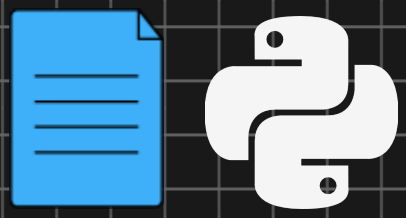

Guía rápida de fundamentos en Python

Con más de 200 términos

mouredev pro

mouredev.pro/recursos

Índice

Introducción

Esta es una guía de referencia que hace un recorrido por los principales fundamentos del lenguaje de programación Python.

Puedes utilizarla para conocer rápidamente las características de Python, apoyar tu aprendizaje y obtener información sobre más de 200 términos de manera rápida y sencilla, consultando nombres, definiciones y sintaxis.

Tipos de datos básicos

Los **tipos de datos** son la base de cualquier programa, ya que determinan cómo se almacenan y manipulan los valores. Existen tipos primitivos como enteros, flotantes, cadenas de texto y booleanos, y estructuras de datos más avanzadas como listas, tuplas, conjuntos y diccionarios. Conocerlos es fundamental para escribir código eficiente y bien estructurado.

Nombre	Palabra reservada	Sintaxis
Primitivos		
Entero	`int`	`0`
Flotante	`float`	`3.14`
Cadena de texto	`str`	`"Hola"`
Booleano	`bool`	`True/False`
Complejo	`complex`	`1 + 2j`
Byte	`bytes`	`b"Hola"`
Nulo	`NoneType`	`None`
Estructuras		
Lista/Pila	`list`	`[1, 2, 3]`
Tupla	`tuple`	`(1, 2, 3)`
Conjunto	`set`	`{1, 2, 3}`
Conj. inmutable	`frozenset`	`frozenset({1, 2, 3})`
Diccionario	`dict`	`{"clave": "valor"}`
Bytearray	`bytearray`	`bytearray(5)`
Rango	`range`	`range(5)`
Memory view	`memoryview`	`memoryview(bytes(5))`

Sintaxis básica

La **sintaxis** de Python es sencilla y legible, lo que lo convierte en un lenguaje fácil de aprender. Aquí encontrarás las estructuras fundamentales del lenguaje, como condicionales, bucles, funciones y clases, junto con su respectiva sintaxis para escribir código claro y organizado.

Nombre	Palabra reservada	Sintaxis
Condicional if	`if`	`if condición:`
Condicional elif	`elif`	`elif otra_condición:`
Condicional else	`else`	`else:`
Bucle for	`for`	`for elemento in secuencia:`
Bucle while	`while`	`while condición:`
Sentencia break	`break`	`break`
Sentencia continue	`continue`	`continue`
Función	`def`	`def nombre(parámetros):`
Clase	`class`	`class NombreClase:`
Variable global	`global`	`global nombre_variable`
Excepciones	`try, except`	`try:...except Excepción:`
Bloque finally	`finally`	`finally:`
Retorno	`return`	`return valor`
Comprobar tipo	`is`	`if variable is tipo:`
Existe	`in`	`if elemento in colección:`
Módulo	`import` `from, import`	`import nombre_módulo` `from módulo import nombre`

Operadores (parte 1 de 3)

Los **operadores** son símbolos que permiten realizar cálculos y comparaciones en Python. Se dividen en varias categorías: aritméticos (suma, resta, multiplicación), de comparación (mayor, menor, igual), lógicos (and, or, not), de asignación (=, +=), bitwise y otros especializados como el operador ternario y el walrus (:=).

