

Script Integración 1 - Introducción a Paradigmas

¿Qué es un programa?



Dos visiones:

Visto desde **1**) es aquel ingenio que logra que una computadora haga lo que uno tiene en mente. Yo le doy un INPUT y el OUTPUT que me devuelve presupone un valor agregado para mi actividad.

Visto desde adentro (de lo que pasa en la compu, **2**): mmm... son todos unos y ceros 😊

Pero qué es lo que uno conoce hasta ahora como “programa”:

- Es una secuencia de pasos o instrucciones
- Las variables son posiciones de memoria que permiten guardar valores intermedios
- Las entidades se representan mediante tipos de datos simples o compuestos (los compuestos me permiten agrupar todas las características de un alumno, un cliente o una factura)
- Para acceder a esos datos se utilizan funciones o procedimientos (el flujo principal del programa está en un procedimiento principal o cuerpo, que tiene llamadas a subprocedimientos). Para organizar un programa tenemos 3 estructuras de control

De secuencia

```
Instrucción 1;  
Instrucción 2;
```

De selección

```
if (a > 8) {  
    ...  
} else {  
    ...  
}
```

De iteración

```
while (true) {  
    ...  
}
```

- Después de la ejecución del programa se llega a un estado final que hay que verificar si es correcto o no (p.ej. mediante una prueba de escritorio)

Bueno, Paradigmas viene a romper un poco con todo esto que conocen. Un programa puede (o no) tener algunas de las ideas que nombramos arriba y que para ustedes eran axiomas a la hora de encarar un desarrollo.

Hasta ahora conocen una herramienta posible para programar: un martillo.



Eso tiene una ventaja... tienen la herramienta adecuada para clavar. Para sacar clavos también les sirve y hasta para picar azulejos va bien. El tema es si quiero poner un tarugo, sacar un tornillo o conectar una llave combinada.

"Cuando tenés un martillo, todo lo que ves son clavos"

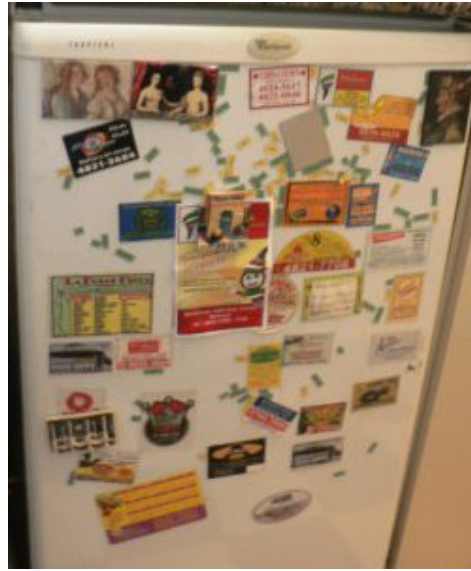
Acá, en Paradigmas de Programación (PDP para los amigos) se van a llevar tres puntos de vista nuevos, con lo que la pregunta original (¿qué es un programa?) no tiene una única respuesta.

¿Y qué es un paradigma?

Un **paradigma** es un conjunto de ideas (herramientas conceptuales) que configura una respuesta a esa pregunta (qué es un programa). Cada paradigma tiene una abstracción principal que define criterios de programación radicalmente diferentes.



vs.



Ejemplo: Tengo hambre y quiero comer empanadas.

- Marco 1: Paradigma del amo de casa
 - Revisar la heladera si hay tapas para empanadas, jamón, queso, etc.
 - Ver si el horno está limpio, si tengo gas, etc.
 - Preparar las empanadas, cocinarlas, comerlas.
- Marco 2: Delivery
 - Revisar la heladera en busca de imanes.
 - Conseguir el imán de las empanadas.
 - Revisar si tengo crédito en el teléfono, si anda, si tengo cambio en la billetera.
 - Llamar, pedir las, esperar, pagarlas, comerlas.

Aquí se ve:

- Que la misma problemática puede pensarse desde distintos puntos de vista y que la solución de cada una es bastante diferente a la de la otra.
- Que en ambos marcos pienso en la heladera, pero la visión que tengo es muy diferente en cada caso:
 - en uno es un contenedor de ingredientes de empanadas
 - en otro es el lugar donde pegar los imanes.

Es el mismo objeto físico, pero la visión cambia mucho, según el marco de resolución del problema.

¿Declarativo o procedimental?

El ejemplo reciente nos sirve para introducir lo que vamos a ver varias veces en esta materia: podemos pensar una solución en forma declarativa o procedimental...

