

Fundamentos de programación

Funciones como ciudadanos de primera clase y abstracción

carlos.andres.delgado@correounivalle.edu.co

Carlos Andrés Delgado S.

Facultad de Ingeniería. Universidad del Valle

Febrero de 2020

Fundamentos de programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

Abstracción
local

Funciones
anónimas

- 1 Funciones como ciudadanos de primera clase
- 2 Abstracción funcional
- 3 Abstracción de tipos
- 4 Abstracción local
- 5 Funciones anónimas

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

Abstracción
local

Funciones
anónimas

- 1 Funciones como ciudadanos de primera clase
- 2 Abstracción funcional
- 3 Abstracción de tipos
- 4 Abstracción local
- 5 Funciones anónimas

Funciones como ciudadanos de primera clase

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

Abstracción
local

Funciones
anónimas

Ejemplo

En algunos casos cuando se desarrollan funciones que reciben parámetros similares, se observa que se parecen.

Funciones como ciudadanos de primera clase

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

Abstracción
local

Funciones
anónimas

Ejemplo

Analicemos el siguiente caso:

- 1 Desarrolle una función para sumar dos números
- 2 Desarrolle una función para restar dos número
- 3 Desarrolle una función para multiplicar dos números
- 4 Desarrolle una función para dividir dos números

¿Que tienen en común?

Funciones como ciudadanos de primera clase

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

Abstracción
local

Funciones
anónimas

Ejemplo

```
;Contrato: sumar: numero,numero->numero  
(define (sumar a b)  
  (+ a b)  
)  
  
;Contrato: restar: numero,numero->numero  
(define (restar a b)  
  (- a b)  
)
```

Funciones como ciudadanos de primera clase

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

Abstracción
local

Funciones
anónimas

Ejemplo

```
;Contrato: multiplicar: numero,numero->numero
(define (multiplicar a b)  )

;Contrato: dividir: numero,numero->numero
(define (dividir a b)    )
(  (/ a b)  )
```

Funciones como ciudadanos de primera clase

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

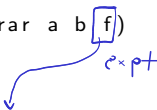
Abstracción
local

Funciones
anónimas

Ejemplo

Debido a que las funciones hacen una operación **con dos números**, podemos generalizar así:

```
;Contrato: operar: numero,numero, (numero,numero->numero)
->numero
(define (operar a b f)
  (f a b)
)
;Probar
(operar 1 2 (+))
(operar 2 3 (-))
(operar 4 5 (/))
(operar 2 2 (*))
```



Funciones como ciudadanos de primera clase

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

Abstracción
local

Funciones
anónimas

Ejemplo

Desarrollar un programa para cambiar de euros, dolares o yenes a pesos, la relación de precios es:

- $1 \text{ yen} = 1000 \text{ pesos}$
- $1 \text{ euro} = 3500 \text{ pesos}$
- $1 \text{ dolar} = 3000 \text{ pesos}$

Funciones como ciudadanos de primera clase

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

Abstracción
local

Funciones
anónimas

Ejemplo

Debido a que las funciones hacen una operación **con dos números**, podemos generalizar así:

```
;Contrato: euros: numero -> numero
(define (euros n)
  (* n 3500)
)

;Contrato: dolares: numero->numero
(define (dolares n)
  (* n 3000)
)

;Contrato: yenes: numero->numero
(define (yenes n)
  (* n 1000)
)
```

Funciones como ciudadanos de primera clase

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

Abstracción
local

Funciones
anónimas

Ejemplo

Debido a que las funciones hacen una operación **con dos números**, podemos generalizar así:

```
;Contrato: convertir: numero, (numero->numero) -> numero
(define (convertir n f)
      (f n)
)
;;Pruebas
(convertir 100 euros)
(convertir 200 dolares)
(convertir 500 yenes)
```

Funciones como ciudadanos de primera clase

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

Abstracción
local

Funciones
anónimas

Filtros

Usando funciones como ciudadanos de primera clase

- 1 Diseñe una función, que permita saber si un número es mayor, menor o igual que 5.
- 2 Diseñe una función, que permita convertir de dolares a yenes, pesos y euros. Utilice el ejemplo anterior.