Nombre	Representación	Sintaxis
Aritméticos		
Suma	+	`a + b`
Resta	-	`a - b`
Multiplicación	*	`a * b`
División	/	`a / b`
División entera	//	`a // b`
Módulo (residuo)	%	`a % b`
Exponenciación	**	`a ** b`
Comparación		
Igual a	==	`a == b`
Distinto de	!=	`a != b`
Mayor que	>	`a > b`
Menor que	<	`a < b`
Mayor o igual que	>=	`a >= b`
Menor o igual que	<=	`a <= b`

Operadores (parte 2 de 3)

Nombre	Representación	Sintaxis
Lógicos		
AND	`and`	`a and b`
OR	`or`	`a or b`
NOT	`not`	`not a`
Asignación		
Asignación	`=`	`x = 5`
A. con suma	`+=`	`x += 3`
A. con resta	`-=`	`x -= 3`
A. con multiplicación	`*=`	`x *= 3`
A. con división	`/=`	`x /= 3`
A. con división entera	`//=`	`x //= 3`
A. con módulo	`%=`	`x %= 3`
A. con exponenciación	`**=`	`x **= 3`
A. con bitwise AND	`&=`	`x &= 3`
A. con bitwise OR	`|=`	`x |= 3`
A. con bitwise XOR	`^=`	`x ^= 3`

Operadores (parte 3 de 3)

Nombre	Representación	Sintaxis
Identidad		
Es	`is`	`a is`
No es	`is not`	`a is not b`
Pertenencia		
Pertenece	`in`	`x in lista`
No pertenece	`not in`	`x not in lista`
Bitwise		
AND bit a bit	`&`	`a & b`
OR bit a bit	`|`	`a | b`
XOR bit a bit	`^`	`a ^ b`
NOT bit a bit	`~`	`~a`
Desplazamiento a la izq.	`<<`	`a << n`
Desplazamiento a la der.	`>>`	`a >> n`
Ternario		
Condicional ternario	`if-else`	`x if cond else y`
Walrus		
Operador Walrus	`:=`	`(x := valor)`

Funciones (parte 1 de 3)

Las **funciones** permiten reutilizar código y hacer que los programas sean más modulares y organizados. Python ofrece una gran variedad de funciones integradas para trabajar con diferentes tipos de datos, desde operaciones matemáticas hasta manipulación de colecciones. Aquí tienes las funciones más comunes que facilitan el desarrollo en el lenguaje.

Nombre	Operación
`print()`	Muestra texto o variables en la consola
`input()`	Permite la entrada de datos desde la consola
`len()`	Devuelve la cantidad de elementos
`type()`	Devuelve el tipo de dato
`range()`	Genera una secuencia de números en un rango
`int(), float(), str()`	Convierte a entero, coma flotante o cadena
`list(), tuple(), set(), dict()`	Crea listas, tuplas, conjuntos o diccionarios
`sum()`	Suma los elementos iterables
`min(), max()`	Encuentra el mínimo o el máximo en iterables
`sorted()`	Ordena elementos en listas y otros iterables
`abs()`	Retorna el valor absoluto de un número
`round()`	Redondea a un número con decimales concretos
`map()`	Aplica una función a cada elemento iterado
`filter()`	Filtra elementos iterables según una condición
`reduce()`	Aplica una función a los elementos iterables para reducirlos a un único valor

Funciones (parte 2 de 3)

Nombre	Operación
`enumerate()`	Agrega un índice a cada elemento iterable
`zip()`	Une iterables en pares de elementos
`open()`	Abre un archivo para leer, escribir o modificar
`isinstance()`	Comprueba si una variable es de un tipo concreto
`issubclass()`	Comprueba relaciones entre clases
`hasattr()`	Verifica si un objeto tiene un atributo
`getattr()`	Obtiene el valor de un atributo
`setattr()`	Asigna un valor a un atributo
`delattr()`	Elimina un atributo de un objeto
`help()`	Muestra la documentación de un objeto o módulo
`id()`	Devuelve la identidad única en memoria
`next()`	Retorna el siguiente valor de un iterador
`iter()`	Transforma un iterable en un iterador
`chr()`, `ord()`	Trabaja con valores Unicode de caracteres
`bin()`, `oct()`, `hex()`	Convierte números enteros a diferentes bases
`any()`, `all()`	Evalúa condiciones en iterables para al menos uno o todos los elementos

Funciones (parte 3 de 3)

