

# Fundamentos de programación

## Procesamiento de datos simples I

Facultad de Ingeniería. Universidad del Valle

Mayo 2019



# Contenido

- 1 Receta de diseño de programas
- 2 Elementos de un programa
- 3 Expresiones condicionales

# Contenido

1 Receta de diseño de programas

2 Elementos de un programa

3 Expresiones condicionales

# Receta de diseño de programas

## Definición

En este punto vamos a establecer una metodología para diseñar nuestros programas. En otras palabras una serie de pasos para solucionar un problema utilizando un computador.

# Receta de diseño de programas

## Receta de diseño

Fase	Objetivo	Actividad
Autor(es)	Indicar quien crea el programa	Especificar autores y fecha de creación
Contrato	<ol style="list-style-type: none"><li>1 Dar nombre a su función</li><li>2 Describir el propósito</li></ol>	<ol style="list-style-type: none"><li>1 Escoger un nombre que se ajuste al problema</li><li>2 Estudiar el problema y determinar entradas y salidas</li><li>3 Formular el contrato</li></ol>

# Receta de diseño de programas

## Receta de diseño

Fase	Objetivo	Actividad
Ejemplos	Caracterizar la relación entrada-salida	<ol style="list-style-type: none"><li>1 Buscar ejemplos del problema</li><li>2 Crear ejemplos</li><li>3 Validar resultados</li></ol>
Cuerpo	Definir la función	Formular en un lenguaje de programación
Pruebas	Buscar errores	Aplicar varias entradas y observar resultados

Variable.

$x = 5$

$x$

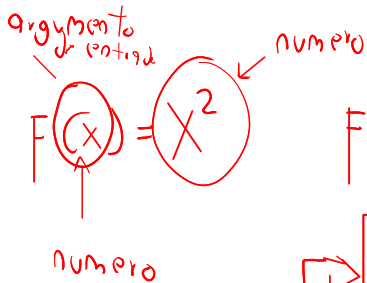
(define x 5)

(define y 8)

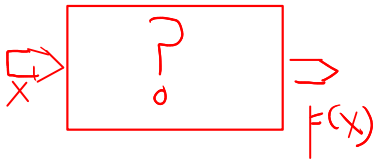
(+ x y)

A handwritten mathematical expression in blue ink: 
$$8 \sqrt[3]{5\sqrt{7} + 8^{\frac{1}{2}} + 7^2}$$
 The entire expression is enclosed in a red oval. A red arrow points from the number 9 above to the square root symbol. Below the main expression, there is another red oval containing the text  $\sqrt[3]{3^2 + \frac{1}{c} (3^2 + 8^2 + \sqrt{9})}$ . A red arrow points from the 1 in the numerator to the 9 in the square root of the denominator.

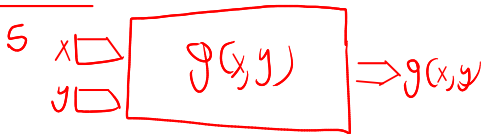
A handwritten mathematical expression in blue ink: 
$$8 \sqrt{\frac{9}{6}}$$



$$F: \underline{\text{numero}} \rightarrow \underline{\text{numero}}$$



$$g(x, y) = \frac{x^2 + 6y}{5}$$



$$g: \text{numero}, \text{numero} \rightarrow \underline{\text{numero}}$$

contrato

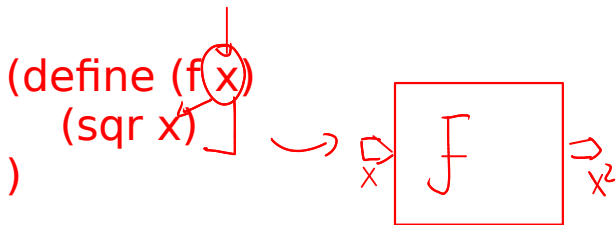


$$h(x, y, z) = \log$$

$$h: \text{numero}, \text{numero}, \text{numero} \rightarrow \text{Simbolo}$$

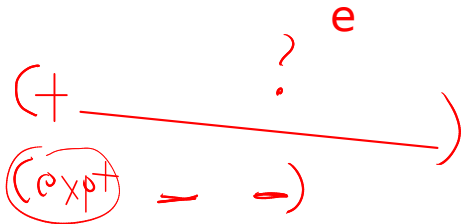
$$i(x, y, z, w, q, b, c) = \frac{x^2 + y^2}{z}$$

i: numero, numero, numero, numero, numero, numero, numero  $\rightarrow$  numero



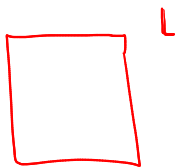
`(f 5)`

`(f 8)`



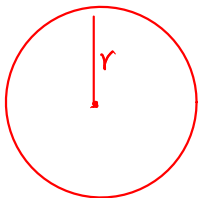
$$F(a, b, c, d, e) = \frac{a^2 + \sqrt{\frac{b}{c}} + d^2}{\sqrt[3]{d^8 - \frac{9}{c + \sqrt{e}}}}$$

