



# Introducción a la Computadora

Universidad Católica Boliviana

MSc, José Jesús Cabrera Pantoja

# Hoy veremos

- Anatomía de la computadora
- Representación de información
- Programa de computadora
- Ciclo de vida de un programa
- Programa
- Variables





# Anatomía de la computadora

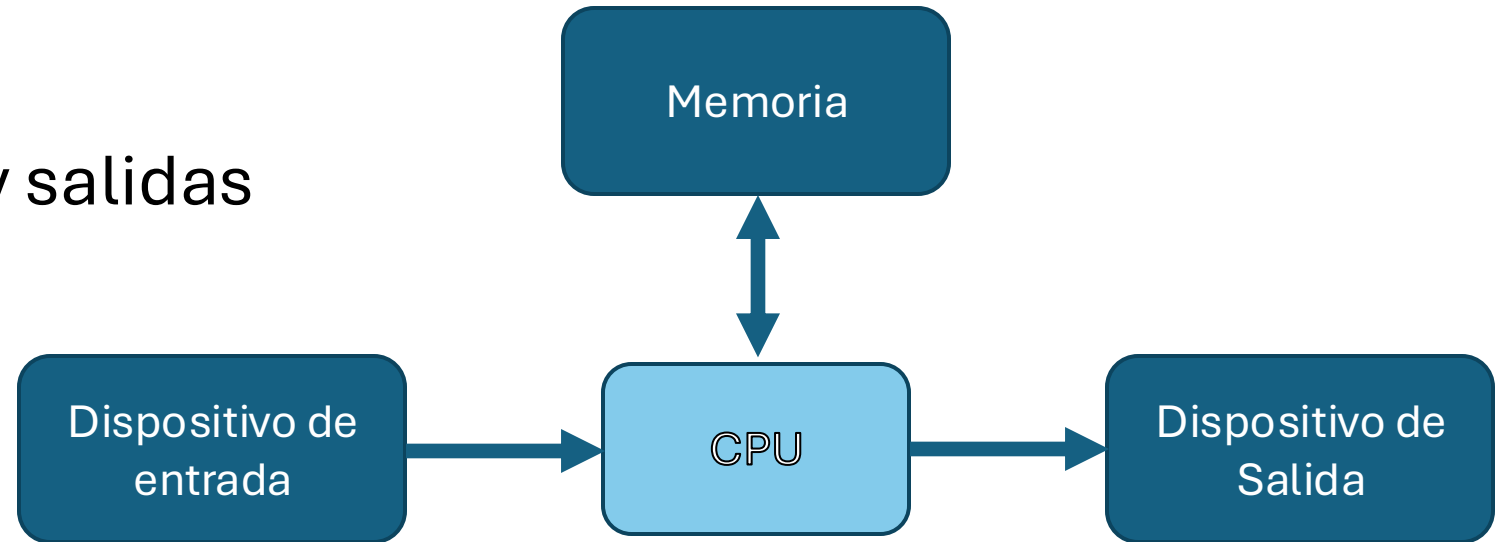
# Que es una Computadora?

- Calculadora?
- Diferencias entre Calculadora y Computadora



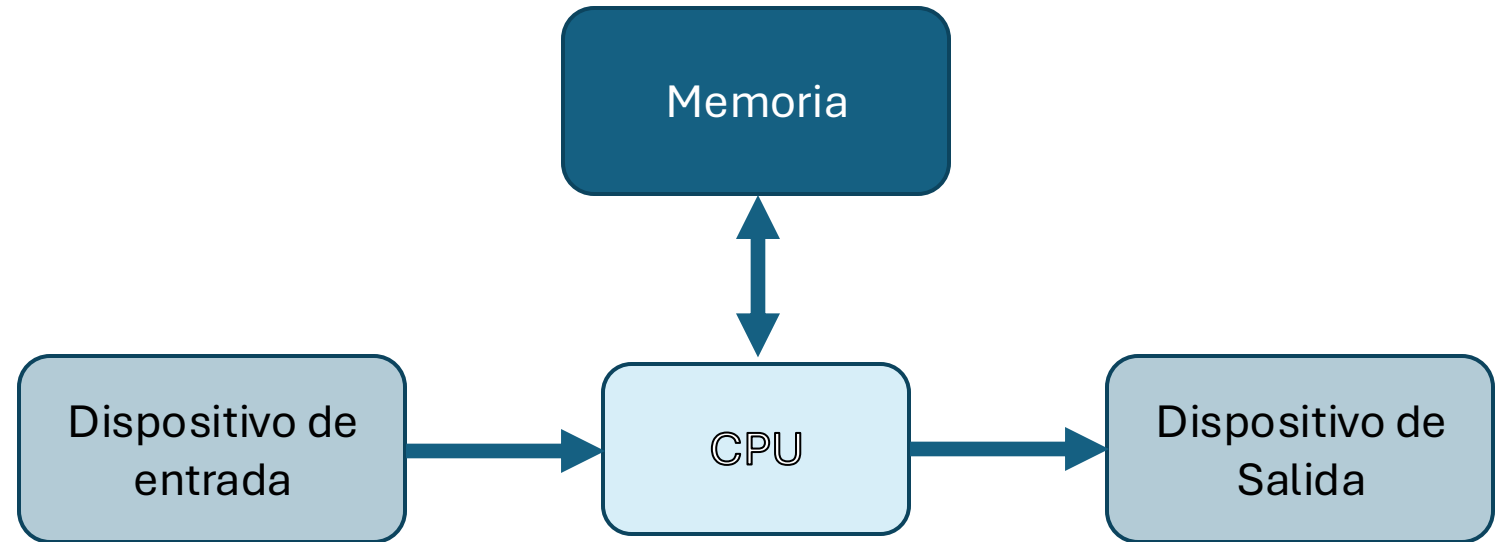
# Componentes de una computadora

- Unidad de procesamiento Central (CPU)
- Memoria
- Dispositivos de entradas y salidas (I/O Devices)



# Memoria

- Memoria principal
  - RAM (Random Access Memory)
  - ROM (Read Only Memory)
- Memoria secundaria
  - HardDrive
    - SSD





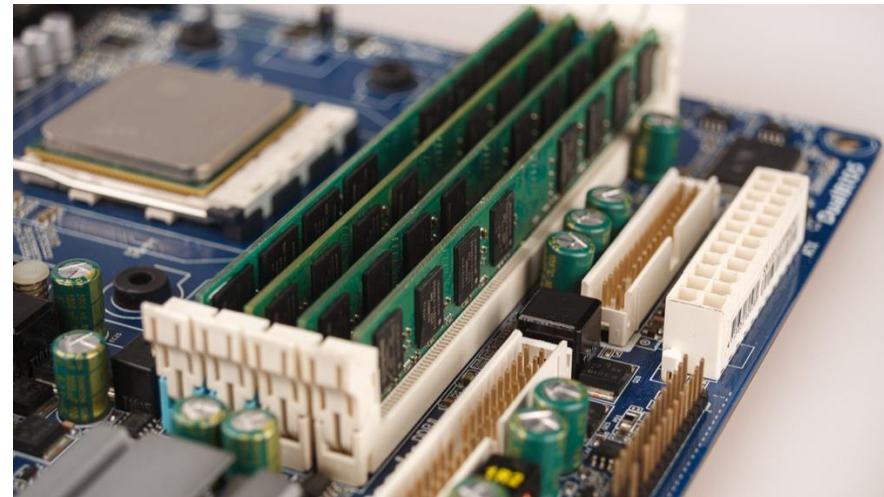
# Memoria principal

- **RAM**

- Volatile
- Rapida

- **ROM**

- No volatile
- Mas lenta que la RAM



# Memoria principal

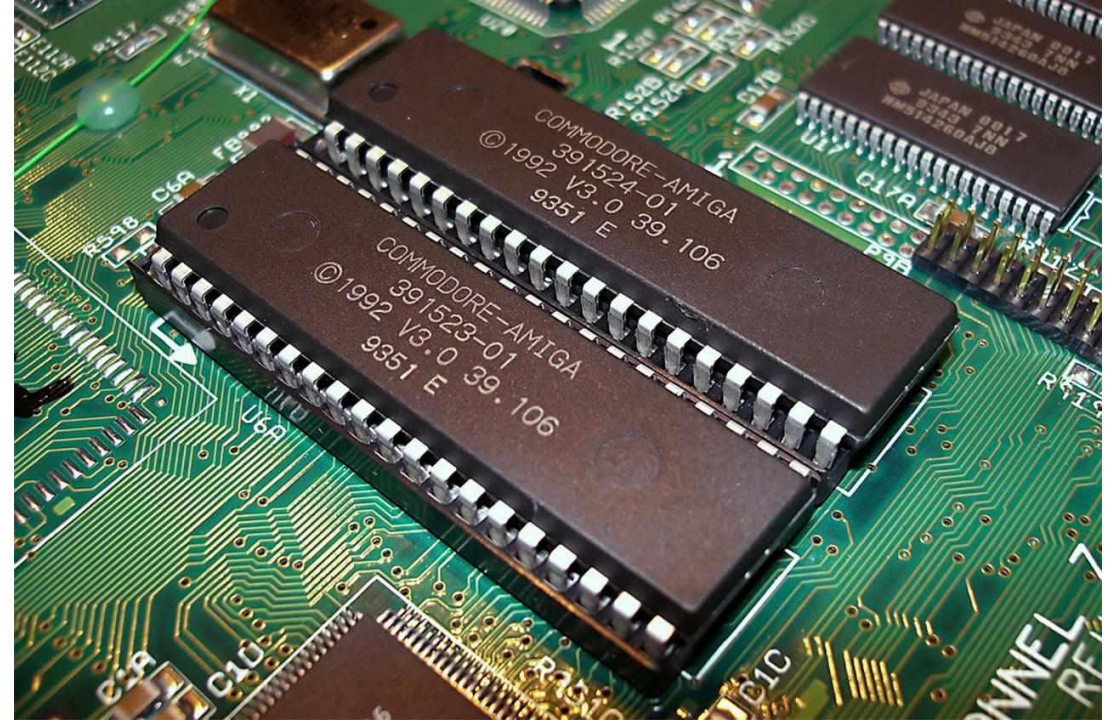
- **RAM**
  - Volatile
  - Rapida
- **ROM**
  - No volatile
  - Mas lenta que la RAM





