




CTOP - PYTHON

# Unidad 01

## Conceptos básicos de Python

The logo for Fomento Ocupacional FOC is a large, light gray circular emblem. It features the words "Fomento Ocupacional" in a serif font, arched along the top inner edge of the circle. In the center of the circle, the letters "FOC" are prominently displayed in a large, bold, sans-serif font. Behind the "FOC" text, there is a faint, stylized graphic of a globe or a sphere with intersecting lines.

*Los documentos, elementos gráficos, vídeos, transparencias y otros recursos didácticos incluidos en este contenido pueden contener imprecisiones técnicas o errores tipográficos. Periódicamente se realizan cambios en el contenido. Fomento Ocupacional FOC SL puede realizar en cualquier momento, sin previo aviso, mejoras y/o cambios en el contenido.*

*Es responsabilidad del usuario el cumplimiento de todas las leyes de derechos de autor aplicables. Ningún elemento de este contenido (documentos, elementos gráficos, vídeos, transparencias y otros recursos didácticos asociados), ni parte de este contenido puede ser reproducida, almacenada o introducida en un sistema de recuperación, ni transmitida de ninguna forma ni por ningún medio (ya sea electrónico, mecánico, por fotocopia, grabación o de otra manera), ni con ningún propósito, sin la previa autorización por escrito de Fomento Ocupacional FOC SL.*

*Este contenido está protegido por la ley de propiedad intelectual e industrial. Pertenecen a Fomento Ocupacional FOC SL los derechos de autor y los demás derechos de propiedad intelectual e industrial sobre este contenido.*

*Sin perjuicio de los casos en que la ley aplicable prohíbe la exclusión de la responsabilidad por daños, Fomento Ocupacional FOC SL no se responsabiliza en ningún caso de daños indirectos, sean cuales fueren su naturaleza u origen, que se deriven o de otro modo estén relacionados con el uso de este contenido.*

© 2025 Fomento Ocupacional FOC SL todos los derechos reservados.

# Índice

<b>1. Objetivos</b>	<b>4</b>
<b>2. Introducción</b>	<b>5</b>
<b>3. Paradigmas de programación</b>	<b>6</b>
3.1 Programación imperativa.....	6
3.2 Programación orientada a objetos.....	7
3.3 Programación funcional.....	7
3.4 Programación declarativa.....	7
<b>4. Algunos conceptos sobre Python</b>	<b>8</b>
4.1. ¿Por qué Python?.....	8
4.2. Python 2 vs. Python 3.....	8
4.3. Implementaciones de Python.....	8
<b>5. Entorno de trabajo en Python</b>	<b>9</b>

# Unidad 01

## Conceptos básicos de Python

### 1. Objetivos

Después de completar esta unidad, será capaz de:

- Conocer las características principales del lenguaje Python.
- Identificar correctamente los componentes de Python.
- Diferenciar correctamente los distintos paradigmas de programación.
- Conocer las peculiaridades del entorno de trabajo de Python.
- Aplicar la sintaxis básica en ejemplos prácticos en Python.

## 2. Introducción

**Python** es un lenguaje de programación **de alto nivel**, **interpretado** y **de propósito general**, conocido por su sintaxis clara y legible. Decimos que es de alto nivel porque usa caracteres y palabras del lenguaje humano, por contraposición a los lenguajes de bajo nivel (como el lenguaje máquina), que codifican las instrucciones que se ejecutan más directamente en el procesador. A diferencia de **lenguajes compilados** como C/C++, Scala o Rust, el código fuente en Python no se traduce a código máquina, sino que debe ser “**interpretado**” antes de poder ejecutarse; para ello, es necesario tener instalado previamente un entorno de ejecución, llamado **intérprete de Python** (la implementación más común es **CPython**). Python es hoy un lenguaje muy popular en desarrollo web, análisis de datos, inteligencia artificial y automatización de procesos.