Cuando la solución tiende a lo procedimental (imperativo)

- tenemos secuencia
- decimos cómo resolver un problema
- tenemos mayor control sobre el algoritmo (decidimos más cosas → definimos más cosas)

Cuando la solución tiende a lo declarativo

- expresamos características del problema

- alguien termina resolviendo el algoritmo para la máquina (necesitamos de alguna “magia”, algún mecanismo externo)
- tenemos menor control sobre el algoritmo (pero hay que pensar en menos cosas)

Pregunta para el lector: ¿cuál de las dos formas de resolver el requerimiento de las empanadas tiende a ser más declarativa? ¿por qué, qué pasa con la otra solución?

Vamos a resolver otro requerimiento: queremos obtener los elementos pares de una lista de números, lo implementamos primero en pseudocódigo.

Pseudocódigo

```

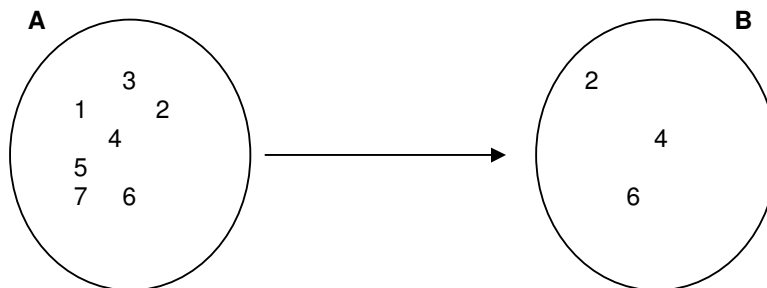
type
  LISTA = array[1..longitudMaxima] of integer;

procedure pares(listaNumeros: LISTA, var listaPares: LISTA)
begin
  j ← 0;
  recorrer i de 1 a longitudMaxima
  begin
    if (listaNumeros[i] es par)
    begin
      listaPares[j] = listaNumeros[i];
      j ← j + 1;
    end
  end
end
end

```

¿Cómo lo pienso desde una óptica matemática?

- Vector de números → noción de conjuntos
- “Recorrer” un vector y tomar los que son pares → definir un subconjunto en base al conjunto original.



$$B = \{ x / x \in A \wedge x \text{ es par} \}$$

Les presentamos la misma solución que filtra los pares de un conjunto de números, en el lenguaje Haskell (que vamos a ver en la cursada de PDP):

pares A = [x | x <- A , even x]

¿Cuál sería el dominio e imagen de la función? De un conjunto de números a otro conjunto de números. Como hemos visto la solución en pseudocódigo y en Haskell son bastante diferentes.

- ¿En cuál solución tengo mayor control sobre el algoritmo?
- ¿En cuál solución tengo mayor foco en lo que tengo que resolver?

La solución planteada en pseudocódigo *tiende a* ser más procedimental y la solución en Haskell *tiende a* ser más declarativa.

Declaratividad y expresividad

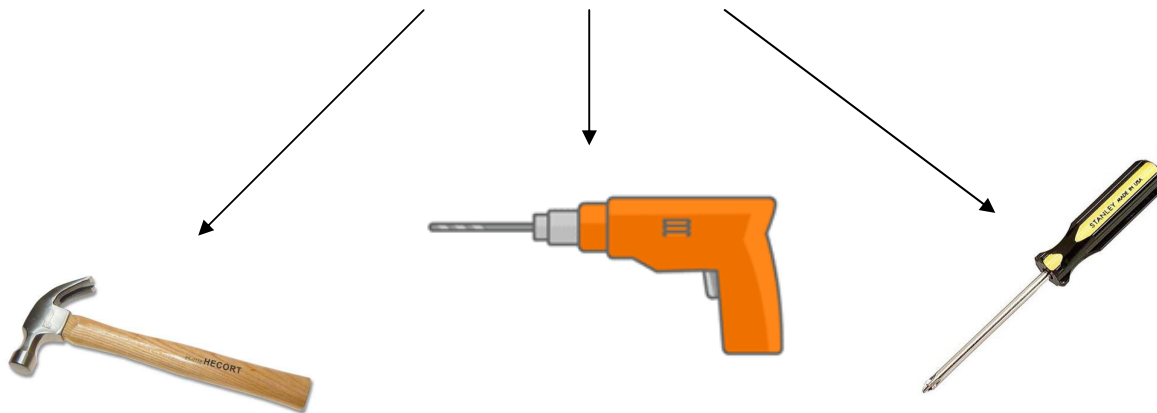
Muchas veces se asocia declaratividad con expresividad o claridad, por el momento digamos que:

- no siempre una solución declarativa es más clara que otra no declarativa
- que son conceptos diferentes. Decimos que una solución es más expresiva que otra si resuelve un problema en forma más clara y menos compleja (y por consiguiente es más fácil de entender). Por otro lado una solución es más declarativa que otra en tanto se tiene menor grado de información sobre los detalles algorítmicos de cómo se resuelve y donde la idea de secuencia pierde relevancia.

