriormente, pero habilitando la ejecución (de alguna) de las funciones embebidas para cuando sea ejecutado desde terminal.

Módulos de Python - librería estándar

Algunos módulos de la librería estándar de Python a tener en consideración:

- sys - Acceso a parámetros y funciones específicos de sistema (argv, ps1)
- builtins - Acceso a las funciones y nombres integrados.
- os - Interacción con el sistema operativo
- time - Acceso a tiempo y conversiones
- math, cmath - Funciones matemáticas (y para números complejos).
- decimal - Aritmética decimal de coma tanto fija como flotante.
- fractions - Números racionales
- **random** - Generar números pseudoaleatorios
- statistics - Funciones de estadística matemática.

Una lista más completa se puede encontrar en: <https://docs.python.org/es/3/library/index.html>

Actividad 8: Probar definiciones de módulos estándar. Aprender a generar números aleatorios con random.