Entre sus características podemos destacar:

- **Sencillo y legible:** Sintaxis cercana al lenguaje natural, fácil de aprender.
- **Multiparadigma:** Soporta programación orientada a objetos, imperativa y funcional.
- **Interpretado:** No requiere compilación, se ejecuta línea por línea.
- Con una **extensa biblioteca estándar:** Módulos para casi cualquier tarea (redes, bases de datos, matemáticas, etc.).
- **Comunidad activa:** Gran cantidad de frameworks y librerías de terceros (Django, NumPy, TensorFlow, etc.).
- **Multiplataforma:** Funciona en Windows, macOS, Linux y más.

Un lenguaje de programación, como cualquier otro lenguaje, se compone de:

- Un **alfabeto:** conjunto de símbolos o caracteres utilizados para formar las palabras del lenguaje (p.ej. el alfabeto latino para el español o el inglés, el kanji para el japonés).
- Un **léxico:** conjunto de palabras válidas en el lenguaje (p.ej. la palabra "ordenador", mientras que "ordnedro" no).
- Una **sintaxis:** conjunto de reglas que determinan si una oración o frase (cadena de palabras) es válida.
- Una **semántica:** conjunto de reglas que determinan si una frase tiene sentido (p.ej. una frase del tipo «ayer por la tarde amanecí» tiene una sintaxis válida, pero quizá no tendría sentido... más allá del poético).



Imagen: Logo de Python

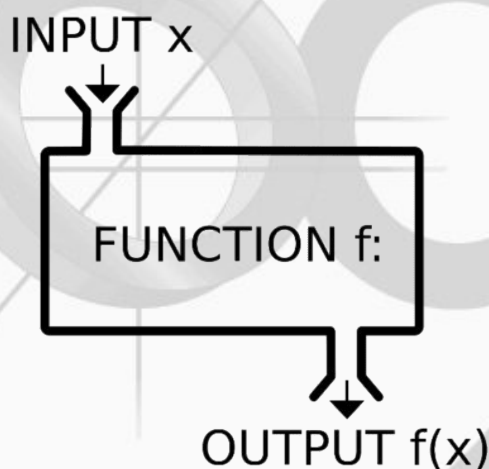
## 3.Paradigmas de programación

Dentro de la programación en Python existe una amplia oferta de tipos y paradigmas para explorar. Desde la **programación orientada a objetos (POO)** hasta la **programación funcional**, cada paradigma conlleva unos pros y contras que debemos sopesar antes de abordar nuestros proyectos. Se trata de crear aplicaciones que no solo sean eficientes, sino también elegantes y flexibles. Conocer y profundizar en estos estilos o paradigmas de programación te permitirá expandir tus habilidades y destacar en el ámbito laboral.

Python es un lenguaje tremendamente versátil, que soporta múltiples paradigmas de programación. Los principales paradigmas o tipos de programación que se pueden implementar en Python incluyen:

- **Programación Imperativa**
- **Programación Orientada a Objetos**
- **Programación Funcional**
- **Programación Declarativa**

A continuación, describiremos cada uno de estos paradigmas de programación.



*Imagen: Programación funcional en Python*

### 3.1 Programación imperativa

La **programación imperativa** se centra en la idea de que el programa consta de una serie de instrucciones que, por lo general, se ejecutan de manera secuencial. En este enfoque, el programa va pasando por diferentes estados a medida que se ejecutan las instrucciones. Siguiendo este paradigma, en Python es habitual utilizar estructuras de control de tipo: **if**, **for** y **while** para controlar el flujo de ejecución.

## 3.2 Programación orientada a objetos

La **programación orientada a objetos (POO)** es un paradigma que utiliza entidades llamadas “objetos”, que combinan (“encapsulan”) datos y funciones. Python permite definir **clases** que pueden encapsular datos y comportamientos. Algunas características fundamentales e inherentes a la POO, que Python soporta, son:

- **Encapsulamiento:** Agrupación de datos y métodos.
- **Herencia:** Reutilización de código existente.
- **Polimorfismo:** Uso de una única interfaz para diferentes formas.

