FRAGUA TECH

Lenguajes de Programación

Clase 2 — Introducción a Lenguajes de Programación

↓

¿Qué es un lenguaje de programación?

Una computadora solo entiende un idioma: secuencias de **unos y ceros** (código binario). Los lenguajes de programación son el **puente** entre el pensamiento humano y la ejecución de la máquina.

Alto nivel
Python, JavaScript, Ruby
Cercano al humano

Medio nivel
C, C++, Rust
Balance humano/máquina

Bajo nivel
Assembly
Cercano a la máquina

Máquina
Código binario
Lo que ejecuta la CPU

Alto nivel

Python

Una sola línea basta para imprimir un mensaje. El código se lee casi como inglés.

```
1  print("Hola, mundo!")
```

1 línea de código

Medio nivel

C++

Necesitamos incluir librerías, definir una función `main` y manejar la salida explícitamente.

```cpp
#include <iostream>

int main() {
    std::cout << "Hola, mundo!" << std::endl;
    return 0;
}
```

6 líneas de código

Bajo nivel

Assembly x86

Debemos manejar registros, syscalls y direcciones de memoria directamente.

```
1  section .data
2      msg db "Hola, mundo!", 0xa
3      len equ $ - msg
4  section .text
5      global _start
6  _start:
7      mov rax, 1
8      mov rdi, 1
9      mov rsi, msg
10     mov rdx, len
11     syscall
12     mov rax, 60
13     xor rdi, rdi
14     syscall
```

14 líneas — el mismo resultado requiere más código a menor nivel de abstracción

📚 Material complementario ▾

Compilados, Transpilados e Interpretados

Tres formas distintas de pasar de código fuente a ejecución. TypeScript, por ejemplo, no es un lenguaje compilado: es un lenguaje **transpilado** a JavaScript.

Compilados

`Código fuente → [Compilador] → Ejecutable → CPU`

Se traduce **todo** antes de ejecutarse

Errores detectados al compilar

C++ Rust Go Java

Interpretados

`Código fuente → [Intérprete] → Línea por línea`

Se traduce y ejecuta **en tiempo real**

Cambias el código y lo ejecutas inmediatamente

JavaScript Python Ruby PHP

TypeScript — Transpilado

TypeScript no es un lenguaje compilado tradicional: se **transpila** a JavaScript. Un transpilador traduce código de un lenguaje a otro del **mismo nivel de abstracción**, no a código máquina.

El transpilador `tsc` verifica los tipos y genera un archivo `.js` que luego se interpreta.

```
archivo.ts → [tsc · transpilador] → archivo.js → [Node · intérprete] → Resultado
```

```
$ tsc hola.ts    # transpila
$ node hola.js   # ejecuta
```

Errores al transpilar | Mismo nivel de abstracción | Requiere retranspilar

JavaScript — Interpretado

JavaScript se **interpreta** línea por línea en tiempo real. No necesita un paso de compilación: cambias el código y lo ejecutas directamente.

```
archivo.js → [Intérprete lee línea] → [Traduce] → [Ejecuta] → Siguiente línea
```

```
$ node hola.js   # ejecuta directamente
```

Desarrollo rápido | Sin paso de compilación | Errores en tiempo de ejecución

La realidad es más compleja

Java

Compila a **bytecode** que ejecuta una máquina virtual (JVM)

JavaScript (V8)

Motores modernos usan compilación **Just-In-Time** (JIT)

TypeScript

Se **transpila** a JavaScript, que luego se interpreta

Analogía

📖 **Compilador**

Traduce **todo el libro** antes de leerlo. Tarda más al inicio, pero la lectura es rápida.

🗣️ **Intérprete**

Te va traduciendo **página por página** en voz alta. Empiezas rápido, pero la lectura es más lenta.

📚 **Material complementario** ▾

Sistemas de tipado

Los lenguajes se clasifican en **dos ejes independientes**. Confundirlos es un error muy común.