$F: \text{numero}, \text{numero}, \text{numero}, \text{numero}, \text{numero} \rightarrow \text{numero}$



$$\text{area}(l) = l^2$$

$$\text{area} : \text{numero} \rightarrow \text{numero}$$



$$\text{area } C : \text{numero} \rightarrow \text{numero}$$

$$\pi \times r^2$$

# Receta de diseño de programas

## Receta de diseño

### En nuestro lenguaje de aprendizaje

```
;; Autor: <nombre>
;; Fecha de creación: <día>
;; Contrato: <nombre> <entrada> -> <salida>
;; Propósito Una breve descripción del problema
;; Ejemplo: Ante una entrada dada debe producir una salida
;; Definición
<codigo>
;; Pruebas
<aqui pruebas>
```

# Receta de diseño de programas

## Receta de diseño

Vamos a mirar algunas recetas de diseño:

- 1 Un programa que reciba dos números y retorne su suma
- 2 Un programa que reciba tres números y retorne su multiplicación
- 3 Un programa que reciba cuatro números y retorna la suma de sus cuadrados

# Receta de diseño de programas

## Receta de diseño

Un programa que reciba dos números y retorne su suma

```
;; Autor: Docente curso Fundamentos de Programación
;; Fecha de creación: 20-Agosto-2016
;; Contrato: suma: numero,numero -> numero
;; Propósito: Este programa recibe dos números y retorna su suma
;; Ejemplo: (suma 5 2) retorna 7, (suma -2 3) retorna 1
;; Definición
<codigo>
;; Pruebas
(check-expect (suma 5 2) 7)
(check-expect (suma -2 3) 1)
```

**check-expect** es una función que nos permite validar si el programa funciona correctamente, luego la veremos en acción.

# Receta de diseño de programas

## Receta de diseño

Diseñar una función que reciba tres números y retorne su multiplicación

```
;; Autor: Docente curso Fundamentos de Programación
;; Fecha de creación: 27-Agosto-2018
;; Contrato: multiplica3: numero,numero,numero -> numero
;; Propósito: Este programa recibe 3 número y retorna la multiplicación
               entre ellos
;; Ejemplo: (multiplica3 1 2 3) -> 6, (multiplica 2 4 5) -> 11
;; Definición
<codigo>
;; Pruebas
(check-expect (multiplica3 1 2 3) 6)
(check-expect (multiplica 2 4 5) 11)
```



# Receta de diseño de programas

## Receta de diseño

Diseñe una función que reciba cuatro números y retorna la suma de sus cuadrados.

```
;; Autor: Docente curso Fundamentos de Programación
;; Fecha de creación: 27-Agosto-2018
;; Contrato: sumaCuadrados: numero,numero,numero,numero -> numero
;; Propósito: Este programa recibe 4 número y retorna la suma de los
               cuadrados de ellos
;; Ejemplo: (sumaCuadrados 1 2 3 4) -> 30, (sumaCuadrados 2 3 4 5) -> 54
;; Definición
<codigo>
;; Pruebas
(check-expect (sumaCuadrados 1 2 3 4) 30)
(check-expect (sumaCuadrados 2 3 4 5) 54)
```

# Receta de diseño de programas

## Receta de diseño

Ahora inténtalo:

- 1 Una función **resta2** que reciba 2 números y retorna la resta del primero con el segundo.
- 2 Una función **multiplicación5** que reciba 5 números y retorna su multiplicación.

# Receta de diseño de programas

## Receta de diseño

Un programa que reciba una lista de números y retorne la suma de los elementos

```
;; Autor: Docente curso Fundamentos de Programación
;; Fecha de creación: 20-Agosto-2016
;; Contrato: resta2: numero, numero -> numero
;; Propósito: Este programa recibe dos números y retorna la resta del
               primero con el segundo
;; Ejemplo: (resta2 1 3) -> -2, (resta2 1 4) -> -3
;; Definición
<código>
;; Pruebas
(check-expect (resta2 1 3) -2)
(check-expect (resta2 1 4) -3)
```

# Receta de diseño de programas

## Receta de diseño

Un programa que recibe dos números y retorna una lista con estos dos números elevados al cuadrado.

```
;; Autor: Docente curso Fundamentos de Programación
;; Fecha de creación: 20-Agosto-2016
;; Contrato: multiplicación5: numero,numero,numero,numero, numero ->
            numero
;; Propósito: Este programa recibe 5 números y retorna su multiplicación
;; Ejemplo: (multiplicación5 1 2 3 4 5)-> 120, multiplicación5 2 4 6 8
            10) -> 3840
;; Definición
<código>
;; Pruebas
(check-expect (multiplicación5 1 2 3 4 5) 120)
(check-expect (multiplicación5 1 2 3 4 5) 3840)
```

# Receta de diseño de programas

## ¿Porque seguir estos pasos?

Estamos aprendiendo a programar, cuando cojamos práctica en los semestres que vienen esta receta será implícita para ustedes.

