

# 3. Lenguajes de Programación

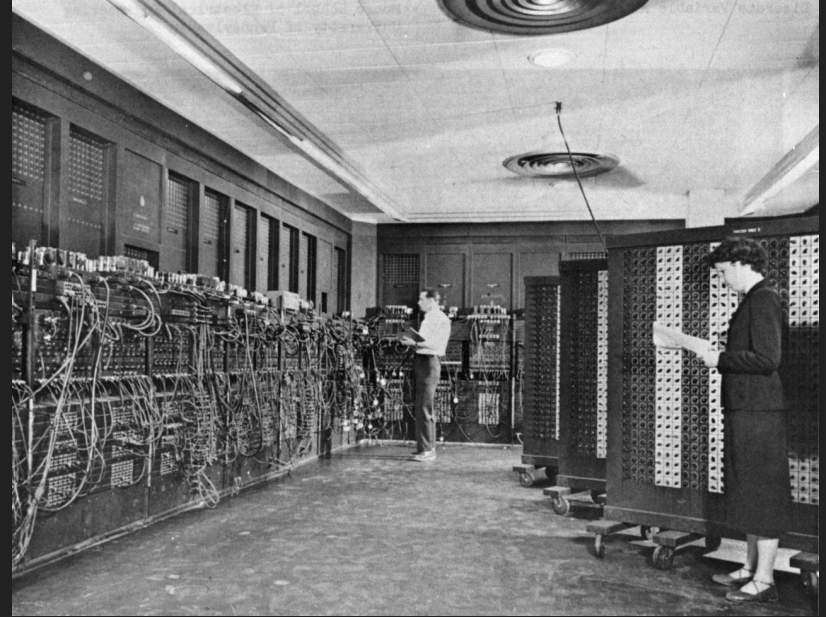
Instructor: Darian Harrison Ragle  
IMJU León  
<https://campusimju.com/>  
León, Gto. Mx.



# Historia

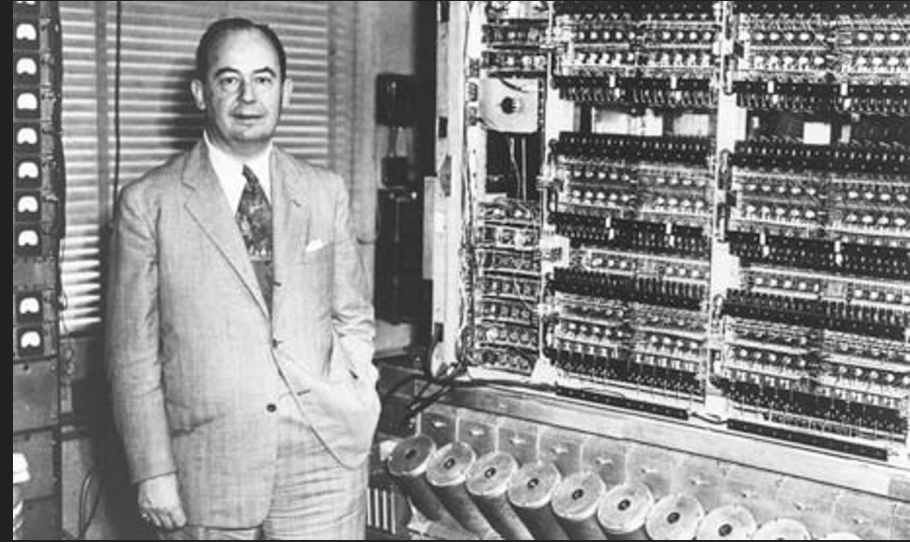
# Programación Electrónica: 1942

- Hasta 1942, la forma de alterar el programa de una computadora era mediante el movimiento físico de objetos
- Finalmente, el movimiento físico fue reemplazado por señales eléctricas cuando el gobierno de EE. UU. Construyó el ENIAC en 1942.
- Solo se podía "programar" preconfigurando interruptores y recableando todo el sistema para cada nuevo " programa "o cálculo.
- Este proceso resultó ser muy tedioso.



# Fundamentos Teóricos de Von Neumann: 1945

- En 1945, John Von Neumann Desarrolló dos conceptos teóricos importantes que mejoraron los lenguajes de programación.
- La primera establece que el hardware de la computadora debe ser simple y no necesita ser cableado manualmente para cada programa. En cambio, se deben usar instrucciones complejas para controlar el hardware simple, lo que permite reprogramarlo mucho más rápido.
- El segundo concepto es la idea de crear pequeños bloques de código que se pueden ejecutar en cualquier orden, en lugar de un solo conjunto de pasos ordenados cronológicamente para que los tomara la computadora. Esto se establece con la idea de que el código informático debería poder hacer declaraciones lógicas como "IF", "THEN", y en ciclo como con una instrucción "FOR".