Nombre	Operación
`pow()`	Calcula una potencia
`divmod()`	Retorna el cociente y el resto de una división
`copy()`	Crea una copia superficial de un objeto
`eval()`	Ejecuta una expresión en forma de string
`exec()`	Ejecuta un bloque de código en forma de string
`format()`	Formatea cadenas con valores personalizados
`reversed()`	Invierte el orden de un iterable
`slice()`	Crea una porción de un iterable sin modificarlo
`callable()`	Verifica si es invocable como una función
`dir()`	Muestra los atributos y métodos de un objeto
`frozenset()`	Crea un conjunto inmutable
`complex()`	Crea números complejos
`bool()`	Transforma un valor en booleano
`super()`	Retorna un proxy que delega en la superclase
`bin()`	Crea la representación binaria de un número
`globals()`	Retorna las variables globales
`locals()`	Retorna las variables locales

Estructuras de datos (parte 1 de 3)

Las listas, tuplas, conjuntos y diccionarios son **estructuras de datos** clave en Python, y cada una tiene operaciones específicas que permiten manipular la información de manera eficiente. Esta sección cubre métodos esenciales para agregar, eliminar, buscar, ordenar y transformar datos en estas estructuras.

Nombre	Operación	Sintaxis
Listas (list)		
`append()`	Agrega un elemento al final	`lista.append(valor)`
`extend()`	Agrega múltiples elementos	`lista.extend(iterable)`
`insert()`	Inserta un elemento en un índice	`lista.insert(i, valor)`
`remove()`	Elimina la primera ocurrencia de un valor	`lista.remove(valor)`
`pop()`	Elimina y devuelve un elemento (por índice o último)	`lista.pop(i)` `lista.pop()`
`index()`	Devuelve el índice de un valor	`lista.index(valor)`
`count()`	Cuenta las ocurrencias de un valor	`lista.count(valor)`
`sort()`	Ordena la lista	`lista.sort()`
`reverse()`	Invierte el orden de la lista	`lista.reverse()`
`copy()`	Crea una copia de la lista	`lista.copy()`
`clear()`	Elimina todos los elementos	`lista.clear()`
Tuplas (tuple)		
`count()`	Cuenta las ocurrencias de un valor	`tupla.count(valor)`
`index()`	Devuelve el índice de un valor	`tupla.index(valor)`

Estructuras de datos (parte 2 de 3)

Nombre	Operación	Sintaxis
Conjuntos (set)		
`add()`	Agrega un elemento al conjunto	`conjunto.add(valor)`
`remove()`	Elimina un elemento (error si no existe)	`conjunto.remove(valor)`
`discard()`	Elimina un elemento (sin error si no existe)	`conjunto.discard(valor)`
`pop()`	Elimina y devuelve un elemento aleatorio	`conjunto.pop()`
`clear()`	Elimina todos los elementos	`conjunto.clear()`
`union()`	Une dos conjuntos	`conjunto.union(otro)`
`intersection()`	Elementos comunes en ambos conjuntos	`conjunto.intersection(otro)`
`difference()`	Elementos en A pero no en B	`conjunto.difference(otro)`
`symmetric_difference()`	Elementos no comunes en ambos	`conjunto.symmetric_difference(otro)`

Estructuras de datos (parte 3 de 3)

Nombre	Operación	Sintaxis
Diccionarios (dict)		
`keys()`	Devuelve las claves del diccionario	`diccionario.keys()`
`values()`	Devuelve los valores del diccionario	`diccionario.values()`
`items()`	Devuelve pares clave-valor	`diccionario.items()`
`get()`	Obtiene el valor de una clave (sin error)	`diccionario.get(clave, defecto)`
`pop()`	Elimina y devuelve un valor	`diccionario.pop(clave)`
`popitem()`	Elimina y devuelve el último par clave-valor	`diccionario.popitem()`
`update()`	Agrega o actualiza claves con valores	`diccionario.update(otro_dict)`
`setdefault()`	Obtiene el valor o lo asigna si no existe	`diccionario.setdefault(clave, valor)`
`copy()`	Crea una copia del diccionario	`diccionario.copy()`
`clear()`	Elimina todos los elementos	`diccionario.clear()`

Manejo de archivos (parte 1 de 2)

El **manejo de archivos** es una tarea común en programación. Python permite leer, escribir, modificar y eliminar archivos de manera sencilla usando funciones como `open()`, read(), `write()`, entre otras. Además, ofrece herramientas para manipular directorios y trabajar con formatos específicos.

