



Guía rápida de fundamentos en **Python**

Con más de 200 términos

mouredev^{pro}

mouredev.pro/recursos

Índice

Introducción	2
Tipos de datos básicos	3
Sintaxis básica	4
Operadores (parte 1 de 3)	5
Operadores (parte 2 de 3)	6
Operadores (parte 3 de 3)	7
Funciones (parte 1 de 3)	8
Funciones (parte 2 de 3)	9
Funciones (parte 3 de 3)	10
Estructuras de datos (parte 1 de 3)	11
Estructuras de datos (parte 2 de 3)	12
Estructuras de datos (parte 3 de 3)	13
Manejo de archivos (parte 1 de 2)	14
Manejo de archivos (parte 2 de 2)	15
Módulos estándar (parte 1 de 2)	16
Módulos estándar (parte 2 de 2)	17
Módulos externos (parte 1 de 2)	18
Módulos externos (parte 2 de 2)	19
¿Quieres aprender más sobre Python?	20

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

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

Funciones (parte 2 de 3)

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

Funciones (parte 3 de 3)

Nombre	Operación
<code>pow()</code>	Calcula una potencia
<code>divmod()</code>	Retorna el cociente y el resto de una división
<code>copy()</code>	Crea una copia superficial de un objeto
<code>eval()</code>	Ejecuta una expresión en forma de string
<code>exec()</code>	Ejecuta un bloque de código en forma de string
<code>format()</code>	Formatea cadenas con valores personalizados
<code>reversed()</code>	Invierte el orden de un iterable
<code>slice()</code>	Crea una porción de un iterable sin modificarlo
<code>callable()</code>	Verifica si es invocable como una función
<code>dir()</code>	Muestra los atributos y métodos de un objeto
<code>frozenset()</code>	Crea un conjunto inmutable
<code>complex()</code>	Crea números complejos
<code>bool()</code>	Transforma un valor en booleano
<code>super()</code>	Retorna un proxy que delega en la superclase
<code>bin()</code>	Crea la representación binaria de un número
<code>globals()</code>	Retorna las variables globales
<code>locals()</code>	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)		
<code>append()</code>	Agrega un elemento al final	<code>lista.append(valor)</code>
<code>extend()</code>	Agrega múltiples elementos	<code>lista.extend(iterable)</code>
<code>insert()</code>	Inserta un elemento en un índice	<code>lista.insert(i, valor)</code>
<code>remove()</code>	Elimina la primera ocurrencia de un valor	<code>lista.remove(valor)</code>
<code>pop()</code>	Elimina y devuelve un elemento (por índice o último)	<code>lista.pop(i)</code> <code>lista.pop()</code>
<code>index()</code>	Devuelve el índice de un valor	<code>lista.index(valor)</code>
<code>count()</code>	Cuenta las ocurrencias de un valor	<code>lista.count(valor)</code>
<code>sort()</code>	Ordena la lista	<code>lista.sort()</code>
<code>reverse()</code>	Invierte el orden de la lista	<code>lista.reverse()</code>
<code>copy()</code>	Crea una copia de la lista	<code>lista.copy()</code>
<code>clear()</code>	Elimina todos los elementos	<code>lista.clear()</code>
Tuplas (tuple)		
<code>count()</code>	Cuenta las ocurrencias de un valor	<code>tupla.count(valor)</code>
<code>index()</code>	Devuelve el índice de un valor	<code>tupla.index(valor)</code>

Estructuras de datos (parte 2 de 3)

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

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

Manejo de archivos (parte 2 de 2)

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

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
<code>os</code>	Permite interactuar con el sistema operativo (archivos, directorios, procesos)
<code>sys</code>	Proporciona acceso a variables y funciones del intérprete de Python
<code>math</code>	Ofrece funciones matemáticas avanzadas (raíces, logaritmos, trigonometría)
<code>random</code>	Genera números aleatorios y selecciona elementos al azar
<code>datetime</code>	Maneja fechas y horas
<code>time</code>	Funciones para manejar el tiempo y pausas en la ejecución
<code>json</code>	Permite trabajar con datos en formato JSON (serialización y deserialización)
<code>csv</code>	Facilita la lectura y escritura de archivos CSV
<code>re</code>	Proporciona herramientas para trabajar con expresiones regulares
<code>collections</code>	Contiene estructuras de datos avanzadas (deque, counter, defaultdict)
<code>itertools</code>	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
<code>requests</code>	Permite realizar peticiones HTTP de manera sencilla
<code>python-dotenv</code>	Carga variables de entorno desde archivos <code>.env</code>
<code>numpy</code>	Librería para cálculo numérico y manipulación de arrays multidimensionales
<code>pandas</code>	Facilita el análisis y manipulación de datos con estructuras como DataFrames
<code>matplotlib</code>	Biblioteca para crear gráficos y visualizaciones
<code>seaborn</code>	Extensión de <code>matplotlib</code> con estilos más avanzados para visualización de datos
<code>scipy</code>	Contiene herramientas avanzadas de cálculo científico y optimización
<code>scikit-learn</code>	Librería para Machine Learning con algoritmos de clasificación, regresión y clustering
<code>tensorflow</code>	Framework para Machine Learning y Deep Learning desarrollado por Google
<code>torch</code>	Biblioteca de Deep Learning desarrollada por Meta (PyTorch)
<code>opencv-python</code>	Procesamiento de imágenes y visión artificial
<code>pillow</code>	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