Clase 008 — Funciones: args, kwargs, lambdas, closures

Parte: 0 — Prerrequisitos · Fuente: Ramalho, Fluent Python 2e — cap. 7 (Functions as First-Class Objects), cap. 9 (Decorators and Closures). · Duración estimada: 90 min.

Clase 008 — Funciones: args, kwargs, lambdas, closures

> *Parte: 0 — Prerrequisitos · Fuente: Ramalho, Fluent Python 2e — cap. 7 (Functions as First-Class Objects), cap. 9 (Decorators and Closures). · Duración estimada: 90 min.*

Objetivo

Que el alumno use funciones como ciudadanos de primera clase: pasarlas como argumento, retornarlas, escribir lambdas cuando aportan, y entender closures — la base de los decoradores que verán más adelante. Sin esto, el código pandas/sklearn parece magia.

Resultados de aprendizaje

Al finalizar la clase, el alumno podrá:

1. Definir funciones con argumentos posicionales, keyword-only, args y *kwargs.
2. Pasar funciones como argumento (callbacks: sorted(xs, key=fn), df.apply(fn)).
3. Usar lambdas donde son legibles (callbacks cortos) y evitarlas donde no (lógica).
4. Explicar y escribir closures (función que captura variables del scope exterior).
5. Anticipar la diferencia entre args y , args (keyword-only marker).

Temas

#	Tema	Por qué importa
1	Argumentos: posicional, keyword, default	Cuatro modos, una sintaxis.
2	args y *kwargs	Funciones que aceptan número variable.
3	Keyword-only con * separador	def f(a, *, b) → b solo nombrado.
4	Funciones como objetos	Asignables, pasables, retornables.
5	Lambdas: dónde sí y dónde no	Callbacks cortos sí; lógica compleja no.
6	Closures: capturando scope	Base mental de los decoradores.

Definiciones y características

First-class object

: En Python, funciones son ciudadanos de primera clase: se asignan a variables (f = saludar), se pasan como argumento (sorted(xs, key=f)), se retornan de otras funciones. Esto habilita callbacks, decoradores y closures.

args / kwargs*

: args captura argumentos posicionales sobrantes en una tupla. kwargs captura argumentos nombrados sobrantes en un dict. Convención: solo el y ** importan; los nombres args/kwargs son convención.

Keyword-only argument

: Argumento que solo puede pasarse nombrado: declarado después de o args en la signatura. def f(a, *, b) obliga a f(1, b=2). Mejora legibilidad en APIs con muchos params.

Lambda

: Función anónima de UNA expresión: lambda x: x*2. Sin nombre, sin docstring, sin múltiples statements. Útil para callbacks cortos (sorted(xs, key=lambda p: p['edad'])). Si necesitas más, usa def.

Closure

: Función que captura variables del scope donde fue definida y las mantiene vivas aunque ese scope termine. Base mental de los decoradores. Para modificar la variable capturada, usa nonlocal.

Decorador

: Función que recibe función y retorna función (típicamente envuelta). Sintaxis: @dec antes de def. Implementado típicamente con closure + @functools.wraps para preservar metadata original.

Dataset / recursos

Datos sintéticos pequeños (lista de dicts simulando ventas). Sin descarga.

Ejercicios

1. Función con todo. Define f(a, b=10, args, c, *kwargs). Llámala de 3 formas distintas que sean válidas. Identifica qué llamadas son inválidas y por qué.

2. sorted con key. Dada list[dict] de personas, ordena por edad (asc) y por nombre alfabético. Usa lambda primero, luego operator.itemgetter.

3. Closure contador. Escribe make_counter() que retorna una función que cada vez que se llama incrementa y retorna un contador interno. ¿Por qué funciona?

4. Memoización manual. Implementa un decorador @memoize usando closure + dict. Aplícalo a Fibonacci recursivo y mide el speedup con %timeit.

5. Compose. Escribe compose(f, g, h) que retorna una función equivalente a lambda x: f(g(h(x))).

Homework verificable

Notebook con: (a) implementación y demo de make_counter explicando con comentario por qué el contador persiste; (b) @memoize aplicado a Fibonacci recursivo con benchmark (N=35) antes/después; (c) ordenamiento de list[dict] por 2 criterios usando itemgetter.

Criterio de aceptación: memoize reduce Fibonacci(35) de segundos a milisegundos. Counter independiente entre instancias.

Errores comunes

Síntoma / mensaje	Causa y cómo arreglar
UnboundLocalError: local variable 'x' refe	Asignaste a x dentro de la función → Pytho
SyntaxError: positional argument follows k	Llamaste f(a=1, 2) — posicionales primero.
TypeError: f() got multiple values for arg	Pasaste x posicional Y nombrado: f(5, x=10
Mi decorador rompe help(funcion)	Sin @functools.wraps(fn), el wrapper pierd
Lambda en loop captura el último valor	[lambda: i for i in range(3)] — todas las

Preguntas frecuentes

¿Cuándo args, kwargs y cuándo argumentos explícitos?*

Argumentos explícitos siempre que conozcas la signatura — el IDE te ayuda y el lector entiende. args, *kwargs solo en wrappers genéricos (decoradores, factories) que deben aceptar cualquier llamada.

¿Lambda o def?

Lambda solo si: (a) cabe en una expresión, (b) la usas inmediatamente (callback), (c) un nombre no aportaría. En todos los demás casos, def con nombre — más debuggeable, soporta docstring y type hints.

¿Closure es lo mismo que decorador?

Decorador suele estar implementado con closure, pero closure ≠ decorador. Closure es cualquier función que captura su entorno; decorador es un patrón específico (función → función).

¿Por qué necesito nonlocal en make_counter?

Sin nonlocal, count += 1 dentro de inner se interpretaría como variable local nueva y daría UnboundLocalError. nonlocal le dice: 'esa variable vive en el scope inmediato exterior, modifícala'.

¿Cuál es el costo de pasar funciones como argumento?

Mínimo (es solo una referencia). Lo costoso es la invocación repetida en bucles tight (cada llamada Python tiene overhead). Para esto, NumPy/Cython/Numba.

Referencias

- Ramalho, Fluent Python 2e — caps. 7 y 9.
- Python docs — More on Defining Functions
- PEP 3102 — Keyword-Only Arguments

Siguiente clase

Clase 009 — Manejo de excepciones y context managers

Apéndice: notebook (primer bloque)

Primera celda ejecutable del notebook de la clase.

```
from operator import itemgetter, attrgetter
import time
from functools import wraps
```

Archivos complementarios

- notebook.ipynb