- 1 Les da idea de que datos esperar de entrada (proporcionados)
- 2 Da idea de que datos esperar de salida
- 3 Ayuda a otras personas que entiendan su código

# Contenido

1 Receta de diseño de programas

2 Elementos de un programa

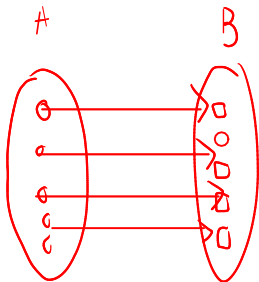
3 Expresiones condicionales

# Elementos de un programa

## Definición

Los elementos básicos que se tienen en un programa son:

- 1 **Variables:** Estas nos permiten almacenar información que va ser utilizada posteriormente
- 2 **Funciones:** Se utilizan para el procesamiento de información, en otras palabras reciben una información y retornan un resultado. Se componen así:
  - Reciben un conjunto de valores. **Dominio**
  - Produce un conjunto de valores. **Rango**
  - Cada valor del dominio es asociado con **uno y sólo uno del rango**



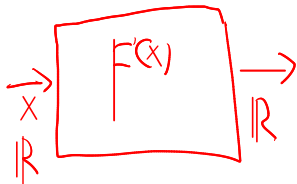
Domina

Codominio  
Rango

$$f(x) = \sqrt{x^3}$$

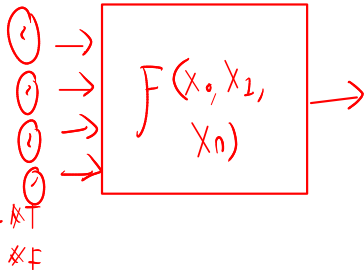
$$\mathbb{R} \rightarrow \mathbb{R}$$

$$\text{num} \times \text{num} \times \dots \rightarrow \text{num}$$



1

Simbolos  
tholo  
Cadenas texto  
" " " " " "  
numeros  
IR  
Booleanos





# Elementos de un programa

## Variables

En nuestro lenguaje de aprendizaje:

```
(define miVariable 4)  
(define miVariableB "hola")
```

Ejecute este programa y muestre los resultados de consultar:

- 1 **miVariable**
- 2 **miVariableB**

# Elementos de un programa

## Palabras reservadas

Hay nombres de variables que no puedes utilizar, a esto se le conoce como **palabras reservadas** los cuales son funciones que provee el lenguaje, intenta:

```
(define sqrt 2)  
(define define "hola")
```

Esto es algo común en todos los lenguajes de programación. En el caso del Dr Racket va generar un error porque no se puede definir de nuevo, sin embargo en algunos lenguajes de programación podrías sobrescribir funciones o procedimientos y tener errores inesperados.

# Elementos de un programa

## Variables

Probemos lo aprendido, en nuestro lenguaje de aprendizaje genere variables que consulten:

- 1 Una variable llamada A que contenga el valor 5
- 2 Una variable llamada B que contenga el valor 8
- 3 Una variable llamada C que contenga el valor 10
- 4 Una variable llamada D que contenga el resultado  $A+B-C$

Cree las variables y consulte su valor

# Elementos de un programa

## Funciones

En nuestro lenguaje de aprendizaje:

```
;; Autor: Docente curso Fundamentos de Programación
;; Fecha de creación: 20-Agosto-2016
;; Contrato: funcion: numero, numero -> numero]
;; Propósito: Este programa recibe dos números y retorna su suma
;; Ejemplo: (funcion 8 6) retorna 14
;; Definición
(define (funcion a b) (+ a b))
;; Pruebas
(check-expect (funcion 8 6) 14)
```

Ejecute este programa y muestre los resultados de consultar:  
(funcion 1 2)

nombre

Entrada

# Elementos de un programa

## Funciones

Al principio no es fácil entender las funciones, este es un concepto fundamental en la programación. Sin ellas no podemos hacer mucho, las ventajas que nos ofrecen son:

- 1 Cuando requerimos realizar operaciones repetidas, se puede diseñar una función y llamarla las veces que se requiera
- 2 Una función puede recibir cero o más entradas. A estos se le conocen como argumentos.

