La receta infalible para diseñar programas

Dante Zanarini

LCC

15 de marzo de 2016

Qué necesitamos para programar (1)

Para programar necesitamos un **lenguaje**. Debemos conocer:

- Su *vocabulario*,
- su *gramática*,
- y el *significado* de sus frases.

Nosotros elegimos racket, y estamos aprendiendo todo lo anterior.

¿Escribir programas me convierte en un buen programador?

Algunas preguntas relacionadas:

- ¿Patear una pelota me convierte en Marco Ruben?
- Sé cómo se mueven las piezas, ¿soy un jugador de ajedrez?
- ¿Usar un cuchillo me convierte en cirujano?
- Ayer hice un huevo frito, ¿soy un chef?

Qué necesitamos para programar (2)

Además de un lenguaje, ¿qué necesitamos para programar?

Necesitamos la habilidad para transformar el enunciado de un problema en un programa.

Algunas preguntas:

- ¿Qué partes del enunciado son relevantes y cuáles no?
- ¿Qúe información recibe en programa? ¿Qué produce? ¿Cómo se relacionan la entrada y la salida?
- ¿Qué herramientas me provee mi lenguaje y sus librerías para resolver el problema?
- Ya tengo el programa, ¿resuelve realmente el problema? Si no, ¿Cómo lo arreglo?

LA receta

- La idea es dar una herramienta para atacar cualquier problema de programación.
- Al tener definida una serie de pasos concretos, es menos probable que nos quedemos en blanco.
- Como toda receta, no sólo indica qué pasos hay que seguir, sino también en qué orden hay que hacerlo!

Los pasos para diseñar programas

1. Diseño de datos;
2. signatura y declaración de propósito;
3. ejemplos;
4. definición de la función;
5. evaluar el código en los ejemplos;
6. realizar modificaciones en caso que el paso anterior genere errores.

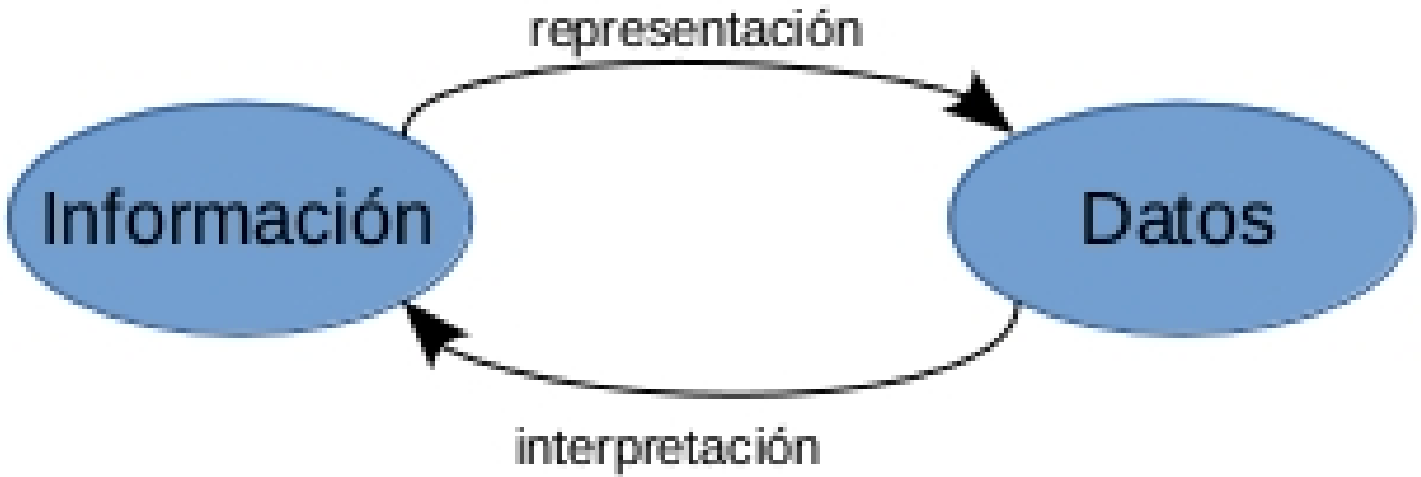
Un problema simple

Para ilustrar los pasos, trabajemos con el siguiente problema:

Escribir un programa que convierta una temperatura medida en un termómetro Fahrenheit a una temperatura en Celsius.

1. Diseño de datos

- La información vive en el mundo real, y es parte del dominio del problema
- Para que un programa pueda procesar información, esta debe representarse a través de datos.
- Asimismo, los datos producidos por un problema, deben poder interpretarse como información.



1. Diseño de datos

- En este paso, definimos la forma de representar la información como datos:

```
;Representamos temperaturas mediante números
```

- A medida que necesitemos representar información más estructurada, esta parte se volverá más interesante.

2. Signatura y declaración de propósito

- La signatura de una función indica qué datos consume (cuántos y de qué clase), y qué datos produce. Ejemplos:

  ```
  ;Number -> Number
  ```

  Signatura para una función que consume un número y produce un número.

  ```
  ;Number String String -> Image
  ```

  Signatura para una función que consume un número y dos strings, y devuelve una imagen

2. Signatura y declaración de propósito

Breve descripción del comportamiento de la función.

¿Qué calcula esta función?

- Cualquier persona que lea un programa, debería entender qué calcula la función sin necesidad de inspeccionar código.
- Para nuestro ejemplo:

```
; recibe una temperatura en Fahrenheit, devuelve su
equivalente en Celsius.
```

3. Ejemplos

A partir de los dos primeros pasos, debemos estar en condiciones de predecir el comportamiento de la función para algunas entradas:

```
; entrada:  32, salida:  0
; entrada:  212, salida:  100
; entrada:  -40, salida:  -40
```

4. Definición de la función (código al fin!)

Ya podemos escribir la definición de nuestra función:

```
(define (far->cel f) (* 5/9 (- f 32)))
```

5. Evaluar el código en los ejemplos

```
> (far->cel 32)
0

> (far->cel 212)
100

> (far->cel (- 40))
-40
```

6. Realizar modificaciones en caso de errores

- ¿Y si no me da?
- Podrían estar mal formulados los ejemplos,
- podría haber un error en el programa,
- o ambas cosas a la vez (muy raro que suceda).
- En general, conviene revisar primero que los ejemplos estén bien.

Juntando todo...

Escribir un programa que convierta una temperatura medida en un termómetro Fahrenheit a una temperatura en Celsius.

```
;Representamos temperaturas mediante números
; far->cel :  Number -> Number
; recibe una temperatura en Fahrenheit, devuelve su
equivalente en Celsius.
; entrada:  32, salida:  0
; entrada:  212, salida:  100
; entrada:  -40, salida:  -40
(define (far->cel f) (* 5/9 (- f 32)) )
```