Programación Funcional

INSTITUTO DE INNOVACIÓN Y TECNOLOGÍA APLICADA

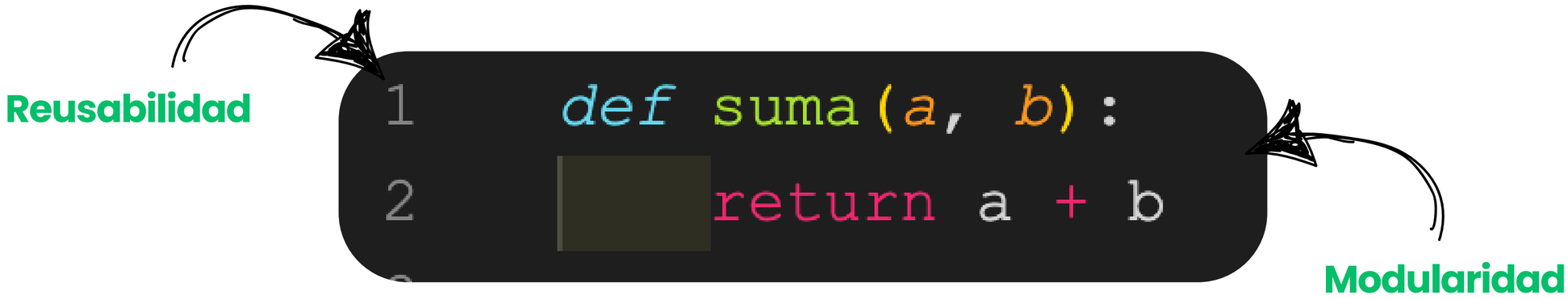
IITA

Contenido

¿Por qué son útiles las funciones?

PREGUNTA INICIAL

BREVE REPASO



Una función es un bloque de código **reutilizable** que realiza una tarea específica. Te permite dividir tu programa en partes más **pequeñas y manejables**, haciendo el código más modular, legible y fácil de mantener.

¿Qué es un Docstring?

PREGUNTA DE ENTREVISTA

DOCUMENTACIÓN

```python
def suma(a, b):
    return a + b
```

No sabes que espera recibir

```python
def sumar(a, b):
    """
    Suma dos números.

    Parámetros:
    a (int, float): El primer número.
    b (int, float): El segundo número.

    Retorna:
    int, float: La suma de los dos números.
    """
    return a + b
```

mejor, no?

Un **docstring** es una **cadena de texto** que aparece al inicio de una función (justo después de la declaración) y **explica qué hace la función**, cuáles son sus parámetros y qué devuelve. Es una práctica recomendada para documentar.

DOCUMENTACIÓN

guía que indica las **convenciones estilísticas** a seguir para escribir código Python

PEP 257

Estándar de Python para docstrings

Usa **comillas triples** incluso si la docstring tiene una sola línea.

La **primera línea** debe ser una **descripción breve**, seguida de una línea en blanco.

Detalla **parámetros y retornos** si la función no es trivial.

¿Qué es List Comprehension?

PREGUNTA DE ENTREVISTA

LIST COMPREHENSION

```python
cuadrados = [x**2 for x in range(5)]
print(cuadrados)
# Salida: [0, 1, 4, 9, 16]
```

Una list comprehension es una forma rápida y concisa de **crear una lista** en Python, en una sola línea de código.

¿Qué es un Generador?

PREGUNTA DE ENTREVISTA

GENERATORS

Cada vez que se encuentra un **yield**, la función **se pausa,** guarda su estado, y **espera** que la vuelvas a llamar.

Así puede **continuar desde donde se quedó**, en vez de empezar desde cero.

```python
def contar():
    yield 1
    yield 2
    yield 3

g = contar()

print(next(g))  # 1
print(next(g))  # 2
print(next(g))  # 3
```

Un **generator expression** se parece mucho a una list comprehension, pero **usa paréntesis** en lugar de corchetes y no crea toda la lista de una vez.

