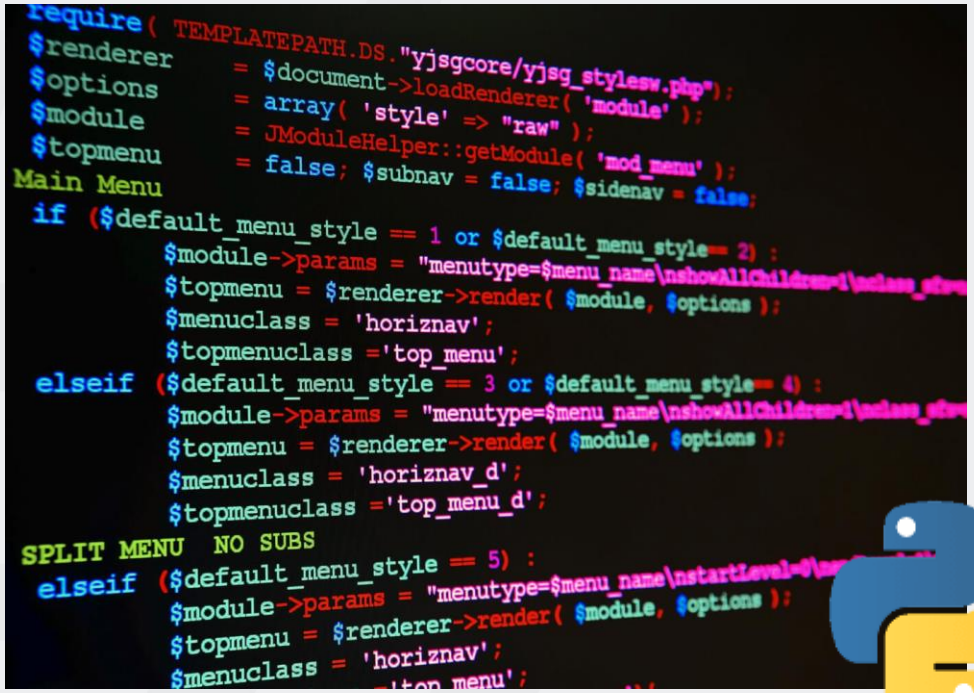


python

Functional Python

Ing. Juan Camilo Correa Chica

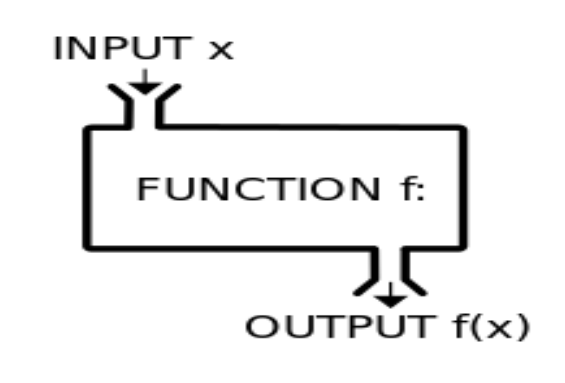
Functional Python

- ¿Qué es el paradigma de Programación Funcional?
- Programación funcional en Python
- Ejemplos

¿Qué es el paradigma de programación funcional?

Los dos paradigmas de programación de software más conocidos son el paradigma procedimental (procedural) característico de lenguajes de programación como C y el Shell de Linux (Unix); y el paradigma de programación orientada a objetos (OOP) característico de lenguajes como C++ y Java. Python es un lenguaje de programación multi-paradigma, con Python se pueden escribir programas y librerías que pueden contener partes procedimentales, orientadas a objetos o "funcionales".

La programación funcional recibe su nombre del hecho de que en este paradigma la característica principal es que la lógica en los programas se describe a través de la implementación de funciones, cada función responde a la descomposición de un problema en distintas características y funcionalidades. Se busca en general la implementación de funciones que reciben "entradas" y producen "salidas" (funciones con parámetros de entrada y valor de retorno) y que no tienen un estado interno que pueda afectar la salida producida por una entrada determinada.



