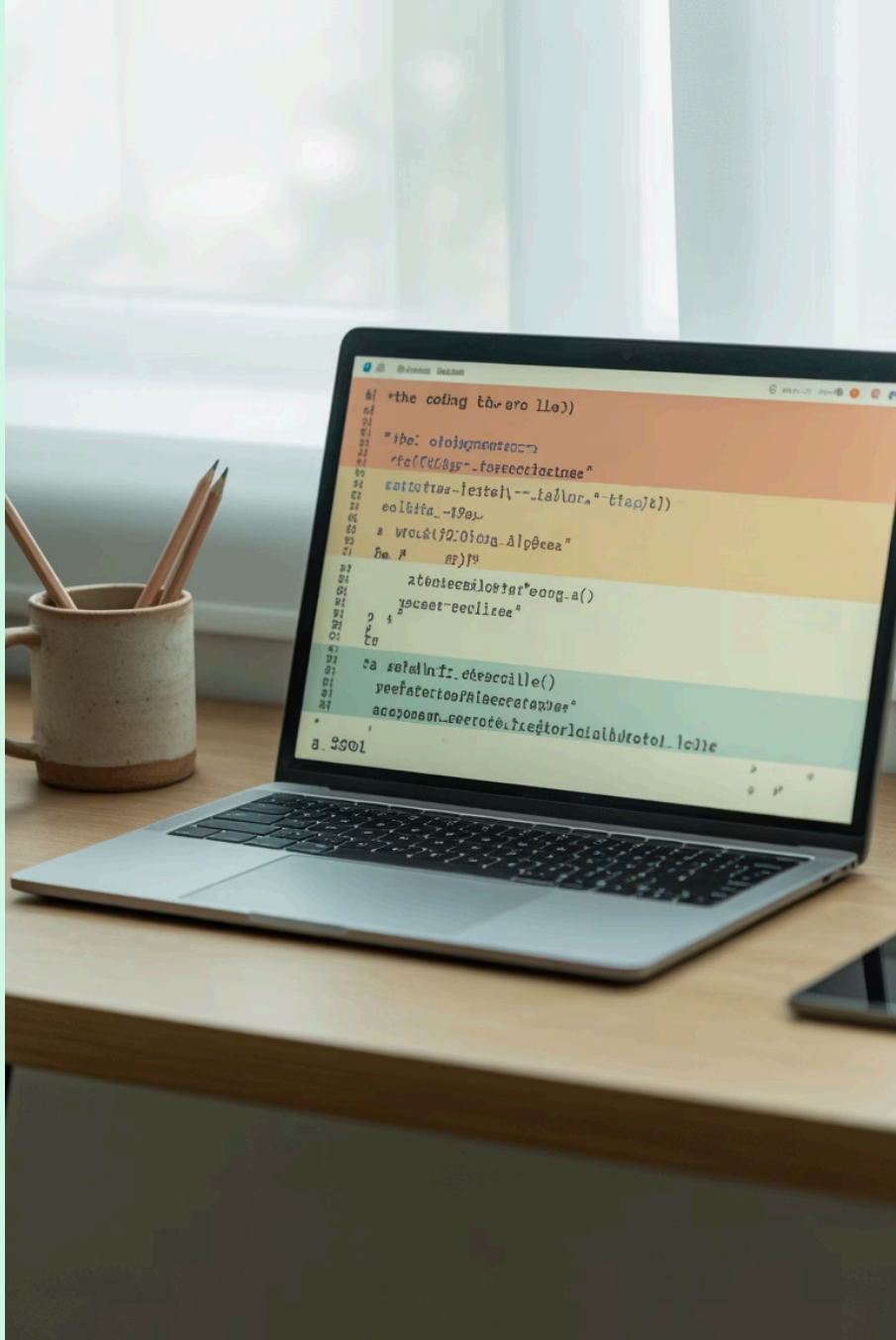
Funciones Lambda en Python

Descubre cómo crear funciones anónimas elegantes y eficientes para optimizar tu código



¿Qué es una función lambda?

En Python, además de definir funciones con `def`, existe otra forma de crear funciones pequeñas y anónimas llamadas **funciones lambda**. Estas funciones se destacan por su sintaxis compacta y su capacidad para expresar operaciones simples en una sola línea.

Se llaman "anónimas" porque no necesitan un nombre definido, aunque podemos asignarlas a una variable si queremos reutilizarlas. Son ideales para operaciones rápidas y puntuales donde crear una función completa sería excesivo.

💡 Dato clave

Las funciones lambda son especialmente útiles cuando necesitas pasar una función como argumento a otra función, como en `map()`, `filter()` o `sorted()`.

