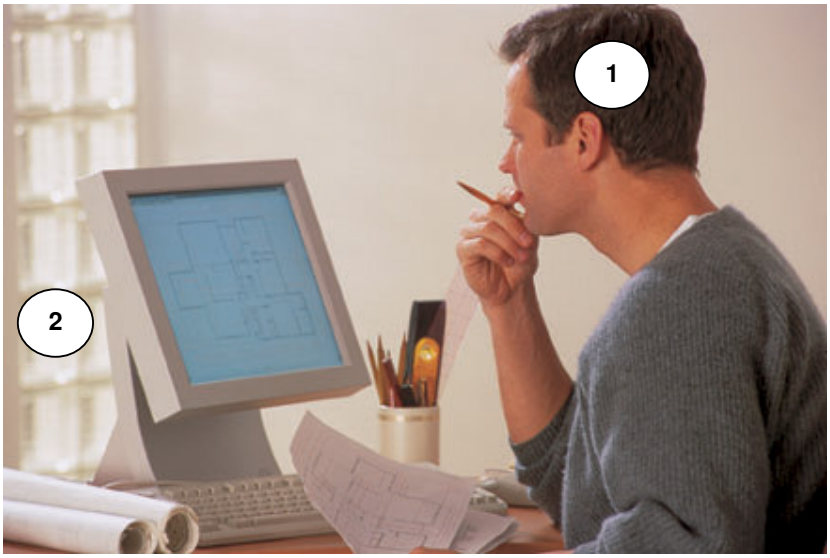
Script Integración 1 - Introducción a Paradigmas

¿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
- Tiene estructuras de control

De secuencia

```
Instrucción 1;
Instrucción 2;
```

De selección

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

De iteración

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

- Las variables se asocian a posiciones de memoria para guardar valores intermedios
- Puedo abstraer entidades mediante tipos de datos compuestos (agrupando así todas las características de un alumno, un cliente o una factura)
- Accedo a esos datos mediante funciones o procedimientos (el flujo principal del programa está en un procedimiento principal o cuerpo, que tiene llamadas a subprocedimientos)
- 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 (dando vuelta el martillo) y hasta para picar azulejos más o menos 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 (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** 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, pedirlas, esperar, pagarlas, comerlas.

Aquí se ve:

- Que la misma problemática puede pensarse desde distintos puntos de vista y tiene formas bien distintas de solucionarla.
- Nótese que en ambos marcos pienso en la heladera, pero la visión que tengo de la heladera es bien distinta para cada caso:
  - en uno es un contenedor de ingredientes de empanadas
  - en el 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

- expreso características del problema
- alguien termina resolviendo el algoritmo para la máquina (necesito de alguna “magia”, algún mecanismo externo)
- tengo menor control sobre el algoritmo (pero tengo que pensar en menos cosas)

Lo vemos con un ejemplo simple: obtener los elementos pares de una lista de números, lo implementamos en pseudocódigo y en Haskell, tirando directamente código Haskell.

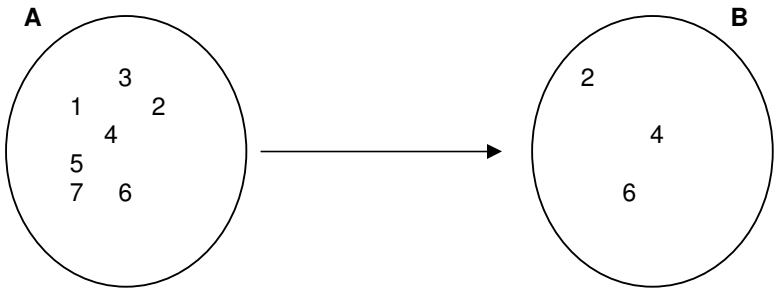
Pseudocódigo

```
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
```

¿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 Є A ^ x 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   xs = [ x | x <- xs , even x ]
```

¿Cuál sería el dominio e imagen de la función? De un conjunto de números a otro conjunto de números

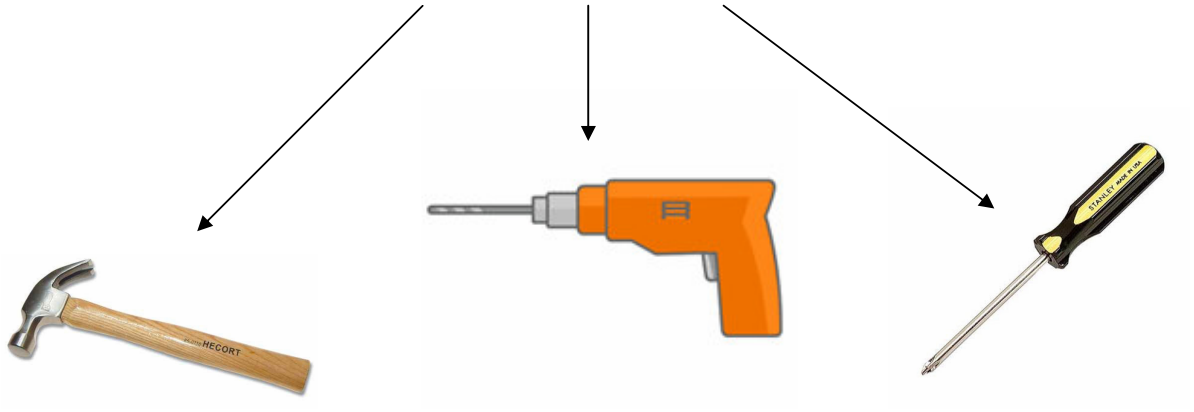
Como hemos visto la solución en pseudocódigo y en Haskell son bien diferentes.
¿En cuál solución tengo mayor control sobre el algoritmo?
¿En cuál solución tengo mayor foco en lo que tengo que resolver?

La solución planteada en pseudocódigo *tiende a* ser más procedimental y la solución en Haskell *tiende a* ser más declarativa. No siempre declaratividad es igual a claridad, pero sí tiene que ver con que no tengo una secuencia de pasos.

Una aclaración importante: los estilos y los paradigmas están en la cabeza de los programadores, los lenguajes son herramientas que me hacen más fácil plasmarlos. Por supuesto, un lenguaje como Haskell nos facilita hacer una solución con estilo declarativo. PDP nos 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

Si volvemos sobre la idea del hombre y la máquina:

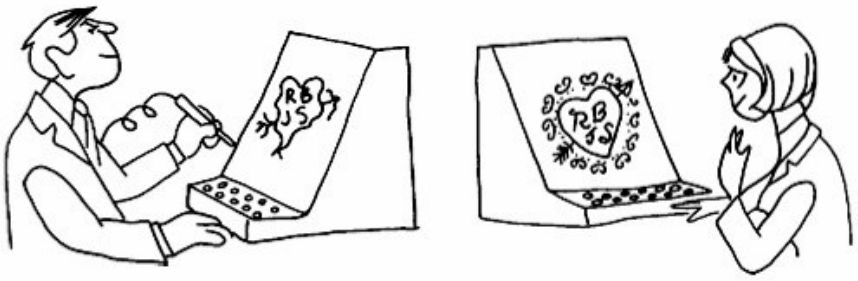
Lenguaje de máquina — Lenguaje del dominio

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 arrancó la historia de 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 (el funcional y el técnico puro ya no existen, o al menos no resultan productivos en los desarrollos actuales, ni siquiera con los sistemas enlatados)

Y más aún, la idea original de la interacción hombre-máquina ya trascendió para dar 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…