# Assembly Language: 1949

- El lenguaje ensamblador era un tipo de lenguaje de programación de bajo nivel que simplificó las instrucciones específicas necesarias para operar una computadora.

```
MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER  PAGE  2

C000                      ORG      ROM+$0000 BEGIN MONITOR
C000 8E 00 70  START      LDS      #STACK

*****
* FUNCTION: INITA - Initialize ACIA
* INPUT: none
* OUTPUT: none
* CALLS: none
* DESTROYS: acc A

0013      RESETA EQU    %00010011
0011      CTLREG EQU    %00010001

C003 86 13      INITA  LDA A  #RESETA  RESET ACIA
C005 B7 80 04      STA A  ACIA
C008 86 11      LDA A  #CTLREG  SET 8 BITS AND 2 STOP
C00A B7 80 04      STA A  ACIA

C00D 7E C0 F1      JMP      SIGNON  GO TO START OF MONITOR

*****
* FUNCTION: INCH - Input character
* INPUT: none
* OUTPUT: char in acc A
* DESTROYS: acc A
* CALLS: none
* DESCRIPTION: Gets 1 character from terminal

C010 B6 80 04  INCH  LDA A  ACIA      GET STATUS
C013 47          ASR A          SHIFT RDRF FLAG INTO CARRY
C014 24 FA      BCC  INCH      RECIEVE NOT READY
C016 B6 80 05  LDA A  ACIA+1    GET CHAR
C019 84 7F      AND A  #$7F    MASK PARITY
C01B 7E C0 79  JMP      OUTCH   ECHO & RTS

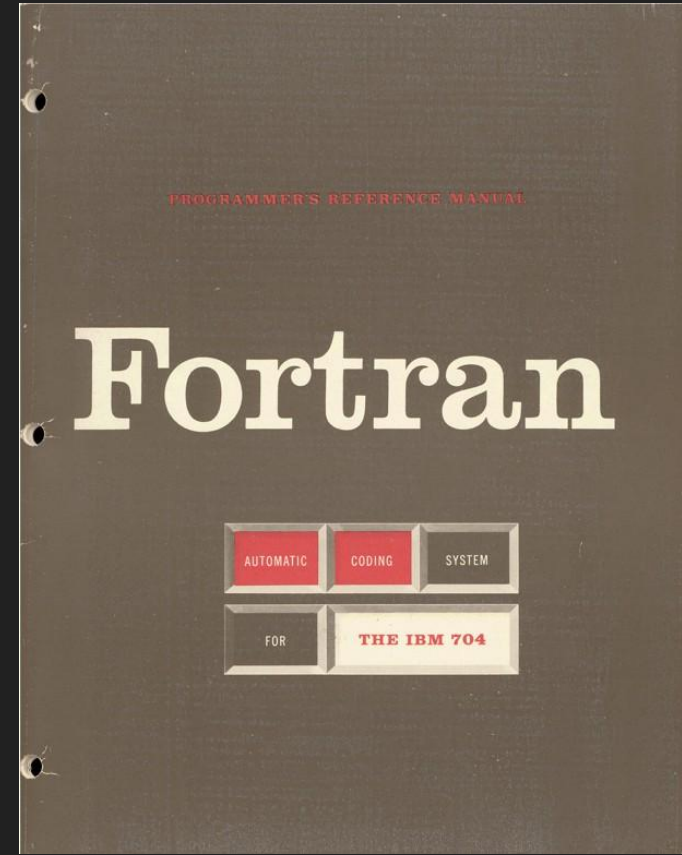
*****
* FUNCTION: INHEX - INPUT HEX DIGIT
* INPUT: none
* OUTPUT: Digit in acc A
* CALLS: INCH
* DESTROYS: acc A
* Returns to monitor if not HEX input

C01E 8D F0      INHEX BSR  INCH      GET A CHAR
C020 81 30      CMP A  #'0      ZERO
C022 2B 11      BMI  HEXERR    NOT HEX
C024 81 39      CMP A  #'9      NINE
C026 2F 0A      BLE  HEXRTS    GOOD HEX
C028 81 41      CMP A  #'A      #A
C02A 2B 09      BMI  HEXERR    NOT HEX
C02C 81 46      CMP A  #'F      #F
C02E 2E 05      BGT  HEXERR    #F
C030 80 07      SUB A  #7      FIX A-F
C032 84 0F      HEXRTS AND A  #$0F  CONVERT ASCII TO DIGIT
C034 39          RTS

C035 7E C0 AF  HEXERR JMP      CTRL  RETURN TO CONTROL LOOP
```

