

# INSTITUTO TECNOLÓGICO DE OAXACA

DEPARTAMENTO DE SISTEMAS Y COMPUTACIÓN

CARRERA: INGENIERÍA EN SISTEMAS COMPUTACIONALES



**PROGRAMACIÓN LÓGICA Y FUNCIONAL**

**CARPETA DE EVIDENCIAS**

**Docente: Ramírez López Sergio Saul**

**Grupo: 7SD**

**Alumno: Jiménez Castillejos Fabián de Jesús**

# **Unidad 1**

# LINEA DEL TIEMPO DE lenguajes de Progra.

1945 C

Considerado el primer lenguaje de programa, aunque no fue implementado hasta décadas después.



1950 Assembly

ASS E M BLY

Lenguaje de bajo nivel que utiliza mnemónicos para la representación de las instrucciones de máquina.

1958 LISP

Pionero en la prog funcional utilizada en Intel. Arti.



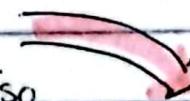
1957 FORTRAN

FORTRAN

Diseñados para cálculos científicos, fue uno de los primeros lenguajes de alto nivel ampliamente utilizados.

1958 COBOL

Diseñado para aplicaciones comerciales, aun en uso en sistemas heredados.



1960 ALGOL

Influyente en el diseño de muchos lenguajes posteriores, introdujo conceptos como bloque y alcance.

1970 PASCAL

Diseñado para enseñar programación estructurada.



1972 C

Lenguaje Bajo Nivel, Eficiente y flexible, base para muchos S.O. y aplicaciones.



1969 BASIC

Diseñado para la enseñanza y uso general, popular en microcomputadoras.

1979 PERL

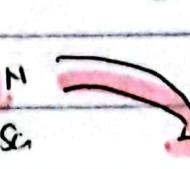
Inicialmente diseñado para el procesamiento de texto, se volvió popular para Script y administración de sistemas.

1980 C++

Extensión de C con características de prog. Orientada a Objetos.

1991 PYTHON

Diseñado para la enseñanza y uso general, popular en microcomputadoras.



1983 Ada 95

Desarrollo para aplicaciones militares con fuerte tipado y concurrencia.

1995 JAVA

Diseñado para ser "escribir una vez, ejecutar en cualquier lugar".

1995 JAVASCRIPT

Esencial para la interactividad en páginas web

1995 PHP

Ampliamente utilizado para el desarrollo web

2000 C#

Desarrollado por Microsoft, orientado a objetos similar a java.

2009 GO

Desarrollado por google  
lenguaje compilado y concurrente

2010 RUST

lenguaje de Sistemas con enfoque en seguridad y concurrencia.

2014 SWIFT

Desarrollado por Apple para iOS y Mac OS

Fabian de Jesus Jimenez castillejos

PLF 7SD

PROGRAMACION  
LOGICA Y  
FUNCIONAL

PROGRAMACIÓN  
ESTRUCTURADA

FABIAN DE JESUS JIMENEZ CASTILLEJOS



# ¿Que es la programación estructurada?

La programación estructurada es una teoría orientada a mejorar la calidad y el tiempo de desarrollo, utilizando únicamente las subrutinas o funciones. Basadas en el teorema de la programación estructurado, propuesto por bohm y jacopini, ha permitido desarrollar software de fácil comprensión.

## ventajas

los programas son desarrollados con la programación estructurada que son más sencillos de entender por su estructura secuencia

-Los programas resultantes tendrán una estructura clara

-Su prueba y depuración son óptimos y sencillos

## Desventajas

-Todo el programa se concentra en un único bloque, debido a que si es demasiado grande es difícil de manejar

-El tipo de lenguaje no permite reutilizar el código debido a que todo está unido

