Introducción a Python

Clase 3

Mateo Suster
mateosuster@gmail.com

Matemática para Economistas III
Instituto de Industria
Universidad Nacional de General Sarmiento

29 de octubre de 2021

¿Qué hace el siguiente programa en Python?

```
1  #ingreso las millas
2  mills = 34.122
3
4  #hago la conversion a kilometros
5  km = mills * 1.6
6
7  km_redondo = round(km, 2)
8
9  #imprimo por pantalla el resultado
10 print(mills, "millas son", km_redondo, "kilometros")
11
12 #la salida queda
13 > 34.122 millas son 54.59 kilometros
```

¿Qué hace el siguiente programa en Python?

```
1  #ingreso las millas
2  mills = 34.122
3
4  #hago la conversion a kilometros
5  km = mills * 1.6
6
7  km_redondo = round(km, 2)
8
9  #imprimo por pantalla el resultado
10 print(mills, "millas son", km_redondo, "kilometros")
11
12 #la salida queda
13 > 34.122 millas son 54.59 kilometros
```

- Un programa sencillo de conversión, que a esta altura nos debe resultar familiar

El mismo programa, pero con listas

```
1  #ingreso las millas
2  mills = [34.122, 17.588, 3.187]
3
4  # Hago la conversion a kilometros
5  km = [mills[0]*1.6, mills[1]*1.6, mills[2]*1.6]
6
7  # Imprimo por pantalla el resultado
8  print(mills[0],"millas son",km[0],"kilometros")
9  print(mills[1],"millas son",km[1],"kilometros")
10 print(mills[2],"millas son",km[2],"kilometros")
```

El mismo programa, pero con listas

```
#ingreso las millas
mills = [34.122, 17.588, 3.187]

# Hago la conversion a kilometros
km = [mills[0]*1.6, mills[1]*1.6, mills[2]*1.6]

# Imprimo por pantalla el resultado
print(mills[0],"millas son",km[0],"kilometros")
print(mills[1],"millas son",km[1],"kilometros")
print(mills[2],"millas son",km[2],"kilometros")
```

La salida:

```
> 34.122 millas son 54.5952 kilometros
> 17.588 millas son 28.1408 kilometros
> 3.187 millas son 5.0992 kilometros
```

El mismo programa, pero con listas

```
1  #ingreso las millas
2  mills = [34.122, 17.588, 3.187]
3
4  # Hago la conversion a kilometros
5  km = [mills[0]*1.6, mills[1]*1.6, mills[2]*1.6]
6
7  # Imprimo por pantalla el resultado
8  print(mills[0],"millas son",km[0],"kilometros")
9  print(mills[1],"millas son",km[1],"kilometros")
10 print(mills[2],"millas son",km[2],"kilometros")
```

La salida:

```
1 > 34.122 millas son 54.5952 kilometros
2 > 17.588 millas son 28.1408 kilometros
3 > 3.187 millas son 5.0992 kilometros
```

- ¿Qué problemas tiene esta codificación?

El mismo programa, pero con listas

```
#ingreso las millas
mills = [34.122, 17.588, 3.187]

# Hago la conversion a kilometros
km = [mills[0]*1.6, mills[1]*1.6, mills[2]*1.6]

# Imprimo por pantalla el resultado
print(mills[0],"millas son",km[0],"kilometros")
print(mills[1],"millas son",km[1],"kilometros")
print(mills[2],"millas son",km[2],"kilometros")
```

La salida:

```
> 34.122 millas son 54.5952 kilometros
> 17.588 millas son 28.1408 kilometros
> 3.187 millas son 5.0992 kilometros
```

- ¿Qué problemas tiene esta codificación? Información duplicada y **sobretodo ineficiencia** para programar un proceso automatizable por medio de una **estructura de control**.