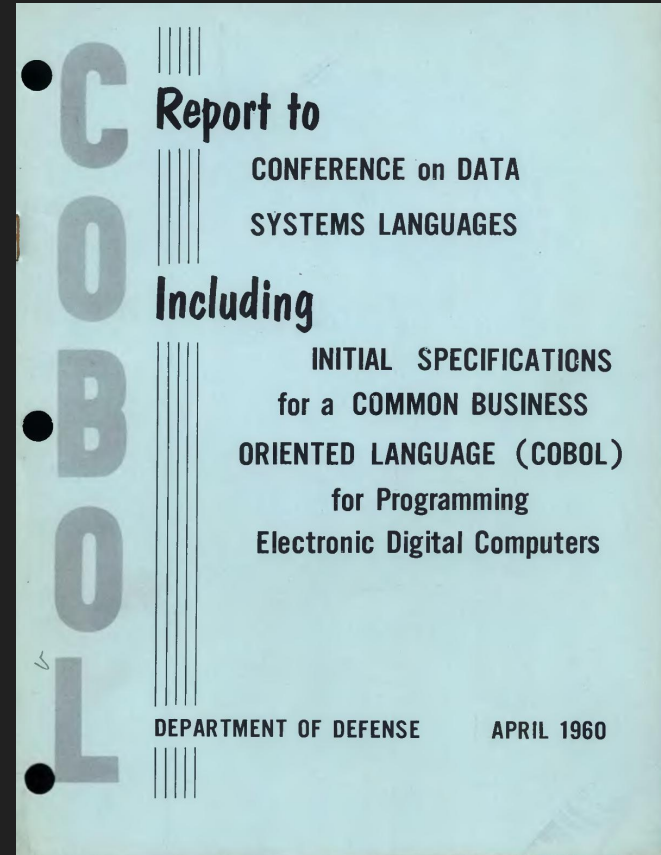
# FORTRAN: 1957

- En 1957, apareció el primero de los principales idiomas en forma de FORTRAN.
- Su nombre son las siglas de FORMula TRANslates system.
- El lenguaje fue diseñado en IBM para computación científica.
- Los componentes eran muy simples y proporcionaban al programador acceso de bajo nivel.



# COBOL: 1959

- La informática empresarial comenzó a despegar en 1959 y, debido a esto, se desarrolló COBOL.
- Fue diseñado desde cero como lenguaje para hombres de negocios.
- Todas estas características se diseñaron para facilitar que las empresas promedio las aprendan y las adopten.



# LISP: 1958

- En 1958, John McCarthy del MIT creó el lenguaje LISt Processing (o LISP).
- Fue diseñado para la investigación de Inteligencia Artificial (IA).
- Debido a que fue diseñado para un campo especializado, la versión original de LISP tenía una sintaxis única: esencialmente ninguna.

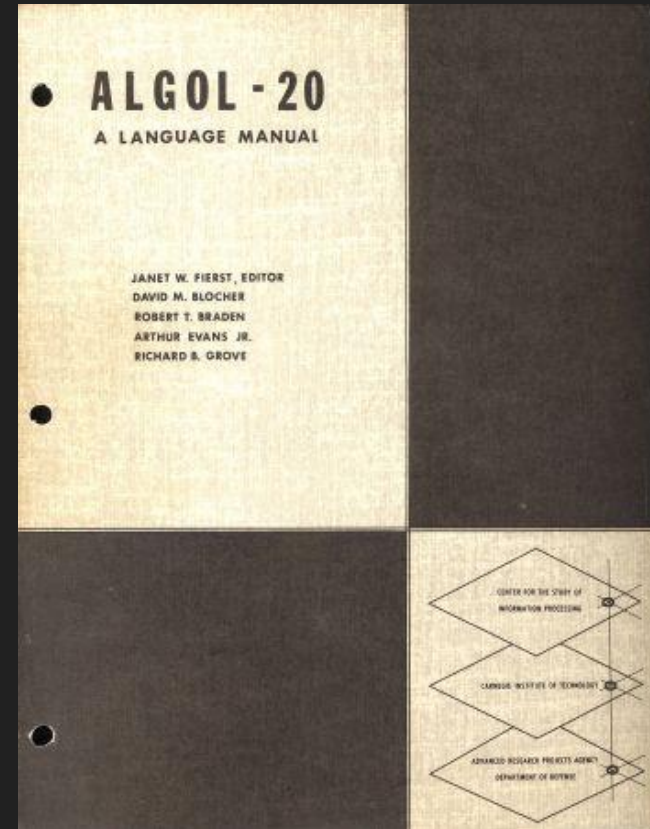
```
DEFINE
(( (RVRSE, (LAMBDA, (L), (COND, ((NULL, L), NIL),
    (T, (CONS, (RVRSE, (CDR, L)), (CONS, (CAR, L), NIL)))))),
(RVDE, (LAMBDA, (L), (REV, L, NIL))),
(REV, (LAMBDA, (J, K), (COND, ((NULL, J), K),
    (T, (REV, (CDR, J), (CONS, (CAR, J), K)))))))
()
RVRSE ((A,B,C,D,E)) ()
RVDE ((A,B,C,D,E)) ()
```