Un **generador** es una forma especial de crear funciones en Python que no devuelven todo de golpe, sino uno a uno, cuando se lo pedís. En vez de usar **return,** usan **yield**.

Cómo es la validación de los datos en python?

PREGUNTA DE ENTREVISTA

ANOTACIONES EN PYTHON

Ayudan a otros desarrolladores a comprender la **intención del código** sin necesidad de leer toda su implementación.

Documentan los **tipos esperados** de argumentos y valores de retorno.

```python
def suma(a: int, b: int) -> int:
    """Suma dos números enteros y retorna un entero."""
    return a + b


print(suma(7, 3))
```

Python no las valida en tiempo de ejecución

son una herramienta poderosa en Python para mejorar la **claridad, la documentación y el análisis del código**.

Veamoslo en código...

¿Qué son Args y Kwargs?

PREGUNTA DE ENTREVISTA

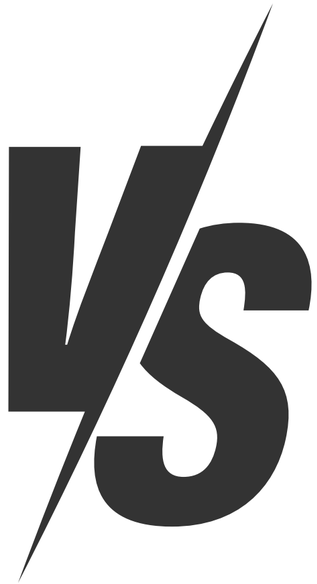
ARGS Y KWARGS

```python
def suma(*args):
    return sum(args)


print(suma(1, 2, 3))
print(suma(10, 20, 30, 40))
```

*Args

Permite pasar un número variable de **argumentos posicionales** a una función. Los argumentos se reciben como una tupla.




```python
def mostrar_datos(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")


mostrar_datos(nombre="Ana", edad=25)
```

**Kwargs

Permite pasar un número variable de argumentos con nombre (o keyword arguments). Los argumentos se reciben como un diccionario.

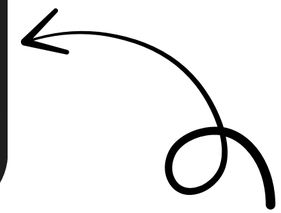
int, str, float

DESEMPAQUETAR EN PYTHON

```python
numeros = [1, 2, 3, 4, 5]

print(*numeros)

agenda = {"nombre": "Juan", "direccion": "Calle 123"}
nueva_agenda = {"apellido": "Perez", "telefono": "1234567"}
print(dict(**agenda, **nueva_agenda))
```



otra forma es
dict1 | dict2

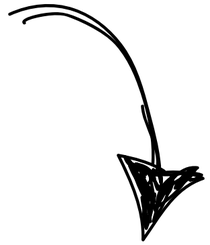
Desempaquetar en programación es el proceso de **extraer o separar los elementos** de una estructura de datos (como listas, tuplas o diccionarios) para usarlos individualmente.

¿Qué son las funciones Lambda o Anonimas?

PREGUNTA DE ENTREVISTA

FUNCIONES ANONIMAS

Solo pueden contener una única expresión.



```
lambda argumentos: expresión
```

No permiten **estructuras complejas** como bucles o múltiples sentencias

```
suma = lambda a, b: a + b
print(suma(2, 4))
```