# ESTRUCTURAS BASICAS

- SECUENCIA:** ESTRUCTURA SECUENCIAL es la que se da de forma natural en el lenguaje
- SELECCION O CONDICIONAL :** Se basa en que una sentencia se ejecuta según el valor que se le atribuye a una variable booleana
- ITERACION (CICLO O BUCLE):** Esta se ejecuta una o un conjunto de variables booleanas sea verdadera, así como las estructuras while o for

# **Unidad 2**

# Funciones :)

Fabian de Jesus  
Jinenez ces tilles

1. Suma

$$\text{Suma } x \ y = x + y$$

2. Resta  $x \ y = x - y$

3.. divide  $x \ y = x / y$

4.. multiplica  $x \ y = x * y$

5.. modulo  $x \ y = x \bmod y$

6.. absoluto  $x = \text{abs } x$

7.. doble  $x = x * 2$

8.. triple  $x = x * 3$

9.. cuadrado  $x = x * x$

10.. Potencia  $x \ y = x^y$

11 Primelemento  $xs = \text{head } xs$

12 Ultimo Elemento  $xs = \text{last } xs$

13 Cola  $xs = \text{tail } xs$

14 Inicio  $xs = \text{init } xs$

15 longitud  $xs = \text{length } xs$

16 Sumar lista  $xs = \text{sum } xs$

17 producto\_de\_lista  $xs = \text{product } xs$

18 maximo xs  $\rightarrow \text{maximum } xs$

19 minimo xs  $= \text{minimum } xs$

20 reversos xs  $= \text{reverse } xs$

21 esPar x  $= \text{even } x$

22 esImpar x  $= \text{odd } x$

23 x mayor y  $= \text{if } x > y \text{ then "el mayor es"} x \text{ else "el mayor es" } y$

24 Lista de cubos  $n = [x^3 \mid x \in [1..n]]$

25 Lista de cuadrados  $n = [x^2 \mid x \in [1..n]]$

26 FiltraMultiplos n listas  $[x \mid x \in \text{listas}, n \bmod 2 = 0]$

27 Sumar\_los\_pares  $= \text{sum } [x \mid x \in xs, x \bmod 2 = 0]$

28 longitudes XSS  $= [\text{length } xs \mid xs \in XSS]$

29 Sumar\_tuplas  $(x_1, y_1) (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$

30 promedio\_triple  $(x, y, z) = (x + y + z) / 3$

31 SumaTupla xs  $= (\text{sum } [x \mid (x, \_) \in xs],$   
 $\text{sum } [y \mid (\_, y) \in xs])$

32 Fun ab  $= [\min ab, \max ab, \text{succ } b]$   
 $+ [a^{1/2}, b^{1/2}, (\text{succ}(\min a(b^{1/2})))]$

33 Fun abc  $= [\max(\frac{a * b}{300}, 1), b^{1/4}, 10^b]$

Fórmula de JESUS  
Jinenc2 cysf: 11ejo

34 doble en lista n = (if  $X > n$  then  $X+2$  else  $X$ ):[]

35 mitad a = (if  $a > 8$  then  $a/2$  else 8):[]

36 los negativos b = (if  $b > 0$  then  $\emptyset$  else b):[]

37 Positivos p = (if  $p > 0$  then p else  $\emptyset$ ):[]

38 Sumar elementos Pares: = if  $S \bmod 2 == \emptyset$  then  
else  $+ S$

39 restar\_elementos R = if  $R \bmod 3 == \emptyset$  then  
else  $R - 3$

40 InvertSigno x = (if  $x < 0$  then  $x * (-x)$   
else  $-x$ ):[]

41 Reemplazar\_mayores M = (if  $m > 0$  then  $\emptyset$   
else  $\{m\}$ ):[]

42 dividir\_Impares D = (if  $x \bmod 2 == 0$   
then "from the vault" show ( $x/2$ )  
else "not tv" show x.