# Memoria principal

- **RAM**
  - Volatile
  - Rapida
- **ROM**
  - No volatile
  - Mas lenta que la RAM



# Memoria secundaria

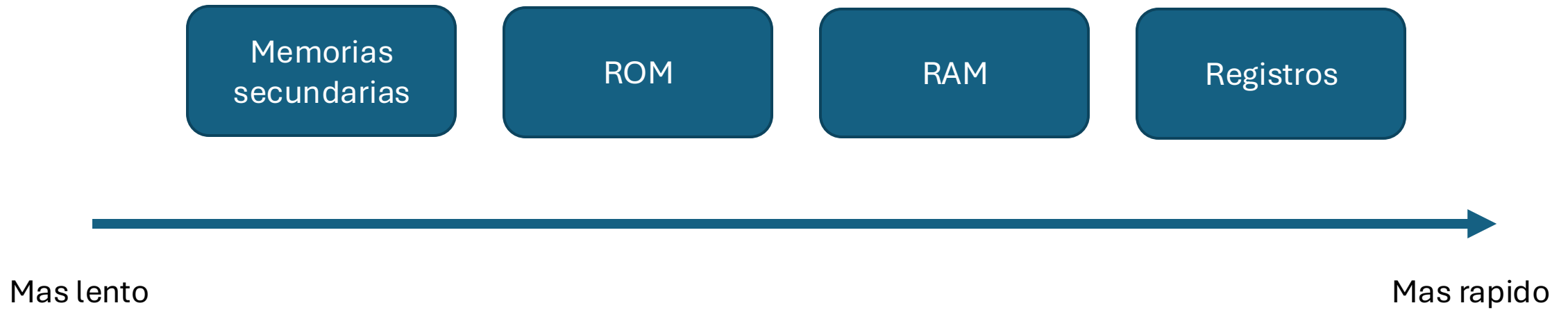
- HardDrive (HDD)
  - Barata
  - Lenta
  - No volatile
- SSD
  - Mas rápida que la HDD
  - No volatile



# Registros

- Memoria sumamente rápida
- Pequeña muy pequeña
- Se encuentra cerca del CPU para realizar operaciones rápidas

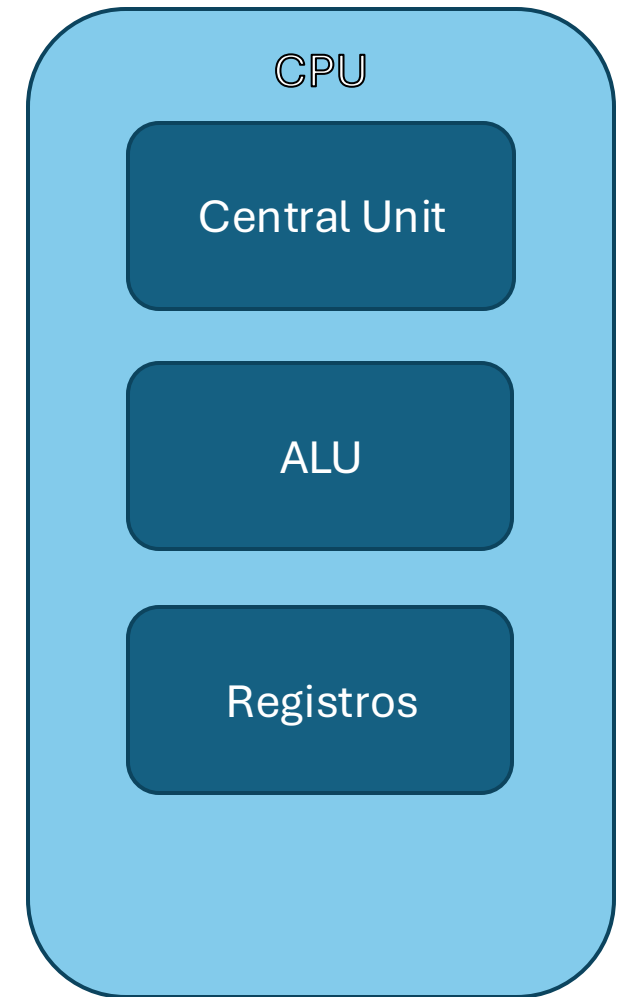
# Memorias





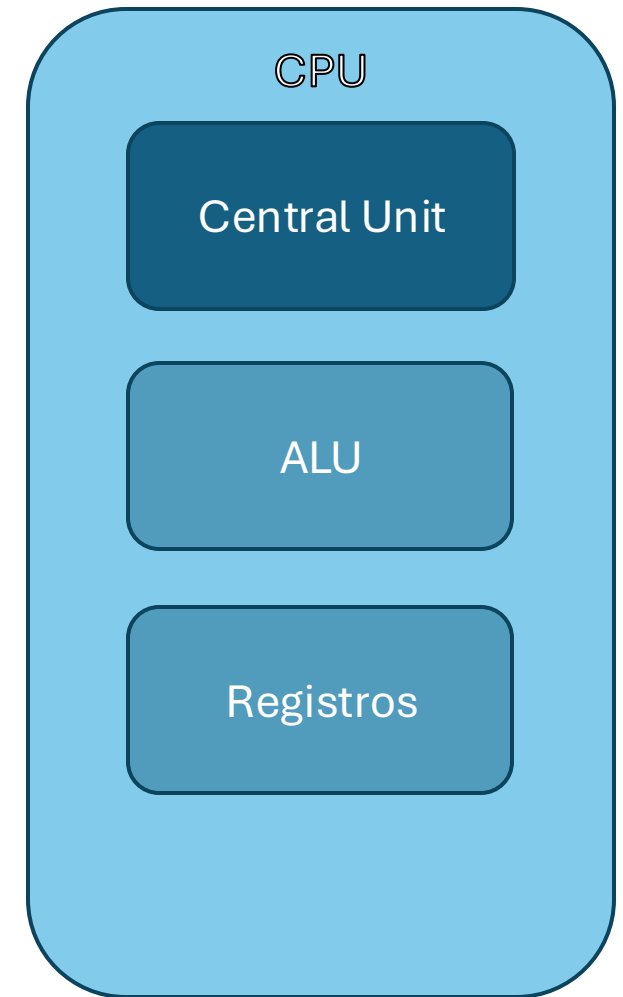
# Central Processing Unit (CPU)

- Unidad de Control
- ALU (Arithmetic Logic Unit)
- Registros



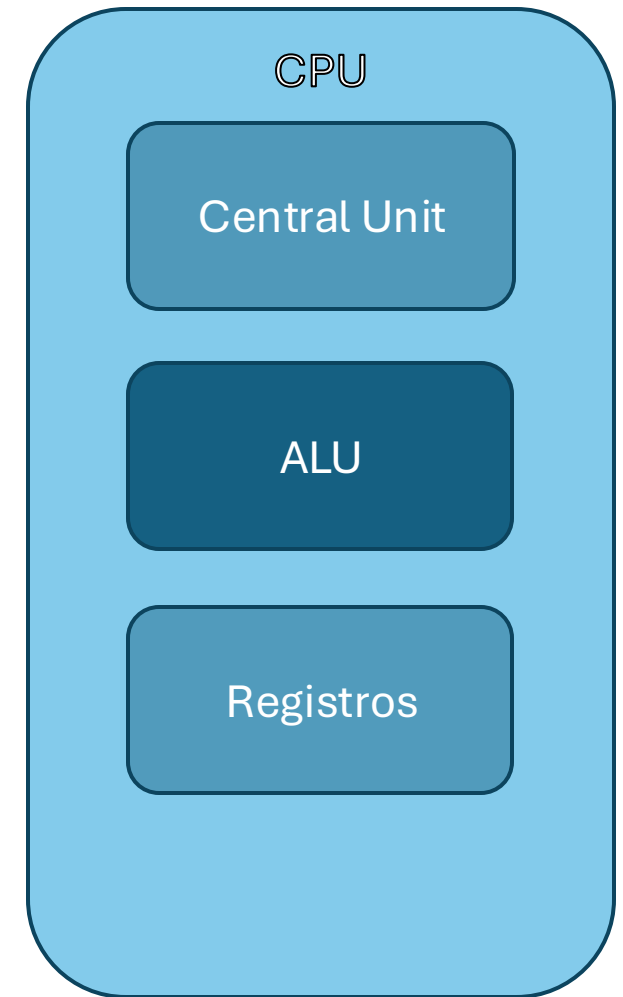
# Unidad de Control

- Carga las instrucciones de la memoria e indica a otros componentes como responder a las instrucciones

