Programación imperativa

Ricardo Pérez López

IES Doñana, curso 2019/2020

Índice general

1. Modelo de ejecución

1.1. Máquina de estados

- La **programación imperativa** es un paradigma de programación basado en el concepto de **sentencia**.
- Un programa imperativo está formado por una sucesión de sentencias que se ejecutan en un orden determinado.
- Una sentencia es una instrucción del programa que lleva a cabo una de estas acciones:

  - **Cambiar el estado interno** del programa, normalmente mediante la llamada **sentencia de asignación**.
  - Cambiar el **flujo de control** del programa, haciendo que la ejecución se bifurque (*salte*) a otra parte del mismo.
- El modelo de ejecución de un programa imperativo es el de una **máquina de estados**, es decir, una máquina que va pasando por diferentes estados a medida que el programa va ejecutándose.
- En programación imperativa, el concepto de **tiempo** cobra mucha importancia: el programa actuará de una forma un otra según el estado en el que se encuentre, es decir, según el momento en el que estemos observando al programa.

  Eso significa que, ante los mismos datos de entrada, una función puede devolver **valores distintos en momentos distintos**.
- En programación funcional, en cambio, el comportamiento de una función no depende del momento en el que se ejecute, ya que siempre devolverá los mismos resultados ante los mismos datos de entrada.
- Eso significa que, para modelar el comportamiento de un programa imperativo, ya **no nos vale el modelo de sustitución**.

1.2. Secuencia de instrucciones

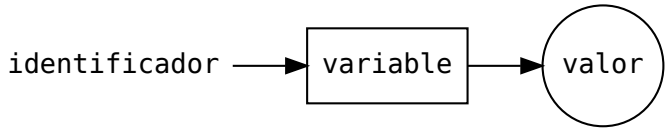
- Un programa imperativo es una **secuencia de instrucciones**, y ejecutar un programa es provocar los **cambios de estado** que dictan las instrucciones en el **orden** definido por el programa.
- Las instrucciones del programa van provocando **transiciones** entre estados, haciendo que la máquina pase de un estado al siguiente.
- Para modelar el comportamiento de un programa imperativo tendremos que saber en qué estado se encuentra el programa, para lo cual tendremos que seguirle la pista desde su estado inicial al estado actual.
- Eso básicamente se logra «ejecutando» mentalmente el programa instrucción por instrucción y llevando la cuenta de los valores ligados a sus identificadores.

2. Asignación destructiva

2.1. Variables

- Una **variable** es un lugar en la **memoria** donde se puede **almacenar un valor**.
- El valor de una variable **puede cambiar** durante la ejecución del programa.
- A partir de ahora, un identificador no se liga directamente con un valor, sino que tendremos:
  - Una **ligadura** entre un identificador y una **variable**.

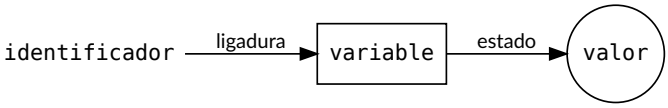
- **La variable almacena el valor**.



2.2. Estado

- La **ligadura** es la asociación que se establece entre un identificador y una variable.
- El **estado de una variable** es el valor que tiene una variable en un momento dado.

  Por tanto, el estado es la asociación que se establece entre una variable y un valor.



- Tanto las ligaduras como los estados pueden cambiar durante la ejecución de un programa imperativo.
- El **estado de un programa** es el conjunto de los estados de todas sus variables.

2.3. Sentencia de asignación

- El estado de una variable se cambia usando la **sentencia de asignación**.
- Es la misma instrucción que hemos estado usando hasta ahora para ligar valores a identificadores, pero ahora, en el paradigma imperativo, tiene otro significado:

```
x = 4
```

  Significa que el identificador `x` está ligado a una variable cuyo valor pasa a ser `4`.
- La asignación es **destructiva** porque al cambiar un valor a una variable se destruye su valor anterior. Por ejemplo, si ahora hacemos:

```
x = 9
```

  El valor de la variable a la que está ligada el identificador `x` pasa ahora a ser `9`, perdiéndose el valor `4` anterior.
- Por abuso del lenguaje, se suele decir:

  «se asigna el valor `9` a la variable `x`»

  en lugar de:

  «se asigna el valor `9` a la variable ligada al identificador `x`»
- Aunque esto simplifica las cosas a la hora de hablar, hay que tener cuidado, porque llegará el momento en el que podamos tener varios identificadores distintos ligados a la misma variable.

- Cada nueva asignación provoca un cambio de estado en el programa.
- En el ejemplo anterior, el programa pasa de estar en un estado en el que la variable `x` vale 4 a otro en el que la variable vale `9`.
- Al final, un programa imperativo se puede reducir a una **secuencia de asignaciones** realizadas en el orden dictado por el programa.
- Este modelo de funcionamiento está estrechamente ligado a la arquitectura de un ordenador: hay una memoria formada por celdas que contienen datos que pueden cambiar a lo largo del tiempo según dicten las instrucciones del programa que controla al ordenador.

2.4. Constantes

- En programación funcional no existen las variables y un identificador sólo puede ligarse a un valor (un identificador ligado no puede re-ligarse a otro valor distinto).
  - En la práctica, eso significa que un identificador ligado actúa como un valor constante que no puede cambiar durante la ejecución del programa.
  - El valor de esa constante es el valor al que está ligado el identificador.
- En programación imperativa, los identificadores se ligan a variables, que son las que realmente contienen los valores.
- Una **constante** en programación imperativa sería el equivalente a una variable cuyo valor no puede cambiar durante la ejecución del programa.
- Muchos lenguajes de programación permiten definir constantes, pero **Python no es uno de ellos**.
- En Python, una constante **es una variable más**, pero **es responsabilidad del programador** no cambiar su valor durante todo el programa.
- Python no hace ninguna comprobación ni muestra mensajes de error si se cambia el valor de una constante.
- En Python, por **convenio**, los identificadores ligados a un valor constante se escriben con todas las letras en **mayúscula**:

  ```
  PI = 3.1415926
  ```

  El nombre en mayúsculas nos recuerda que `PI` es una constante.
- Aunque nada nos impide cambiar su valor (cosa que debemos evitar):

  ```
  PI = 99
  ```

3. Efectos laterales

3.1. Transparencia referencial