43 lista\_de\_RS palabra = [ $x | x \leftarrow x_3, x == 'P'$ ]

44 Eliminar\_P palabra = [ $x | x \leftarrow x_3, x \neq 'P'$ ]

45 mostrarSigno r = if  $r > 0$  then show r ++ "es positivo"  
else r ++ "es negativo"

46 mostrar\_Par X = If  $X \bmod 2 == \emptyset$  then X ++  
"es par" else Show X ++ "no es par"

47 mostrar\_el\_doble R = If  $R > 0$  then Show (R \* 2)  
++ "Es Par" else "no es par"

48 mostrar\_menor XY = if  $x < y$  then Show X ++  
"es menor" ++ Show Y else Show Y ++ "es menor"

49 Sumar\_triples (x, y, z) ( $x_1, y_1, z_1$ ) =  
(( $x_1 + x$ ), ( $y_1 + y$ ), ( $z_1 + z$ ))

50 Funcion\_b = [a \* b, 7 ^ b, a + (b ^ a), b \* 100]

# **Unidad 3**

## **Funciones generadas en clase:**

### **Generar los cuadrados de los números del 1 al 5**

```
cuadrados = [x^2 | x <- [1..5]]
```

### **Duplicar cada elemento de cada sublistas**

Aquí duplicamos cada elemento en cada lista de una lista de listas.

```
duplicados = [[2 * x | x <- xs] | xs <- [[1,2,3], [4,5,6], [7,8,9]]]
```

### **Filtrar solo las palabras largas en una lista de strings**

```
palabrasLargas = [palabra | palabra <- ["hola", "mundo", "Haskell", "es", "genial"], length palabra > 4]
```

### **Obtener la primera posición de cada lista**

Supongamos que tienes una lista de listas, y quieres obtener el primer elemento de cada una de ellas.

```
primeros = [head xs | xs <- [[1,2,3], [4,5,6], [7,8,9]]]
```

### **Filtrar las listas que contienen más de tres elementos**

En este caso, seleccionaremos solo las listas con longitud mayor que tres.

```
listasLargas = [xs | xs <- [[1,2], [3,4,5,6], [7,8,9]], length xs > 3]
```

### **Sumar los elementos de cada lista**

Para cada lista en la lista de listas, calcula la suma de sus elementos.

```
sumaListas = [sum xs | xs <- [[1,2,3], [4,5,6], [7,8,9]]]
```

## **Obtener los pares de cada lista y ponerlos en una lista**

Para cada lista, extraemos los elementos pares.

```
paresEnListas = [[x | x <- xs, even x] | xs <- [[1,2,3], [4,5,6], [7,8,9]]]
```

## **Aplanar una lista de listas (concatenar todas las sublistas en una sola lista)**

Puedes usar una lista intencional para "aplanar" una lista de listas en una sola lista.

```
aplanada = [x | xs <- [[1,2,3], [4,5,6], [7,8,9]], x <- xs]
```

## **funciones con el uso de import:**

```
-- import Data.Char(toLower)
```

```
-- import Data.Char(toUpper)
```

```
-- import Data.Char
```

```
-- -- import Data.Char(chr)
```

```
-- -- import Data.Char(ord)
```

```
-- -- import Data.Char(isLetter)
```

```
-- -- funcion a b c = if(a > b) then (if (a>c) then "mayor"++ Show a else "mayor"++ Show c)  
else (if (b>c) then "mayor"++ Show b else "mayor"++ Show c)
```

```
-- -- SABER SI UN AÑO ES BISIESTO
```

```
-- esBisiesto :: Integer -> String
```

```
-- esBisiesto a =  
--   if a `mod` 400 == 0  
--     then show a ++ " es bisiesto"  
--   else if a `mod` 4 == 0 && a `mod` 100 /= 0  
--     then show a ++ " es bisiesto"  
--   else show a ++ " no es bisiesto"
```

```
-- impar n = if n `mod` 2 /= 0 then True else False
```

```
-- funcionx xs =[x| x <- xs , impar x]
```