Handwritten notes illustrating function calls and arguments:

Left side (arguments):

- (F 7 6)
- (F 5 6)
- (F 8 7)

Middle (calculations and function call):

- $\sqrt{5+6}$  with a red arrow pointing to a box containing 'f'.
- $\sqrt{8+7}$  with a red arrow pointing to the same box containing 'f'.
- Below the box:  $9 \rightarrow$  and  $6 \rightarrow$  pointing to the box, and  $\rightarrow s$  pointing away from the box.

Right side (calculation):

- $\sqrt{7+6}$  with red arrows pointing to the numbers 7 and 6.

# Elementos de un programa

$$(6 + c) - 9$$

## Funciones

Miremos otra función:

```
;; Autor: Docente curso Fundamentos de Programación
;; Fecha de creación: 20-Agosto-2016
;; Contrato: otraFuncion: numero, numero -> numero
;; Propósito: Este programa recibe tres números a,b y c y retorna el
               resultado de b+c-a
;; Ejemplo: (otraFuncion 1 8 6) retorna 13
;; Definición
(define (otraFuncion a b c) (- (+ b c) a))
;; Pruebas
(check-expect (otraFuncion 1 8 6) 13)
```

Ejecute este programa y muestre los resultados de consultar:  
**(otraFuncion 1 2 3)**

# Elementos de un programa

## Funciones

Una función llamada funcionA que reciba 3 números y retorne su suma

```
;; Autor: Docente curso Fundamentos de Programación
;; Fecha de creación: 20-Agosto-2016
;; Contrato: funcionA: numero, numero, numero -> numero
;; Propósito: Este programa recibe tres números y retorna su suma
;; Ejemplo: (funcionA 1 8 6) retorna 15
;; Definición
(define (funcionA a b c) (+ a b c))
;; Pruebas
(check-expect (funcionA 1 8 6) 15)
```

# Elementos de un programa

## Funciones

Una función llamada funcionB que reciba 4 números y retorne su suma

```
;; Autor: Docente curso Fundamentos de Programación
;; Fecha de creación: 20-Agosto-2016
;; Contrato: funcionB: numero, numero, numero, numero -> numero
;; Propósito: Este programa recibe cuatro números y retorna su suma
;; Ejemplo: (funcionB 1 8 6 3) retorna 18
;; Definición
(define (funcionB a b c d) (+ a b c d))
;; Pruebas
(check-expect (funcionB 1 8 6 3) 18)
```



# Elementos de un programa

## Funciones

Vamos a algo más aplicado. Diseñe un programa que calcule el área de un círculo. **Recuerde utiliza la receta de diseño**

1) Autor y Fecha

~~2) Contrato~~

3) Descripción

4) Ejemplos

5) Codificación

6) Pruebas

Contrato: nombre función  $\langle e1 \rangle \langle e2 \rangle \dots \langle en \rangle \rightarrow \langle S \rangle$

# Elementos de un programa

## Funciones

- 1 Nombre del programa **area-circulo**
- 2 ¿Que recibe?: Un número indicando el radio  $r$  del circulo
- 3 ¿Que retorna?: Un número  $\pi * r^2$ . Asumiremos que  $\pi = 3.14$ . Dr Racket tiene un valor más aproximado en la variable **pi**, pero no podremos verificar con check-expect, ya que no es posible verificar expresiones irracionales.

# Elementos de un programa

## Funciones

Una función llamada `funcionB` que reciba 4 números y retorne su suma

```
;; Autor: Docente curso Fundamentos de Programación
;; Fecha de creación: 20-Agosto-2016
;; Contrato: area-circulo: numero -> numero
;; Propósito: Este calcula el área de un círculo
;; Ejemplo: (area-circulo 5) retorna 78.5,
;; (area-circulo 10) retorna 314
;; Definición
(define (area-circulo r) (* 3.14 (expt r 2)))
;; Pruebas
(check-expect (area-circulo 5) 78.5)
(check-expect (area-circulo 10) 314)
```

Pi

llamado

# Elementos de un programa

## Funciones auxiliares

Es común en la programación, que para resolver un problema se requiera más de una función, supongamos que necesitamos resolver la expresión  $(+ a (/ b (+ c a)))$  en dos pasos.

```
;; Autor: Docente curso Fundamentos de Programación
;; Fecha de creación: 27-Agosto-2018
;; Contrato: resolverExpresion: numero,numero,numero -> numero
;; Propósito: Resuelve la expresión ((c + a)*a + (b / (c + a) ))
;; Ejemplo: (resolverExpresion 1 2 3)->4.5, (resolverExpresion 2 4
6)->16.5
```

```
(define (resolverExpresion a b c)
  (+ (* (+ c a) a) (/ b (+ c a))))
```

```
(check-expect (resolverExpresion 1 2 3) 4.5)
(check-expect (resolverExpresion 2 4 6) 16.5)
```

¿Que ven repetido?

