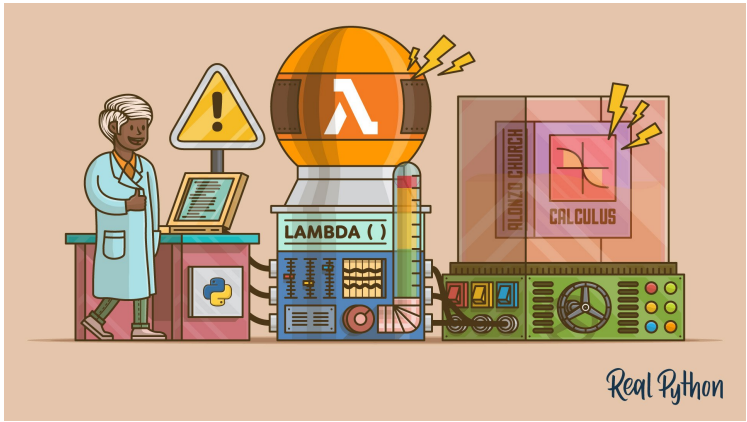
Lenguaje de Programación Python

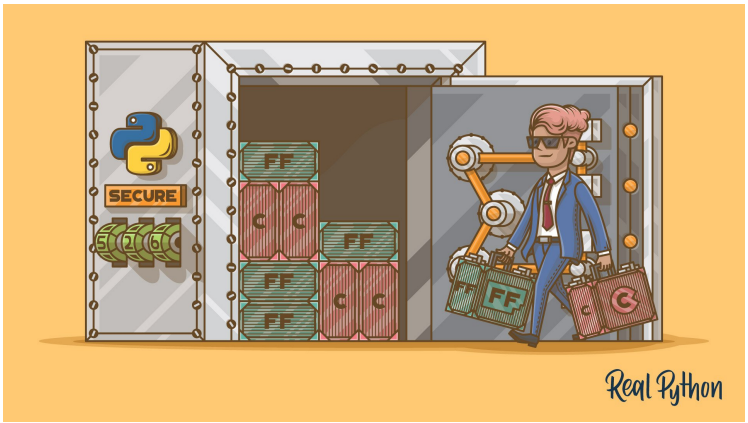
Soporte Funcional: Introducción

Dr. Mario Marcelo Berón
Argentina Programa
Universidad Nacional de San Luis

Programación Funcional

- Las funciones son *first class*. Todo lo que se puede hacer con los datos se puede hacer con funciones (Ej: pasar una función como parámetro de otra función).
- La *Recursión* es la estructura de control primaria.

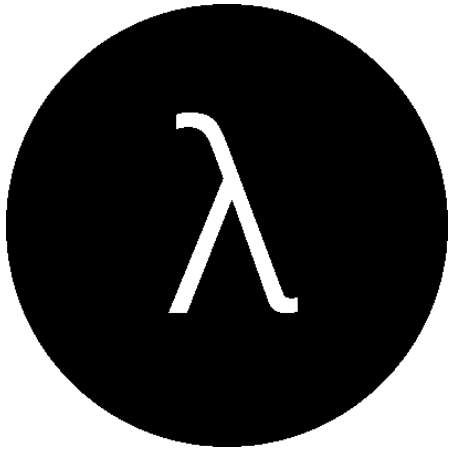


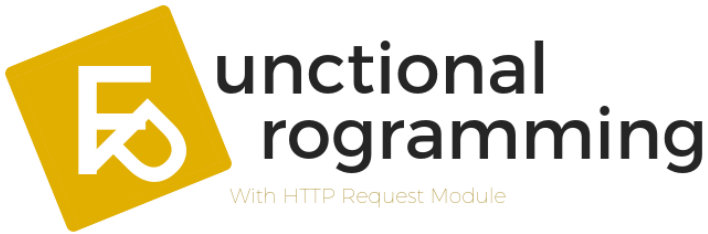


- Existe un foco en el procesamiento de listas. Las listas son frecuentemente usadas con recursión sobre sublistas como un sustituto de los loops.
- Los lenguajes funcionales puros evitan los efectos colaterales.

Programación Funcional

- La Programación Funcional trabaja con evaluación de expresiones.
- La recursión es una estructura de control primaria.

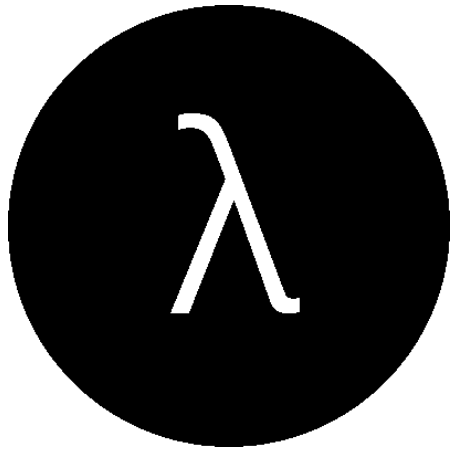


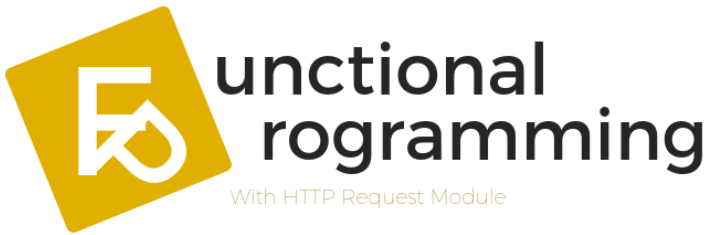


- El foco está en el procesamiento de listas.