Sintaxis de las funciones lambda

lambda

Palabra clave que indica que estamos creando una función anónima

argumentos

Los parámetros que recibe la función, separados por comas si hay varios

expresión

Lo que devuelve la función, siempre en una sola línea de código

```
lambda argumentos: expresión
```

Esta estructura concisa permite definir funciones al vuelo, sin necesidad de utilizar `def` ni especificar un `return` explícito. La expresión después de los dos puntos se evalúa y se devuelve automáticamente.

Comparación: def vs lambda

Función tradicional con def

```
def cuadrado(x):
    return x * x

print(cuadrado(5))  # 25
```

Requiere tres líneas de código, un nombre de función y una declaración `return` explícita.

Función lambda equivalente

```
cuadrado_lambda = lambda x: x * x

print(cuadrado_lambda(5))  # 25
```

Se define en una sola línea, con sintaxis más compacta y return implícito.

Ambas funciones producen exactamente el mismo resultado, pero lambda ofrece una forma más concisa cuando la lógica es simple. Para funciones más complejas o con múltiples líneas, `def` sigue siendo la mejor opción.

¿Cuándo usar funciones lambda?

Las funciones lambda brillan cuando necesitamos operaciones rápidas y puntuales, especialmente como argumentos de funciones de orden superior. Son perfectas para transformaciones simples que no justifican definir una función completa.

map()

Aplica una función a cada elemento de una secuencia, transformando todos los valores de forma simultánea

filter()

Filtra elementos según una condición específica, devolviendo solo aquellos que cumplen el criterio

sorted()

Ordena secuencias aplicando reglas personalizadas mediante una función clave customizada

Ventajas de las funciones lambda

✅ Código más conciso

Reduce varias líneas de código a una sola expresión, mejorando la legibilidad cuando la lógica es simple.

✅ Sin contaminar el espacio de nombres

Al ser anónimas, no añaden nombres innecesarios al scope global, manteniendo el código más limpio.

✅ Perfectas para operaciones puntuales

Ideales cuando necesitas una función que solo usarás una vez, como argumento de otra función.

✅ Facilitan la programación funcional

Permiten escribir código más declarativo y expresivo, siguiendo paradigmas funcionales.

Ejemplo práctico: filtrar números pares

Veamos cómo combinar `filter()` con una función lambda para resolver un problema común: extraer solo los números pares de una lista. Este tipo de operación es muy frecuente en el procesamiento de datos.

01

Definimos la lista original

Creamos una lista con números del 1 al 10 que queremos filtrar

02

Aplicamos filter() con lambda

La función lambda verifica si cada número es divisible entre 2

03

Convertimos el resultado a lista

filter() devuelve un iterador, así que lo convertimos con list()

Código del ejercicio

```
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

pares = list(filter(lambda x: x % 2 == 0, numeros))

print("Números pares:", pares)
```

Salida esperada:

```
Números pares: [2, 4, 6, 8, 10]
```

¿Cómo funciona?

- `lambda x: x % 2 == 0` crea una función que devuelve True si x es par
- `filter()` aplica esa función a cada elemento
- Solo los elementos que devuelven True se incluyen en el resultado
- `list()` convierte el objeto filter en una lista visible

🎯 Ejercicio de práctica

Intenta modificar el código para filtrar números impares, o números mayores que 5. ¡Experimenta con diferentes condiciones!

Más ejemplos de funciones lambda

Elevar al cuadrado con map()

```
numeros = [1, 2, 3, 4, 5]
cuadrados = list(map(
    lambda x: x ** 2,
    numeros
))
# [1, 4, 9, 16, 25]
```

Ordenar por longitud

```
palabras = ["python", "es",
            "genial"]
ordenadas = sorted(
    palabras,
    key=lambda x: len(x)
)
# ["es", "genial", "python"]
```

Sumar dos números

```
suma = lambda a, b: a + b

resultado = suma(10, 5)
# 15
```

Las lambdas pueden recibir múltiples argumentos separados por comas.

Conclusión: El poder de lo simple

1

línea de código

Para expresar funciones simples de forma elegante y eficiente

3

funciones clave

map(), filter() y sorted() son las mejores compañeras de lambda

∞

posibilidades

Dominar lambda abre la puerta a un código más pythónico y funcional

Las funciones lambda son una herramienta esencial en el arsenal de cualquier programador Python. Aunque simples en apariencia, permiten escribir código más limpio, expresivo y funcional. Practica usándolas en tus proyectos para dominar este concepto fundamental.