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

Abstracción
local

Funciones
anónimas

1 Funciones como ciudadanos de primera clase

2 Abstracción funcional }

3 Abstracción de tipos

4 Abstracción local

5 Funciones anónimas

Definición

Muchos problemas manejan la misma información, pero se piden acciones diferentes. Un buen ejemplo de esto son los filtros. Con esto nos evitamos el **copiar y pegar**.

¿Que es un filtro?

Un filtro es una función que toma una lista con n elementos y retorna los k elementos que cumplen con una función predicado f . Ejemplo, de una lista de números filtre los que son pares.

Filtros

```
;Contrato: filtrar-pares: lista-numeros -> lista-numeros
(define (filtrar-pares l)
  (cond
    [(empty? l) empty]
    [(even? (first l)) (cons (first l) (
      filtrar-pares (rest l)))]
    [else (filtrar-pares (rest l))])
  )
```

Si cumple con
la condición
incluye
el primer
elemento

Si no cumple la condición lo descarta

Filtros

Y ahora si quiero filtrar impares

```
;Contrato: filtrar-impares: lista-numeros ->
           lista-numeros
(define (filtrar-impares l)
  (cond
    [(empty? l) empty]
    [(odd? (first l)) (cons (first l) (
      filtrar-impares (rest l)))]
    [else (filtrar-impares (rest l))])
  )
)
```

Filtros

Y si quiero los mayores que 5

```
;Contrato: filtrar-mayores5: lista-numeros ->
           lista-numeros
(define (filtrar-mayores5 l)
  (cond
    [(empty? l) empty]
    [(> (first l) 5) (cons (first l) (
      filtrar-mayores5 (rest l)))]
    [else (filtrar-mayores5 (rest l))])
  )
)
```

Filtros

Y si quiero los menores que 3

```
;Contrato: filtrar-menores3: lista-numeros ->
           lista-numeros
(define (filtrar-menores3 l)
  (cond
    [(empty? l) empty]
    [(<= (first l) 3) (cons (first l) (
      filtrar-menores3 (rest l)))]
    [else (filtrar-menores3 (rest l))])
  )
)
```

Filtros

¿Como podemos abstraer esto?

```
;Contrato: menor-que-3: numero -> bool
(define (menor-que-3 n)
  (< n 3)
)
;Contrato: mayor-que-5: numero -> bool
(define (mayor-que-5 n)
  (> n 5)
)
```

Filtros

¿Como podemos abstraer esto?

```
;;Contrato: filtrar-num: lista-numeros, (numero->bool) ->
  lista-numeros
(define (filtrar-num l f)
  (cond
    [(empty? l) empty]
    [(f (first l)) (cons (first l) (filtrar-num (
      rest l) f))]
    [else (filtrar-num (rest l) f)]
  )
)

;;pruebas
(filtrar-num (list 1 2 3 4 5) odd?)
(filtrar-num (list 1 2 3 4 5) even?)
(filtrar-num (list 1 2 3 4 5) menor-que-3)
(filtrar-num (list 1 2 3 4 5) mayor-que-5)
```

Funciones predicado
→ booleano
porque los vamos a
usar en un condicional

Filtros

Diseñe utilizando abstracción funcional:

- 1 Una función que filtre una lista de símbolos de acuerdo: si es igual a un símbolo, si es diferente a un símbolo, si es mayor que un símbolo o si es menor que un símbolo **Pista:** `symbol <?`, para usar esta función en la primera línea del código escriba **(require racket/base)**
- 2 Una función que filtre una lista de estructuras, tipo producto, un producto tiene un nombre que es un símbolo y un valor que es un número. Se requiere filtrar un producto si su valor es mayor a un valor, si es igual a un valor o si es menor que un valor.