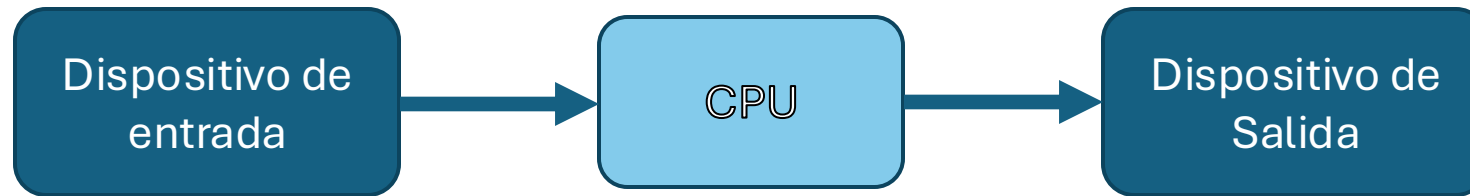


# ALU

- Encargado de realizar las operaciones matemáticas y lógicas. Por ejemplo, suma, resta, operaciones lógicas. etc

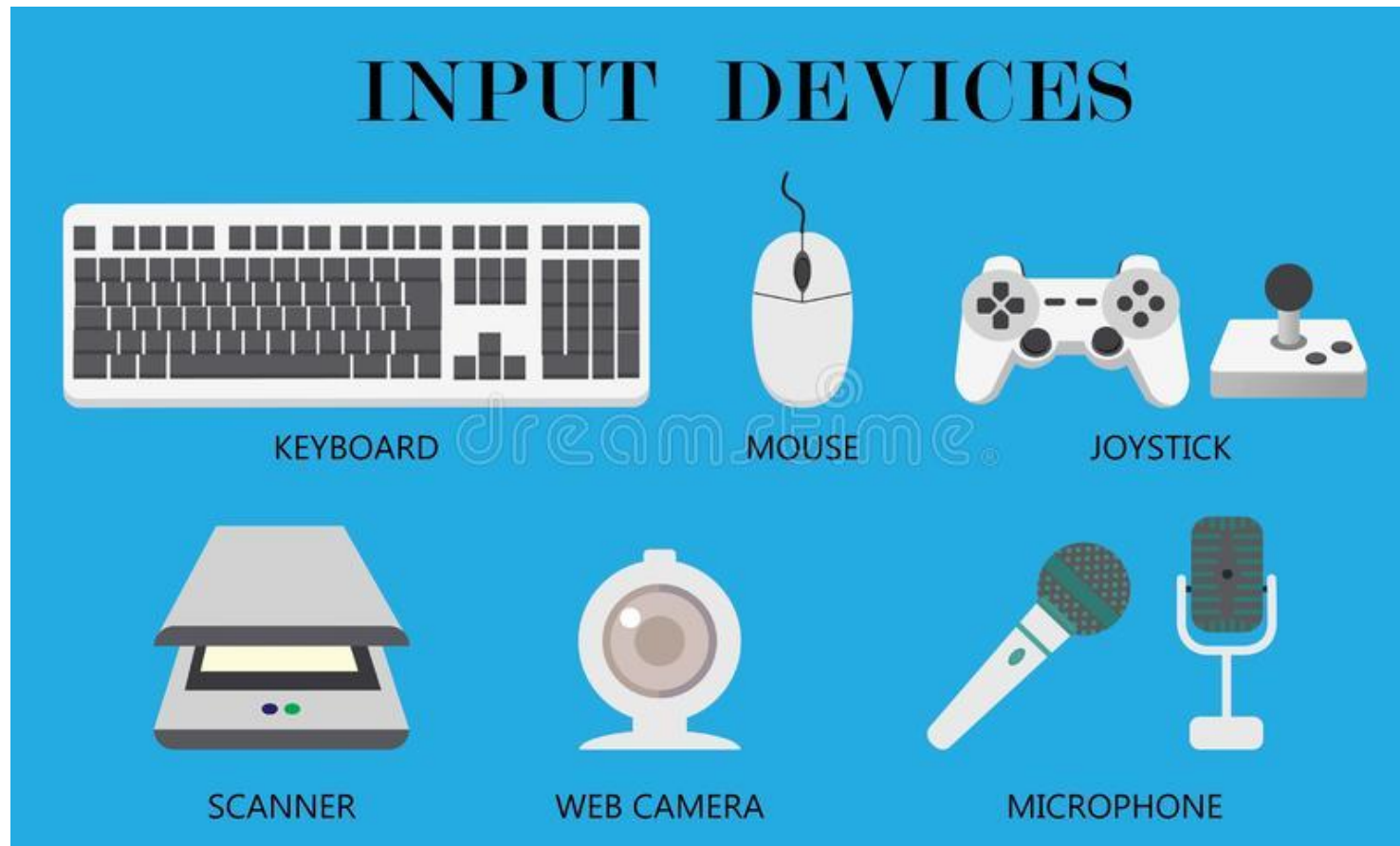


# Dispositivos de entrada-salida





# Dispositivos de entrada



# Dispositivos de salida

## OUTPUT DEVICES



MONITOR



PRINTER



SPEAKER



HEADPHONE



PROJECTOR

# Dispositivos de entrada y salida

- Puede pensar en algún dispositivo que sea de salida y de entrada a la vez?

# Dispositivos de entrada y salida

- Puede pensar en algún dispositivo que sea de salida y de entrada a la vez?







# Representación de la Información

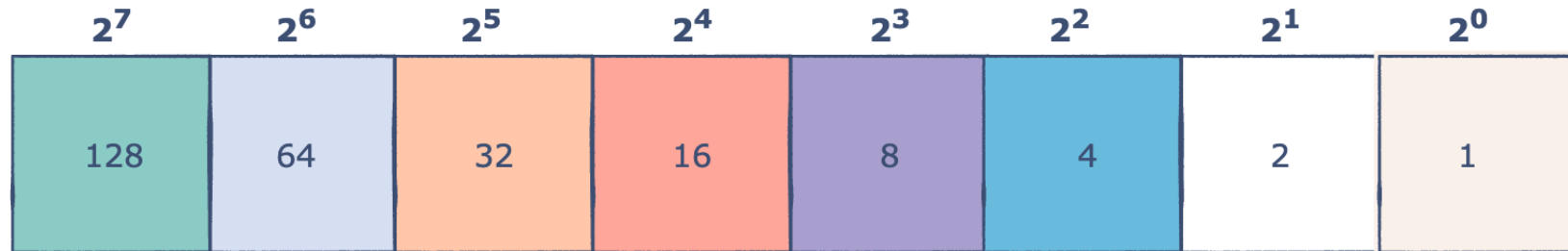
# Representación de la información

- Números Binarios:
- Bits – 0 – 1

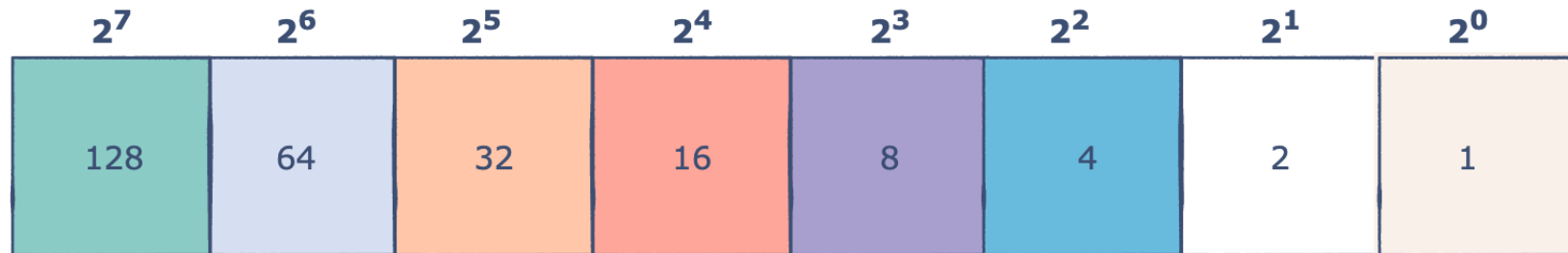
$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
128	64	32	16	8	4	2	1

# Representación de números en decimal

- 27

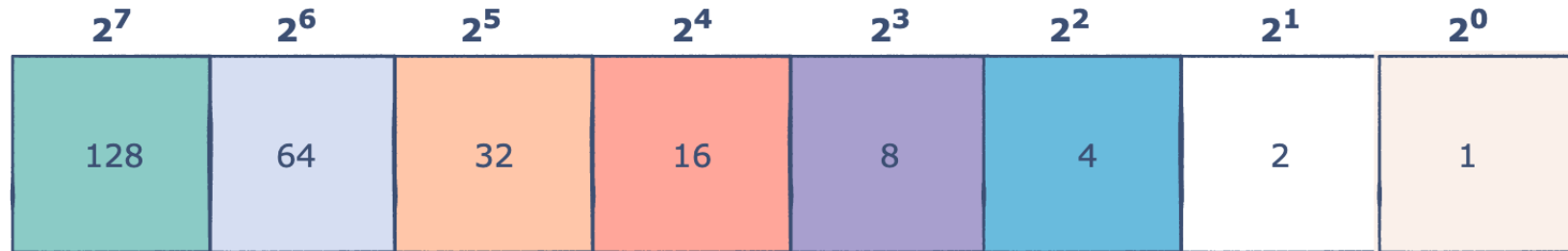


- 10

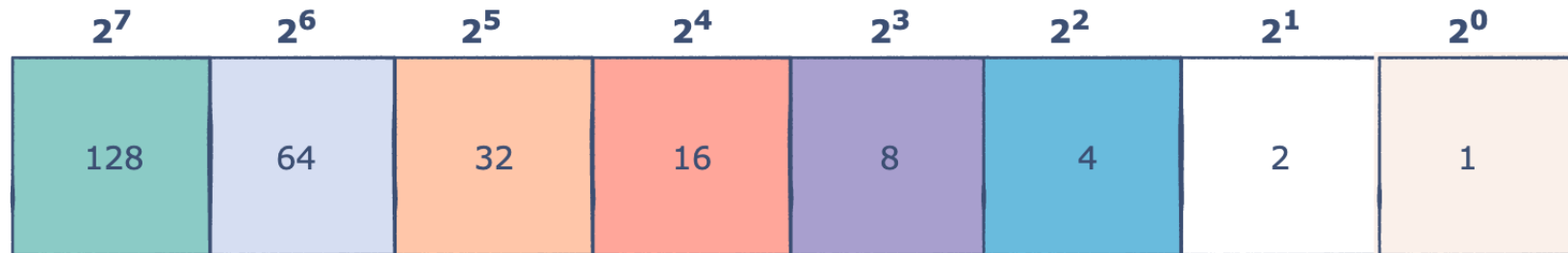


# Representación de números en decimal

- 23



- 40





# Representación de texto

- Ya que las computadoras manejan solo los números binarios se ha buscado la forma de estandarizar el mapeo de texto a números decimales
- American Standard Code for Information Interchange (ASCII)
- Los primeros treinta y dos caracteres de la tabla ASCII no se pueden imprimir. Corresponden a cosas como "comienzo de texto".



