

# Simulación computacional con Java

Escuela de Código PILARES

**Dr. Juan Claudio Toledo Roy**

Instituto de Ciencias Nucleares UNAM

[juan.toledo@nucleares.unam.mx](mailto:juan.toledo@nucleares.unam.mx)

# ¿Qué veremos en este curso?

- Java: un lenguaje de programación general multi-plataforma
- Curso relámpago de programación en Java
- Simulación computacional: ¿qué es y para qué sirve?
- Algunas simulaciones:
  - La paradoja del cumpleaños
  - Caída vertical (con resistencia del aire)
  - Simulación de un gas ideal
  - Modelo de percolación (incendios forestales)
  - Modelo de tráfico


# Java: un lenguaje de programación general multi-plataforma

- Creado en 1995 por Sun Microsystems (ahora parte de Oracle)
- Última versión: Java 24, marzo de 2025
- Propiedades:
  - Programación general de alto nivel
  - Orientado a objetos
  - Multi-plataforma (JVM)
  - Sintaxis y arquitectura similar a C/C++, pero “más limpia”
  - Manejo automático de memoria



# Java: un lenguaje de programación general multi-plataforma

- Aún sigue siendo muy popular

Aug 2025	Aug 2024	Change	Programming Language	Ratings	Change
1	1		 Python	26.14%	+8.10%
2	2		 C++	9.18%	-0.86%
3	3		 C	9.03%	-0.15%
4	4		 Java	8.59%	-0.58%
5	5		 C#	5.52%	-0.87%
6	6		 JavaScript	3.15%	-0.76%
7	8	⬆	 Visual Basic	2.33%	+0.15%
8	9	⬆	 Go	2.11%	+0.08%
9	25	⬆	 Perl	2.08%	+1.17%
10	12	⬆	 Delphi/Object Pascal	1.82%	+0.19%

# Java: un lenguaje de programación general multi-plataforma

- Aplicaciones principales:
  - Android (aunque ahora se usa Kotlin)
  - Aplicaciones "enterprise"
  - "Big Data" (p. ej. Apache Spark)
  - Back-end y servicios en la nube



# Java: un lenguaje de programación general multi-plataforma

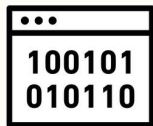
- ¿Es un lenguaje compilado o interpretado?

## Compilado



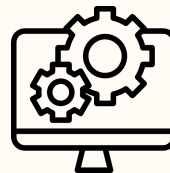
Código fuente

Compilación



Código de máquina (binario)

Ejecución



Ventajas:

- Código *muy* rápido
- Código empaquetado (fácil distribución)

Desventajas:

- Compilado para un sistema específico

## Interpretado



Código fuente

Interpretación  
(línea por línea)



Intérprete  
(programa separado)



Ventajas:

- No hace falta compilar
- No específico a un sistema operativo

Desventajas:

- Más lento (debe checar código cuando se ejecuta)
- Hay que distribuir código fuente

# Java: un lenguaje de programación general multi-plataforma

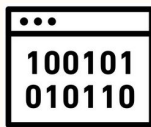
- ¿Es un lenguaje compilado o interpretado?

## Java: Bytecode



Código fuente

Compilación a  
bytecode

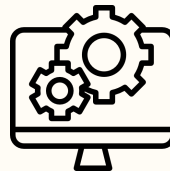


Bytecode  
(código para la JVM)

Interpretación  
por JVM



Java Virtual  
Machine



## Ventajas:

- Programa pre-compilado pero multi-plataforma
- Bastante más rápido que interpretado
- Optimizaciones en tiempo real
- Código (semi) empaquetado