**LISP I Programmer's Manual, 1960**



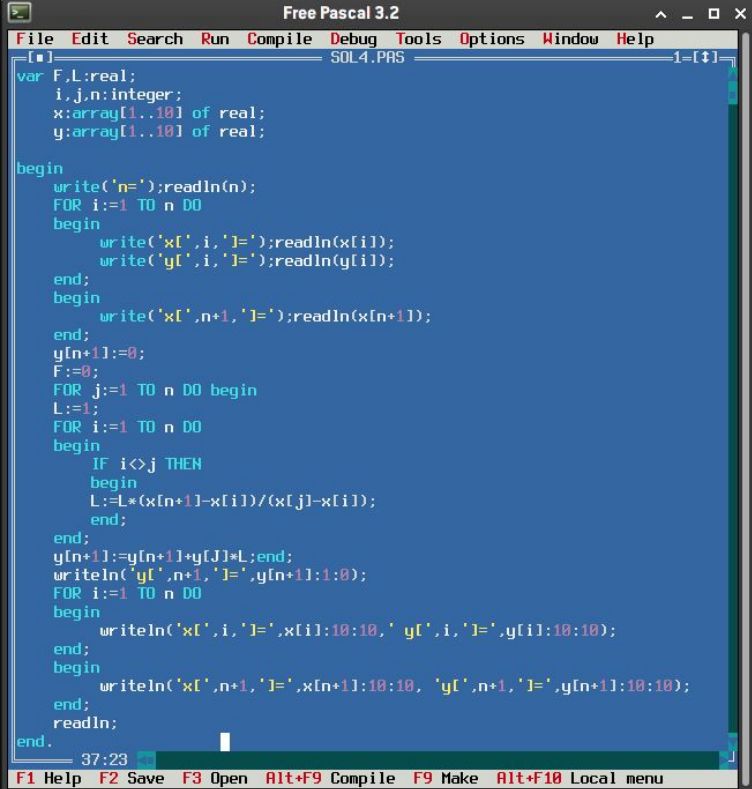
# Algol: 1958

- El lenguaje Algol fue creado por un comité para uso científico en 1958.
- Su principal contribución es ser la raíz del árbol que ha llevado a lenguajes como Pascal, C, C ++ y Java.
- Aunque Algol implementó algunos conceptos novedosos, se volvió difícil de usar. Esto llevó a la adopción de otros lenguajes más sencillos y compactos, como Pascal.



# Pascal: 1968

- Pascal se inició en 1968 por Niklaus Wirth.
- Fue diseñado con un enfoque muy ordenado, combinó muchas de las mejores características de los lenguajes en uso en ese momento, COBOL, FORTRAN y ALGOL.
- Limpiaron muchas de las irregularidades y declaraciones extravagantes de estos lenguajes.
- La combinación de características, entrada / salida y sólidas características matemáticas, lo convirtió en un lenguaje de gran éxito.

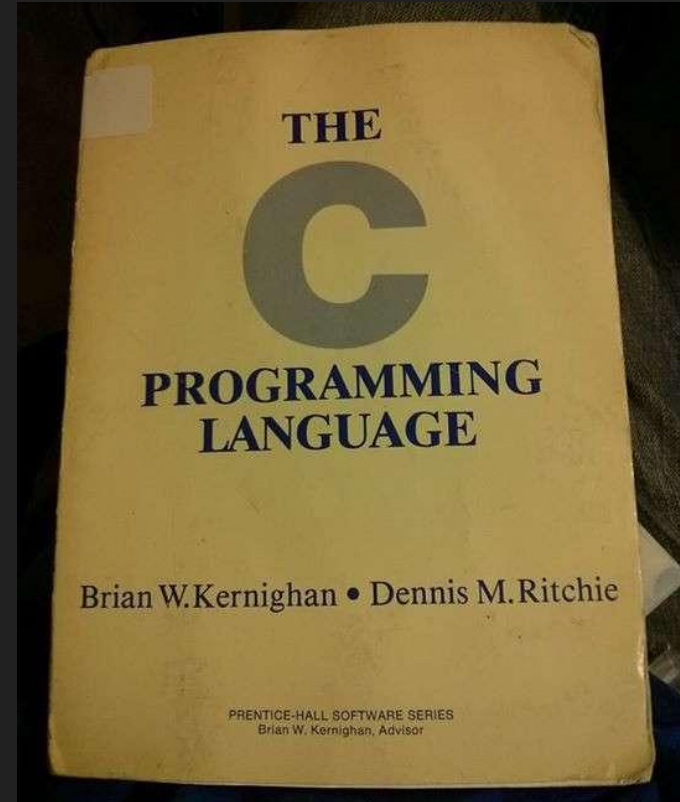


```
Free Pascal 3.2
File Edit Search Run Compile Debug Tools Options Window Help
[.] SOL4.PAS 1-[+]
var F,L:real;
    i,j,n:integer;
    x:array[1..10] of real;
    y:array[1..10] of real;

begin
    write('n=');readln(n);
    FOR i:=1 TO n DO
    begin
        write('x[i],i,']=');readln(x[i]);
        write('y[i],i,']=');readln(y[i]);
    end;
    begin
        write('x[i],n+1,']=');readln(x[n+1]);
    end;
    y[n+1]:=0;
    F:=0;
    FOR j:=1 TO n DO begin
        L:=1;
        FOR i:=1 TO n DO
        begin
            IF i<>j THEN
            begin
                L:=L*(x[n+1]-x[i])/(x[j]-x[i]);
            end;
        end;
        y[n+1]:=y[n+1]+y[j]*L;end;
    writeln('y[i],n+1,']=',y[n+1]:1:0);
    FOR i:=1 TO n DO
    begin
        writeln('x[i],i,']=',x[i]:10:10, ' y[i],i,']=',y[i]:10:10);
    end;
    begin
        writeln('x[i],n+1,']=',x[n+1]:10:10, 'y[i],n+1,']=',y[n+1]:10:10);
    end;
    readln;
end.
37:23
F1 Help F2 Save F3 Open Alt+F9 Compile F9 Make Alt+F10 Local menu
```