# Representación de texto

Character	ASCII code
A	65
B	66
C	67
D	68
E	69

# Representación de texto

- Busque los ASCII code para 32 y 127.
- ¿Que representan estos números?

Character	ASCII code
A	65
B	66
C	67
D	68
E	69

# Sistemas Numéricos

- Decimal
- Binario

# Sistemas Numéricos

- Decimal
- Binario
- Hexadecimal

# Sistemas Numericos

Sistema numérico	Base	Símbolos	Interpretación de "11"
Binario	2	0 1	3
Decimal	10	0 1 2 3 4 5 6 7 8 9	11
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 a b c d e f	17

# Sistemas Numericos: Hex

- Cada numero Hexadecimal tiene 4 bits:
  - 0b1111 -> 0xF
  - 0b0000 -> 0x0
  - 0b0000 1111 -> 0x0F



# Sistemas Numericos

Sistema numérico	Base	Símbolos	Interpretación de "11"
Binario	2	0 1	3
Decimal	10	0 1 2 3 4 5 6 7 8 9	11
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 a b c d e f	17

$$153_b = 1 * b^2 + 5 * b^1 + 3 * b^0$$

$$153_{16} = 1 * 16^2 + 5 * 16^1 + 3 * 16^0$$

```
1 int myNumber = 339;  
2 int myNumber = 0x153;  
3 int myNumber = 0b101010011;
```

# Sistemas Numericos

- 0b0110 en decimal
- 0b1110 en hexadecimal
- 0xC en binario
- 0xD3 en binario

# Ejercicios

0b0110

$$\textit{Decimal} \Rightarrow 0 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0$$

# Ejercicios

- 0b1110 en hexadecimal

# Ejercicios

- 0b1110 en hexadecimal

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

# Ejercicios

- 0xD3 en binario

$$\mathbf{0xD = 13 = 8 + 4 + 1 = 0b1101}$$

$$\mathbf{0x3 = 3 = 2 + 1 = 0b0011}$$

$$\mathbf{0b1101\ 0011}$$



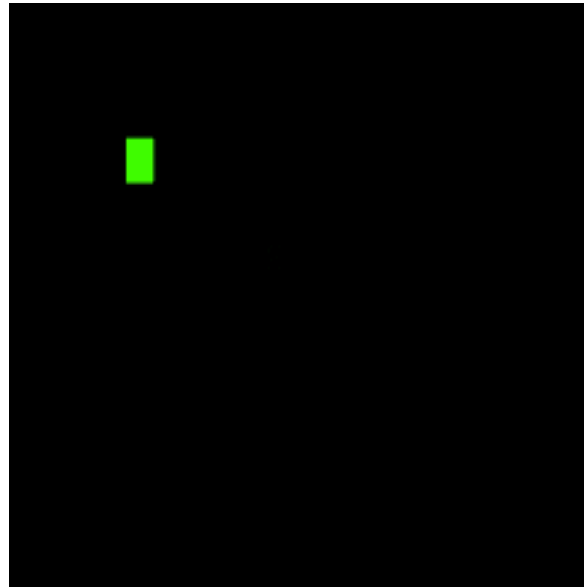
# Programa de computadora



Que es un programa de  
computadora?

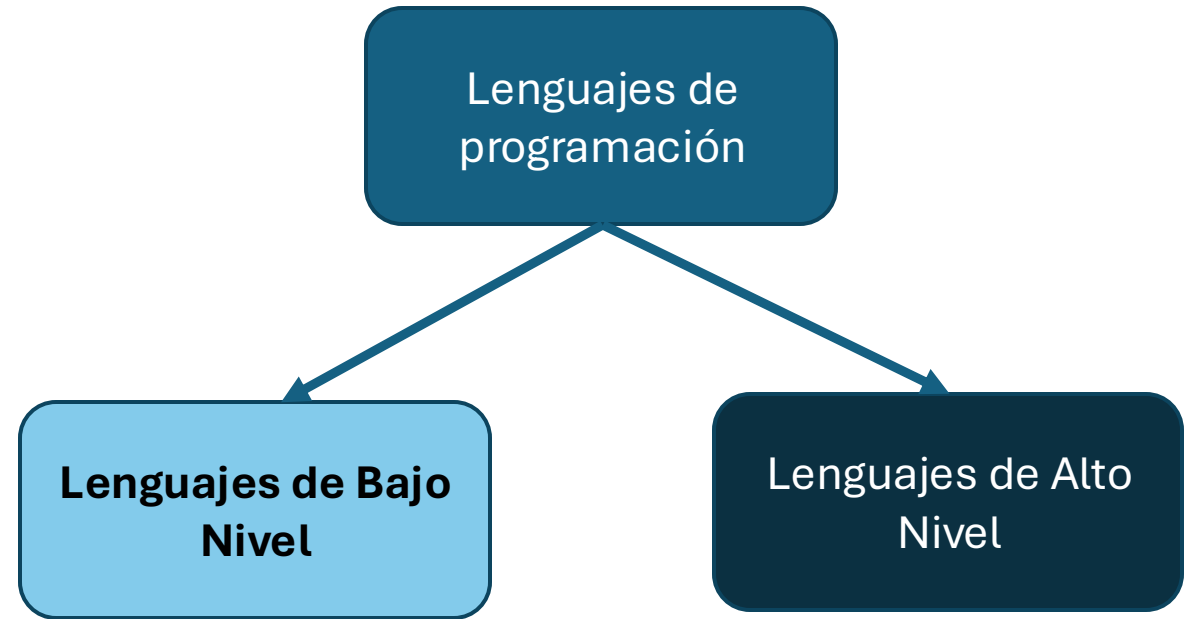
# Que es un programa de computadora?

- El conjunto de instrucciones que debe ejecutar una computadora para resolver un problema en particular se llama programa



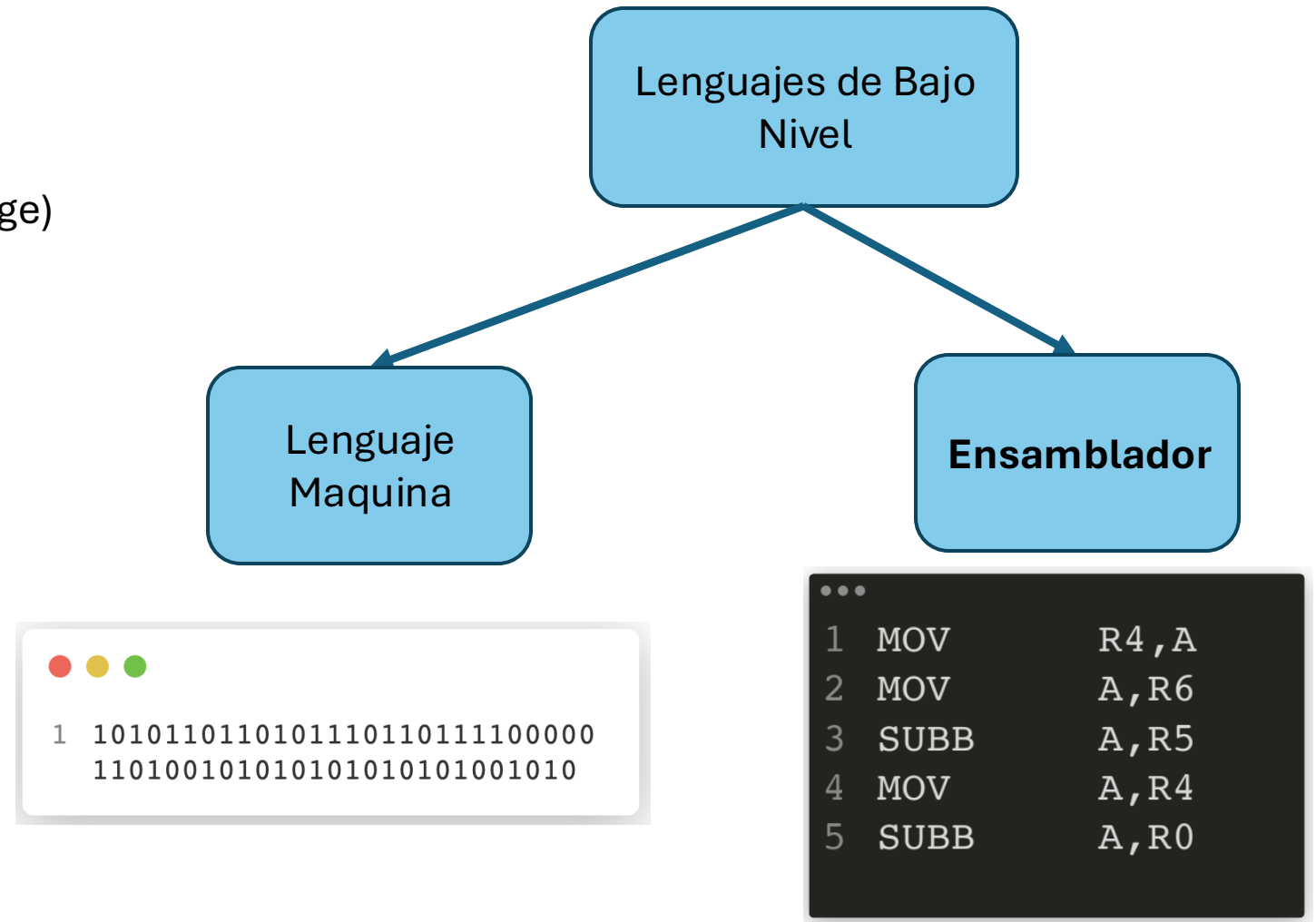
# Categorías

- Hay varias formas de categorizar los lenguajes de programación:
  - Lenguajes de Bajo Nivel
  - Lenguajes de Alto Nivel