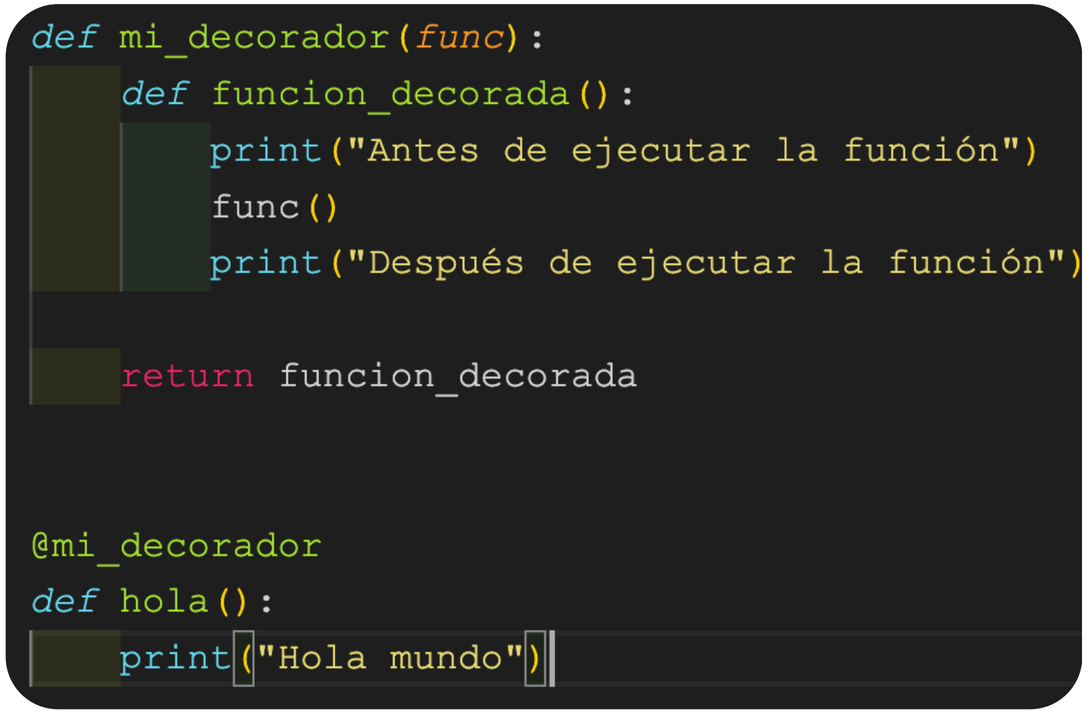
son herramientas compactas y útiles para definir **funciones sencillas de manera rápida**, especialmente cuando no necesitas reutilizar esa lógica en otro lugar.

¿Qué es un decorador? ¿Cómo funciona?

PREGUNTA DE ENTREVISTA

DECORADORES EN PYTHON

El @ es solo una forma más rápida de decirle a Python: **"Aplica este decorador a mi función antes de que sea ejecutada."**



En **Python**, los decoradores sirven para modificar o añadir funcionalidades a una función, sin necesidad de cambiar el código original de la función.

Veamoslo en código...

¿Qué son las primitivas funcionales?

PREGUNTA DE ENTREVISTA

PROGRAMACIÓN FUNCIONAL

Funciones como ciudadanos de primera clase

se pueden almacenar en variables, pasarse como argumentos

Ausencia de bucles tradicionales

se evita usar ciclos por map, filter, reduce

Inmutabilidad

los datos tienden a ser inmutables

Funciones puras

dado el mismo input, siempre devuelve el mismo output

La **programación funcional** es un paradigma de programación en el que **las funciones** son las principales herramientas para escribir código. Sin embargo, el concepto va más allá de simplemente "usar funciones", ya que implica seguir ciertos principios clave que distinguen a este paradigma de otros

PROGRAMACIÓN FUNCIONAL

Map

Aplica una función a cada elemento

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

filter

Filtra elementos de un iterable que cumplan con una condición

```python
numeros = [1, 2, 3, 4, 5]
pares = filter(lambda x: x % 2 == 0, numeros)
print(list(pares))  # [2, 4]
```

reduce

Aplica una función acumulativa a los elementos de un iterable

```python
from functools import reduce
numeros = [1, 2, 3, 4]
suma = reduce(lambda x, y: x + y, numeros)
print(suma)  # 10
```

IITA

¡Muchas Gracias!

Por su atención