# C: 1972

- C fue desarrollado en 1972 por Dennis Ritchie mientras trabajaba en Bell Labs en Nueva Jersey.
- Todas las características de Pascal, incluidas las nuevas, como la instrucción CASE, están disponibles en C.
- Debido a que C solucionó la mayoría de los errores que tenía Pascal, se ganó a los antiguos usuarios de Pascal con bastante rapidez.
- Ritchie desarrolló C para el nuevo sistema Unix que se estaba creando al mismo tiempo. Debido a esto, C y Unix van de la mano.
- Debido a esto, C se usa con mucha frecuencia para programar sistemas operativos como Unix, Windows, MacOS y Linux.



# SQL (SEQUEL at the time): 1974

- SQL fue desarrollado por primera vez por los investigadores de IBM Raymond Boyce y Donald Chamberlain.
- SQL se utiliza para ver y cambiar la información que se almacena en las bases de datos.
- Hay una gran cantidad de empresas que utilizan SQL y algunas de ellas incluyen Microsoft y Accenture.



```
SELECT clause { SELECT  
                first_name  
FROM clause { FROM  
                employees  
WHERE clause { WHERE  
                YEAR(hire_date) = 2000
```

# C++: 1979-1983

- A finales de la década de 1970 y principios de la de 1980, se estaba desarrollando un nuevo método de programación conocida como Programación Orientada a Objetos o OOP.
- Bjarne Stroustrup decidió agregar características orientadas a objetos (además de otras mejoras) a C.
- A medida que se desarrolló el lenguaje, Stroustrup lo nombró C++ en 1983.



```
1  #include <iostream>
2
3  int main()
4  {
5      std::cout << "Hello, world!\n";
6  }
```

# HTML: 1990

- Tim Berners-Lee, un físico del instituto de investigación CERN en Suiza, inventó el HTML en 1990. Esta primera versión constaba de 18 etiquetas HTML.
- HTML es uno de los lenguajes de programación más populares y utilizados en el mundo.



```
<!DOCTYPE html>
<html>
  <head>
    <title>This is a title</title>
  </head>
  <body>
    <div>
      <p>Hello world!</p>
    </div>
  </body>
</html>
```

# Haskell: 1990

- Haskell es un lenguaje de programación puramente funcional, lo que significa que es principalmente matemático.
- Se utiliza en múltiples industrias, particularmente aquellas que se ocupan de cálculos complicados, registros y procesamiento de números.
- Como muchos otros lenguajes de programación de esta era, no es demasiado común ver a Haskell en uso para aplicaciones conocidas.



```
module Main (main) where

main :: IO ()
main = putStrLn "Hello, World!"
```

# Python: 1991

- Python fue desarrollado por Guido Van Rossum.
- Es un lenguaje de programación de alto nivel y propósito general creado para admitir una variedad de estilos de programación y ser divertido de usar.
- Python es, hasta el día de hoy, uno de los lenguajes de programación más populares del mundo y es utilizado por empresas como Google, Yahoo y Spotify.