Estático

Tipos verificados **antes de ejecutar** (al compilar o transpilar)

TypeScript, C++, Rust, Java

Dinámico

Tipos verificados al **ejecutar**

Python, JavaScript, Ruby

Fuerte

No convierte tipos automáticamente

TypeScript, Python, Rust

Débil

Convierte tipos implícitamente

JavaScript, C

Estático vs Dinámico

TypeScript (estático)

```typescript
let edad: number = 25;
edad = "veinticinco"; // ERROR al transpilar
```

El error se detecta **antes** de ejecutar

JavaScript (dinámico)

```javascript
let edad = 25;
edad = "veinticinco";  // Válido
let r = edad + 1; // ERROR en ejecución
```

El error aparece **al ejecutar**

Fuerte vs Débil

Python (fuerte): se rehúsa a adivinar

```python
resultado = "5" + 3  # TypeError!
```

JavaScript (débil): convierte automáticamente

```javascript
let resultado = "5" + 3;
// "53" (concatenó!)
let otro = "5" - 3;
// 2 (restó!)
```

Error común: Confundir estático/dinámico con fuerte/débil. Son dos ejes independientes. Python es dinámico **Y** fuerte. JavaScript es dinámico **Y** débil.

Tabla resumen de tipado

Lenguaje	Estático / Dinámico	Fuerte / Débil
TypeScript	Estático	Fuerte
JavaScript	Dinámico	Débil
Python	Dinámico	Fuerte
C++	Estático	Moderado
Rust	Estático	Muy fuerte

📚 **Material complementario** ▾

Panorama de lenguajes en 2026

Lenguaje	Uso principal	¿Por qué aprenderlo?
JavaScript / TypeScript	Web (frontend y backend)	Es el lenguaje de la web
Python	IA/ML, scripting, backend	Fácil de aprender, enorme ecosistema
Rust	Sistemas, WebAssembly	Seguridad de memoria, el futuro de sistemas
Go	Backend, DevOps	Simple, rápido, excelente para la nube
Java / Kotlin	Enterprise, Android	Masivo en empresas grandes
Swift	iOS, macOS	Apps para Apple
C#	Enterprise, videojuegos (Unity)	Ecosistema Microsoft y gaming

¿Por qué TypeScript en este curso? TypeScript combina los beneficios del tipado estático con el ecosistema de JavaScript, haciéndolo ideal para aprender conceptos de transpilación y desarrollo web al mismo tiempo.

📚 **Material complementario** ▼

Fundamentos de Programación

Antes de escribir código, entendamos los **bloques fundamentales** que comparten todos los lenguajes de programación.

Variables | Operadores | Arreglos | Condicionales | Bucles | Funciones | Clases

Variables

Una variable es un **contenedor con nombre** que almacena un valor en memoria. Piensa en ella como una caja etiquetada.

```
// Declaración con tipo explícito
let nombre: string = "Ana";
let edad: number = 22;
let activo: boolean = true;

// const = no se puede reasignar
const PI: number = 3.1416;
```

string	number	boolean
Texto	Números	true / false

📚 Material complementario ▾

Operadores

Símbolos que realizan operaciones sobre valores y variables.

Aritméticos

```
+  5 + 3   // 8
-  10 - 4  // 6
*  3 * 7   // 21
/  20 / 4  // 5
%  10 % 3  // 1 (residuo)
```

Comparación

```
===  // igual estricto
!==  // diferente
>    // mayor que
<    // menor que
>=   // mayor o igual
```

Lógicos

```
&&  // AND (y)
||  // OR (o)
!   // NOT (negación)
```

Combinan condiciones booleanas

Arreglos (Arrays)

Una **lista ordenada** de elementos accesibles por su posición (índice, empezando en 0).

```ts
const frutas: string[] = ["manzana", "banana", "naranja"];

console.log(frutas[0]);    // "manzana"
console.log(frutas.length); // 3

// Métodos comunes
frutas.push("uva");        // agregar al final
frutas.pop();              // quitar del final

// Recorrer con for...of
for (const fruta of frutas) {
  console.log(fruta);
}
```

.push() .pop() .length .map() .filter()

📚 Material complementario ▾

Condicionales

Permiten que el programa tome **decisiones** ejecutando código diferente según una condición.

if / else if / else

```
const nota = 85;

if (nota >= 90) {
  console.log("Excelente");
} else if (nota >= 70) {
  console.log("Aprobado");
} else {
  console.log("Reprobado");
}
```

Operador ternario

```
// condición ? sí : no

const mensaje = nota >= 70
  ? "Aprobado"
  : "Reprobado";
```

Forma compacta para condiciones simples con dos resultados posibles.