Características de la programación funcional

La programación funcional evita el "estilo imperativo" característico de los paradigmas procedimentales y orientado a objetos (descripción paso a paso del cambio de estado desde el inicio hasta el resultado) y por el contrario fomenta un "estilo funcional" en el que se aplican transformaciones y se basa en composición de funciones.

```
#Imperative style: actions that change state from initial state to result

expr, res = "28+32+++32++39", 0
for t in expr.split("+"):
    if t != "":
        res += int(t)
print(res)

#Functional style: apply transformation (and compositions)

from operator import add

expr = "28+32+++32++39"
print(reduce(add, map(int, filter(bool, expr.split("+")))))
```

Características de la programación funcional

- La implementación de la lógica se logra solo a través de funciones.
- La programación funcional requiere que las funciones no tengan estados y que dependan solo en las entradas para producir salidas (concepto de funciones puras).
- El beneficio de utilizar funciones “puras” es la reducción de “efectos colaterales”.
- Los “efectos colaterales” se dan cuando se producen cambios dentro de la operación de una función que están fuera de su alcance (scope).
- La programación funcional disuade del uso de funciones con “efectos colaterales” que modifican el estado interno o hacen cambios que no se visualizan explícitamente en el valor de retorno de la función.
- La programación funcional evita la implementación de la lógica como una secuencia de cambios de estado y promueve la implementación de la lógica como datos que van fluyendo entre funciones.
- Las funciones que no generan “efectos colaterales” se consideran “funcionalmente puras”, y en general son funciones que no recurren al uso de estructuras de datos que puedan ser actualizadas conforme el programa se ejecuta; y además su retorno depende estrictamente de las entradas de la función.

Funciones "funcionalmente puras"

```
#Functional purity

#Pure:
def is_interesting(topic):
    return topic.contains("FP")

#Not Pure:
def speak(topic):
    print topic

#Quite impure:
def set_talk(speaker, topic):
    speaker["talk"] = topic
    return speaker
```

Ventajas teóricas y prácticas del estilo funcional:

- Susceptibilidad a prueba formal: el estilo funcional facilita el proceso de probar matemáticamente la exactitud de una función. Es decir, si se define un conjunto de invariantes se espera que ese conjunto permanezca inalterado antes y después de la ejecución del programa.

- Modularidad: Pequeñas funciones que realizan procedimientos simples, facilitando la legibilidad del código y el chequeo de errores.

- Facilidad de depuración y prueba: La depuración y evaluación de funciones simples se facilita ya que solo se necesita confeccionar las entradas adecuadas para la función y se espera que los retornos sean los esperados (se evita la intervención de eventos externos).

- Composición: A pesar de que se definen funciones especializadas, es posible utilizar funciones y utilidades para ensamblar nuevos programas.

- Paralelización: Los programas funcionales permiten correr el mismo código sin necesidad de sincronización y varios procesos concurrentes se pueden ejecutar en procesadores diferentes.

- Evaluación perezosa: Es una técnica de compilación que evita correr funciones solo hasta que su resultado sea necesario.

- Determinismo: Luego de repetidas ejecuciones de un programa funcional (para el mismo conjunto de entradas claro está) se obtienen resultados iguales.

Propiedades y requerimientos de los programas funcionales

Características del lenguaje de programación que ayudan a la programación funcional:

- Inmutabilidad de los datos: Datos que no pueden ser cambiados.
- Funciones “first-class”: Se dice que un lenguaje tiene funciones “first-class” cuando permite el paso de funciones como parámetros de entrada, retornar funciones o guardar funciones en estructuras de datos.
- Funciones “high-order”: Son funciones que reciben una o más funciones como parámetros de entrada y retornan una función.
- Optimización “tail call”: Se refiere a la reutilización de la misma porción del “stack” para una única secuencia de llamadas recursivas de una misma función. (Ausente en Python)

Técnicas de programación para escribir código funcional:

- Mapping: Evitar el uso de bucles iterativos mediante el mapeo de una función a los elementos de un iterable.
- Reducing: Evitar el uso de bucles iterativos en operaciones de acumulación sobre los elementos de un iterable.
- Pipelining: Encadenamiento de la ejecución de un programa mediante el paso de la salida de una función como entrada a la siguiente función y así en adelante.
- Recursing: Habilidad de una función para poder invocarse a sí misma, permite dividir un problema en varios sub-problemas de la misma naturaleza que pueden ser solucionados por la misma función.
- Currying:

Propiedades y requerimientos de los programas funcionales

El estilo funcional de programación requiere de funciones conocidas como de “high-order”, son funciones que a las que se les puede pasar otras funciones como parámetros de entrada y también pueden retornar funciones. Un importante conjunto de funciones “high-order” incluidas en Python son aquellas que toman como parámetros de entrada una función y un iterable, de modo que la función es aplicada a cada uno de los elementos del iterable. Por lo general estas funciones tienen la forma general:

function_name(function_to_apply, iterable_of_elements)

La programación funcional en Python se basa en tres componentes fundamentales:

1. Operadores: Funciones propias del lenguaje que reemplazan las operaciones y operadores estándar.
2. Iteradores: Son objetos que representan a un flujo de datos contiguos y retornan un único dato a la vez.
3. Generadores: Son funciones que en general simplifican la tarea de escribir iterables.

Para la declaración de funciones sencillas, la programación funcional prefiere el uso de “declaraciones lambda” y evitar el uso de la sintaxis con “def”.

```
custom_function = lambda x,y:x+y
z = custom_function(3,5)
```

Funciones para implementar el estilo funcional usando Python

Map: Esta función toma como parámetros de entrada un iterable y una función. La función retorna un objeto especial de tipo "map" al cual se le debe hacer el "casting" adecuado para obtener otro iterable en el cual los elementos son los mismos del iterable de entrada pero luego de ser transformados por la función de entrada.

```
nums = [1,2,3,4,5,6,7]
suma = list(map(lambda x:x+10, nums)) #casting is needed
print(nums) # -> [11,12,13,14,15,16,17]
```

Filter: Esta función aplica a un iterable una función booleana que retorna "True" o "False". La función retorna un iterable con los elementos del iterable de entrada que retornan "True" luego de ejecutar la función booleana.

```
squares = list(map(lambda x: x**2, range(10)))
special_squares = list(filter(lambda x: x > 5 and x < 50, squares))
print(special_squares) #-> [9, 16, 25, 36, 49]
```

Funciones para implementar el estilo funcional usando Python

Reduce: Esta función reduce el iterable de entrada a un único valor de salida luego de ejecutar algún calculo u operación sobre un acumulador y los elementos del iterable. Las funciones lambda utilizadas en la función **reduce** tienen dos parámetros de entrada, el primero es un acumulador y el otro recibe el valor del elemento que se está iterando.

```
from functools import reduce

product = reduce((lambda a, x: a * x), [1, 2, 3, 4])
print(product) #-> 24
```

Any: Esta función retorna "True" si al menos un elemento en el iterable evalúa una condición como verdadera. Retorna "False" cuando todos los elementos evalúan la condición como falsa o si el iterable está vacío.

```
lista = [14, 34, 2, 7, 8, 44, 65]
print(any((i < 10) for i in lista)) #-> True
print(any(not(i % 3) for i in lista)) #-> False
```

Funciones para implementar el estilo funcional usando Python

All: Esta función retorna "True" si todos los elementos en el iterable evalúan una condición como verdadera. Retorna "False" cuando al menos un elemento evalúa la condición como falsa o si el iterable está vacío.

```
leest = ['1','2','3,'4','5','6','7']
print(all((i < 5) for int(i) in leest)) #-> False
print(all((i > 0) for int(i) in leest)) #-> True
```