## 3.3 Programación funcional

La **programación funcional** es un paradigma de programación que se centra en el uso de funciones matemáticas para transformar datos y resolver problemas, evitando el uso de estados mutables. En la programación funcional el flujo principal consiste en funciones encadenadas, donde la salida de una función alimenta la entrada de otra. Es un poco más compleja que la programación imperativa (la más común) porque requiere declarar y llamar a funciones dentro de otras funciones, por lo que es fácil perder la noción de lo que se está haciendo, ya que no vemos claramente el “paso a paso” que permite la programación imperativa. Python soporta este enfoque mediante el uso de funciones como **map**, **filter** y **reduce**. Además, permite el uso de funciones **lambda**, que son funciones anónimas que pueden usarse para crear pequeñas funciones **inline**.

## 3.4 Programación declarativa

En la **programación declarativa**, el enfoque se centra en el “**qué se debe hacer**” en lugar del “**cómo hacerlo**”. Se basa en describir el problema declarando propiedades y reglas que deben cumplirse, en lugar de instrucciones. Esto significa que el programador especifica el resultado deseado sin definir los pasos para lograrlo. Algunos de los primeros ejemplos de lenguajes funcionales fueron Lisp y Prolog. Python permite este tipo de programación a través de *frameworks* como SQLAlchemy y Django ORM para la interacción con bases de datos.



## 4. Algunos conceptos sobre Python

### 4.1. ¿Por qué Python?

La popularidad de Python tiene que ver con múltiples razones: es fácil de aprender, fácil de enseñar, fácil de entender, de obtener y de utilizar. Es gratuito, abierto y multiplataforma. A diferencia de otros lenguajes, la curva de aprendizaje es muy suave. Como **lenguaje interpretado** (también llamado **lenguaje de scripting**), Python tiene dos competidores directos:

- **Perl:** un lenguaje de scripting clásico, diseñado por Larry Wall en 1987. Perl toma características del lenguaje C, del lenguaje interpretado bourne shell (sh), de AWK, sed, y Lisp, y está especializado en el procesamiento de texto.
- **Ruby:** un lenguaje de scripting diseñado por Yukihiro Matsumoto en 1993. Combina una sintaxis inspirada en Python y Perl, con características de programación orientada a objetos similares a Smalltalk.

### 4.2. Python 2 vs. Python 3

Existen dos tipos principales de Python, llamados **Python 2** y **Python 3**. El desarrollo de Python 2 se ha estancado, aunque aún existen actualizaciones. Python 3 es la versión más nueva (y la actual) del lenguaje, a través de la cual sigue evolucionando. Es importante saber que estas dos versiones de Python **no son compatibles entre sí**. Los programas escritos en Python 2 no se ejecutarán en un entorno de Python 3 y viceversa. Python 3 no es solo una versión mejorada de Python 2, es un lenguaje diferente, aunque similar a su predecesor. Para nuevos proyectos de Python, deberías usar **Python 3**. La buena noticia es que todas las versiones recientes de Python 3 (3.12, 3.11 o 3.10) son compatibles con las versiones anteriores de Python 3.x (3.9, 3.7 o 3.6).

### 4.3. Implementaciones de Python

La implementación tradicional de Python, llamada **CPython**, es la versión de referencia del lenguaje creado por Guido van Rossum. Guido van Rossum usó el **lenguaje de programación C** para implementar la primera versión de su lenguaje y esta decisión sigue vigente. CPython es la implementación de Python más influyente de todas.

Otra implementación alternativa es **Jython** (la "J" es de **Java**). Se trata de una implementación del lenguaje Python escrita en Java en lugar de C. Resulta útil, por ejemplo, si desarrollas sistemas grandes y complejos escritos completamente en Java. Java y C viven en mundos completamente diferentes y no comparten muchas ideas comunes, por lo que Jython puede comunicarse con la infraestructura Java existente de manera más efectiva. Es por esto que algunos proyectos lo encuentran útil y necesario. Lo malo es que la implementación actual de Jython no soporta Python 3.