### -- -- INDICE DE MASA CORPORAL

```
-- imc :: Double -> Double -> String  
-- imc p a =  
--   let imcValue = transformarP p / (transformar a ^ 2)  
--   in if imcValue < 18.5  
--     then "Bajo Peso"  
--   else if imcValue <= 24.9  
--     then "Peso normal"  
--   else "SobrePeso"
```

```
-- transformarP :: Double -> Double
```

```
-- transformarP p = p / 1000
```

```
-- transformar :: Double -> Double
```

```
-- transformar a = a / 100
```

```
-- funcion xs = init(tail xs)
```

```
-- :
```

```
-- fun xs = ccf (tail xs)
```

```
-- ccf ys = (ys !! 0) !! 1 : (ys !! 1 )!!1 : (ys!! 2)!!1:[]
```

```
--lista ood
```

```
-- boomBangs xs = [ if x < 10 then "BOOM!" else "BANG!" | x <- xs, odd x]
```

```
--lista de listas
```

```
-- fun xxss = [ [ x | x <- xs, x =='a' ] | xs <- xxss]
```

```
-- fun x = [x | x <- xs, x =='-' ]
```

```
-- fun = [x + y | x<-[1..3], y <-[5..10]]
```

```

-- fun xs = [toLower x | x<-xs]

-- fun2 xs = [toUpper x | x<-xs]

-- funcion l= chr l

-- function n = ord n

-- -- fun a b c = if ((b^2) - 4*(a*c)) > 0 then print "si se puede" else print "no se puede"

-- dis a b c = (b^2) - 4*(a*c)

-- x1 a b c = (-b + sqrt(dis a b c)) / ( 2*a)

-- x2 a b c = (-b - sqrt(dis a b c)) / ( 2*a)

-- fun a b c = if (dis a b c ) > 0 then "x1=" ++ show(x1 a b c ) ++ ", x2=" ++ show (x2 a b c)
--      else "NO SE PUEDE"

-- mun xs = [ fun a b c | (a,b,c)<-xs]

```

```
dis y1 y2 x1 x2 = "la distancia es" ++ show((y2-y1)^ 2) + ((x2-x1)^ 2)
```

```
doid y1 y2 x1 x2 = sqrt(dis y1 y2 x1 x2)
```

```
fun y1 y2 x1 x2 = if (doid y1 y2 x1 x2 ) > 0 then "la distancia es" ++ show(doid y1 y2 x1 x2 )  
else "NO SE PUEDE"
```

```
,
```

```
mun xs = [ doid y1 y2 x1 x2 | (y1, y2,x1, x2)<-xs]
```

# **Unidad 4**

# Funciones realizadas en clase:

charName 'a' = "Albert"

charName 'b' = "Broseph"

charName 'c' = "Cecil"

charName x="hola"

first (x, g, f) = x

list (a:b:x:xs) = a+b+x

list2 (\_:x:xs) = x+1

fun x

| mod x 3== 0 && mod x 5==0 = (x, "BizzBuzz")

| mod x 3== 0= (x, "Bizz")

| mod x 5==0 = (x, "Buzz")

|otherwise = (x, "")

funn xs=[fun x|x <-xs]

factorial 0=1

factorial 1=1

factorial x= x \* factorial (x-1)

-- Numero fibonacci de un numero

fibonacci 0=0

fibonacci 1=1

fibonacci x= fibonacci (x-1) + fibonacci (x-2)

fibo x=[fibonacci y |y<-[0..x]]

-- Encuentra el Maximo elemento de una lista

maximum' [] = error "lista vacía"

maximum' [x] = x

maximum' (x:xs)

| x > maxTail = x

| otherwise = maxTail

where maxTail = maximum' xs

-- Encuentra el Maximo elemento de una lista

minimum' [] = error "Máximo de una lista vacía"

minimum' [x] = x

minimum' (x:xs)

| x < minTail = x

| otherwise = minTail

where minTail = minimum' xs

```
repeat' x = x : repeat' x
```

-- Determina la suma de los elementos de una lista.

```
sumlista [] = 0
```

```
sumlista [x] = x
```

```
sumlista (x:xs)= x + sumlista xs
```