Modos de apertura en `open()`:

- `"r"` → Lectura (por defecto)
- `"w"` → Escritura (borra contenido previo)
- `"a"` → Agregar contenido al final
- `"rb"` / `"wb"` → Lectura y escritura en modo binario

Nombre	Representación	Sintaxis
`open()`	Abre un archivo en diferentes modos de acceso	`archivo = open("archivo.txt", "r")`
`read()`	Lee el contenido completo del archivo	`contenido = archivo.read()`
`readline()`	Lee una única línea del archivo	`línea = archivo.readline()`
`readlines()`	Lee todas las líneas y las devuelve en una lista	`líneas = archivo.readlines()`
`write()`	Escribe datos en un archivo (sobrescribe)	`archivo.write("Texto")`
`writelines()`	Escribe múltiples líneas en un archivo	`archivo. writelines(lista_de_texto)`
`close()`	Cierra el archivo para liberar recursos	`archivo.close()`
`flush()`	Fuerza la escritura de datos del buffer en el archivo	`archivo.flush()`

Manejo de archivos (parte 2 de 2)

Nombre	Representación	Sintaxis
`with open()`	Abre un archivo con manejo automático de cierre	`with open("archivo.txt") as f:`
`seek()`	Mueve el cursor a una posición específica	`archivo.seek(0)`
`tell()`	Devuelve la posición actual del cursor	`pos = archivo.tell()`
`truncate()`	Corta el archivo a un tamaño específico	`archivo.truncate(50)`
`exists()`	Comprueba si un archivo existe	`os.path.exists("archivo.txt")`
`remove()`	Elimina un archivo	`os.remove("archivo.txt")`
`rename()`	Renombra un archivo	`os.rename("viejo.txt", "nuevo.txt")`
`mkdir()`	Crea un directorio	`os.mkdir("nueva_carpeta")`
`rmdir()`	Elimina un directorio vacío	`os.rmdir("nueva_carpeta")`
`listdir()`	Lista los archivos en un directorio	`os.listdir("ruta")`

Es recomendable usar `with open()` en lugar de `open()` + `close()` para evitar problemas de fugas de memoria.

Módulos estándar (parte 1 de 2)

Python cuenta con una **biblioteca estándar** que incluye **módulos** integrados listos para ser usados sin instalación adicional. Estos módulos facilitan tareas como manipulación de archivos (`os`, `shutil`), operaciones matemáticas (`math`, `random`), manejo de fechas (`datetime`), y muchas más. Conocer estos módulos ayuda a optimizar el desarrollo sin necesidad de librerías externas.

Nombre	Descripción
`os`	Permite interactuar con el sistema operativo (archivos, directorios, procesos)
`sys`	Proporciona acceso a variables y funciones del intérprete de Python
`math`	Ofrece funciones matemáticas avanzadas (raíces, logaritmos, trigonometría)
`random`	Genera números aleatorios y selecciona elementos al azar
`datetime`	Maneja fechas y horas
`time`	Funciones para manejar el tiempo y pausas en la ejecución
`json`	Permite trabajar con datos en formato JSON (serialización y deserialización)
`csv`	Facilita la lectura y escritura de archivos CSV
`re`	Proporciona herramientas para trabajar con expresiones regulares
`collections`	Contiene estructuras de datos avanzadas (deque, counter, defaultdict)
`itertools`	Ofrece herramientas para trabajar con iteradores y combinaciones de datos

Módulos estándar (parte 2 de 2)