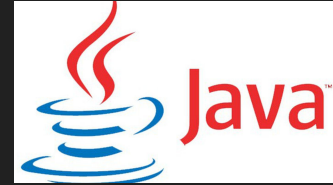


```
print('Hello, world!')
```



# Java: 1995

- A principios de la década de 1990, Sun Microsystems desarrolló un lenguaje de programación para los televisores interactivos.
- Este lenguaje se convirtió en Java 1994
- El equipo del proyecto Java cambió su enfoque a la web, que se estaba teniendo mucho crecimiento.
- Al año siguiente, Netscape obtuvo la licencia de Java para su uso en su navegador de Internet, Navigator.
- Java es un lenguaje de alto nivel de propósito general y es de los lenguajes más populares del mundo.
- Java se puede encontrar en todas partes, desde computadoras hasta teléfonos inteligentes y parquímetros.



```
1 public class HelloWorldApp {  
2     public static void main(String[] args) {  
3         System.out.println("Hello World!");  
4     }  
5 }
```

# PHP: 1995

- PHP fue desarrollado por Rasmus Lerdorf.
- Sus usos principales incluyen la creación y el mantenimiento de páginas web dinámicas, así como el desarrollo del lado del servidor.
- Algunas de las empresas más grandes de todo el mundo utilizan PHP, incluidas Facebook, Wikipedia, Digg, WordPress y Joomla.



```
hello.php
1 <?php
2
3 echo "hello world". "\t";
4 echo "\n";
5
6 ?>
```

# JavaScript: 1995

- JavaScript fue creado por Brendan Eich.
- Este lenguaje se usa principalmente para desarrollo web dinámico
- Casi todos los sitios web importantes utilizan JavaScript. Gmail, Adobe Photoshop y Mozilla Firefox incluyen algunos ejemplos bien conocidos.



```
console.log("Hello World!");
```

# Scala: 2003

- Desarrollado por Martin Odersky, Scala combina programación funcional matemática y programación orientada a objetos.
- La compatibilidad de Scala con Java lo hace útil para muchos tipos de desarrollo.
- LinkedIn, Twitter, Foursquare y Netflix y Google son solo algunos ejemplos de las muchas empresas que utilizan Scala en sus pilas de tecnología.



```
object HelloWorld extends App {  
  println("Hello, World!")  
}
```

# Go: 2009

- Go fue desarrollado por Google para abordar problemas que ocurren debido a grandes sistemas de software.
- Debido a su estructura simple y moderna, Go ha ganado popularidad entre algunas de las empresas de tecnología más grandes del mundo, como Google, Uber, Twitch y Dropbox.



```
package main

import "fmt"

func main() {
    fmt.Println("Hello, world!")
}
```

# Swift: 2014

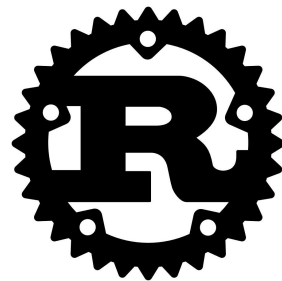
- Desarrollado por Apple como reemplazo de C, C ++ y Objective-C,
- Swift fue desarrollado con la intención de ser más fácil que los lenguajes mencionados anteriormente y permitir menos margen de error.
- La versatilidad de Swift significa que se puede utilizar para aplicaciones de escritorio, móviles y en la nube.



```
1 //  
2 //  main.swift  
3 //  HelloWorld  
4 //  
5 //  Created by TUTORIALKART on 05/03/21.  
6 //  
7  
8 import Foundation  
9  
10 print("Hello, World!")  
11  
12
```

# Rust: 2015

- Rust es un lenguaje de programación multiparadigma, de alto nivel y de propósito general diseñado para el rendimiento y la seguridad, especialmente la concurrencia segura.
- Rust es sintácticamente similar a C++
- El lenguaje surgió de un proyecto personal iniciado en 2006 por el empleado de Mozilla Graydon Hoare.
- Mozilla comenzó a patrocinar el proyecto en 2009 y el lenguaje logró compilarse por primera vez con éxito en 2011.



**The Rust  
Programming  
Language**

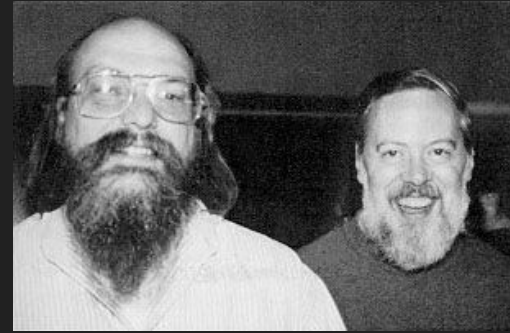
```
fn main() {  
    println!("Hello, World!");  
}
```