List comprehensions: Son procedimientos que proporcionan una forma concisa para la creación de listas de un modo más "pythonico", es decir, evitando el uso de las funciones map y filter. Las sintaxis de una "list comprehension" está delimitada por corchetes "[ ]" para indicar que el resultado será una lista. Y como valores de entrada tienen una expresión a evaluar, un bucle iterativo que proporciona los datos que evaluaran la expresión y opcionalmente pueden agregarse otros bucles y sentencias condicionales.

```
squares = [i**2 for i in range(10)]
special_squares = [i**2 for i in range(10) if i**2 > 5 and i**2 < 50]
```

Funciones para implementar el estilo funcional usando Python

Partial functions: Es una importante funcionalidad que proporciona Python para implementar nuevas funciones a partir de otras mediante el "silenciamiento" de los parámetros de entrada. En general, se implementa una nueva función con un número inferior de parámetros de entrada al asignar valores por defecto a los parámetros "silenciados".

```
from functools import partial

def add(a, b):
    return a + b

add_two = partial(add, 2)
add_ten = partial(add, 10)

print(add_two(4)) #-> 6
print(add_ten(4)) #-> 14
```

Mejorando el estilo funcional....

Conforme se gana más experiencia en la programación funcional se puede cada vez ir mejorando la implementación del estilo funcional en los programas.

```
#Bad
ss = ["UA", "PyCon", "2018"]
reduce(lambda acc, s: acc + len(s), ss, 0)

#Not Bad
ss = ["UA", "PyCon", "2018"]
reduce(lambda l,r: l+r, map(lambda s: len(s), ss))

#Good
ss = ["UA", "PyCon", "2018"]
reduce(operator.add, map(len, ss))
```

Ejercicios prácticos

1. Dados 3 enteros X, Y, Z que representan las dimensiones de un tetraedro. Y dado un número N. Imprima todas las posibles coordenadas dadas por (i,j,k) en una cuadricula tridimensional donde la suma de i+j+k no es igual a N. De modo que: $0 \leq i \leq X$; $0 \leq j \leq Y$; $0 \leq k \leq Z$

2. Dada una lista de enteros separados por espacio. Determine si todos los números de la lista son positivos y si todos los números de la lista son números "palidromos"

3. Dada la siguiente lista de strings cuente cuántas veces se repite la palabra "Python" en toda la lista.

```
leest = ["We are learning Python", "Functional programming in Python", "What are this Python functions for?", "Do we really need Python?", "Python rulez!"]
```

4. ¿Cuál es la suma de los primeros 50 números positivos cuyo cuadrado es divisible por 5?

5. Imprima una lista con los cubos de los primeros 30 números de Fibonacci

6. Suponga que recibe cadenas de caracteres de la siguiente forma: '12223311', donde un carácter 'c' aparece consecutivamente x veces. Reemplace las ocurrencias consecutivas con tuplas de estilo (x, c), donde x es el número de ocurrencias y c el carácter. (Sugerencia: groupby)

Ejm: 12223311 -> (1,1)(3,2)(2,3)(2,1)

Enlaces útiles Functional Programming

https://docs.python.org/3/howto/functional.html
https://www.programiz.com/python-programming/list-comprehension
https://www.pythonforbeginners.com/basics/list-comprehensions-in-python
https://python-reference.readthedocs.io/en/latest/docs/functions/reduce.html
https://docs.python.org/3/library/functools.html
http://book.pythontips.com/en/latest/map_filter.html
http://www.u.arizona.edu/~erdmann/mse350/topics/list_comprehensions.html
https://maryrosecook.com/blog/post/a-practical-introduction-to-functional-programming
https://hackernoon.com/functional-programming-with-python-for-people-without-time-1eebdbd9526c
https://www.dataquest.io/blog/introduction-functional-programming-python/
https://docs.python.org/3/library/functional.html
https://hackernoon.com/10-data-structure-algorithms-and-programming-courses-to-crack-any-coding-interview-e1c50b30b927
https://hackernoon.com/learn-functional-python-in-10-minutes-to-2d1651dece6f
https://www.programiz.com/python-programming/methods/built-in/any
https://www.programiz.com/python-programming/methods/built-in/all
https://stackoverflow.com/questions/19389490/how-do-pythons-any-and-all-functions-work
https://github.com/sfermigier/awesome-functional-python