Introducción a la programación

Usando Python

G. Sebastián Pedersen

Instituto de Industria
Universidad Nacional de General Sarmiento

Matemática para Economistas III, 1er. cuat. 2019

`https://sebasped.github.io/pythonungs/`

Repaso clase anterior

Evitar duplicar información

```
# m2k_v2.py
# Ingreso las millas (en float para no hacer división entera)
mills = 34.122

# Hago la conversión a kilómetros
kms = mills*1.6

# Imprimo por pantalla el resultado
print mills,"millas son",kms,"kilómetros"
```

- Duplicación de información corregida: 34.122 lo estoy guardando en mills, y después uso la variable mills y no vuelvo a poner 34.112 en kms

Repaso clase anterior

La última versión usaba el ciclo `while` para que el programa funcione para listas de *cualquier longitud*.

```
# m2k_v8.py
import matplotlib.pyplot as plt

# Ingreso las millas
mills = [1.1, 33.4, 13.2, 60.0, 17.35]  # la ingreso float para no hacer división entera

# guardo la longitud de la lista
long = len(mills)

# Hago la conversión a kilómetros
kms = [0]*long  # lista de ceros de longitud adecuada

i = 0 # variable para posición en la lista
while i < long : # mientras que la posición no se salga de la lista
    kms[i] = mills[i]*1.6  # convertir
    i = i + 1  # avanzar una posición

# Imprimo el resultado en un gráfico
plt.plot(mills,kms,'*')
plt.show()
```

Particionando el problema y encapsulando en funciones

¿Qué hace el siguiente programa?

```
# m2k_v9.py
import matplotlib.pyplot as plt

def convertir(lista):
   long = len(lista)  #guardo la longitud de la lista

    # Hago la conversión
    listaConvertida = [0]*long  # lista de ceros de longitud adecuada
    i = 0
    while i < long :
        listaConvertida[i] = lista[i]*1.6
        i = i + 1

    return listaConvertida

# El resto del código que más prolijo
mills = [1.1, 33.4, 13.2, 60.0, 17.35]

kms = convertir(mills)

plt.plot(mills,kms,'*')
plt.show()
```

- La función `convertir` encapsula el problema de la conversión a kms.
- El resto del código encapsula el problema de usar la conversión y graficar el resultado.
- Ventajas de esto:
  - Particionamos el problema es dos problemas de menor complejidad.
  - Los dos problemas pueden sen resueltos por diferentes personas y al mismo tiempo.

Particionando el problema y encapsulando en funciones

¿Cuál es la diferencia con el anterior programa?

```
# m2k_v10.py
import matplotlib.pyplot as plt

def convertir(lista,factor):
    long = len(lista)  #guardo la longitud de la lista

    # Hago la conversión
    listaConvertida = [0]*long  # lista de ceros de longitud adecuada
    i = 0
    while i < long :
        listaConvertida[i] = lista[i]*factor
        i = i + 1

    return listaConvertida

# El resto del código que más prolijo
mills = [1.1, 33.4, 13.2, 60.0, 17.35]

fc = 1.6
kms = convertir(mills,fc)

plt.plot(mills,kms,'*')
plt.show()
```

- La función `convertir` ahora toma dos parámetros de entrada.
- La ventaja de esto es que ahora la función `convertir` funciona para cualquier lista y para cualquier factor de conversión.

¡Hacking time!

Tener presente que:

- Cualquier *valor que quiera ser reutilizado* (para un cálculo posterior, para una salida, etc.) debe ser almacenado previamente en una variable.
- Siempre que sea posible no duplicar información en el código.
- Siempre que sea posible particionar el problema y encapsularlo en problemas menos complejos mediante funciones.

Ejercicios:

1. Escribir un script en un archivo d2p_v3.py. El programa tiene que tomar como entrada una lista de dólares, y convertirla a una lista de sus equivalentes en pesos (tomar algún tipo de cambio).
   - El programa tiene que funcionar para listas de cualquier longitud.
   - El programa debe utilizar una función que realice la conversión.
   - La salida del programa tiene que ser un gráfico Dólares vs. Pesos.
2. Modificar el script del punto 1. y guardarlo en un archivo d2p_v4.py. Ahora la función que hace la conversión debe también funcionar para cualquier tipo de cambio.