# Elementos de un programa

## Funciones auxiliares

```
;; Autor: ....  
(define (resolverExpresion a b c)  
  (+ (* (suma c a) a) (/ b (suma c a))))  
  
;; Autor: Docente curso Fundamentos de Programación  
;; Fecha de creación: 27-Agosto-2018  
;; Contrato: suma: numero,numero,numero -> numero  
;; Propósito: Resuelve la expresión ((c + a)*a + (b / (c + a) ))  
;; Ejemplo: (suma 1 2)->3, (suma 2 4)->6  
(define (suma b c)  
  (+ b c))  
(check-expect (resolverExpresion 1 2 3) 4.5)  
(check-expect (resolverExpresion 2 4 6) 16.5)
```

$$F(x) = X \times g(x)$$

$$g(x) = x + 3$$

$$F(8) = 8 \times g(8)$$

$$8 \times 11 = 88$$

$$F_2(x) = x^2 \times g\left(\frac{x}{2}\right)$$

$$g_2(x) = 6 + x$$

$$F_2(8)$$

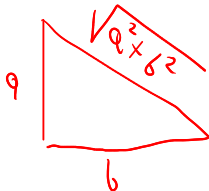
$$64 \times g(4)$$

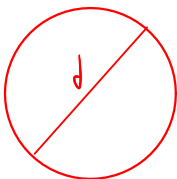
$$64 \times 10 = 640$$

# Elementos de un programa

## Un reto

Diseñe un programa para calcular la hipotenusa de un triángulo, conociendo el valor de sus catetos.





$$\text{area}(d) = \pi \left( \frac{d}{2} \right)^2$$

;;Autor: Carlos A Delgado

;;Fecha: 4 - Nov - 2019

;;Contrato: area:num->num

;;Descripción: Esta función  
calcula el area usando el  
diámetro

;;Ejemplos (area 6) -> 9\*pi

;; (area 8) -> 16\*pi

----Código---

---Pruebas---



# Contenido

1 Receta de diseño de programas

2 Elementos de un programa

3 Expresiones condicionales

# Expresiones condicionales

## Definición

- 1 Los programas no solamente contienen funciones como vimos anteriormente
- 2 Al programar podemos controlar lo que queremos se ejecute dada alguna situación
- 3 El manejo de condicionales da gran poder expresivo a los lenguajes y permite construir programas más complejos

# Expresiones condicionales

## Ejemplos

- 1 ¿El número de estudiantes matriculados en el curso es superior a 15?
- 2 ¿El avión tiene suficiente combustible para llegar a Tuluá?
- 3 ¿El estudiante con código 1654563 aprobó Fundamentos de programación?

# Expresiones condicionales

## Definición

Las respuestas a estas preguntas son **verdadero** o **falso**. A este tipo de datos se le conoce como **booleanos**

## Operadores

Podemos ir más allá y utilizar operadores cuyas respuestas son **verdadero** o **falso**.

# Expresiones condicionales

## Operadores

Algunos operadores son:

- 1 **Numéricos:**  $>$   $>=$   $<=$   $=$   $<$   $>$   $\neq$
- 2 **Lógicos** and, or y not
- 3 **Generales:** equal?

Estos operadores retornan verdadero (true) o falso (false)

Qnd

X	y	X and y	not X and y	X	y	z	X and y and z
F	F	F	V	F	F	F	F
F	V	F	V	F	F	V	F
V	F	F	V	F	V	F	F
V	V	V	F	F	V	V	F
				V	F	F	F
				V	F	V	F
				V	V	F	F
				V	V	V	V

\ or

$x$	$y$	$x \text{ or } y$	$\text{not}(x \text{ or } y)$	$x$	$y$	$\neg$	$x \text{ or } y \text{ or } \neg$
F	F	F	V	F	F	F	F
F	V	V	F	F	F	V	V
V	F	V	F	F	V	F	V
V	V	V	F	F	V	V	V
				V	F	F	V
				V	F	V	V
				V	V	F	V
				V	V	V	V

Hoy voy a almorzar sancocho o sopa de tortilla.

