Programación funcional I

Ricardo Pérez López

IES Doñana, curso 2019/2020

Índice general

1. El lenguaje de programación Python

1.1. Historia

1.2. Características principales

2. Modelo de ejecución

2.1. Modelo de ejecución

- Cuando escribimos programas (y algoritmos) nos interesa abstraernos del funcionamiento detallado de la máquina que va a ejecutar esos programas.
- Nos interesa buscar una *metáfora*, un símil de lo que significa ejecutar el programa.
- De la misma forma que un arquitecto crea modelos de los edificios que se pretenden construir, los programadores podemos usar modelos que *simulan* en esencia el comportamiento de nuestros programas.
- Esos modelos se denominan **modelos de ejecución**.
- Definición:

  Modelo de ejecución:

  Es una herramienta conceptual que permite a los programadores simular el funcionamiento de un programa sin tener que ejecutarlo directamente en el ordenador.

- Podemos definir diferentes modelos de ejecución dependiendo, principalmente, de:

  - El paradigma de programación utilizado (ésto sobre todo).
  - El lenguaje de programación con el que escribamos el programa.
  - Los aspectos que queramos estudiar de nuestro programa.

2.2. Modelo de sustitución

- En programación funcional, **un programa es una expresión** y lo que hacemos al ejecutarlo es **evaluar dicha expresión**, usando para ello las definiciones de operadores y funciones predefinidas por el lenguaje, así como las definidas por el programador y que el código fuente del programa.
- La **evaluación de una expresión**, en esencia, consiste en **sustituir**, dentro de ella, unas *subexpresiones* por otras que, de alguna manera, estén más cerca del valor a calcular, y así hasta calcular el valor de la expresión al completo.
- Por ello, la ejecución de un programa funcional se puede modelar como un **sistema de reescritura** al que llamaremos **modelo de sustitución**.
- La ventaja de este modelo es que no necesitamos recurrir a pensar que debajo de todo esto hay un ordenador con una determinada arquitectura *hardware*, que almacena los datos en celdas de la memoria principal, que ejecuta ciclos de instrucción en la CPU, que las instrucciones modifican los datos de la memoria, etc.
- Todo resulta mucho más fácil que eso.
- **Todo se reduce a evaluar expresiones**.

3. Expresiones

3.1. Evaluación de expresiones

- Ya hemos visto que la ejecución de un programa funcional consiste, en esencia, en evaluar una expresión.
- ***Evaluar* una expresión** consiste en determinar el **valor** de la expresión. Es decir, una expresión *representa* o **denota** un valor.
- En programación funcional, el significado de una expresión es su valor, y no puede ocurrir ningún otro efecto, ya sea oculto o no, en ninguna operación que se utilice para calcularlo.
- Una característica de la programación funcional es que **toda expresión posee un valor definido**, a diferencia de otros paradigmas en los que, por ejemplo, existen las *sentencias*, que no poseen ningún valor.
- Además, el orden en el que se evalúe no debe influir en el resultado.
- Podemos decir que las expresiones:
  - 3

  - $1 + 2$
  - $5 - 3$

  denotan el mismo valor (el número abstracto 3).
- Es decir: todas esas expresiones son representaciones diferentes del mismo ente abstracto.
- Lo que hace el sistema es buscar **la representación más simplificada o reducida** posible (en este caso, 3).
- Por eso a menudo usamos, indistintamente, los términos *reducir*, *simplificar* y *evaluar*.

3.1.1. Transparencia referencial

- En programación funcional, el valor de una expresión depende, exclusivamente, de los valores de sus sub-expresiones constituyentes.
- Dichas sub-expresiones, además, pueden ser sustituidas libremente por otras que tengan el mismo valor.
- A esta propiedad se la denomina **transparencia referencial**.
- En la práctica, eso significa que la evaluación de una expresión no puede provocar **efectos laterales**.
- Formalmente, se puede definir así:

  Transparencia referencial:

  Si $p = q$, entonces $f(p) = f(q)$.

