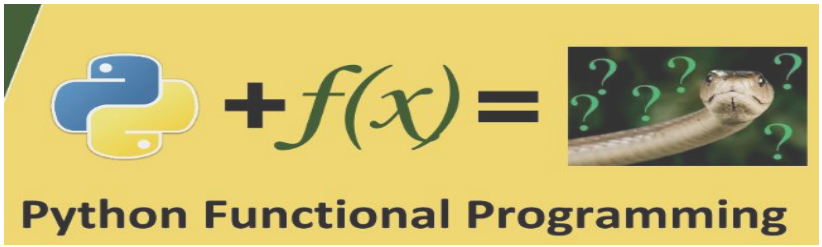
Python Básico

Programación funcional

Enero 2018

Paradigma funcional

El paradigma de programación funcional es uno de los fundamentales entre los llamados de programación declarativa. La programación funcional tiene sus raíces en el cálculo lambda, un sistema formal desarrollado en los años 1930 para investigar la definición de función, la aplicación de las funciones y la recursión.



Características

- Ausencia de efectos colaterales, todas las variables son inmutables
- El valor de una expresión sólo depende de los valores de sus subexpresiones, si las tiene.

```
resultado = func1(a) + func2(a)
```

*Podríamos ejecutar func1 y func2 en paralelo porque sabemos que "a" no se va a modificar