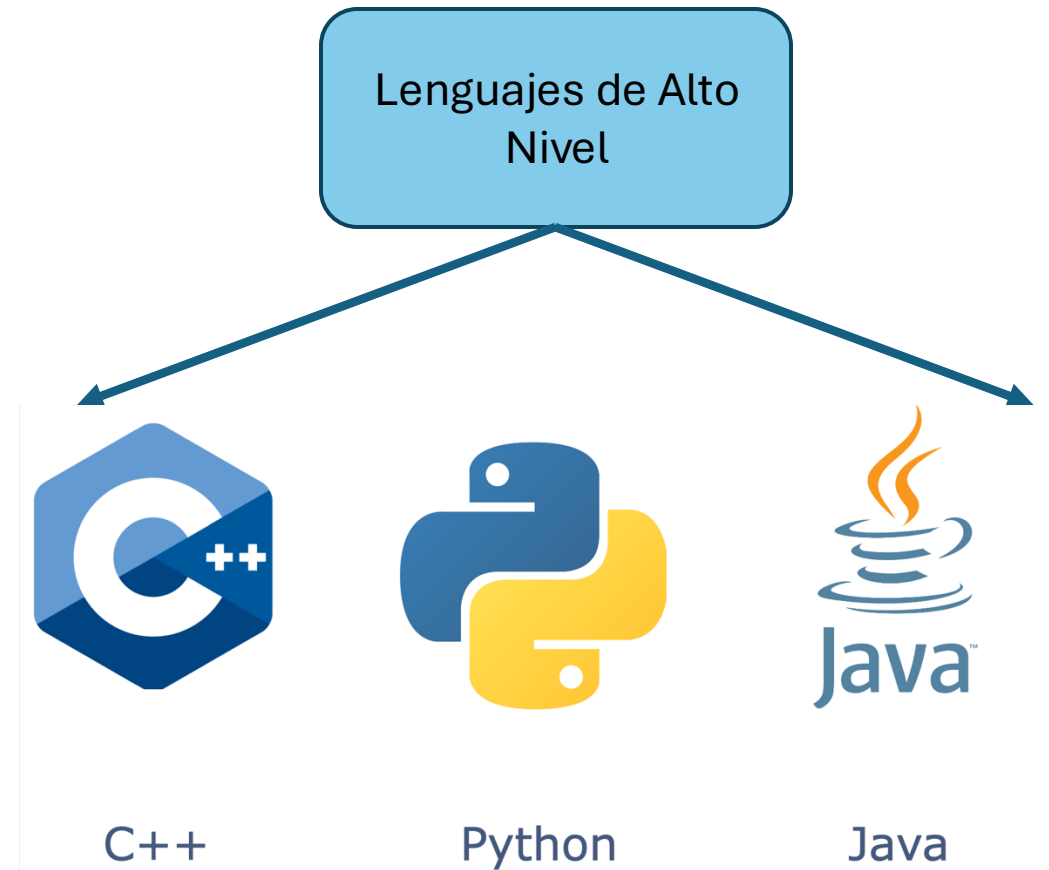
# Lenguajes de Bajo Nivel

- Código maquina (Machine Code)
- Lenguaje Ensamblador (Assembly Language)



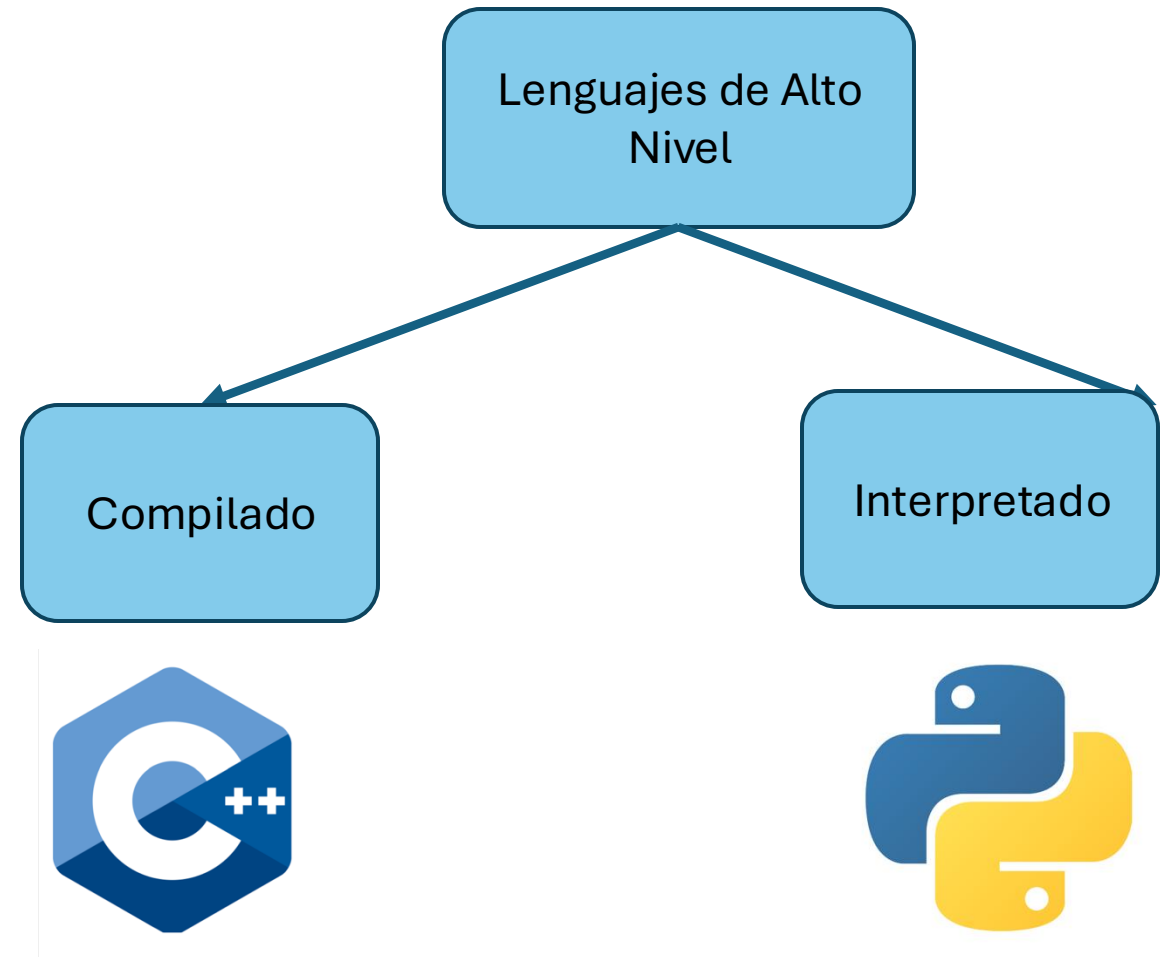
# Lenguajes de Alto Nivel

- Python
- Java
- C/C++
- JavaScript
- Y muchos otros :D



# Compilado vs Interprete

- Generalmente, los lenguajes compilados son más rápidos de ejecutar que los lenguajes interpretados.