# Sistemas Operativos



# Unix en C: 1973

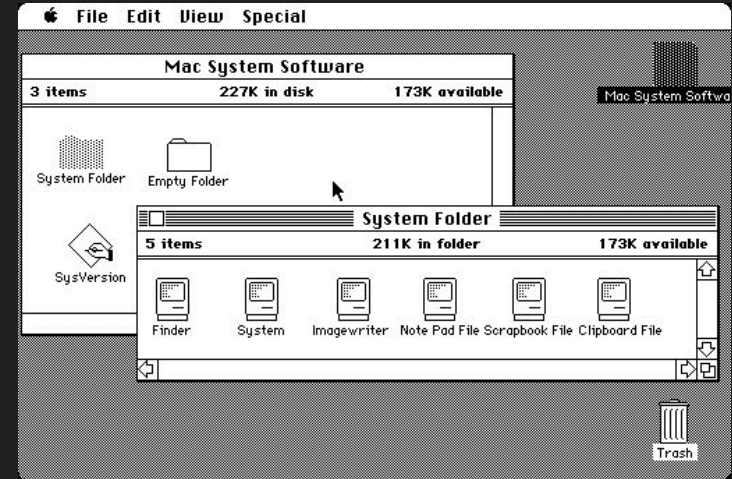
- El desarrollo de UNIX se inició en 1969 en Bell Labs con el fin de abordar las necesidades de programación, control de trabajos y uso de recursos.
- La primera versión de Unix se escribió en lenguaje ensamblador y luego se reescribió en C en 1973. En 1975, Unix se volvió libre y se distribuyó con su código fuente.



```
Terminal
-rwxr-xr-x 1 sys      52850 Jun  8 1979 hptmunix
drwxrwxr-x 2 bin      320 Sep 22 05:33 lib
drwxrwxr-x 2 root     96 Sep 22 05:46 mdec
-rwxr-xr-x 1 root    50990 Jun  8 1979 rkunix
-rwxr-xr-x 1 root    51982 Jun  8 1979 rl2unix
-rwxr-xr-x 1 sys     51790 Jun  8 1979 rphtunix
-rwxr-xr-x 1 sys     51274 Jun  8 1979 rptmunix
drwxrwxrwx 2 root      48 Sep 22 05:50 tmp
drwxrwxr-x12 root    192 Sep 22 05:48 usr
# ls -l /usr
total 11
drwxrwxr-x 3 bin      128 Sep 22 05:45 dict
drwxrwxrwx 2 dmr      32 Sep 22 05:48 dmr
drwxrwxr-x 5 bin      416 Sep 22 05:46 games
drwxrwxr-x 3 sys      496 Sep 22 05:42 include
drwxrwxr-x10 bin      528 Sep 22 05:43 lib
drwxrwxr-x11 bin      176 Sep 22 05:45 man
drwxrwxr-x 3 bin      208 Sep 22 05:46 mdec
drwxrwxr-x 2 bin       80 Sep 22 05:46 pub
drwxrwxr-x 6 root      96 Sep 22 05:45 spool
drwxrwxr-x13 root    208 Sep 22 05:42 src
# ls -l /usr/dmr
total 0
#
```

# macOS: 1984

- La primera versión de macOS es llamada "Classic Mac OS" duró de 1984 a 2001, era un sistema operativo creado por Apple que tomaba prestados conceptos de UNIX y otros conceptos de sistemas desarrollados en Xerox Parc.
- en 2001 macOS transiciona a un sistema que en su núcleo es a un sistema operativo compatible con POSIX (compatible con UNIX) construido sobre el kernel XNU que a su vez está basada en Unix.
- Tiene funciones estándar de Unix disponibles desde la interfaz de línea de comandos.



# Windows: 1985

- Windows publicó su primer lanzamiento en 1985.
- En 1991 se introdujeron versiones estables de Excel y Word que les ayudaron a dominar el mercado de los sistemas operativos.



# Linux: 1991

- En 1991, mientras estudiaba informática en la Universidad de Helsinki, Linus Torvalds inició un proyecto que más tarde se convirtió en el kernel de Linux.
- El desarrollo inicial se realizó en MINIX que está basado en UNIX y usó el compilador GNU C.
- En 1992, el software lo puso a disposición de cualquier persona de forma gratuita (Licencia pública general GNU).



# Lenguajes de Programación hoy en día

# Estructura de Proyectos

Generalmente cada lenguaje tiene una estructura de directorios estándar en donde por lo general se describen ciertos archivos comunes que se comparten para todo el programa, tales como:

- Dependencias Externas
- Configuración
- Testeo
- Código Fuente

```
├── my_project
│   ├── .gitignore
│   ├── LICENSE
│   ├── my_project
│   │   └── __init__.py
│   ├── README.md
│   ├── requirements.txt
│   ├── setup.py
│   └── tests
│       └── test_my_project.py
```

Python

```
├── README.md
├── example/
│   ├── README.md
│   ├── package.json
│   └── public/
│       ├── index.html
│       └── manifest.json
├── src/
│   ├── App.js
│   ├── index.css
│   └── index.js
└── yarn.lock
```

