

LA RECETA INFALIBLE EN PYTHON

Cátedra Redictado Programación II

Marzo 2022

1. Construcción de Programas

Como vimos en **Programación I**, la receta que aplicamos consta de los siguientes pasos:

1. diseño de datos; (análisis del problema)
2. signatura y declaración de propósito; (análisis del problema)
3. ejemplos; (análisis del problema)
4. definición de la función; (código)
5. evaluar el código de los ejemplos; (testing)
6. realizar modificaciones en caso que el paso anterior genere errores. (debugging)

¿Cómo aplicamos La Receta en Python? La receta la aplicaremos en todas las funciones que compondrán mi programa. Un programa diseñado, es aquel en el cual aplicamos La Receta a todas las funciones que lo componen. Veremos, sobre distintos ejemplos, cómo diseñar nuestros programas. Una vez explicado el tema, los ejercicios que resolvamos de aquí en más, **serán todos usando La Receta**. Los ejercicios resueltos sin la mencionada metodología no serán considerados como resolución completa, en las evaluaciones serán comentados como soluciones incompletos a los problemas.

Problema 1: Traducción de temperaturas de Farenheit a Celsius

Enunciado

Diseñar un programa que convierta una temperatura medida en un termómetro Fahrenheit a una temperatura en Celsius. Para ello tener en cuenta la siguiente fórmula de conversión:

$$C = (F - 32) \times \frac{5}{9}$$

1.1. Item 1: Diseño de datos

¿ *Cómo representamos la información* ? Es decir, que tipos de datos vamos a usar del lenguaje Python para representar las entidades del “mundo del problema” para poder abstraerlas en el “mundo computacional”.



```
1 #Representamos temperaturas mediante números float
2 #temperatura: float
```

1.2. Item 2: Signatura y declaración de propósito

La **signatura** de una función indica qué parámetros recibe (cuántos y de qué tipo), y qué datos retorna y de qué tipo. En el caso de los problemas sencillos que abordaremos, deberemos decidir:

- a) cuáles son los datos de entrada que se nos proveen,
- b) cuáles son las salidas que debemos producir, y
- c) cuál es la relación entre todos ellos.

Es importante tener presente que puede suceder que una función en Python no tome argumentos o no retorne valores. Más adelante veremos cómo representar esto.

La **declaración de propósito** indica de forma resumida ¿Qué hace? la función. También incluye información sobre convenciones que el programador haya tenido en cuenta a la hora de diseñar el programa.



```
1 #far_cel: float -> float
2 #El parámetro representa una temperatura en Fahrenheit y retorna su equivalente↔
   en Celsius.
```

1.3. Item 3: Ejemplos

¿Qué sabemos de nuestra función? ¿Qué nos debería devolver bajo ciertas entradas?. Luego de los pasos anteriores, nos sigue uno de los pasos mas importantes que tiene el diseño de los programas, los resultados que sabemos que nuestra función debe devolver en determinados casos. Estos resultados los conocemos de antemano a partir de domino del problema que estamos resolviendo con nuestro programa. En nuestro caso podríamos pensar en que contamos con ciertos valores de referencia extraídos de un libro de física.

En este caso sería:



```
1 #Ejemplos:
2 # far_cel(32)  = 0
3 # far_cel(212) = 100
4 # far_cel(-40) = -40
5 # far_cel(90)  = 32.22
```

1.4. Item 4: Definición del programa

¡Escribimos código! Es decir, vamos a traducir a un lenguaje de programación (en nuestro caso, y por el momento, Python) el diseño que pensamos para resolver nuestro problema.



```
1 def far_cel(temp_f):  
2     temp_c = (temp_f-32)*5/9  
3     return temp_c
```

1.5. Item 5: Evaluar el código en los ejemplos

¿ *Qué significa* ? Aplicar el código generado en los ejemplos que se diseñaron y, verificar que los resultados obtenidos coincidan con lo esperado.



```
>>> far_cel(32)  
0  
>>> far_cel(212)  
100  
>>> far_cel(-40)  
-40  
>>> far_cel(90)  
32.22222222222222
```

1.6. Item 6: Realizar modificaciones en caso de error

¿ *Qué significa* ? Encontrar los problemas que se hubieran detectado y solucionarlos. Esto puede implicar revisar cada y corregir cada una de las etapas antes descriptas. Los errores que encontremos pueden ser lógicos o sintácticos.

1.7. El Programa Completo


Una forma de diseño de la función empleamos los comentarios para escribir los pasos de la receta. La información aparece asociada a la función pero de forma implícita. Mostramos aquí abajo como nos quedaría el diseño completa bajo esta consideración.



```
1 #Representamos temperaturas mediante números float  
2 #temperatura: float  
3 #far_cel: float -> float  
4 #El parámetro representa una temperatura en Fahrenheit y retorna su equivalente↔  
   en Celsius.  
5 #Ejemplos:  
6 # far_cel(32) = 0  
7 # far_cel(212) = 100  
8 # far_cel(-40) = -40
```

```
9 # far_cel(90) = 32.22
10
11 def far_cel(temp_f):
12     temp_c = (temp_f-32)*5/9
13     return temp_c
```

Otra forma de presentar el diseño es asociando al receta a la función de forma explícita, usando los comentarios conocidos como "docstrings". Este tipo de comentarios quedan vinculados a la función, se escriben con tres comillas simples, y se ubica a continuación del nombre de la función. Veamos como resulta en este caso:



```
1 def far_cel(temp_f):
2     '''Representamos temperaturas mediante números float
3     temperatura: float
4     far_cel: float -> float
5     El parámetro representa una temperatura en Fahrenheit y
6     retorna su equivalente en Celsius.
7     Ejemplos:
8     far_cel(32) = 0
9     far_cel(212) = 100
10    far_cel(-40) = -40
11    far_cel(90) = 32.22 '''
12    temp_c = (temp_f-32)* 5/9
13    return temp_c
```