# Lenguajes de Alto Nivel



```
#include <stdio.h>

int main() {
    int revenue = 100;
    int expenses = 50;
    int profit = revenue - expenses;

    printf("El profit es: %i", profit);

    return 0;
}
```



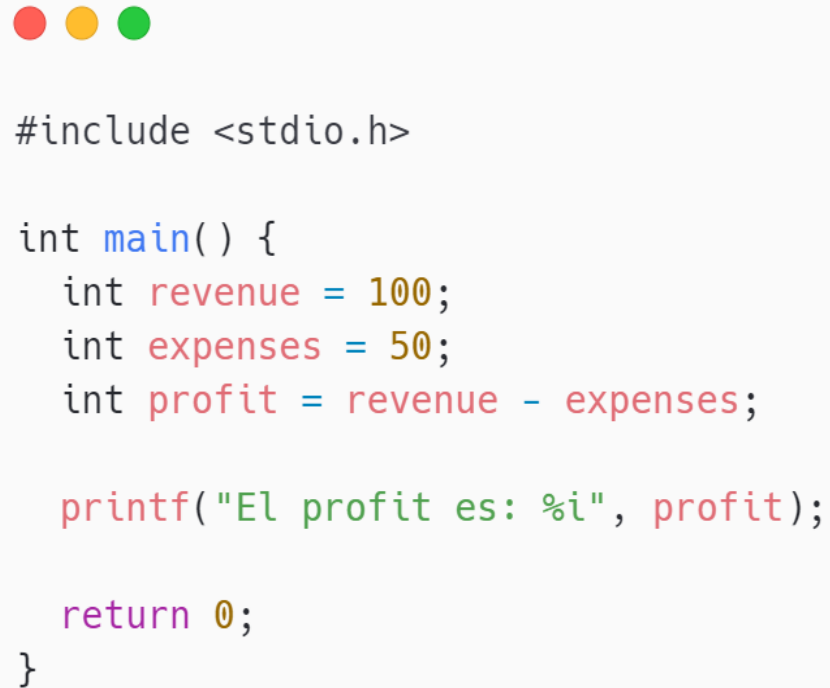
# Lenguajes de Alto Nivel



```
1 revenue = 100
2 expenses = 50
3 profit = revenue - expenses
4 print(profit)
```

# Lenguajes de Alto Nivel

C



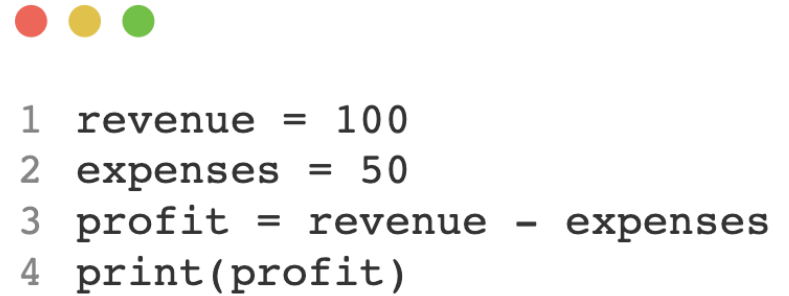
```
#include <stdio.h>

int main() {
    int revenue = 100;
    int expenses = 50;
    int profit = revenue - expenses;

    printf("El profit es: %i", profit);

    return 0;
}
```

Python



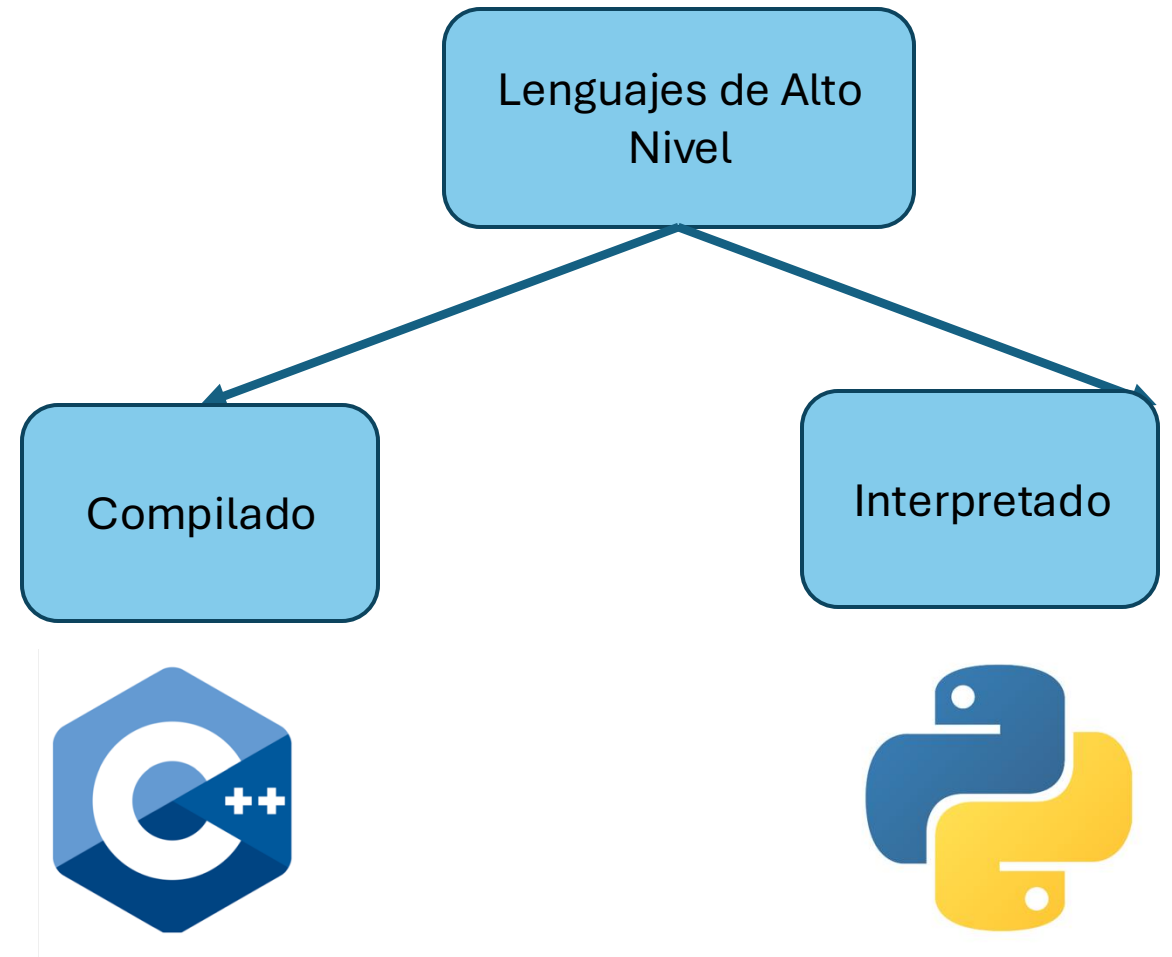
```
1 revenue = 100
2 expenses = 50
3 profit = revenue - expenses
4 print(profit)
```



# Ciclo de vida de un programa

# Compilado vs Interprete

- El CPU de una computadora solo puede comprender y ejecutar instrucciones de código de máquina.
- Se utiliza lenguajes de alto nivel como C++ y Python para escribir código.



# Compilador

- Fases de un programa Compilado
  - Source Code (Codigo Fuente)
  - Object File
  - Executable file



**Compiler**



# Compilador



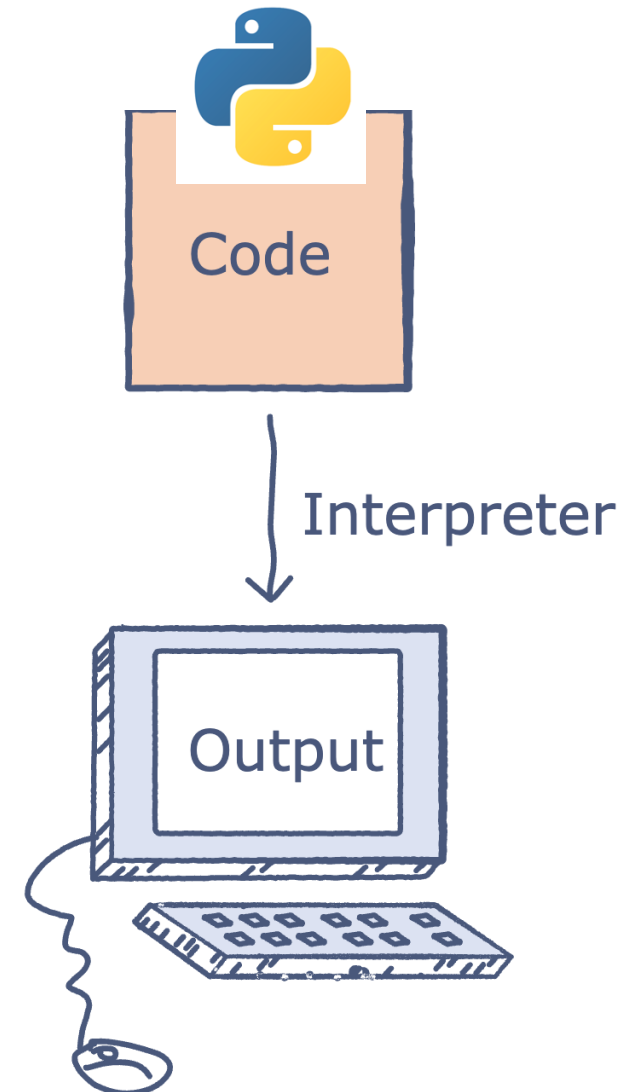
- Source Code

- Object File
- Executable File

# Interprete



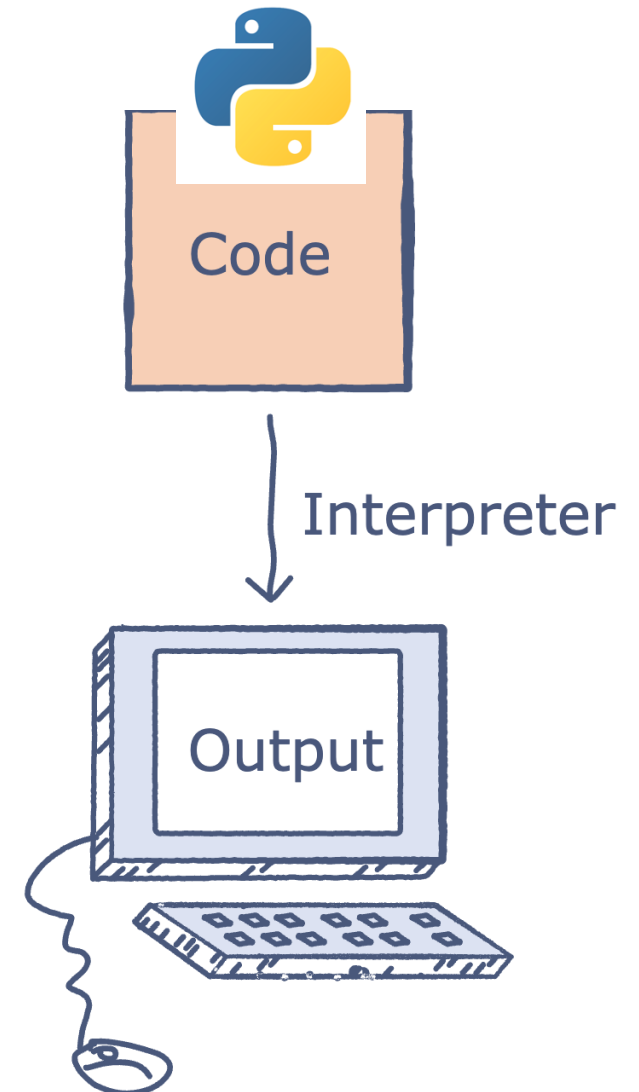
```
1 revenue = 100 ←  
2 expenses = 50  
3 profit = revenue - expenses  
4 print(profit)
```