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

Abstracción
local

Funciones
anónimas

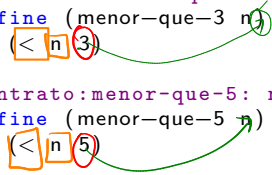
- 1 Funciones como ciudadanos de primera clase
- 2 Abstracción funcional
- 3 Abstracción de tipos
- 4 Abstracción local
- 5 Funciones anónimas

Definición

Muchas veces se tienen funciones que se comportan de forma similar, pero se diferencian en los datos que reciben. Miremos el siguiente ejemplo:

```
;Contrato: menor-que-3: numero -> bool
(define (menor-que-3 n)
  (< n 3))

;Contrato: menor-que-5: numero -> bool
(define (menor-que-5 n)
  (< n 5))
```



¿Que tienen en común?

Abstracción de tipos

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

Abstracción
local

Funciones
anónimas

Definición

Esto lo podemos reescribir esta forma:

```
;Contrato: menor-que-numero: numero, numero -> numero
(define (menor-que-numero a b)
  (< a b)
)
;;Pruebas
(menor-que-numero 1 2)
(menor-que-numero 2 1)
```

¿Que más podemos abstraer?

Definición

Utilizando abstracción funcional

```
;Contrato: comparar-numeros: numero, numero, (numero,  
           numero->bool)-> numero  
(define (comparar-numeros a b f)  
  (f a b))  
  
(comparar-numeros 1 2 <)  
(comparar-numeros 2 1 <)  
(comparar-numeros 1 2 =)  
(comparar-numeros 2 1 >)
```

Definición

Diseñar una función que reciba una lista de números y realice el filtro de:

- 1 Números menores que un x
- 2 Números mayores que un y
- 3 Números mayores que un z y que son pares

Definición

Empecemos por las funciones:

```
;Contrato comparar-num: numero, numero, (numero, numero->
  bool) -> bool
(define (comparar-num a b (f))
  (f a b))

;Contrato comparar-num-ypares: numero, numero, (numero,
  numero->bool) -> bool
(define (comparar-num-ypares a b f)
  (and (f a b) (even? a)))
```

Filtros

¿Cómo podemos abstraer esto?

```
;Contrato: filtrar-num: lista-numeros, numero, (numero,
           numero->bool), (numero, numero, (numero,numero->bool)
           -> bool) -> lista-numeros
(define (filtrar-num l n g f)
  (cond
    [(empty? l) empty]
    [(f (first l) n g) (cons (first l) (filtrar-num
                               (rest l) n g f))]
    [else (filtrar-num (rest l) n g f)])
)

;;pruebas
(filtrar-num (list 1 2 3 4 5) 5 > comparar-num)
(filtrar-num (list 1 2 3 4 5) 5 < comparar-num)
(filtrar-num (list 1 2 3 4 5) 4 > comparar-num-ypares)
(filtrar-num (list 1 2 3 4 5) 4 < comparar-num-ypares)
```

Fundamentos de programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

Abstracción
local

Funciones
anónimas

- 1 Funciones como ciudadanos de primera clase
- 2 Abstracción funcional
- 3 Abstracción de tipos
- 4 Abstracción local**
- 5 Funciones anónimas

Definición

¡También dentro de las funciones hay cosas repetidas!

```
;Contrato: filtrar-num: lista-numeros, (numero->bool) ->
  lista-numeros
(define (filtrar-num l f)
  (cond
    [(empty? l) empty]
    [(f (first l)) (cons (first l) (filtrar-num (rest l) f))]
    [else (filtrar-num (rest l) f)])
  )
)
```

Definición

Si observan:

```
(filtrar-num (rest l) f)
```

Está repetido

Definición

Podemos usar local para evitar líneas de código repetidas así:

```
;Contrato: filtrar-num: lista-numeros, (numero->bool) ->
  lista-numeros
(define (filtrar-num l f)
  (cond
    [(empty? l) empty]
    [(list? l)
     (local
      [
        (define resto (filtrar-num (rest l) f))
      ]
      (cond
        [(f (first l)) (cons (first l) resto)]
        [else resto]
      )
     )
    ]
  )
)
```

Filtros

En este ejercicio, usemos abstracción funcional, de tipos y local.

