Un método para diseñar programas

Dante Zanarini

10 de abril de 2020

¿Qué necesitamos para programar? (1)

Para programar necesitamos un
lenguaje de programación. Debemos conocer:

- *Su vocabulario,*
- *su gramática (o sintaxis),*
- *y el significado de sus frases (o semántica).*

¿Qué necesitamos para programar? (2)

En nuestro caso elegimos racket como lenguaje,
y estamos aprendiendo todo lo anterior.

¿Escribir programas me convierte en programador?

¿Qué necesitamos para programar? (2)

En nuestro caso elegimos racket como lenguaje,
y estamos aprendiendo todo lo anterior.

¿Escribir programas me convierte en programador?

¿Patear una pelota me convierte
en Giovanni Lo Celso?
¿Usar un cuchillo me hace cirujano?

Ayer hice un huevo frito, ¿soy chef?

¿Qué necesitamos para programar? (3)

Además de un lenguaje,

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

¿Qué necesitamos para programar? (3)

Además de un lenguaje,

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

Algunas preguntas:

- ¿ Qué partes del enunciado son relevantes y cuáles no?
- ¿Qué información recibe el programa? ¿Qué produce? ¿Cómo se relacionan la entrada y la salida?
- ¿Qué herramientas me provee el lenguaje y sus bibliotecas para resolver el problema?
- Ya tengo el programa, ¿resuelve realmente el problema? ¿Cómo encuentro errores?, y ¿Cómo los elimino?

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) validación en los ejemplos;

(6) realizar modificaciones en caso de error.

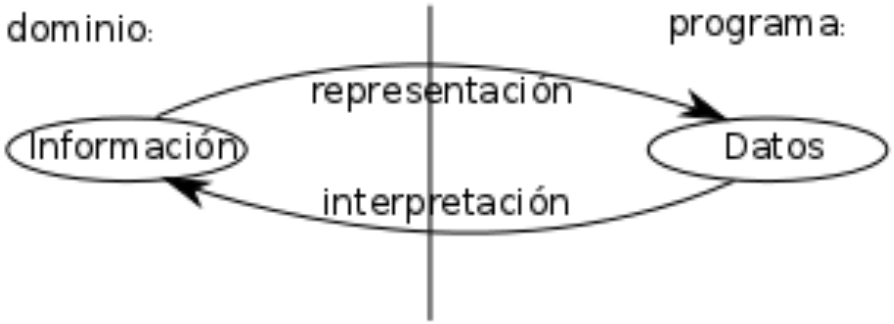
Un problema simple

Trabajaremos con el siguiente problema:

Dadas las longitudes (expresadas en metros)
de las aristas de un prisma rectangular,
calcular su área.

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 que produce un problema, deben poder interpretarse como información



1. Diseño de datos

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

```
; Representamos longitudes mediante números,
; siendo 1 el equivalente a un metro.
```

- 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 tipo) y qué datos produce. Por ejemplo:
- Signatura de una función que consume un booleano y un string y devuelve un número

```
; Bool String -> Number
```

- Signatura de una función que consume tres números y devuelve un número

```
; Number Number Number -> Number
```

2. Signatura y declaración de propósito

- La **declaración de propósito** es un breve comentario que describe el comportamiento de la función
- Podemos pensarlo como la respuesta más corta posible a la pregunta *¿Qué calcula esta función?*
- Cualquiera que lea nuestro programa debería saber qué hace cada función sin necesidad de leer su código. En el ejemplo:

```
; Dados tres números que representan el ancho,
; largo y alto de un prisma rectangular,
; calcula su área
```

Ejemplos, casos de prueba o test de unidad

- Antes de resolver el problema para todos los posibles valores de entrada, lo resolvemos para algunos ejemplos concretos
- Esto tiene una doble ventaja:
  1. nos ayuda a deducir una regla general para el problema
  2. nos permite validar nuestra solución una vez definida

```
; Entrada: 4,5,6; salida: 148
; Entrada: 1,1,1; salida: 6
```

Definición de la función

- Con toda la información de más arriba, podemos escribir un template de nuestra función:

```
(define (área a b c) ...)
```

- Ahora sí, viene la parte creativa. Tendríamos que poder definir el código, ayudados tal vez por los ejemplos.

Evaluación en los ejemplos

- Evaluamos la expresión en los ejemplos para verificar que (al menos en esos casos) la respuesta es la esperada.

```
> (área 4 5 6)
148
> (área 1 1 1)
6
```

¿Y si no me da?

- ¿Qué ocurre si el resultado esperado difiere del que obtengo?

¿Y si no me da?

- ¿Qué ocurre si el resultado esperado difiere del que obtengo?
- Pueden darse tres situaciones:
  - Los ejemplos fueron mal calculados
  - La función calcula de forma incorrecta el resultado
  - Ocurren ambas

¿Y si no me da?

- ¿Qué ocurre si el resultado esperado difiere del que obtengo?
- Pueden darse tres situaciones:
  - Los ejemplos fueron mal calculados
  - La función calcula de forma incorrecta el resultado
  - Ocurren ambas
- En cualquier caso, deberíamos ser capaces de encontrar y corregir el error

Juntando todo...

```
;  Representamos longitudes mediante números,
; siendo 1 el equivalente a un metro.
;
; área: Number Number Number -> Number
; Dados tres números que representan el ancho,
; largo y alto de un prisma rectangular,
;  calcula su área.
; Entrada: 4,5,6. Salida: 148
; Entrada: 1,1,1. Salida: 6
(define
 (área a b c)
 (+ (* 2 a b) (* 2 b c) (* 2 a c)))
```