Bucles

Repiten un bloque de código **múltiples veces**, evitando escribir lo mismo una y otra vez.

for

```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```

Cuando sabes cuántas veces repetir

while

```
let n = 1;
while (n <= 10) {
  console.log(n);
  n++;
}
```

Mientras una condición sea verdadera

for...of

```
const nums = [10, 20, 30];
for (const n of nums) {
  console.log(n);
}
```

Para recorrer arreglos directamente

📚 **Material complementario** ▾

Funciones

Bloques de código **reutilizables** que realizan una tarea específica. Reciben datos (parámetros) y pueden devolver un resultado.

Función clásica

```typescript
function saludar(nombre: string): string {
  return `Hola, ${nombre}`;
}

console.log(saludar("Ana"));
// "Hola, Ana"
```

Arrow function

```typescript
const sumar = (a: number, b: number)
  : number => {
  return a + b;
};

// Forma corta (una línea)
const doble = (n: number) => n * 2;
```

Principio clave: Si copias y pegas código, probablemente necesitas una función.

📚 **Material complementario** ▾

Clases

Una clase es un **plano o molde** para crear objetos. Define qué propiedades y métodos tendrá cada objeto creado a partir de ella.

```
class Persona {
  nombre: string;
  edad: number;

  constructor(nombre: string, edad: number) {
    this.nombre = nombre;
    this.edad = edad;
  }

  presentarse(): string {
    return `Soy ${this.nombre}, tengo ${this.edad} años`;
  }
}

const ana = new Persona("Ana", 22);
console.log(ana.presentarse());
```

class · constructor · this · new · métodos

Herencia

Clases — Parte 2

Una clase puede **heredar** propiedades y métodos de otra clase padre, evitando duplicar código.

```
class Animal {
  nombre: string;
  constructor(nombre: string) {
    this.nombre = nombre;
  }
  hablar(): string {
    return "...";
  }
}

// Perro hereda de Animal
class Perro extends Animal {
  hablar(): string {
    return `${this.nombre} dice: Guau!`;
  }
}

const rex = new Perro("Rex");
console.log(rex.hablar());
// "Rex dice: Guau!"
```

extends

super()

La clase hija hereda todo del padre

Polimorfismo

Clases — Parte 3

Distintas clases pueden **responder al mismo mensaje de forma diferente**. El mismo método, múltiples comportamientos.

```
class Gato extends Animal {
  hablar(): string {
    return `${this.nombre} dice: Miau!`;
  }
}

// Mismo tipo, diferente comportamiento
const animales: Animal[] = [
  new Perro("Rex"),
  new Gato("Michi"),
];

for (const a of animales) {
  console.log(a.hablar());
}
// "Rex dice: Guau!"
// "Michi dice: Miau!"
```

Polimorfismo = "muchas formas". Cada subclase redefine `hablar()` a su manera, pero se invocan con la misma interfaz.

📚 Material complementario ▾

Práctica con TypeScript

Primero necesitamos **Node.js** y **TypeScript** instalados.

```
$ # Instalar Node.js desde nodejs.org
$ npm install -g typescript ts-node
$ tsc --version    # verificar instalación
```

Windows: Descargar de nodejs.org | Mac: `brew install node` | Linux: `sudo apt install nodejs npm`

Tu primer programa

Crea un archivo `hola.ts`:

```
1  console.log("Hola, mundo!");
```

Transpila y ejecuta:

```
$  tsc hola.ts
$  node hola.js
>  Hola, mundo!
```

O usa `npx ts-node hola.ts` para transpilar y ejecutar en un solo paso.

Variables y tipos

```typescript
const nombre: string = "Juan";
let edad: number = 25;
const altura: number = 1.75;
const esEstudiante: boolean = true;

console.log(`Nombre: ${nombre}`)
console.log(`Edad: ${edad}`)
```

string · number · boolean · let — reasignable · const — inmutable

Entrada del usuario

```
import * as readline from "readline";

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

rl.question("¿Cómo te llamas? ", (nombre) => {
  rl.question("¿Cuántos años tienes? ", (input) => {
    const edad: number = parseInt(input);
    const nacimiento = 2026 - edad;
    console.log(`Hola ${nombre}! Naciste en ~${nacimiento}.`);
    rl.close();
  });
});
```

Condicionales

```ts
const nota: number = 85;

if (nota >= 90) {
  console.log("Excelente!");
} else if (nota >= 70) {
  console.log("Aprobado.");
} else {
  console.log("Necesitas estudiar más.");
}
```

La estructura `if / else if / else` es idéntica en TypeScript, JavaScript, Java, C++ y muchos más.
Lo que cambia entre lenguajes es la sintaxis, no el concepto.

Bucles

for

```for
for (let i = 1; i <= 5; i++) {
  console.log(`Iteración ${i}`)
}
```

while