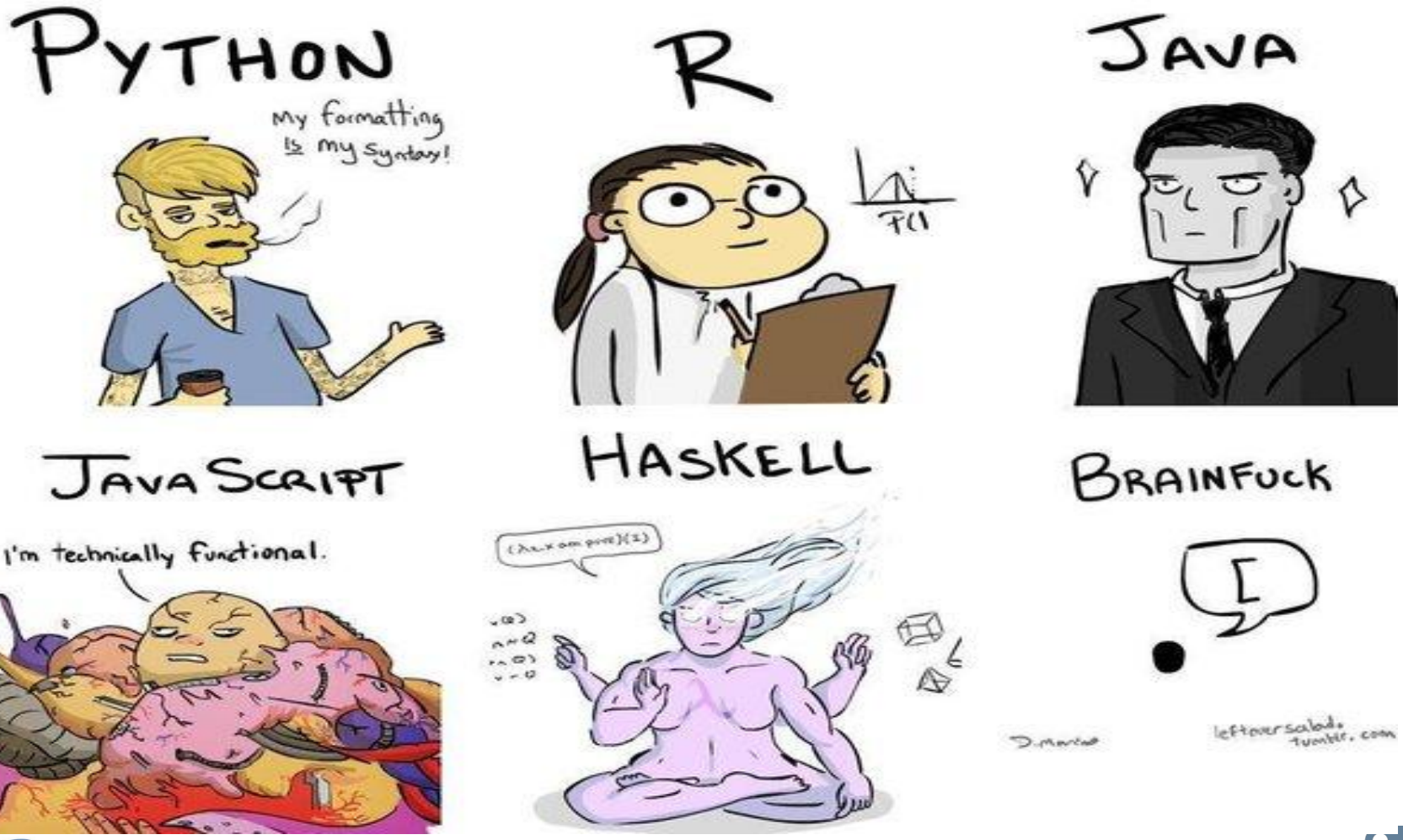
Características

- Funciones de primera clase y de orden superior
- Funciones puras
- Recursividad
- Evaluación estricta frente a la no estricta
- Sistemas de tipos

Ventajas

- El orden de ejecución de las funciones está libre de efectos secundarios para permitir la simultaneidad (procesamiento en paralelo).
- Y debido a que las funciones en lenguajes funcionales comportan de manera muy similar a las funciones matemáticas, es muy fácil de traducir, aquellas a los lenguajes funcionales. En algunos casos, esto puede hacer que el código sea más legible.