Este tipo de documentación permite que en el intérprete se pueda consultar la ayuda de la función usando el comando `help('<Funcion>')` el cual brindará como respuesta la cadena docstring asociada a la función.

```
> help(far_cel)
Help on function far_cel in module __main__:

far_cel(temp_f)
    Representamos temperaturas mediante números float
    temperatura: float
    far_cel: float -> float
    El parámetro representa una temperatura en Fahrenheit y
    retorna su equivalente en Celsius.
    Ejemplos:
    far_cel(32) = 0
    far_cel(212) = 100
    far_cel(-40) = -40
    far_cel(90) = 32.22
```

EJERCICIO 1. Vamos a completar el problema solicitando que el usuario ingrese la temperatura a convertir. ¿Cómo modificaríamos nuestro programa?

EJERCICIO 2. Complete el programa anterior:

- Agregando algunas funciones más de conversión de temperatura.
- El usuario no sólo brindará la temperatura a convertir sino también la unidad en la que esta dada. Por ejemplo “F” para Fahrenheit, “C” para Celsius, “K” para Kelvin, etc.

Dejamos a continuación una tabla de conversión para esta actividad.

FÓRMULAS DE CONVERSIÓN DE TEMPERATURA		
CONVERSIÓN DE	A	FÓRMULA
Grados Celsius	Grados Fahrenheit	$^{\circ}\text{F} = ^{\circ}\text{C} \times 1.8 + 32$
Grados Celsius	Kelvin	$\text{K} = ^{\circ}\text{C} + 273.15$
Grados Celsius	Rankine	$\text{R} = (^{\circ}\text{C} + 273.15) \times 1.8$
Grados Fahrenheit	Grados Celsius	$^{\circ}\text{C} = (^{\circ}\text{F} - 32) / 1.8$
Grados Fahrenheit	Kelvin	$\text{K} = (^{\circ}\text{F} + 459.67) / 1.8$
Grados Fahrenheit	Rankine	$\text{R} = ^{\circ}\text{F} + 459.67$
Kelvin	Grados Celsius	$^{\circ}\text{C} = \text{K} - 273.15$
Kelvin	Grados Fahrenheit	$^{\circ}\text{F} = 9\text{K} - 459.67$
Rankine	Grados Celsius	$^{\circ}\text{C} = (\text{R} / 1.8) - 273.15$

EJERCICIO 3. Revisar los ejercicios realizados en las prácticas 1 y 2 para adecuar los mismos al formato y estilo de construcción de programas indicado por La Receta.

2. Testing - Pytest

Para automatizar los casos de prueba en Python vamos a utilizar la librería **pytest**. Una vez instalada la misma podemos invocar al testeo desde la línea de comandos, escribiendo:

```
1 >>> python -m pytest <nombre_del_programa>.py
```

no haciendo falta poner el `import` en el código del programa.

En el siguiente ejemplo, **prueba.py** mostramos escribir nuestra implementación para utilizar la librería **pytest** para testear la función `fmaximo`.

```
1 def fmaximo(x, y):
2     '''
3     número: Int
4     fmaximo: Int Int -> Int
5     Función que determina el máximo entre dos números.
6     fmaximo (1,5) = 5
7     fmaximo (1,0) = 1
8     fmaximo (1,1) = 1
9     fmaximo (-1,1) = 1
10    fmaximo (-1,-5) = -1
11    '''
12    if x > y:
13        return x
```

```

14     else:
15         return y
16
17 def test_fmaximo():
18     '''
19     Función de prueba de la función fmaximo
20     '''
21     assert fmaximo(3,4) == 4
22     assert fmaximo(-4,92) == 92
23     assert fmaximo(-10,2) == 2
24     assert fmaximo(-5,72) == 72

```

La función de prueba `test_fmaximo()` sigue el estilo de los "check expect" en Dr. Rackett. Se deben definir distintos `assert` igualando el llamado a la función con ciertos argumentos que se consideren interesantes y relevantes. En la práctica, generalmente, les daremos los argumentos con los cuales deben testear las funciones.

Dr. Racket		Pytest	
1	<code>(define (fmaximo x y)</code>	1	<code>def fmaximo(x y):</code>
2	<code> (if (> x y) x y))</code>	2	<code> if (x > y):</code>
3	<code>(check-expect(fmaximo 5 6) 6)</code>	3	<code> return x</code>
4	<code>(check-expect(fmaximo 0 -5) 0)</code>	4	<code> else:</code>
		5	<code> return y</code>
		6	
		7	<code>def test_fmaximo():</code>
		8	<code> assert fmaximo (5,6) == 6</code>
		9	<code> assert fmaximo (0,-5) == 0</code>

EJERCICIO 4. Testear la función anterior, escribiendo en el interprete el siguiente comando:

```

1 >>> python -m pytest prueba.py

```

EJERCICIO 5. Modificar la implementación anterior, agregando los siguientes casos de prueba:

```

1 assert fmaximo(5,6) == 5
2 assert fmaximo(3,0) == 4

```

¿Qué sucede?, ¿Se chequean todos los asserts?

EJERCICIO 6. Completar el diseño de la función `far_cel` utilizando el modulo **Pytest** para realizar el testing unitario.

EJERCICIO 7. Completar el diseño de las funciones de las prácticas 1 y 2 utilizando el módulo **Pytest** para realizar el testing unitario.