Universidad Tecnológica de Pereira

Misión TIC 2022

El futuro digital es de todos

MinTIC

Programación Funcional Python

Funciones de Primera Clase, Lambda y Orden Superior

Hechos QUE CONECTAN

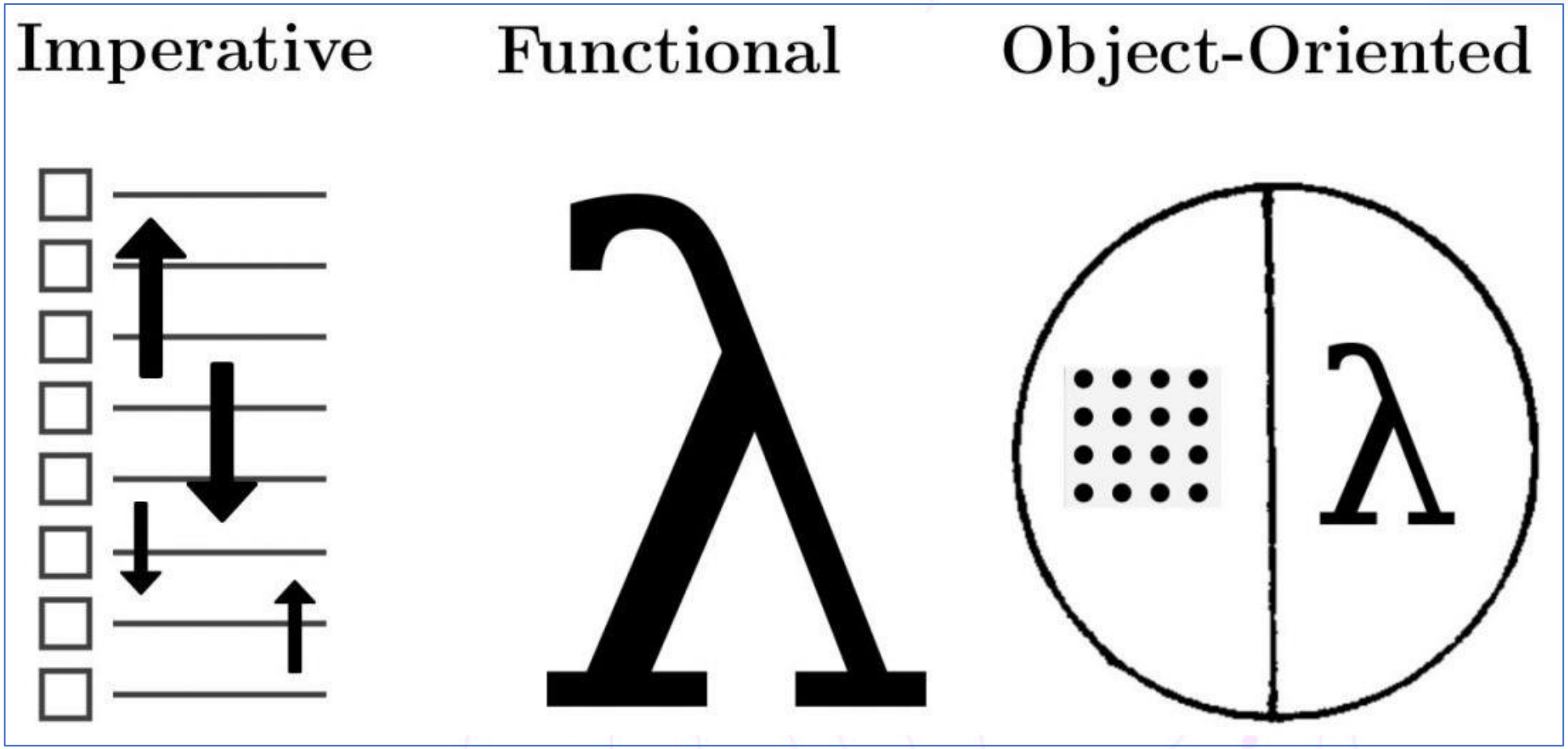
Introducción

- El Paradigma de Programación Funcional ha tomado fuerza en los últimos años, antes un enfoque académico, ahora una herramienta de desarrollo que disminuye las asignaciones o estados de las variables (facilita el debugging o depuración).

- Python es ***multiparadigma*** y se pueden combinar las comodidades del paradigma imperativo, con lo compacto del paradigma de programación funcional.