XOR

X	Y	$X \oplus Y$
F	F	F
F	V	V
V	F	V
V	V	F

$(X \oplus Y) \oplus Z$  ✓

$X \oplus (Y \oplus Z)$

X	Y	Z	$X \oplus Y$	
F	F	F	F	F
F	F	V	F	V
F	V	F	V	V
F	V	V	V	F
V	F	F	V	V
V	F	V	V	F
V	V	F	F	F
V	V	V	F	V



$x$	$y$	$\bar{z}$	$y \text{ xor } z$		
F	F	F	F	F	F
F	F	V	V	V	V
F	V	F	V	V	V
F	V	V	F	F	F
V	F	F	F	V	F
V	F	V	V	F	F
V	V	F	V	F	V
V	V	V	F	V	

$$(x \text{ xor } y) \text{ xor } (\bar{z} \text{ xor } w)$$



not		not
x		x
V		F
F		V

# Expresiones condicionales

## Operadores lógicos

Los operadores lógicos son:

- **and**: Su resultado es verdadero si todas sus entradas son verdaderas.
- **or**: Su resultado es verdadero si al menos una de sus entradas es verdadera.
- **not**: Sólo recibe una entrada, esta retorna falso si la entrada es verdadera y verdadero si la entrada es falso.

# Expresiones condicionales

## Operadores

Prueba en el lenguaje de aprendizaje lo siguiente:

```
(and false false true) ← ✗ F
(and true true true true) ← ✗ T
(or true false false false) ← ✗ T
(or false false false) ← ✗ F
(not false) ← ✗ T
(not true) ← ✗ F
```

$(xor \ true \ false) \leftarrow \text{true}$   
 $(xor \ true \ true) \leftarrow \text{false}$

# Expresiones condicionales

## Operadores

Prueba en el lenguaje de aprendizaje lo siguiente:

```
(= 3 2) ← ✗ F  
(> 3 4) ← ✗ F  
(>= 3 3) ← ✗ T  
(not (>= 3 3)) ← ✗ F  
(or (>= 3 3) (>= 2 3)) ← ✗ T  
(and (>= 3 3) (>= 2 3)) ← ✗ F  
(equal? "hola" 3) ← ✗ F  
(equal? "hola" "hola") ← ✗ T
```

(xor (>= 3 4) (= 4 6))

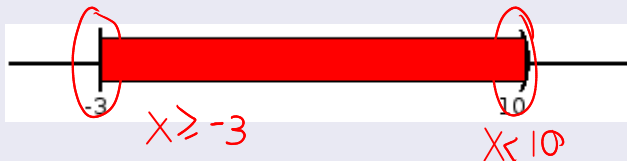
(xor true (>= 6 8))

(equal? "Hola" "hola") ← ✗ F

# Expresiones condicionales

## Rangos

Podemos usar combinaciones de operadores lógicos con relacionales para validar si un número está en un rango.



**Importante:** [ o ] son rangos cerrados, es decir incluye el número y ( o ) son rangos abiertos es decir que no lo incluyen.

¿Como validamos que un número se encuentra en este rango?

# Expresiones condicionales

## Rangos



Debe verificar:

- El número debe ser mayor o igual que -3, esto lo hacemos ( $\geq x -3$ )
- El número debe ser menor estricto que 10, esto lo hacemos ( $< x 10$ )
- Y debe cumplir las dos anteriores, es decir (and ( $\geq x -3$ ) ( $< x 10$ ))

¿Porque usar **or** no es correcto?

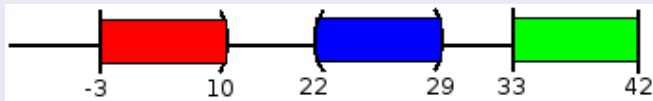


Universidad  
del Valle

# Expresiones condicionales

## Rangos

De acuerdo a esto podemos verificar lo siguiente:

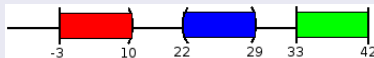


¿Que proponen?

(or  
(and ( $\geq x -3$ ) ( $< x 10$ ))  
(and ( $> x 22$ ) ( $< x 29$ ))  
(and ( $\geq x 33$ ) ( $\leq x 42$ )) )

# Expresiones condicionales

## Rangos



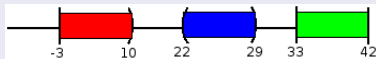
Para abordar este problema:

- Defina el primer rango  $[-3, 10]$
- Defina el segundo rango  $(22, 29)$
- Define el tercer rango  $[33, 42]$
- ¿Con que los unimos y porque?



# Expresiones condicionales

## Rangos



La solución es:

```
(or  
  (and (>= x -3) (< x 10))  
  (and (> x 22) (< x 29))  
  (and (>= x 33) (<= x 42))  
)
```

¿Porque se usa or de esta manera?