El mismo programa, pero con listas

```
1  #ingreso las millas
2  mills = [34.122, 17.588, 3.187]
3
4  # Hago la conversion a kilometros
5  km = [mills[0]*1.6, mills[1]*1.6, mills[2]*1.6]
6
7  # Imprimo por pantalla el resultado
8  print(mills[0],"millas son",km[0],"kilometros")
9  print(mills[1],"millas son",km[1],"kilometros")
10 print(mills[2],"millas son",km[2],"kilometros")
```

La salida:

```
1 > 34.122 millas son 54.5952 kilometros
2 > 17.588 millas son 28.1408 kilometros
3 > 3.187 millas son 5.0992 kilometros
```

- ¿Qué problemas tiene esta codificación? Información duplicada y **sobretodo ineficiencia** para programar un proceso automatizable por medio de una **estructura de control**.
- Tener más codigo no es síntoma de ser un buen programador

El mismo programa, pero con ciclo while

```
#ingreso las millas
mills = [1.1, 33.4, 34.122, 17.588, 3.187, 50.]

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

#hago la conversion a kilometros
km = [0]*longitud #lista de ceros de longitud adecuada

i = 0 # variable para indicar la posicion en la lista
while i < longitud: #condicion
    km[i] = mills[i]*1.6 #conversion
    i = i +1 #avance de posicion
print(km)
#la salida queda igual a:
> [1.8, 53.4, 54.6, 28.1, 5.1, 80.0]
```

El mismo programa, pero con ciclo while

```
1  #ingreso las millas
2  mills = [1.1, 33.4, 34.122, 17.588, 3.187, 50.]
3
4  #guardo la longitud de la lista
5  longitud = len(mills)
6
7  #hago la conversion a kilometros
8  km = [0]*longitud #lista de ceros de longitud adecuada
9
10 i = 0 # variable para indicar la posicion en la lista
11 while i < longitud: #condicion
12     km[i] = mills[i]*1.6 #conversion
13     i = i +1 #avance de posicion
14 print(km)
15 #la salida queda igual a:
16 > [1.8, 53.4, 54.6, 28.1, 5.1, 80.0]
```

- Un programa automatizado de conversión, con el que se tienen que familiarizar.

El mismo programa, pero con ciclo while

```
1  #ingreso las millas
2  mills = [1.1, 33.4, 34.122, 17.588, 3.187, 50.]
3
4  #guardo la longitud de la lista
5  longitud = len(mills)
6
7  #hago la conversion a kilometros
8  km = [0]*longitud #lista de ceros de longitud adecuada
9
10 i = 0 # variable para indicar la posicion en la lista
11 while i < longitud: #condicion
12     km[i] = mills[i]*1.6 #conversion
13     i = i +1 #avance de posicion
14 print(km)
15 #la salida queda igual a:
16 > [1.8, 53.4, 54.6, 28.1, 5.1, 80.0]
```

- Un programa automatizado de conversión, con el que se tienen que familiarizar.
- ¿Pero se puede automatizar aún más?

El mismo programa, pero con ciclo while

```
1  #ingreso las millas
2  mills = [1.1, 33.4, 34.122, 17.588, 3.187, 50.]
3
4  #guardo la longitud de la lista
5  longitud = len(mills)
6
7  #hago la conversion a kilometros
8  km = [0]*longitud #lista de ceros de longitud adecuada
9
10 i = 0 # variable para indicar la posicion en la lista
11 while i < longitud: #condicion
12     km[i] = mills[i]*1.6 #conversion
13     i = i +1 #avance de posicion
14 print(km)
15 #la salida queda igual a:
16 > [1.8, 53.4, 54.6, 28.1, 5.1, 80.0]
```

- Un programa automatizado de conversión, con el que se tienen que familiarizar.
- ¿Pero se puede automatizar aún más? Sí.

Funciones

Funciones

- Una función es una unidad de código que aísla una parte de un cómputo. **Es un programa dentro de un programa.**

Funciones

- Una función es una unidad de código que aísla una parte de un cómputo. **Es un programa dentro de un programa.**
- Poseen argumentos o parámetros (inputs), los procesa y devuelve una salida (output).

Funciones

- Una función es una unidad de código que aísla una parte de un cómputo. **Es un programa dentro de un programa.**
- Poseen argumentos o parámetros (inputs), los procesa y devuelve una salida (output). (Piensen en las funciones matemáticas)

Funciones

- Una función es una unidad de código que aísla una parte de un cómputo. **Es un programa dentro de un programa.**
- Poseen argumentos o parámetros (inputs), los procesa y devuelve una salida (output). (Piensen en las funciones matemáticas)
- Características de las funciones:

Funciones

- Una función es una unidad de código que aísla una parte de un cómputo. **Es un programa dentro de un programa.**
- Poseen argumentos o parámetros (inputs), los procesa y devuelve una salida (output). (Piensen en las funciones matemáticas)
- Características de las funciones:

1. Permite **dividir** un problema en problemas más simples.

Funciones

- Una función es una unidad de código que aísla una parte de un cómputo. **Es un programa dentro de un programa.**
- Poseen argumentos o parámetros (inputs), los procesa y devuelve una salida (output). (Piensen en las funciones matemáticas)
- Características de las funciones:

1. Permite **dividir** un problema en problemas más simples.
2. Permite **ordenar conceptualmente** el código para que sea más fácil de entender.