- En resumen, empalma la inmutabilidad de las funciones del paradigma funcional con los efectos colaterales de los paradigmas imperativos (procedural y orientado a objetos) para desarrollar soluciones de software.

Paradigmas de Programación



Paradigmas de Programación

Podemos desarrollar software utilizando cualquiera de los siguientes paradigmas:

- Paradigma Imperativo (cómo se hacen las cosas paso a paso)
  - Spaghetti (Nombre despectivo para código no modular)
  - Procedural/Estructurado
  - Orientado a Objetos
- Paradigma Declarativo (qué se debe hacer)
  - Lenguajes para bases de datos
  - Programación Lógica
  - Programación Funcional

Paradigma Híbrido: Imperativo + Funcional

Conceptos Programación Funcional

- Se dice que en un lenguaje (como es el caso de Python) las funciones son de primera clase (o que son "objetos de primera clase"), cuando se pueden tratar como cualquier otro valor del lenguaje, es decir, cuando se pueden almacenar en variables, pasar como parámetro y devolver desde funciones, sin ningún tratamiento especial.

- Cuando una función no recibe otras funciones como parámetro, se la denomina de primer orden.

- En el caso en el que sí las reciba, se llama de orden superior.

Paradigma Híbrido:
Imperativo + Funcional

Funciones para Colecciones de Datos

- Python ofrece unas funciones híbridas entre ambos paradigmas, muy versátiles para trabajar con grandes colecciones de datos, que son funciones de orden superior.
- Las funciones más utilizadas de este tipo, para realizar operaciones sobre listas principalmente, sin utilizar ciclos, al estilo del paradigma funcional, son las siguientes:
  - Map
  - Reduce
  - Filter
  - Zip

Paradigma Híbrido:
Imperativo + Funcional

Envoltura de Funciones en Python

- Algo interesante de las funciones en Python es que estas pueden ser asignadas a ***variables***.

- Las funciones pueden ser utilizadas como argumento de otras funciones.

- Las funciones pueden retornar otras funciones.

Envoltura de Funciones en Python

```python
def suma(val1=0, val2=0):
    return val1 + val2

def operacion(funcion, val1=0, val2=0):
    return funcion(val1, val2)

funcion_suma = suma
resultado = operacion(funcion_suma, 10, 20)
print(resultado)

```

Envoltura de Funciones en Python

- Función *suma* almacenada en la variable *funcion_suma*, variable que es utilizada como argumento en la función *operacion*.

- Esta función se encarga de ejecutar nuestro argumento y retorna el resultado de la operación.

- Con esto nuestra función *operacion* puede ser utilizada para ejecutar sumas, restas, multiplicaciones o cualquier tipo de operación que necesitemos. Esta función puede actuar como un ***wrapper***.

Ejemplo 2 Envoltura Funciones

```python
def crear_funcion(operador):
    if operador == '+':
        def suma(val1=0, val2=0):
            return val1 + val2
        return suma

def operacion(funcion, val1=0, val2=0):
    return funcion(val1, val2)

funcion_suma = crear_funcion('+')
resultado = operacion(funcion_suma, 10, 20)
print(resultado)
```

Ejemplo 2 Envoltura Funciones

- En esta ocasión definimos la función, *crear_funcion*.

- Esta función tiene la capacidad de crear nuevas funciones a partir del valor del parámetro.

Funciones Anónimas

- Habrá ocasiones en las cuales necesitemos crear funciones de manera rápida, en tiempo de ejecución.

- Funciones las cuales realizan una tarea en concreto, regularmente pequeña.

- En estos casos haremos uso de ***funciones lambda***.

Funciones Anónimas

```
lambda argumentos : cuerpo de la función
```

Ejemplos Funciones Anónimas

```
1  suma = lambda val1=0, val2=0: val1 + val2
2  operacion = lambda operacion, val1=0, val2=0 : operacion(val1, val2)
3
4  resultado = operacion(suma, 10, 20)
5
6  print(resultado)
```

Ejemplos Funciones Anónimas

```
1  sin_parametros = lambda : True
2
3  con_valores = lambda val, val1=10, val2=10 : val + val1 + val2
4
5  con_asterisco = lambda *args : args[0]
6
7  con_doble_asterisco = lambda **kwargs : args[0]
8
9  con_asteriscos = lambda *args , **kwargs : kwargs.get('key', False)
```