Diferencia con otros lenguajes

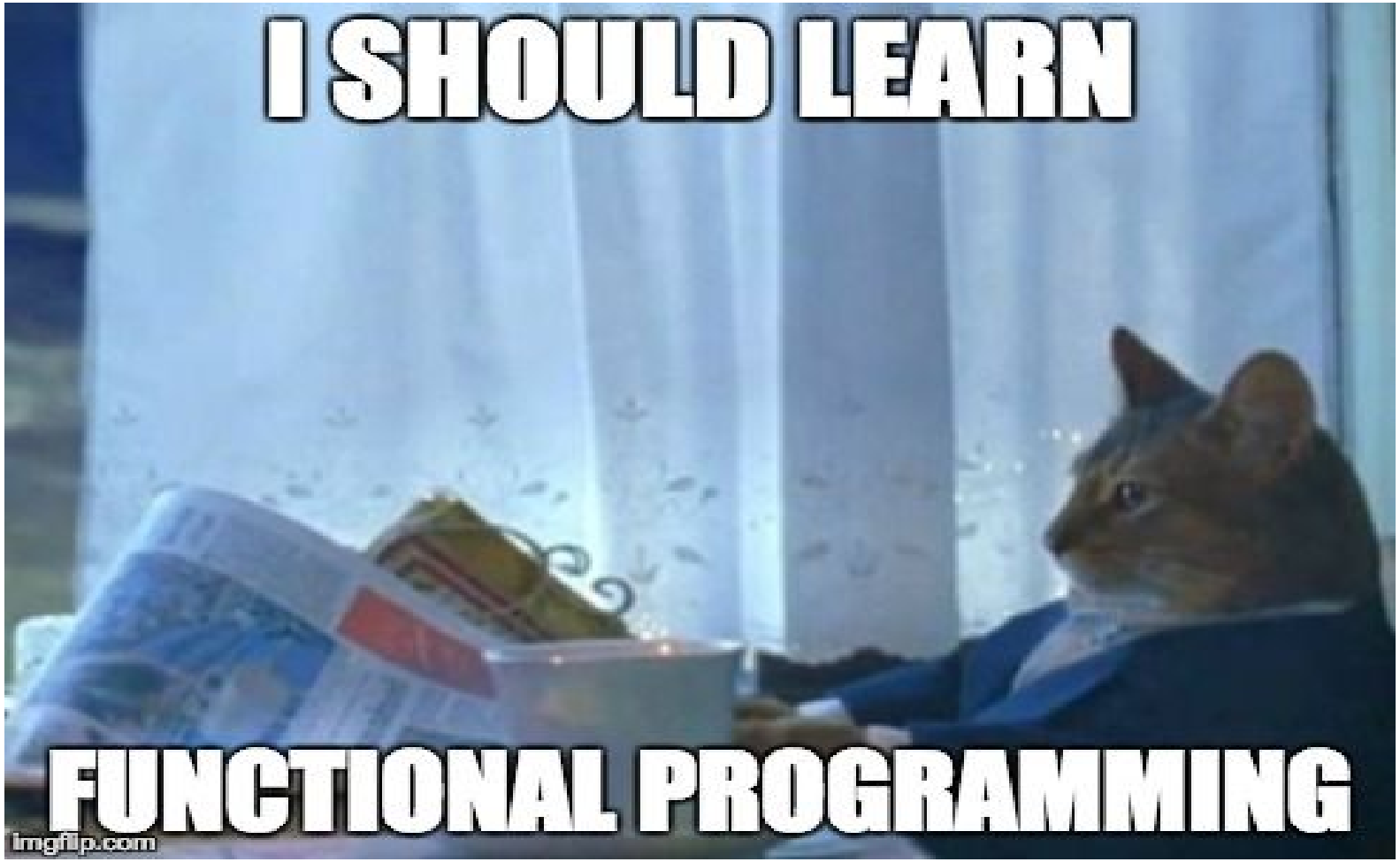


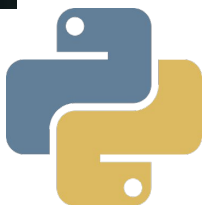
Ventajas?

Quizás no se vea ahora el impacto de esta forma de programar, pero ayuda a ciertas cosas:

- Una forma de encontrar nuevas abstracciones (funciones superiores) que a su vez… permiten reutilizar código.
- Separar la lógica por niveles y así tener funciones "incompletas" en cuanto a que les falta una parte de su comportamiento que se completará con parámetros. Es decir que estamos parametrizando las funciones.

Así que…





Buenas Prácticas

Para asegurarse de que estamos usando las bases de este paradigma podemos seguir estas buenas prácticas:

- Todas las funciones al menos deben recibir un argumento
- Todas las funciones deben regresar un dato u otra función
- ¡No hay loops!

Funciones

Son un conjunto de líneas de código que realizan una tarea específica y puede retornar un valor. Además las funciones pueden recibir parámetros/argumentos que modifiquen su funcionamiento.

Argumentos de una función

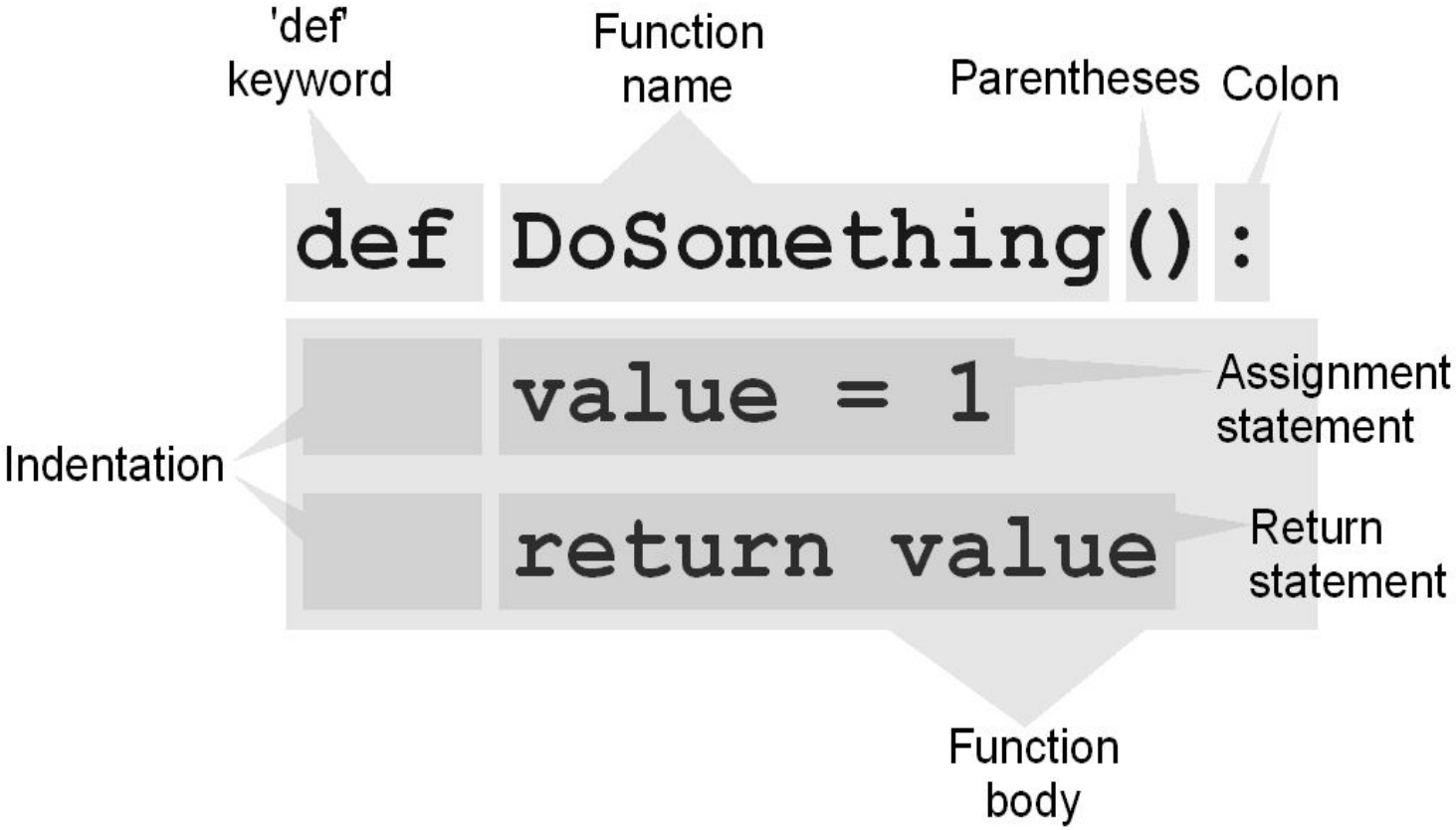
Un argumento es un valor que la función espera recibir cuando sea llamada (invocada), a fin de ejecutar acciones en base al mismo. Una función puede esperar uno o más argumentos (que irán separados por una coma) o ninguno.