Universidad  
del Valle

## Operadores

Los operadores ayudan a estructurar soluciones a diferentes problemas, por ejemplo:

- 1 Diseñar un programa para verificar si la edad está entre 10 y 20 años
- 2 Diseñar un programa para verificar si el salario es 2000 o 3000

3. Diseñe un programa que valide si  $x$  está en  $(-10, 8]$  o  $(10, 20]$  o  $(33, 66]$  o  $(100, 200]$  o  $[500, 1000]$  o  $[2000, 3000]$  o  $[5000, +\infty)$

# Expresiones condicionales

## Operadores

Diseñar un programa para verificar si la edad está entre 10 y 20 años

```
;; Autor: Docente curso Fundamentos de Programación
;; Fecha de creación: 20-Agosto-2016
;; Contrato: verificar-edad: numero -> booleano
;; Propósito: Verifica si la edad está entre 10 y 20 años
;; Ejemplo: (verificar-edad 5) retorna falso,
;; (verificar-edad 10) retorna verdadero
;; Definición
(define (verificar-edad edad) (and (>= edad 10) (<= edad 20)
  ) )
;; Pruebas
(check-expect (verificar-edad 5) #F)
(check-expect (verificar-edad 10) #T)
```



# Expresiones condicionales

## Operadores

Diseñar un programa para verificar si el salario es 2000 o 3000

```
;; Autor: Docente curso Fundamentos de Programación
;; Fecha de creación: 20-Agosto-2016
;; Contrato: verificar-salario: numero -> booleano
;; Propósito: Verifica si el salario es 2000 o 3000
;; Ejemplo: (verificar-salario 5000) retorna falso,
;; (verificar-salario 2000) retorna verdadero
;; Definición
(define (verificar-salario salario) (or (= salario 2000) (=
  salario 3000)))
;; Pruebas
(check-expect (verificar-salario 5000) #F)
(check-expect (verificar-salario 2000) #T)
```

# Expresiones condicionales

## Ejercicio

Diseñar un programa para verificar si el valor dado de área de un cuadrado de lado  $L$  es correcto. **verificar-area**

# Expresiones condicionales

## Operadores

Diseñar un programa para verificar si el salario es 2000 o 3000

```
;; Autor: Docente curso Fundamentos de Programación
;; Fecha de creación: 20-Agosto-2016
;; Contrato: verificar-area: numero, numero -> booleano
;; Propósito: Verifica si el área dada de un cuadrado de lado L es
               correcta
;; Ejemplo: (verificar-area 10 300) retorna falso,
;; (verificar-area 10 100) retorna verdadero
;; Definición
(define (verificar-area lado area) (= (* lado lado) area))
;; Pruebas
(check-expect (verificar-area 10 300) #F)
(check-expect (verificar-area 10 100) #T)
```

# Expresiones condicionales

number?

## Predicados

Los predicados son funciones que permiten calcular determinar si se cumple o no cierta propiedad.

- 1 Su salida es un booleano
- 2 Su nombre termina con ?

list?

# Expresiones condicionales

## Predicados

Algunas funciones de predicados son:

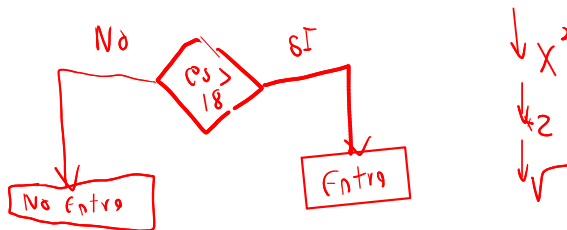
- 1 **number?** Determina si un valor es un número
- 2 **odd?** Determina si un número es impar
- 3 **even?** Determina si un número es par



# Expresiones condicionales

## Usando los condicionales

Los condicionales y predicados nos van a servir para tomar decisiones dentro de nuestros programas



# Expresiones condicionales

## cond

Para evaluar las expresiones condiciones nuestro lenguaje de aprendizaje cuenta con la función **cond**

# Expresiones condicionales

# T  
X F

cond

La estructura del cond es la siguiente

```
( cond  
  [pregunta respuesta]  
  [pregunta respuesta]  
  ...  
  [pregunta respuesta]  
)
```

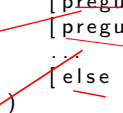
$$F(x) = \begin{cases} x^2 & \underline{x < 0} & (-\infty, 0) \\ x & \underline{x \geq 0 \text{ y } x \leq 5} & [0, 5] \\ x^3 & x > 5 & (5, +\infty) \end{cases}$$

# Expresiones condicionales

## cond

Es recomendado **else**, debido a que pueden existir casos que usted no considere y si en la ejecución ningún caso es válido y no existe else producirá un error

```
( cond
  [pregunta respuesta]
  [pregunta respuesta]
  .
  [else respuesta]
)
```



# Expresiones condicionales

## cond

- ¿Que significa pregunta? Especifica la condición que se debe cumplir para dar una respuesta
- ¿Que significa respuesta? Cuando la pregunta se cumple (es verdadera), se produce una respuesta
- ¿Que significa else? Si ninguna pregunta se ha cumplido, entonces se emite esta respuesta

# Expresiones condicionales

funcion: numero  $\rightarrow$  texto

cond

Pruebe el siguiente caso

```
(define (funcion a)
  (cond
    [(< a 2) "soy menor que 2"]
    [(and (>= a 2) (<= a 5)) "estoy entre 2 y 5"]
    [(and (> a 5) (<= a 10)) "soy mayor que 5 y menor o
      igual que 10"]
    [else "soy mayor que 10"]
  ))
```

¿Que pasa si evalua (funcion 3), (funcion 5), (funcion 11)?