Otro ejemplo de implementación es **PyPy**, un Python dentro de un Python. Representa un entorno de ejecución escrito en un lenguaje similar a Python, llamado **RPython (Restricted Python)**. En realidad es un subconjunto de Python. PyPy es compatible con el lenguaje Python 3.



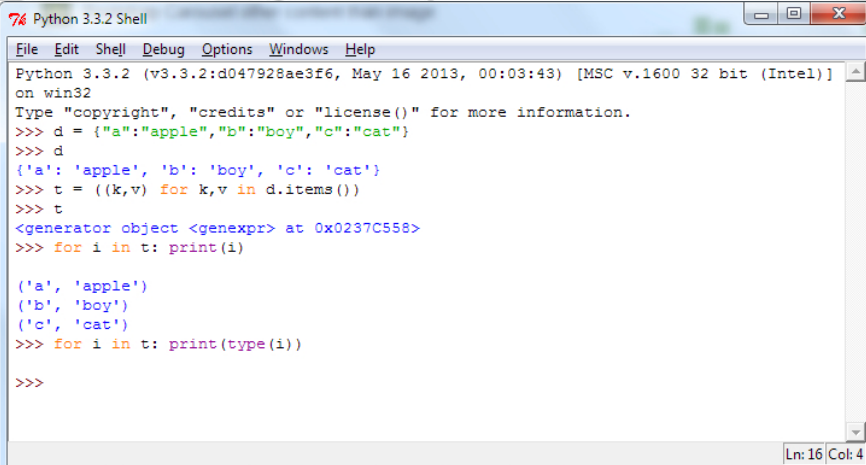
Por último, **MicroPython** es una implementación eficiente de software de código abierto de Python 3, optimizada para ejecutarse en microcontroladores. Incluye un pequeño subconjunto de la biblioteca estándar de Python, pero posee una gran cantidad de funciones, como mensajes interactivos o números enteros de precisión arbitraria, así como módulos que dan acceso al programador a hardware de bajo nivel.

## 5. Entorno de trabajo en Python

Una vez que tengas Python 3 instalado, es hora de verificar por ti mismo que el entorno de Python es completo y funcional. Existen muchas formas de utilizar Python, especialmente si vas a ser un desarrollador de Python. Para comenzar tu trabajo, necesitas las siguientes herramientas:

- Un **editor** que te ayudará a escribir el código (resaltado de sintaxis, etc).
- Una **consola** en la que puedas ejecutar tu código recién escrito y forzar su parada cuando lo requieras (Ctrl-C).
- Una herramienta llamada **depurador (debugger)**, capaz de ejecutar tu código paso a paso e inspeccionarlo a lo largo de su ejecución.

La instalación de Python 3 proporciona de serie este conjunto de herramientas, combinadas en una aplicación llamada **IDLE**. IDLE es un acrónimo de *Integrated Development and Learning Environment* (**Entorno de Desarrollo y Aprendizaje Integrado**). Junto a este programa se incluye una consola interactiva (**REPL**, acrónimo de **Read-Eval-Print Loop**), que permite ejecutar las instrucciones en modo intérprete de línea de comandos, de modo que el usuario puede ejecutar sentencias de Python una detrás de otra, y obtener los resultados inmediatamente, sin tener que esperar a que se ejecute todo el código fuente de un programa o *script* escrito en Python.



```
Python 3.3.2 Shell
File Edit Shell Debug Options Windows Help
Python 3.3.2 (v3.3.2:d047928ae3f6, May 16 2013, 00:03:43) [MSC v.1600 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.
>>> d = {"a": "apple", "b": "boy", "c": "cat"}
>>> d
{'a': 'apple', 'b': 'boy', 'c': 'cat'}
>>> t = ((k,v) for k,v in d.items())
>>> t
<generator object <genexpr> at 0x0237C558>
>>> for i in t: print(i)

('a', 'apple')
('b', 'boy')
('c', 'cat')
>>> for i in t: print(type(i))

>>>
```

Imagen: IDLE