Una aclaración importante: los estilos y los paradigmas están en la cabeza de los programadores, los lenguajes son herramientas que hacen más fácil plasmarlos. Por supuesto, un lenguaje como Haskell facilita hacer una solución con estilo declarativo. PDP va a permitir abrir la mente a nuevas formas de programar.

El objetivo

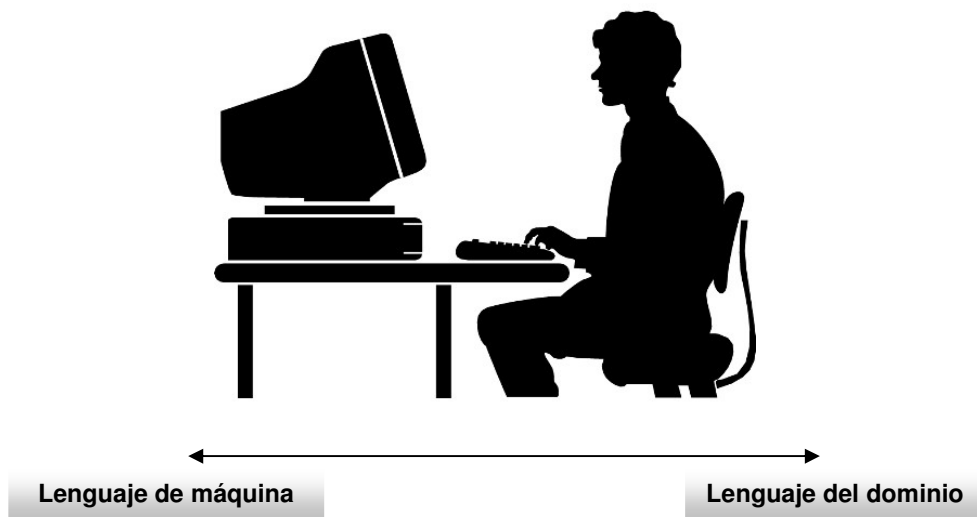
Después de la cursada vamos a poder, como profesionales, elegir las herramientas de programación que más nos convenga para desarrollar una aplicación.



Sí, y eso no dependerá de la tecnología que usemos comercialmente... los conceptos que vemos están en los lenguajes de programación desde hace 40 años...

Volviendo sobre los lenguajes

Retomamos la relación que tienen el hombre y la máquina:

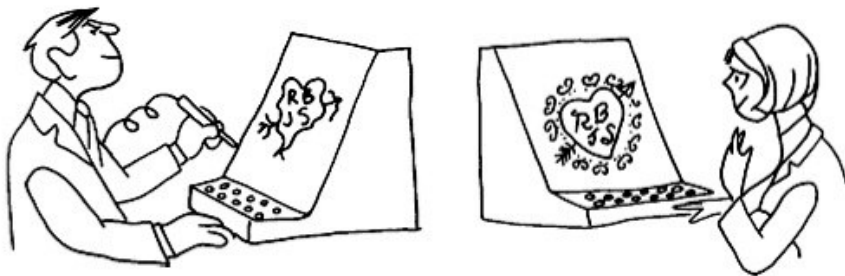


El gap semántico es la distancia entre el lenguaje de la persona que necesita la solución y el lenguaje que termina implementándose en la máquina.

Desde que comenzó la programación, los lenguajes han ido incorporando abstracciones de mayor nivel, eso permitió:

- acercar los lenguajes de alto nivel al dominio del problema
- tener mayor expresividad (decir lo mismo con menos esfuerzo, ser más claro en lo que digo)
- cambiar el skill de las personas (la división entre perfiles funcionales y técnicos es cada vez más difusa, no alcanza con tener gente que hable con el usuario y gente que programe, es importante poder contar con profesionales que sean un mix de ambos mundos)

Más aún, la idea original de la interacción hombre-máquina le está dando paso a lenguajes que son en realidad soporte para comunicar personas y para transmitir ideas...



A communication system should make a positive contribution to the discovery and arousal of interests.

¡Bienvenidos a Paradigmas! y disfruten de su viaje...