

# 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

# 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**

# 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 a 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)**

# 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.

# Elementos de un programa

## 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**

# 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)
```

# 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)
```

# Elementos de un programa

## Un reto

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

$$h^2 = C_1^2 + C_0^2$$

$$h = \sqrt{C_1^2 + C_0^2}$$

;;Autor: Carlos A Delgado

;;Fecha: 4 Jun 2019

;;Contrato: calc-h: numero,numero-> numero

;;Propósito: Calcular la hipotenusa a partir

;; los dos catetos

;;Ejemplos (calc-h 4 5) -> 6 (calc-h 3 4) -> 5

(define (calc-h a b)

(sqrt (calc-haux a b))

)

;;Función auxiliar

;;calc-haux: numero, numero -> numero

;;Ejemplo (calc-haux 4 5)-> 36 (calc-h 3 4)-> 25

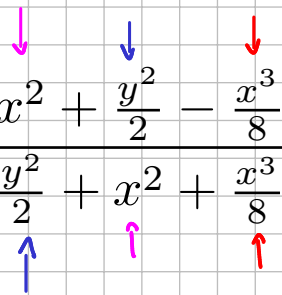
(define (calc-haux a b)

(+ (sqr a) (sqr b)

)



Diseñe una función que reciba dos numeros  
y permite calcular la siguiente expresión

$$f(x, y) = \frac{x^2 + \frac{y^2}{2} - \frac{x^3}{8}}{\frac{y^2}{2} + x^2 + \frac{x^3}{8}}$$


The diagram illustrates the mapping of variables between the numerator and denominator of the fraction. Colored arrows indicate the following correspondences:

- Blue arrows:** Point from the  $x^2$  term in the numerator to the  $x^2$  term in the denominator, and from the  $\frac{y^2}{2}$  term in the denominator to the  $\frac{y^2}{2}$  term in the numerator.
- Red arrows:** Point from the  $\frac{x^3}{8}$  term in the numerator to the  $\frac{x^3}{8}$  term in the denominator, and from the  $\frac{x^3}{8}$  term in the denominator to the  $\frac{x^3}{8}$  term in the numerator.
- Green arrows:** Point from the  $\frac{y^2}{2}$  term in the numerator to the  $\frac{y^2}{2}$  term in the denominator, and from the  $\frac{y^2}{2}$  term in the denominator to the  $\frac{y^2}{2}$  term in the numerator.

<https://pastebin.com/1Q8LFhGH>

$$f(x, y, z, w) = \frac{\overbrace{\frac{x^2+y^3+z^3}{x^3+w^3}}^{\text{green}} + \overbrace{\frac{x^5+\sqrt{x+y}}{x+y}}^{\text{blue}}}{\frac{x^2+y^3+z^3}{x^3+w^3} + \frac{x+y}{x^5+\sqrt{x+y}}}$$

# 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

$$F(x, y) = \begin{cases} \underline{x^2} & x < 0 & y & y < 0 \\ \underline{x^2 + y} & x \geq 0 & y & y < 0 \\ x^3 + 3y & \text{otherwise} \end{cases}$$

$$x = 4$$

$$y = -7$$

# Expresiones condicionales

## Ejemplos

- 1 ¿El número de estudiantes matriculados en el curso es superior a 15?  $\leftarrow F, V$
- 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:** > >= <= = >  $\odot$

2 **Lógicos** and, or y not

3 **Generales:** equal?

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

$(= 5 3) \leftarrow F$

equal? "hola" "hola,1"  
↓  
F



# Expresiones condicionales

## Operadores lógicos

Los operadores lógicos son:

- **and**: Su resultado es verdadero si todas sus entradas son verdaderas.  $(\text{and } \text{false } \text{true } \text{true}) \rightarrow \text{false}$
- **or**: Su resultado es verdadero si al menos una de sus entradas es verdadera.  $(\text{or } \text{false } \text{true } \text{true}) \rightarrow \text{true}$
- **not**: Sólo recibe una entrada, esta retorna falso si la entrada es verdadera y verdadero si la entrada es falso.

$(\text{not } \text{false}) \rightarrow \text{true}$

# 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
```

# 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
```

# Expresiones condicionales

## Rangos

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

$$x \geq -3 \text{ y } x \leq 10$$



**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 ( $\text{and } (\geq x -3) (< x 10)$ )

¿Porque usar **or** no es correcto?

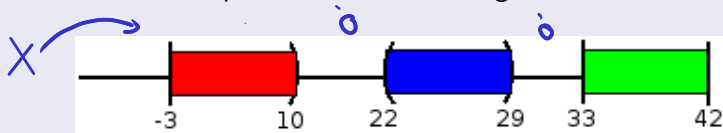


Universidad  
del Valle

# Expresiones condicionales

## Rangos

De acuerdo a esto podemos verificar lo siguiente:

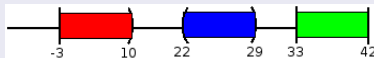


¿Que proponen?

or

# Expresiones condicionales

## Rangos

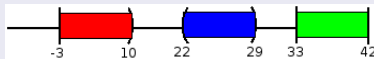


Para abordar este problema:

- 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:

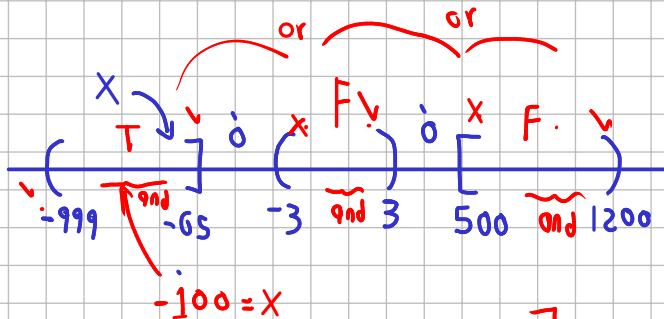
```
(or  
  (and (>= x -3) (< x 10))  
  (and (> x 22) (< x 29))  
  (and (>= x 33) (<= x 42))  
)
```

¿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

# 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**

$area, lado$

verificar-area: numero, numero  $\rightarrow$  booleano

$(= area \quad lado^2)$

# 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

Cequis?

## 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 ?

# 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**

if y cond

```
(if <pred?>  
  <res-true>  
  <res-false>  
)
```

# Expresiones condicionales

V o F

cond

La estructura del cond es la siguiente

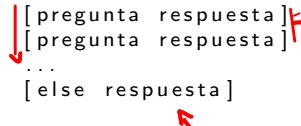
```
( cond  
  [pregunta respuesta]  
  [pregunta respuesta]  
  ...  
  [pregunta respuesta]  
)
```

# 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] F
  ↓ [pregunta respuesta]
  ...
  [else respuesta]
)
```



## 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

## 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

## 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"**



j; verificar-edad: numero  $\rightarrow$  texto

(define (verificar-edad edad)

(cond

$\rightarrow$  [ (< edad 0) "Edad inválida" ]

$\rightarrow$  [ (< edad 5) ("Eres un niño") ]  
pregunta respuesta

[ (and ( $\geq$  edad 5) (< edad 10))  
"Eres un niño grande" ]

[ (and ( $\geq$  edad 10) (< edad 20))  
"Eres un adolescente" ]

[ else "Eres un adulto" ] )

(verificar-edad 10)  $\rightarrow$  "Eres un adolescente"

(verificar-edad -2)  $\rightarrow$  ~~"Eres un niño"~~  
"Edad inválida"

# 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 %

# ¿Preguntas?

¿Preguntas?