NodeJS

```
├── Cargo.toml
├── config
│   └── Default.toml
├── dataset
│   ├── Mutag-backup.edges
│   └── Mutag.edges
├── src
│   ├── lib.rs
│   ├── main.rs
│   └── prep
│       ├── gcn.rs
│       ├── load_data.rs
│       ├── mod.rs
│       ├── pre_process.rs
│       └── settings.rs
```

Rust

# Dependencias Externas

- Las dependencias externas nos permiten definir y utilizar código que ha sido escrito por terceros dentro de nuestro proyecto.

requirements.txt

```
xmldict  
absl-py==0.12.0  
astor==0.8.1  
cachetools==4.2.1  
certifi==2020.06.20  
chardet==3.0.4  
cyclr==0.10.0  
decorator==4.4.2  
gast==0.3.3  
google-auth==1.28.1
```

Python

Cargo.toml

```
[package]  
name = "ragle"  
version = "0.2.0"  
authors = ["Darian Harrison <darianharrison89@gmail.com>"]  
edition = "2018"  
description = "Resilient Approximate Graph Learning"  
  
[dependencies]  
petgraph = "0.6.0"  
tch = "0.5.0"  
config = "0.11"  
rand = { version = "0.8.4", features = ["std_rng"] }  
serde = { version = "1.0", features = ["derive"] }
```

Rust

package.json

```
{  
  "name": "my-app",  
  "version": "0.1.0",  
  "private": true,  
  "dependencies": {  
    "@testing-library/jest-dom": "^5.15.0",  
    "@testing-library/react": "^11.2.7",  
    "@testing-library/user-event": "^12.8.3",  
    "react": "^17.0.2",  
    "react-dom": "^17.0.2",  
    "react-scripts": "4.0.3",  
    "web-vitals": "^1.1.2"  
  },  
  "scripts": {  
    "start": "react-scripts start",  
    "build": "react-scripts build",  
    "test": "react-scripts test",  
    "eject": "react-scripts eject"  
  },  
  "eslintConfig": {  
    "extends": [  
      "react-app",  
      "react-app/jest"  
    ]  
  },  
  "browserslist": {  
    "production": [  
      ">0.2%",  
      "not dead",  
      "not op_mini all"  
    ],  
    "development": [  
      "last 1 chrome version",  
      "last 1 firefox version",  
      "last 1 safari version"  
    ]  
  }  
}
```

NodeJS

# Manejadores de Paquetes

- Los manejadores de paquetes nos ofrecen una interfaz sencilla para instalar, actualizar o eliminar dependencias externas. Para descargar programas en diferentes lenguajes solo basta con escribir un solo comando en la terminal
- npm para NodeJS:
  - `npm install <nombre-de-paquete>`
- pip para python:
  - `pip install <nombre-de-paquete>`
- Cargo para Rust:
  - `Cargo install <nombre-de-paquete>`



# Utilizar/Importar dependencias en nuestro código

Para utilizar dependencias externas:

- Primero, hay que instalarlas con algún manejador de paquetes o bien escribir el código
- Segundo, debemos importar la dependencia al contexto de este programa
- Tercero, utilizar utilizar algunas funcionalidad de la dependencia en tu programa

```
sample2.py x
1 import numpy as np
2
3 pred = np.array([1, 2, 3, 4, 2, 5, 1,2])
4 print(len(pred))
5 print(pred[7])

8
2
[Finished in 1.5s]
```

Python utiliza la dependencia “numpy”

```
var MongoClient = require('mongodb').MongoClient;
var url = 'mongodb://localhost/EmployeeDB';

MongoClient.connect(url, function(err, db) {

    var cursor = db.collection('Employee').find();

    cursor.each(function(err, doc) {

        console.log(doc);

    });

});
```

NodeJS utiliza la dependencia “mongodb”

```
use tch::{nn::Init, nn::VarStore, Device, Kind, Tensor};
use tch::manual_seed;

fn main() {

    let a_mod: Tensor = pre_process::a_mod(
        &induced_subgraph,
        &n_classes
    );
}
```

Rust utiliza algunos componentes la librería “tch”

# Nuestro Codigo Scripts

En general, los pasos para crear un programa simple son:

1. Crear un archivo propio con la terminación de cada lenguaje
  - a. Ejemplo.py (python)
  - b. Ejemplo.rs (rust)
  - c. ejemplo.js (nodejs)
2. Importar dependencias locales o externas
3. Escribir la lógica de nuestro programa
4. Ejecutar
5. Nota: el modelo de ejecución dependerá de cada lenguaje de programación

# Aspectos a considerar al elegir un lenguaje

- Dinámicos vs Estáticos
- No-tipados vs Tipados
- Interpretado vs Compilado
- OOP vs Funcional

# Prctica

# Preparar las siguientes 3 clases

Nota: La práctica la haremos juntos durante la última parte de la sesión.

- Instalar python 3.x
  - Ejecutar hola-mundo
- Instalar Nodejs
  - Ejecutar hola-mundo