Programación Funcional

- Los lenguajes funcionales puros evitan los efectos colaterales.
- La Programación Funcional se preocupa por el ***¿qué?*** antes que por el ***¿cómo?***.
- Desalienta el uso de sentencias en su lugar trabaja con la evaluación de expresiones.





- Utilizan High Order Functions es decir funciones que reciben funciones como parámetro y retornan funciones como resultado.

Programación Funcional

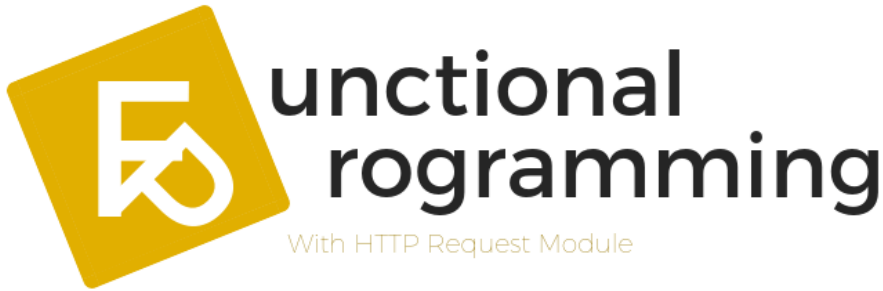
- El desarrollo es más rápido, más corto y menos propenso a errores.
- Es más fácil probar propiedades que en los lenguajes imperativos.

Python

- Los efectos colaterales están dispersos en los programas Python.
- Las variables se ligan varias veces.
- Las colecciones cambian su contenido.
- La entrada salida está entrelazada con la computación.

Python... No es un Lenguaje Funcional

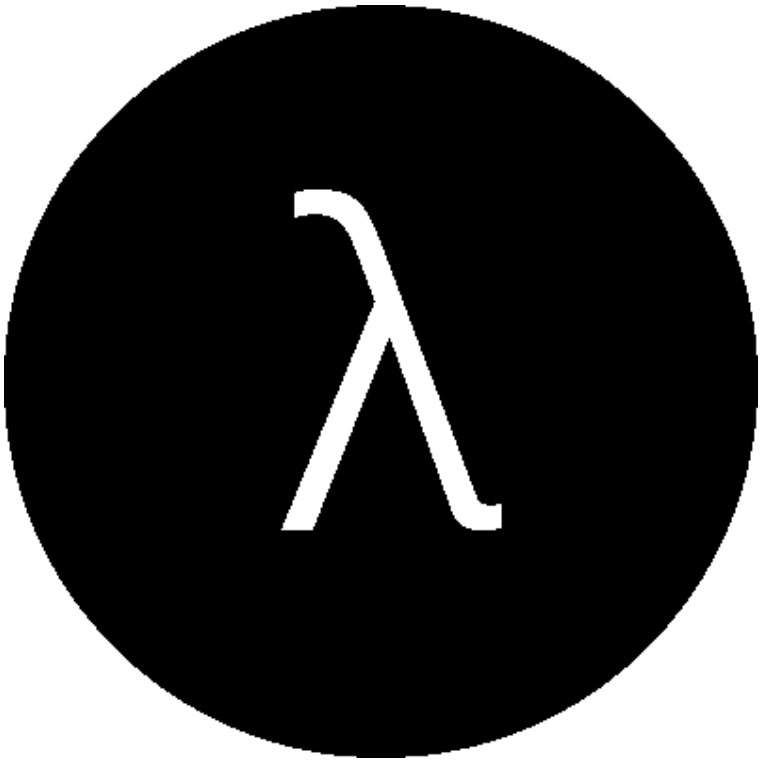
Programación Funcional



Python

Python es: un lenguaje con soporte *Multiparadigma* que permite usar conceptos de programación funcional cuando el programador lo desea y los puede mezclar con otros estilos de programación.