# Interprete



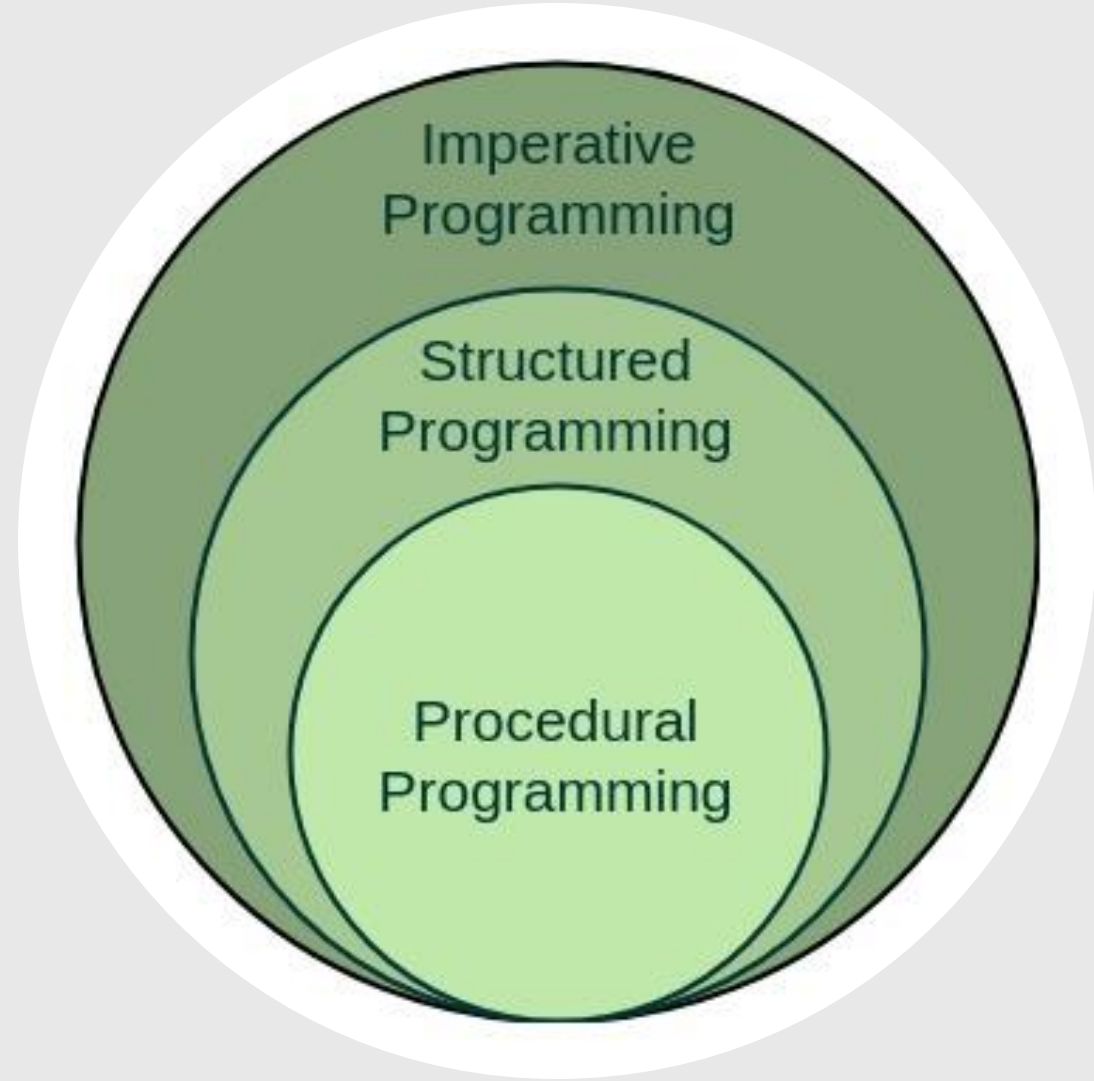
```
1 revenue = 100
2 expenses = 50
3 profit = revenue - expenses
4 print(profit)
```





# Programación procedimental

- Secuencia tras secuencia
- Una instrucción a la vez en un orden específico (Una tras otra)



# Programa



# C



```
1 #include <stdio.h>
2
3 int main() {
4     printf("Hello World!\n");
5     return 0;
6 }
```

# C++



```
1 #include <stdio.h>
2
3 int main() {
4     printf("Hello World!\n");
5     return 0;
6 }
```

Utilidades (Incluirlas) stdio

# C++



```
1 #include <stdio.h>
2
3 int main( ) {
4     printf( "Hello World!\n" );
5     return 0;
6 }
```

Punto de entrada de nuestro programa. Se llama programa principal o función principal (main)

# C++

```
4 printf( "Hello World!\n" );
```

Funcion  
para  
mostrar  
contenido  
al usuario  
en la  
terminal  
**Content  
Output**

Inicio de los  
argumentos  
de la  
funcion.

Entre comillas se  
especifica lo que se  
mostrar en la terminal.  
Hello World!

Nueva linea

Final de la  
instrucción!

# C++



```
1 #include <stdio.h>
2
3 int main() {
4     printf("Hello World!\n");
5     return 0;
6 }
```

Fin del programa. En caso  
de una ejecución exitosa

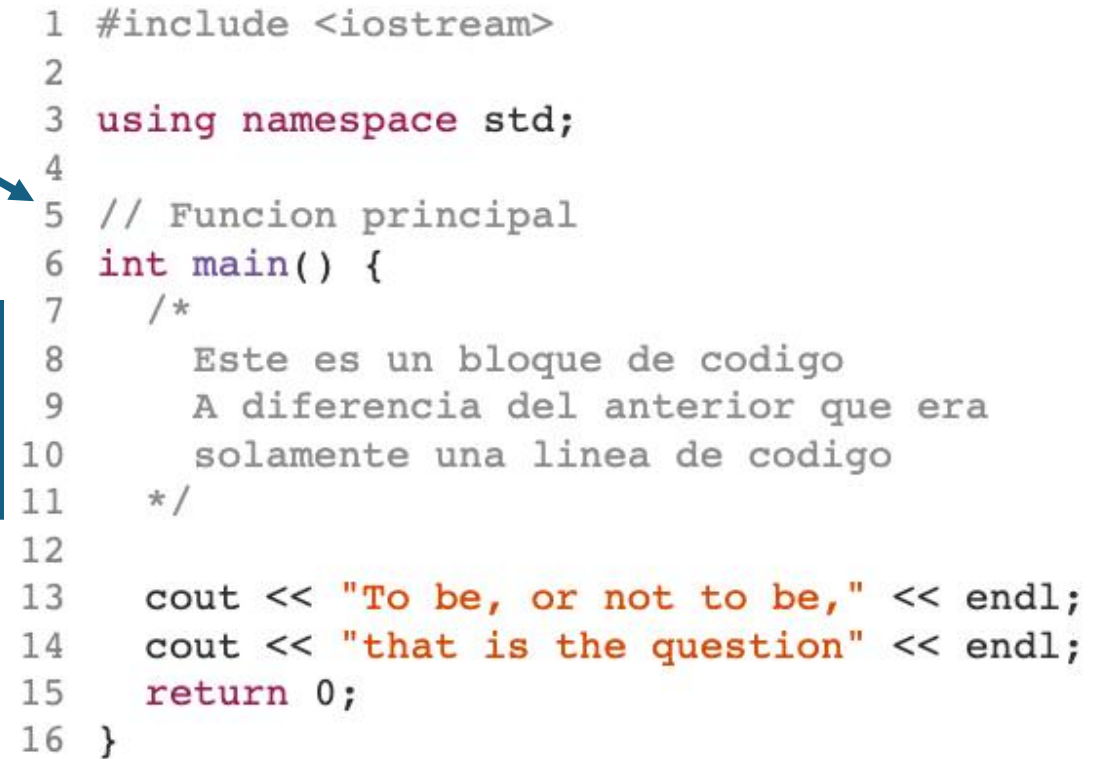
# Comentarios

- Comentarios en línea (//)
- Bloques de comentarios

/\*

aquí su comentario

\*/



```
1 #include <iostream>
2
3 using namespace std;
4
5 // Funcion principal
6 int main() {
7     /*
8         Este es un bloque de codigo
9         A diferencia del anterior que era
10        solamente una linea de codigo
11    */
12
13    cout << "To be, or not to be," << endl;
14    cout << "that is the question" << endl;
15    return 0;
16 }
```