Nombre	Descripción
`functools`	Permite funciones de orden superior y optimización con caché
`operator`	Proporciona funciones rápidas para operaciones matemáticas y lógicas
`statistics`	Contiene funciones estadísticas como media, mediana y desviación estándar
`hashlib`	Permite generar hashes criptográficos (SHA256, MD5, etc.)
`logging`	Proporciona herramientas para registrar eventos y errores en aplicaciones
`argparse`	Facilita el manejo de argumentos en la línea de comandos
`shutil`	Permite la manipulación de archivos y directorios (copiar, mover, eliminar)
`socket`	Soporta la comunicación en red mediante sockets
`threading`	Permite ejecutar múltiples hilos de forma concurrente
`multiprocessing`	Soporta la ejecución de múltiples procesos en paralelo
`subprocess`	Permite ejecutar comandos del sistema y capturar su salida
`tkinter`	Biblioteca estándar de Python para interfaces gráficas (GUIs)

Módulos externos (parte 1 de 2)

Además de la biblioteca estándar, Python permite instalar **módulos externos** para ampliar sus capacidades. Algunas de las librerías más populares incluyen `requests` para peticiones web, `numpy` y pandas para análisis de datos, `matplotlib` para gráficos, y `django` o `reflex` para desarrollo web. Estas son algunas de las librerías de terceros más utilizadas.

Nombre	Descripción
`requests`	Permite realizar peticiones HTTP de manera sencilla
`python-dotenv`	Carga variables de entorno desde archivos .env
`numpy`	Librería para cálculo numérico y manipulación de arrays multidimensionales
`pandas`	Facilita el análisis y manipulación de datos con estructuras como DataFrames
`matplotlib`	Biblioteca para crear gráficos y visualizaciones
`seaborn`	Extensión de matplotlib con estilos más avanzados para visualización de datos
`scipy`	Contiene herramientas avanzadas de cálculo científico y optimización
`scikit-learn`	Librería para Machine Learning con algoritmos de clasificación, regresión y clustering
`tensorflow`	Framework para Machine Learning y Deep Learning desarrollado por Google
`torch`	Biblioteca de Deep Learning desarrollada por Meta (PyTorch)
`opencv-python`	Procesamiento de imágenes y visión artificial
`pillow`	Biblioteca para manipulación y procesamiento de imágenes

Módulos externos (parte 2 de 2)

Nombre	Descripción
`beautifulsoup4`	Facilita la extracción de datos de páginas web (Web Scraping)
`selenium`	Automatización de navegación web mediante scripts
`fastapi`	Framework moderno y rápido para crear APIs
`flask`	Framework ligero para desarrollo web
`django`	Framework robusto para desarrollo web
`reflex`	Framework para crear aplicaciones web full-stack
`sqlalchemy`	ORM para interactuar con bases de datos SQL de manera eficiente
`pymongo`	Cliente para bases de datos MongoDB
`pytest`	Librería para realizar pruebas unitarias
`loguru`	Biblioteca mejorada para logging
`typer`	Permite crear aplicaciones de línea de comandos fácilmente
`rich`	Agrega colores y estilos a la terminal de Python
`kivy`	Framework para crear aplicaciones gráficas multiplataforma (Windows, Linux, macOS, Android, iOS)
`flet`	Framework para crear aplicaciones Flutter multiplataforma (escritorio, web y móvil)

Todos estos módulos deben instalarse con `pip install nombre_del_modulo` antes de ser utilizados.

¿Quieres aprender más sobre Python?

Aquí tienes mis cursos para aprender Python desde cero.

Curso gratis en YouTube:
https://mouredev.link/python

Curso con extras (lecciones por tema, ejercicios, soporte, comunidad, test y certificado) en el campus de estudiantes mouredev pro:
https://mouredev.pro/cursos/python-desde-cero

(Utiliza el cupón "PRO" para acceder con un 10% de descuento a todas las suscripciones y cursos del campus).

mouredev pro

Estudia programación y desarrollo de software de manera diferente

mouredev.pro