Funciones

- Una función es una unidad de código que aísla una parte de un cómputo. **Es un programa dentro de un programa.**
- Poseen argumentos o parámetros (inputs), los procesa y devuelve una salida (output). (Piensen en las funciones matemáticas)
- Características de las funciones:

1. Permite **dividir** un problema en problemas más simples.
2. Permite **ordenar conceptualmente** el código para que sea más fácil de entender.
3. Permite **reutilizar soluciones** a problemas pequeños en la solución de problemas mayores.

Estructura de las funciones

Estructura de las funciones

- Funciones que retornan un valor

```
1 def NOMBRE(ARGUMENTOS):
2     ENUNCIADOS
3     return VALOR
```

Estructura de las funciones

- Funciones que retornan un valor

```
def NOMBRE(ARGUMENTOS):
    ENUNCIADOS
    return VALOR
```

- Funciones que no devuelven valor alguno (también llamadas Procedimiento)

```
def NOMBRE(ARGUMENTOS):
    ENUNCIADOS
    print(VALOR)
```

Estructura de las funciones

- Funciones que retornan un valor

```
def NOMBRE(ARGUMENTOS):
    ENUNCIADOS
    return VALOR
```

- Funciones que no devuelven valor alguno (también llamadas Procedimiento)

```
def NOMBRE(ARGUMENTOS):
    ENUNCIADOS
    print(VALOR)
```

- Nuevamente, atención con el bloque indentado!

Estructura de las funciones

- Funciones que retornan un valor

```
def NOMBRE(ARGUMENTOS):
    ENUNCIADOS
    return VALOR
```

- Funciones que no devuelven valor alguno (también llamadas Procedimiento)

```
def NOMBRE(ARGUMENTOS):
    ENUNCIADOS
    print(VALOR)
```

- Nuevamente, atención con el bloque indentado!
- Tanto en los enunciados (donde se realizan las operaciones sobre los argumentos), como en la salida, puede ir cualquier cosa.

Estructura de las funciones

- Funciones que retornan un valor

```
def NOMBRE(ARGUMENTOS):
    ENUNCIADOS
    return VALOR
```

- Funciones que no devuelven valor alguno (también llamadas Procedimiento)

```
def NOMBRE(ARGUMENTOS):
    ENUNCIADOS
    print(VALOR)
```

- Nuevamente, atención con el bloque indentado!
- Tanto en los enunciados (donde se realizan las operaciones sobre los argumentos), como en la salida, puede ir cualquier cosa.
- Importante: `def` y `return` son dos palabras reservadas.

Estructura de las funciones

- Funciones que retornan un valor

```
def NOMBRE(ARGUMENTOS):
    ENUNCIADOS
    return VALOR
```