# Blindaje

Error!!!




```
#include <stdio.h>
```

```
int main() {  
    printf("Quote from \"Hamlet\":");  
    printf("To be, or not to be,");  
    printf("that is the questions");  
  
    return 0;  
}
```

# Blindaje

- El símbolo \ ayuda a decirle al compilador "el siguiente carácter muéstralo y no lo conviertas en una instrucción"
- Si queremos mostrar el símbolo \ en consola. Se debe:
- \\hola\\
- Output: \hola\



```
#include <stdio.h>

int main() {
    printf("Quote from \"Hamlet\":");
    printf("To be, or not to be,");
    printf("that is the questions");

    return 0;
}
```



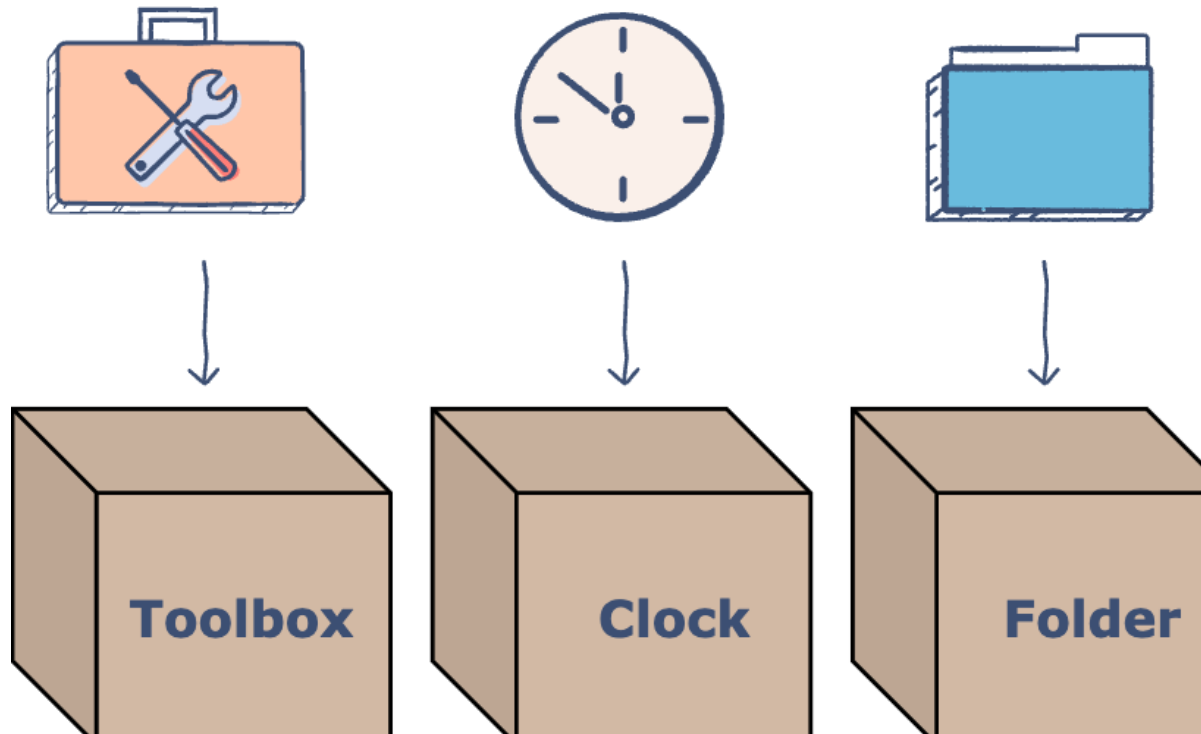
Variables

# Variables

- En términos de lenguaje de programación, una variable es una ubicación en la memoria de la computadora donde podemos almacenar datos. El valor de estos datos se puede cambiar durante la ejecución de un programa.
- Cada variable tiene un nombre único y significativo que se conoce como **identificador**.

# Variables

- Una variable almacena un valor y tiene un nombre para identificarlo.



# Variables en C++

- Declaración de una variable:
  - **tipo\_de\_variable** **nombre\_de\_variable**
- Inicialización de una variable:



```
#include <stdio.h>

int main() {
    int number;

    number = 10;

    return 0;
}
```

# Variables en C++

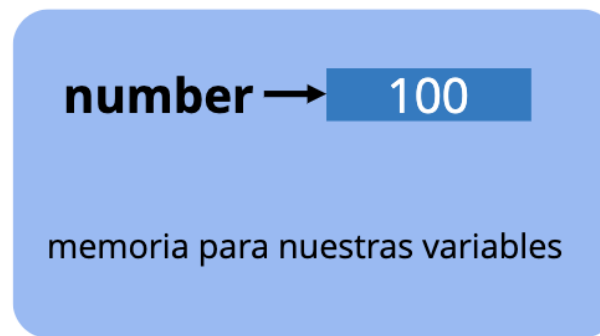
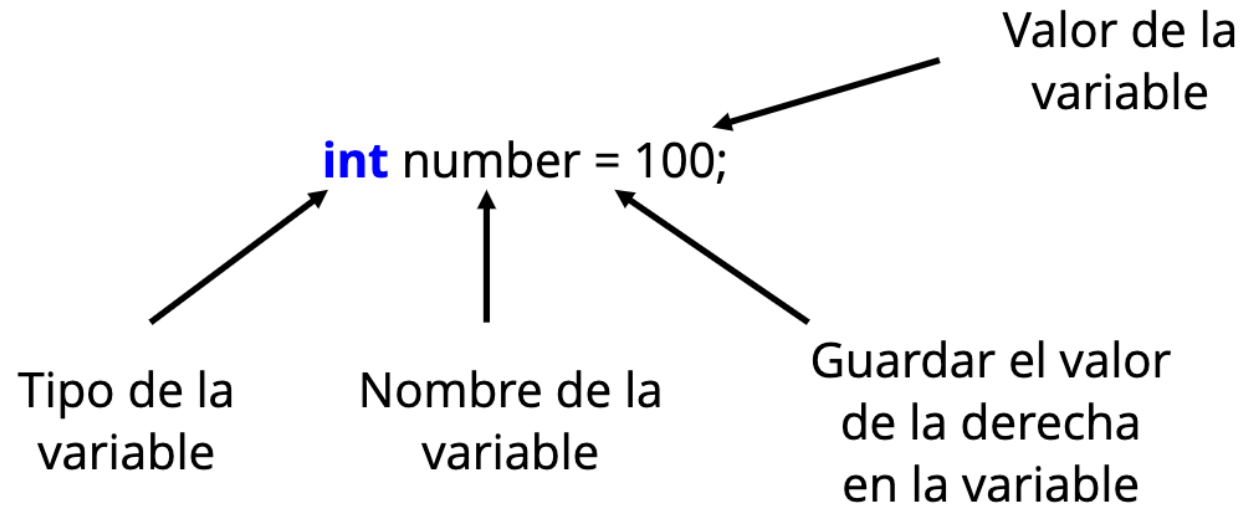
- Declaración e inicialización de una variable:
  - **tipo\_de\_variable nombre\_de\_variable = 100;**



```
#include <stdio.h>
```

```
int main() {  
    int number = 10;  
  
    return 0;  
}
```

# Variables en C



RAM



# Variables



```
#include <stdio.h>

typedef char* string;

int main() {
    // Declaracion de un variable
    // y definicion de su valor
    string text = "Hello from C++);

    // Uso de la variable
    printf("", text);

    return 0;
}
```

# Variables



```
#include <stdio.h>

typedef char* string;

int main() {
    // Declaracion de un variable
    // y definicion de su valor
    string text = "Hello from C++);

    // Uso de la variable
    printf("%s", text);

    return 0;
}
```

# Variables



```
#include <stdio.h>

typedef char* string;

int main() {
    // Declaracion de un variable
    // y definicion de su valor
    string text = "Hello from C++);
    string text2 = "Otro texto";

    // Uso de la variable
    printf("%s %s", text, text2);

    return 0;
}
```

# Variables



```
#include <stdio.h>

typedef char* string;


int main() {
    // Declaracion de un variable
    // y definicion de su valor
    string text = "Hello from C++);
    string text2 = "Otro texto";

    // Uso de la variable
    printf("%s %s", text, text2);

    // Escribimos un nuevo valor en la varibale text
    // El valor de la variable text cambia
    text = "Happy coding!";
    printf("%s", text);

    return 0;
}
```

# Variables



```
#include <stdio.h>

typedef char* string;

int main() {
    // Declaracion de un variable
    int current_amount;

    // Definicion de su valor
    current_amount = 100;

    // Mostrar el valor de la variable current_amount
    printf("%i", current_amount);

    // Actualizar el valor de la variable current amount
    current_amount = 120;

    // Mostrar el nuevo valor de la variable
    printf("%i", current_amount);

    return 0;
}
```

# Analice el código

- En una hoja escriba los resultados esperados del siguiente código

```
#include <stdio.h>

int main() {
    // Declaracion de un variable
    int revenue = 100;
    int expenses;

    expenses = 50;

    int profit = revenue - expenses;

    printf("%i\n", profit);

    profit = profit * profit;
    printf("%i\n", profit);

    profit += profit;
    printf("%i\n", profit);

    return 0;
}
```

# Variables

- Dos tipos de datos hemos revisado:
  - Carácter (**char**)
  - Conjunto de Caracteres (**string**)
  - Enteros (**int**)
- Otros tipos de datos:
  - Flotante (**float**)
  - Dobles (**double**)
  - Booleanos (**bool**)

# C Compilador Online