3.1.2. Valores, expresión canónica y forma normal

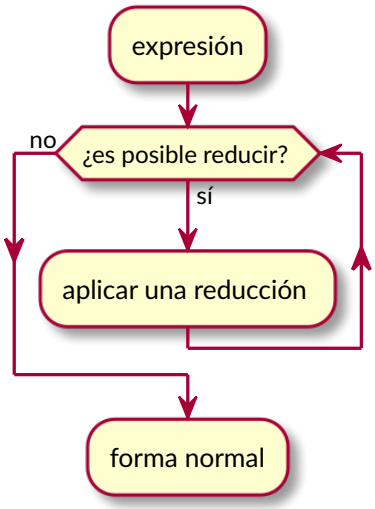
- Los ordenadores no manipulan valores, sino que sólo pueden manejar representaciones concretas de los mismos.
  - Por ejemplo: utilizan la codificación binaria en complemento a 2 para representar los números enteros.
- Pidamos que la **representación del valor** resultado de una evaluación sea **única**.
- De esta forma, seleccionemos de cada conjunto de expresiones que denoten el mismo valor, a lo sumo una que llamaremos **expresión canónica de ese valor**.
- Además, llamaremos a la expresión canónica que representa el valor de una expresión la **forma normal de esa expresión**.
- Con esta restricción pueden quedar expresiones sin forma normal.
- Ejemplo:
  - De las expresiones anteriores:
    * 3

    * $1 + 2$
    * $5 - 3$

    que denotan todas el mismo valor abstracto 3, seleccionamos una (la expresión 3) como la **expresión canónica** de ese valor.
  - Igualmente, la expresión 3 es la **forma normal** de todas las expresiones anteriores (y de cualquier otra expresión con valor 3).
  - Es importante no confundir el valor abstracto 3 con la expresión 3 que representa dicho valor.
- Hay valores que no tienen expresión canónica:
  - Las funciones (los valores de tipo *función*).
  - El número π no tiene representación decimal finita, por lo que tampoco tiene expresión canónica.
- Y hay expresiones que no tienen forma normal:
  - Si definimos $inf = inf + 1$, la expresión *inf* (que es un número) no tiene forma normal.
  - Lo mismo ocurre con $\frac{1}{0}$.

3.1.3. Formas normales y evaluación

- A partir de todo lo dicho, la ejecución de un programa será el proceso de encontrar su forma normal.
- Un ordenador evalúa una expresión (o ejecuta un programa) buscando su forma normal y mostrando este resultado.
- Con los lenguajes funcionales los ordenadores alcanzan este objetivo a través de múltiples pasos de reducción de las expresiones para obtener otra equivalente más simple.
- El sistema de evaluación dentro de un ordenador está hecho de forma tal que cuando ya no es posible reducir la expresión es porque se ha llegado a la forma normal.



3.2. Literales

3.3. Operaciones, operadores y operandos

3.3.1. Precedencia y asociatividad de operadores

3.4. Tipos de datos

3.4.1. Concepto

3.4.1.1. Tipo de un valor

3.4.1.2. Tipo de una expresión

3.4.2. Tipos de datos básicos

3.4.2.1. Números

3.4.2.1.1. Operadores aritméticos

3.4.2.2. Cadenas

5.6. Evaluación de expresiones con variables

5.7. Constantes

6. Documentación interna

6.1. Identificadores significativos

6.2. Comentarios

6.3. Docstrings

Respuestas a las preguntas

6.3.0.1. Respuestas a las preguntas

Bibliografía

Abelson, Harold, Gerald Jay Sussman, and Julie Sussman. 1996. *Structure and Interpretation of Computer Programs*. 2nd ed. Cambridge, Mass. : New York: MIT Press ; McGraw-Hill.

Blanco, Javier, Silvina Smith, and Damián Barsotti. 2009. *Cálculo de Programas*. Córdoba, Argentina: Universidad Nacional de Córdoba.