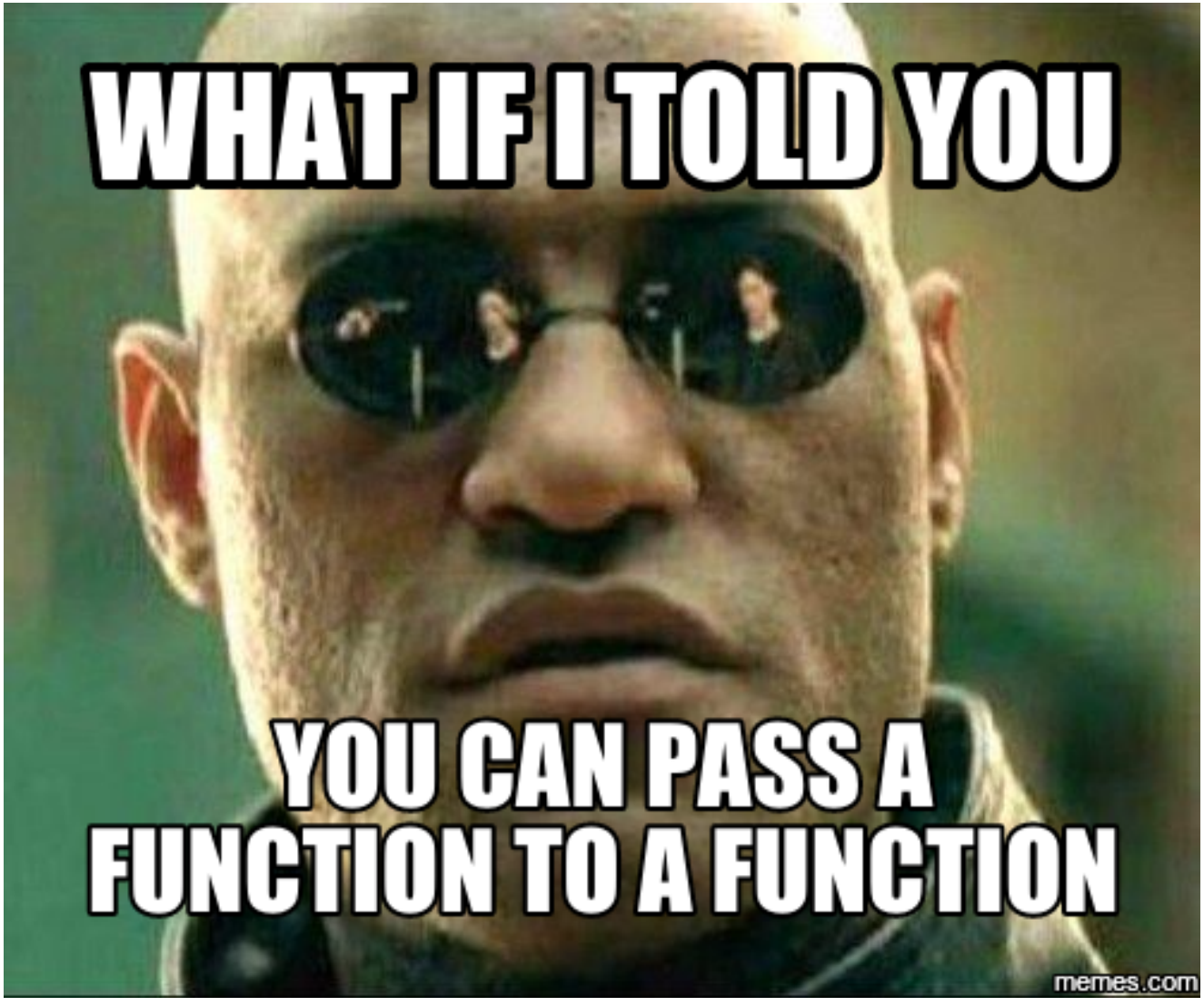
- Funciones que no devuelven valor alguno (también llamadas Procedimiento)

```
def NOMBRE(ARGUMENTOS):
    ENUNCIADOS
    print(VALOR)
```

- Nuevamente, atención con el bloque indentado!
- Tanto en los enunciados (donde se realizan las operaciones sobre los argumentos), como en la salida, puede ir cualquier cosa.
- Importante: `def` y `return` son dos palabras reservadas. Se puede retornar más de un valor

En los enunciados puede ir cualquier cosa cualquier cosa...

En los enunciados puede ir cualquier cosa cualquier cosa...



El mismo programa, pero en una función

El mismo programa, pero en una función

```
1  def convertir(lista):
2      longitud = len(lista)
3     # Hago la conversion
4      listaConvertida = [0]*longitud # lista de ceros
5      i = 0
6      while i < longitud:
7          listaConvertida[i] = lista[i]*1.6
8          i = i + 1
9      return listaConvertida
10
11 # El resto del codigo queda mas prolijo
12 mills = [1.1, 33.4, 13.2, 60.0, 17.35]
13 kms = convertir(mills)
14 print(kms)
15 > [1.8, 53.4, 54.6, 28.1, 5.1, 80.0] #  la salida
```

El mismo programa, pero en una función

```
1  def convertir(lista):
2      longitud = len(lista)
3     # Hago la conversion
4      listaConvertida = [0]*longitud # lista de ceros
5      i = 0
6      while i < longitud:
7          listaConvertida[i] = lista[i]*1.6
8          i = i + 1
9      return listaConvertida
10
11 # El resto del codigo queda mas prolijo
12 mills = [1.1, 33.4, 13.2, 60.0, 17.35]
13 kms = convertir(mills)
14 print(kms)
15 > [1.8, 53.4, 54.6, 28.1, 5.1, 80.0] #  la salida
```

- La función `convertir` **encapsula** el problema de la conversión a kilómetros.