```while
let numero: number = 1;

while (numero <= 10) {
  console.log(numero);
  numero++;
}
```

Funciones

```
function sumar(a: number, b: number): number {
  return a + b;
}

function esPar(numero: number): boolean {
  return numero % 2 === 0;
}

console.log(`5 + 3 = ${sumar(5, 3)}`)
if (esPar(4)) console.log("4 es par")
```

Nota importante: No necesitas memorizar la sintaxis. Lo importante es entender los **conceptos**: variables, tipos, condicionales, bucles, funciones. Existen en **todos** los lenguajes.

Resumen de la Clase 2

Concepto	Definición
Lenguaje compilado	Se traduce completamente a código máquina antes de ejecutarse (C++, Rust, Go)
Lenguaje interpretado	Se traduce y ejecuta línea por línea en tiempo real (JavaScript, Python)
Lenguaje transpilado	Se traduce a otro lenguaje del mismo nivel (TypeScript → JavaScript)
Tipado estático	Los tipos se verifican antes de ejecutar (al compilar o al transpilar)
Tipado dinámico	Los tipos se verifican al ejecutar
Tipado fuerte	El lenguaje no convierte tipos automáticamente
Tipado débil	El lenguaje convierte tipos implícitamente

Básico

Ejercicio 2.1: Hola Mundo en tres lenguajes

Compara cómo se escribe y ejecuta el mismo programa en TypeScript, Python y JavaScript. Haz clic en cada lenguaje para ver las diferencias.

TypeScript | **Python** | **JavaScript**

```
1  console.log("Hola, mundo!");
```

```
$  npx ts-node hola.ts
>  Hola, mundo!
```

TypeScript	Python	JavaScript
1	1	1
línea	línea	línea

El mismo resultado en los tres. La diferencia está en cómo se **ejecuta**: TS compila primero, JS y Python se interpretan directamente.

Básico

Ejercicio 2.2: Quiz de tipado

¿Puedes predecir qué devuelve cada expresión? Escribe tu respuesta y revela el resultado.

JavaScript

1 / 8 — Aciertos: 0

```
> "5" + 3
```

¿Qué resultado esperas?

Revelar

Intermedio

Ejercicio 2.3: Calculadora interactiva

Prueba la calculadora aquí y luego recréala en TypeScript por tu cuenta. A la derecha puedes ver cómo se estructura el código.

Así se vería en TypeScript:

```typescript
function calcular(
  a: number,
  b: number,
  op: string
): number {
  switch (op) {
    case "+": return a + b;
    case "-": return a - b;
    case "*": return a * b;
    case "/": return a / b;
  }
}
```

Intermedio

Ejercicio 2.4: Clasificador de lenguajes

Clasifica cada lenguaje usando los selectores. Cuando termines, verifica tus respuestas.

Lenguaje	Compilación	Tipado	Fuerza
Rust	?	?	?
Kotlin	?	?	?
Dart	?	?	?
Lua	?	?	?
Elixir	?	?	?

Verificar respuestas

Proyecto: Juego de Adivinanza

Crea un juego en **TypeScript** donde la computadora elige un número aleatorio y el usuario tiene que adivinarlo.

Requisitos

- Número aleatorio entre 1 y 100
- El usuario ingresa intentos
- Indica "mayor" o "menor"
- Cuenta intentos
- Muestra total al final

Conceptos que usarás

- Variables y tipos
- Entrada del usuario (readline)
- Condicionales (if/else)
- Bucles (while)
- Funciones

Estructura sugerida

```
// 1. Generar número aleatorio
const secreto = Math.floor(Math.random() * 100) + 1;

// 2. Crear interfaz readline
// ... (como en la práctica)

// 3. Función para pedir intento
function pedirIntento(): void {
  // Leer input del usuario
  // Comparar con secreto
  // Mostrar "mayor" o "menor"
  // Si acierta, mostrar intentos
  // Si no, llamar pedirIntento()
}

// 4. Iniciar el juego
pedirIntento();
```

Bonus y criterios

Bonus

⭐ Límite de intentos (10 máximo)

⭐ Elegir dificultad (1-50 / 1-100 / 1-500)

⭐ Manejar entrada inválida sin romperse

Criterios de evaluación

☑️ Compila y ejecuta sin errores

☑️ Genera número aleatorio correctamente

☑️ Da pistas correctas (mayor/menor)

☑️ Cuenta los intentos

☑️ Usa al menos una función propia

☑️ Usa tipos de TypeScript

A practicar

Los conceptos son universales. La sintaxis se aprende con la práctica. **Escribe código todos los días**.

Fragua Tech — Clase 2