¿Que observa?

# Expresiones condicionales

(F --)      edad: numero  $\rightarrow$  texto

cond

Ahora si, vamos con toda :). Diseñe una función que reciba la edad de una persona y:

- 1 Si la edad es menor que 5, retorne **Eres un niño**
- 2 Si la edad es mayor o igual que 5 y menor que 10, retorne **Eres un niño grande**
- 3 Si la edad es mayor o igual que 10 y menor que 20, retorne **Eres un adolescente**
- 4 Si la edad es mayor o igual que 20, retorne **Eres un adulto**



Vamos a codificar una función que recibe la edad (numero), nombre (string), marchael21 (bool)

- 1) Edad [0, 10). Si marchael21 entonces la respuesta es "Eres muy pequeño para marchar "+nombre, si no "Apoya en Twitter "+nombre.
- 2) Edad [10, 16) Si marchael21 entonces la respuesta es "Aun eres pequeño para marchar", si no "Apoya en Instragram"+nombre
- 3) Edad [16, 28) Si marchael21 entonces la respuesta es "Ok", si no "#Noapoyoelparo "+nombre
- 4) Edad  $\geq 28$ , si marchael21 respuesta "vayatrabaje "+nombre si no "veaRCN "+nombre

(emitir-mensaje 10 "Carlos" true)  
"Aun eres pequeño para marchar Carlos"

# Expresiones condicionales

## Operadores

```
;; Autor: Docente curso Fundamentos de Programación
;; Fecha de creación: 20-Agosto-2016
;; Contrato: verificar-edad: numero -> texto
;; Propósito: De acuerdo a la edad retorna un texto
;; Ejemplo: (verificar-edad 10) retorna ".Eres un adolescente",
;; (verificar-edad 80) retorna ".Eres un adulto"
;; Definición
(define (verificar-edad edad)
  (cond
    [(< edad 5) "eres un niño"]
    [(and (>= edad 5) (< edad 10)) "Eres un niño grande"]
    [(and (>= edad 10) (< edad 20)) "Eres un
      adolescente"]
    [else "Eres un adulto"]
  )
)
;; Pruebas
(check-expect (verificar-edad 10) "Eres un adolescente")
(check-expect (verificar-edad 80) "Eres un adulto")
```



# Expresiones condicionales

## Operadores

Debemos realizar validaciones para evitar dar respuestas incorrectas.

```
;; Autor: Docente curso Fundamentos de Programación
;; Fecha de creación: 20-Agosto-2016
;; Contrato: verificar-edad: numero -> texto
;; Propósito: De acuerdo a la edad retorna un texto
;; Ejemplo: (verificar-edad 10) retorna ".Eres un adolescente",
;; (verificar-edad 80) retorna ".Eres un adulto"
(define (verificar-edad edad)
  (cond
    [(and (>= edad 0) (< edad 5)) "Eres un niño"]
    [(and (>= edad 5) (< edad 10)) "Eres un niño grande"]
    [(and (>= edad 10) (< edad 20)) "Eres un
      adolescente"]
    [(>= edad 20) "Eres un adulto"]
    [else "Edad no valida"]
  )
)
(check-expect (verificar-edad -9) "Edad no valida")
```

# Expresiones condicionales

## cond

Un amigo suyo quiere calcular el precio de la venta de CDs de acuerdo a un precio variable de acuerdo al número que compra el cliente:

- 1 Si el cliente compra menos de 2 CDs, cada CD cuesta 4000
- 2 Si el cliente compra entre 2 y 5 CD, cada CD cuesta 3500
- 3 Si el cliente compra más de 5 CD, cada CD cuesta 3000

# Expresiones condicionales

cond

Ahora con funciones auxiliares, vamos a calcular el precio de venta de unos CDs.

- 1 Si el cliente compra menos de 2 CDs, cada CD cuesta 4000 y el IVA es 20 %
- 2 Si el cliente compra entre 2 y 5 CD, cada CD cuesta 3500 y el IVA es 15 %
- 3 Si el cliente compra más de 5 CD, cada CD cuesta 3000 y el IVA es 13 %

$$3 \rightarrow 10500 \times 1.15$$
$$8 \rightarrow 24000 \times 1.13$$

# ¿Preguntas?

¿Preguntas?