El mismo programa, pero en una función

```
1  def convertir(lista):
2      longitud = len(lista)
3     # Hago la conversion
4      listaConvertida = [0]*longitud # lista de ceros
5      i = 0
6      while i < longitud:
7          listaConvertida[i] = lista[i]*1.6
8          i = i + 1
9      return listaConvertida
10
11 # El resto del codigo queda mas prolijo
12 mills = [1.1, 33.4, 13.2, 60.0, 17.35]
13 kms = convertir(mills)
14 print(kms)
15 > [1.8, 53.4, 54.6, 28.1, 5.1, 80.0] #  la salida
```

- La función `convertir` **encapsula** el problema de la conversión a kilómetros.
- Ventajas?

El mismo programa, pero en una función

```
1  def convertir(lista):
2      longitud = len(lista)
3     # Hago la conversion
4      listaConvertida = [0]*longitud # lista de ceros
5      i = 0
6      while i < longitud:
7          listaConvertida[i] = lista[i]*1.6
8          i = i + 1
9      return listaConvertida
10
11 # El resto del codigo queda mas prolijo
12 mills = [1.1, 33.4, 13.2, 60.0, 17.35]
13 kms = convertir(mills)
14 print(kms)
15 > [1.8, 53.4, 54.6, 28.1, 5.1, 80.0] #  la salida
```

- La función `convertir` **encapsula** el problema de la conversión a kilómetros.
- Ventajas? **Particionamos** el problema en otros dos de menor complejidad y el programa **puede ser resuelto por diferentes personas y al mismo tiempo**

¿Cuál es la diferencia con el programa anterior?

¿Cuál es la diferencia con el programa anterior?

```
1  def convertir(lista,factor):
2      longitud = len(lista) #guardo la longitud de la
       lista
3     # Hago la conversion
4      listaConvertida = [0]*longitud # lista de ceros
5      i = 0
6      while i < longitud:
7          listaConvertida[i] = lista[i]*factor
8          i = i + 1
9      return listaConvertida
10
11 # El resto del codigo queda mas prolijo
12 mills = [1.1, 33.4, 13.2, 60.0, 17.35]
13 fc = 1.6
14 kms = convertir(mills,fc)
```

¿Cuál es la diferencia con el programa anterior?

```
1  def convertir(lista,factor):
2      longitud = len(lista) #guardo la longitud de la
       lista
3     # Hago la conversion
4      listaConvertida = [0]*longitud # lista de ceros
5      i = 0
6      while i < longitud:
7          listaConvertida[i] = lista[i]*factor
8          i = i + 1
9      return listaConvertida
10
11 # El resto del codigo queda mas prolijo
12 mills = [1.1, 33.4, 13.2, 60.0, 17.35]
13 fc = 1.6
14 kms = convertir(mills,fc)
```

- La función convertir ahora toma dos parámetros de entrada.

¿Cuál es la diferencia con el programa anterior?

```
1  def convertir(lista,factor):
2      longitud = len(lista) #guardo la longitud de la
       lista
3     # Hago la conversion
4      listaConvertida = [0]*longitud # lista de ceros
5      i = 0
6      while i < longitud:
7          listaConvertida[i] = lista[i]*factor
8          i = i + 1
9      return listaConvertida
10
11 # El resto del codigo queda mas prolijo
12 mills = [1.1, 33.4, 13.2, 60.0, 17.35]
13 fc = 1.6
14 kms = convertir(mills,fc)
```

- 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.

A coedear se a dicho!

A coedear se a dicho!

Nuestro nuevo aprendizaje:

A coedear se a dicho!

Nuestro nuevo aprendizaje:

- Siempre que sea posible particionar el problema y encapsularlo en problemas menos complejos mediante funciones