--Determina el producto de una lista.

```
productolista [] = 0
```

```
productolista [x] = x
```

```
productolista (x:xs)= x * productolista xs
```

--Encuentra la longitud de una lista sin usar length.

```
lengthlista []=0
```

```
lengthlista [x]=1
```

```
lengthlista (x:xs)= 1 + lengthlista xs
```

--Determina si un elemento está en una lista.

```
buscar x []=False
```

```
buscar x (y:xs)
```

```
 | x == y = True
```

```
 | otherwise = buscar x xs
```

-- Implementa una función para revertir una lista.

```
reversa [] = []
```

```
reversa (x:xs) = reversa xs ++ [x]
```

```
palindromo xs
```

```
| xs == reversa xs = "Es palindromo"
```

```
| otherwise = "No es Palindromo"
```

--Lista donde cada elemento de la lista original aparece dos veces.

```
listados [] = []
```

```
-- listados (x:xs) = x:x:listados xs
```

```
listados (x:xs) = [x,x]++listados xs
```

--Implementa una función que elimine todas las apariciones de un elemento dado en una lista.

```
eliminarList _ [] = []
```

```
eliminarList y (x:xs)
```

```
| y == x = eliminarList y xs
```

```
| otherwise = x : eliminarList y xs
```

-- crea una lista de n elementos

```
listaEnteros 0 = []
```

```
listaEnteros n = listaEnteros (n - 1) ++ [n]
```

-- Funcion para quitar las vocales de un texto

```
quitarvocales [] = []
```

```
quitarvocales (x:xs)
```

```
| x `elem` ['a','e','i','o','u'] = quitarvocales xs
```

```
| otherwise = x: quitarvocales xs
```

-- Dada una lista de listas, obtener una nueva lista donde cada elemento es la suma de los elementos de la lista interna correspondiente.

```
sumarListas [] = []
```

```
sumarListas (xs:xss) = sum xs : sumarListas xss
```

-- Combinar todas las listas internas en una sola lista.

```
soloLista [] = []
```

```
soloLista (xs:xss) = xs ++ soloLista xss
```

-- Quitar todas las listas vacías de una lista de listas.

```
eliminarVacias [] = []
```

```
eliminarVacias (xs:xss)
```

```
| null xs = eliminarVacias xss
```

```
| otherwise = xs : eliminarVacias xss
```

-- Contar el número total de elementos de las listas internas.

```
contarlistas [] = 0
```

```
contarlistas (xs:xss)=1 + contarlistas xss
```

-- Contar el número total de elementos en todas las listas internas.

```
contarlistas2 [] = 0
```

```
contarlistas2 (xs:xss)=length xs + contarlistas2 xss
```

--dada una lista de numeros enteros del 20 al 50 debulva una lista de duplas, y en ella el numero y el factorial

tita = [(x,(factorial(x)))|x<-[20..50]];

--2.- Dada una lista de listas de números, multiplicar todos los elementos de cada lista interna y devolver una lista con los resultados.

multi [] = 0

multi [x] = x

multi (x:xs) = x \* multi xs

recorreListas [] = []

recorreListas (xs:xss) = multi xs : recorreListas xss

---3.- Determinar si un elemento está presente en alguna de las listas internas.

entrar [] = []

entrar x (xs:xss) = presente x xs : entrar x xss

presente x [] = False

presente x (y:xs)

| x == y = True

| otherwise = presente x xs

--4.- Retornar solo las listas internas cuya suma sea mayor a un valor dado.

-- 5.- Pregunta de rescate: Sustituir todos los números negativos en las listas internas por 0.

entrarParaCambiar [] = []

entrarParaCambiar (xs:xss) = cambiesito xs : entrarParaCambiar xss

cambiesito [] = []

cambiesito (x:xs)

| x < 0 = 0 : cambiesito xs

| otherwise = x : cambiesito xs