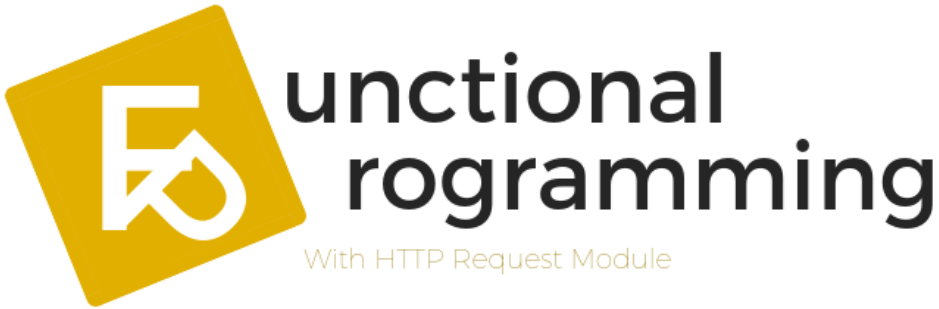
Programación Funcional



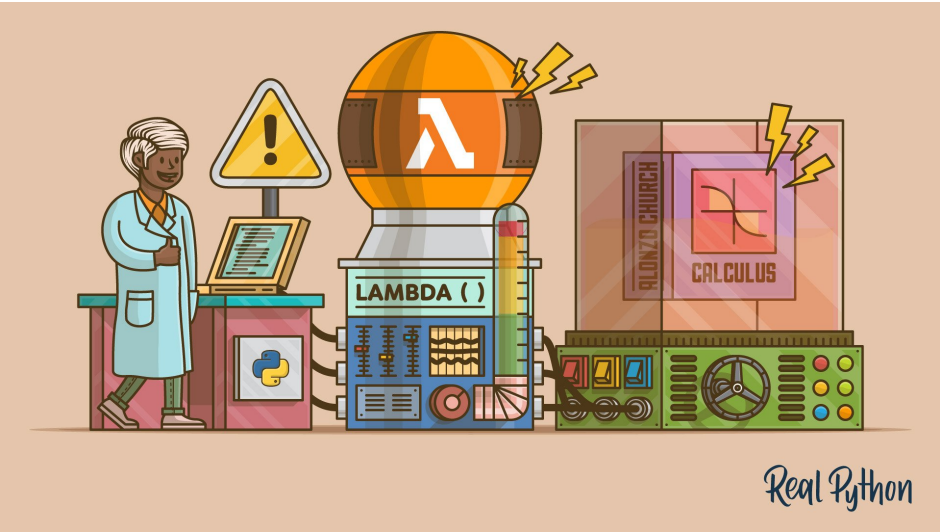
- En lenguajes imperativos y orientados a objetos el código consiste de clases y métodos que contienen sentencias while, for, etc.
- Las setencias modifican las estructuras de datos tales como listas, diccionarios, etc.
- Lo anterior parece natural pero los problemas surgen con los efectos colaterales.

Programación Funcional

- Las variables y estructuras de datos modelan la realidad pero no permiten razonar precisamente respecto del estado de los datos en un punto específico del programa.
- Una solución consiste en no focalizar en la construcción del dato sino en *Qué* consiste la colección de datos.



Programación Funcional



- Cuando se piensa en *En qué se necesita hacer con los datos?* antes que *Cómo construirlos?* es posible un razonamiento más directo.
- El flujo de control imperativo enfatiza el *Cómo?* y no el *Qué?*

Encapsulación

Importante

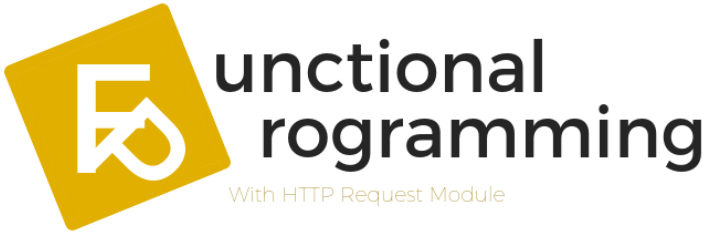
Una forma simple de focalizar más en el *Qué?* que en el *Cómo?* consiste en refactorizar el código y colocar la construcción de los datos en un lugar más aislado.

Encapsulación

Ejemplo - Sin Encapsular

```
colección = obtenerEstadoInicial()
estadoVariable = None
for dato in conjuntoDato:
 if condición(estadoVariable):
  estadoVariable = calcularDeEste(dato)
  nuevo = modificar(dato, estadoVariable)
  colección.add_to(nuevo)
 else:
  nuevo = modificarDiferente(dato)
  colección.add_to(nuevo)
```

Encapsulación



Ejemplo - Sin Encapsular

```
for cosa in colección:
  procesar(cosa)
```

Encapsulación

Ejemplo

```
def crearColección(conjuntoDeDato):
 colección = obtenerEstadoInicial()
 estadoVariable = None
 for dato in conjuntoDeDato:
  if condición(estadoVariable):
   estadoVariable= calcular(deste, estadoVariable)
   nuevo = modificar(dato, estadoVariable)
   coleción.add_to(nuevo)
```

Encapsulación

unctional
rogramming
With HTTP Request Module

Ejemplo

```
    else:
      nuevo = modificarDiferente(dato)
      colección.add_to(nuevo)
  return colección
for cosa in crearColección(conjuntoDeDato):
  procesar(cosa)
```