Argumentos de una función

Los argumentos, se indican entre los paréntesis, a modo de variables, a fin de poder utilizarlos como tales, dentro de la misma función.

```
def ejemplo(nombre,apellido):
    #Código
```

Estructura de una función



Argumentos posicionales

Estos son los parámetros que se tienen que pasar a la función si queremos que funcione debidamente, en caso de que llegue a faltar alguno tendremos un error.

```
def func( a , b ):
    -cuerpo de la función
```

Si llamamos la función así func(4) nos indica que nos falta un parámetro posicional.

```
TypeError: func() missing 1 required positional argument: 'b'
```

Ejercicio 1

Realice una función que me solicite el nombre completo de una persona, además sus datos como son: dirección, teléfono casa, teléfono celular, #cuenta o RFC.
Y que me retorne una lista con todos los datos.

Ejercicio 2

Del ejercicio anterior, mande la lista retornada a otra función que se llame “formato()” esta recibirá la lista y la imprimirá de la siguiente manera:

```
0 Jorge
1 Chávez
2 Delgado
3 Zapata #15
4 123456789
5 987654321
6 1
```

Lambdas

Funciones de una sola línea también funciones anónimas
Su sintaxis es la siguiente:

funcion=(**lambda** args : **operación**)

En la operación no puede haber más que funciones, no podemos usar for o while.

Ejemplo:

cubo=(**lambda** x : **x**3**)

Funciones de alto orden: map, filter y reduce

- Estas funciones nos permiten pasar una función completamente sobre todos los elementos de una lista
- Regresan una lista nueva sin afectar la que usemos

- Su sintaxis es la siguiente:

  funcionAltoOrden(**función**, **lista**)

- **función** puede ser una lambda

*funcionAltoOrden=map,filter,reduce o zip

Funciones de alto orden: map, filter y reduce

- Estas funciones sufren un cambio en versiones posteriores a python 2.7
- **map( )** y **filter( )** regresan iteradores, pueden castearse a listas así: **list(map(…))**
- **reduce( )** tiene que importarse desde el módulo **functools**

functools.reduce(…)

Listas por Comprensión

Forma potente de crear listas

- Aplica una función a cada elemento de la lista
- Su sintaxis es la siguiente:

- [**expresión** for **nombre** in **lista**]

- La **expresión** debe ser capaz de operar sobre toda la **lista**

- Si **lista** (puede ser un generador) contiene objetos indexables entonces **nombre** debe de tener un patrón que "empate" con sus elementos.

Generadores

- Forma potente de crear iteradores
- Como una función pero usan **yield** en vez de **return**
- Cada vez que se llama a **__next__( )** el generador continua donde se quedó
- Permanece un estado de las variables en la memoria, y la última línea que se ejecutó.

Decoradores

- Funciones que toman como parámetro una función y regresan una función
- Crea un "envoltura" alrededor de la función a decorar.

- Se declaran como funciones **@decorador** y se coloca esta etiqueta sobre la función a decorar.
- Cambian el comportamiento de un función o clase sin modificarla.

Ligas de interés

- Documentación más extensa
- https://docs.python.org/3/howto/functional.html
- ¿Qué define a un lenguaje funcional?
- https://clojurefun.wordpress.com/2012/08/27/what-defines-a-functional-programming-language/
- ¿Porqué python no es muy bueno para Programación Funcional?
- http://stackoverflow.com/questions/1017621/why-isnt-python-very-good-for-functional-programming