- 1 Una función que filtre una lista de símbolos de acuerdo: si es igual a un símbolo, si es diferente a un símbolo, si es mayor que un símbolo o si es menor que un símbolo **Pista:** *symbol <?*
- 2 Una función que filtre una lista de estructuras, tipo producto, un producto tiene un nombre que es un símbolo y un valor que es un número. Se requiere filtrar un producto si su valor es mayor a un valor, si es igual a un valor o si es menor que un valor.

Fundamentos de programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

Abstracción
local

Funciones
anónimas

- 1 Funciones como ciudadanos de primera clase
- 2 Abstracción funcional
- 3 Abstracción de tipos
- 4 Abstracción local
- 5 Funciones anónimas**

Funciones anónimas

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

Abstracción
local

Funciones
anónimas

Definición

¿Es posible evitar nombrar funciones?, es decir crearlas y después usarlas.

La respuesta es sí.

Definición

Se puede utilizar la expresión **lambda**

```
;;(lambda (<entradas> <salida>)  
  (lambda (x) +(x 2)))  
;;Prueba en tu DrRacket  
((lambda (x y) (+ x y)) 1 2)
```

Importante: Hora de cambiar al ¡max level!. Selecciona
Estudiante avanzado :D

Definición

¿Como la usamos dentro de funciones?. Pensemos en este caso:

```
;Contrato: filtrar-num: lista-numeros, (numero->bool) ->
           lista-numeros
(define (filtrar-num l f)
  (cond
    [(empty? l) empty]
    [(list? l)
     (local
      [
        (define resto (filtrar-num (rest l) f))
      ]
      (cond
        [(f (first l)) (cons (first l) resto)]
        [else resto]
      )
     )
    ])
  )))
```

Definición

¿Como la usamos dentro de funciones?. Pensemos en este caso:

```
(filtrar-num (list 1 2 3 4 5) (lambda (x) (< x 5)))  
(filtrar-num (list 1 2 3 4 5) (lambda (x) (> x 5)))  
(filtrar-num (list 1 2 3 3 4) (lambda (x) (and (odd? x)  
(< x 3)))))
```

Definición

Para que más sirva. Existe en Dr Racket, la función Map que permite aplicar una función a una lista. Prueba:

```
(map (lambda (x) (* x x)) (list 1 2 3 4))  
(map (lambda (x) (+ x x)) (list 1 2 3 4))  
(map (lambda (y) (+ y 2)) (list 1 2 3 4))
```


Filtros

- 1 Una función que filtre símbolos. Se ingresa una lista de símbolos, una función y un símbolo. La función que ingresa recibe dos símbolos y retorna un booleano, donde se valida si es un símbolo menor, mayor o igual.
- 2 Una función que filtre números. Se ingresa una lista de números, una función y un número. La función que ingresa recibe dos números y retorna un booleano, se valida si los dos números son iguales o diferentes.

Filtros

- 1 Una función que filtre personas por edad. Se ingresa una lista de personas(nombre y edad) y una función. La función que ingresa recibe una persona y un número, y retorna un booleano, se valida si la edad de la persona es menor, mayor o igual.

Filtros

- 1 Una función que cuente los numeros que pasen un filtro

Solución

```
(define (contarFiltro lista funcion)
  (cond
    [(empty? lista) empty]
    [(list? lista)
     (local
      (
        (define (contarFiltro—acc lista funcion acc)
          (cond
            [(empty? lista) acc]
            [(function (first lista)) (contarFiltro—acc (rest lista) funcion
                                                         (+ acc 1))]
            [else (contarFiltro—acc (rest lista) funcion acc)])
          )))
      (contarFiltro—acc lista funcion 0)
    )]))

;; Prueba
(contarFiltro (list 1 2 3 4 5 6 8)
              (lambda (n)(cond[(> n 2) true][else false])))
```

Fundamentos
de
programación

Carlos Andrés
Delgado S.

Funciones
como
ciudadanos de
primera clase

Abstracción
funcional

Abstracción de
tipos

Abstracción
local

Funciones
anónimas

VAMO A