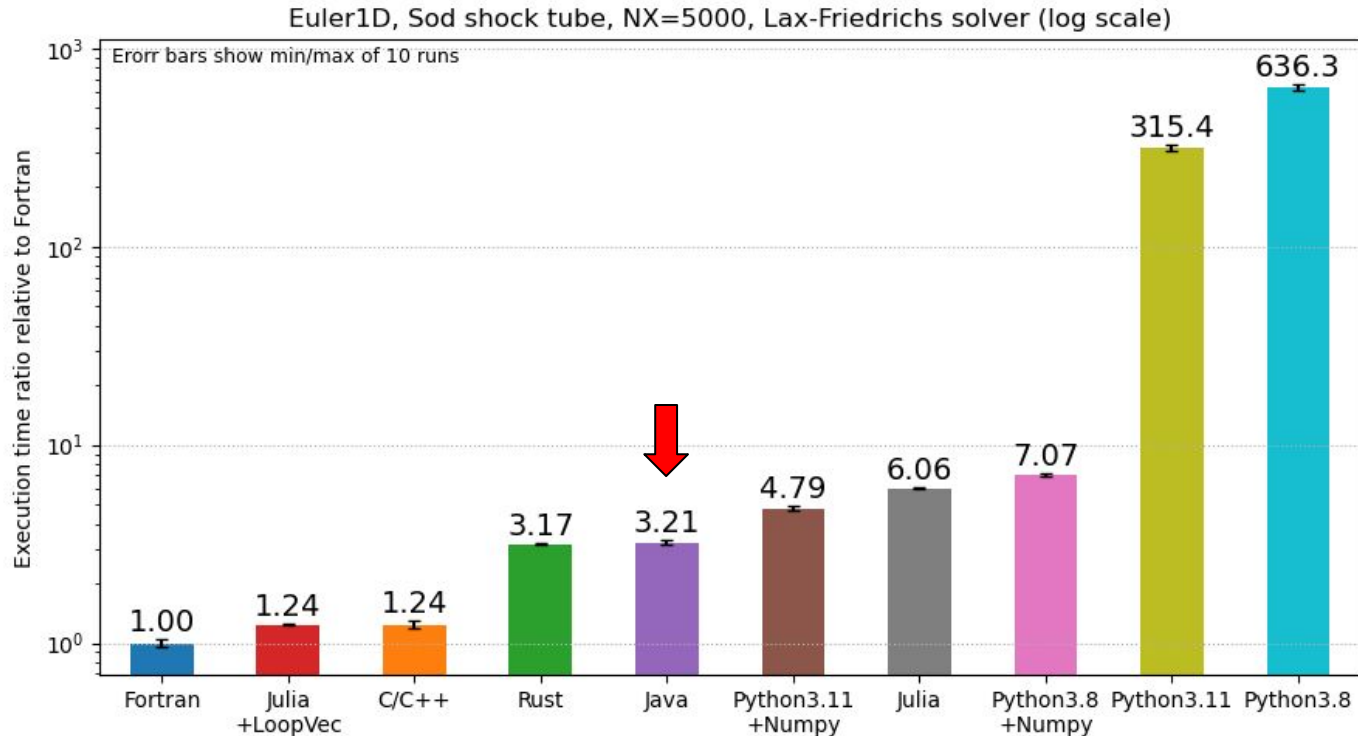
## Desventajas:

- Necesidad de una JVM
  - Debe estar disponible e instalada
  - Requiere un poco de recursos (p.ej. memoria RAM)

# Java: un lenguaje de programación general multi-plataforma

- Una comparación de velocidad (código hidrodinámico)

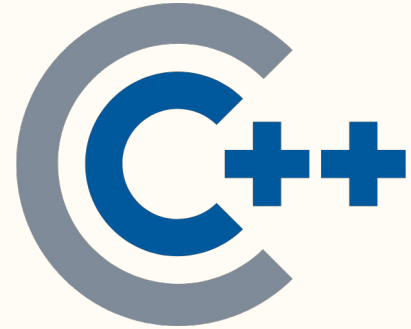
[https://github.com/meithan/Euler1D\\_Benchmark](https://github.com/meithan/Euler1D_Benchmark)





# Java: un lenguaje de programación general multi-plataforma

- “Como C/C++ pero más limpio”
  - Lenguaje de más alto nivel, portable
  - No hay apuntadores
  - No hay reserva/liberación manual de memoria RAM (*garbage collector*)
  - Puramente orientado a objetos
  - Librería estándar muy amplia



# Java: un lenguaje de programación general multi-plataforma

- No tiene relación directa con JavaScript
- Sintaxis parecida a Java, pero lenguaje bien diferente
- JavaScript muy usado en web (corre en el browser)

**JavaScript**



# Cómo instalar Java

- Java tiene dos componentes:
  - Java Runtime Environment (JRE): *ejecutar* programas (JVM)
  - Java Development Kit (JDK): *compilar* programas
- Compatible con Windows, MacOS y Linux
- JDK: <https://www.oracle.com/java/technologies/downloads/> (incluye JRE)
- Solamente JRE: <https://www.java.com/en/download/>

# Cómo compilar y ejecutar Java

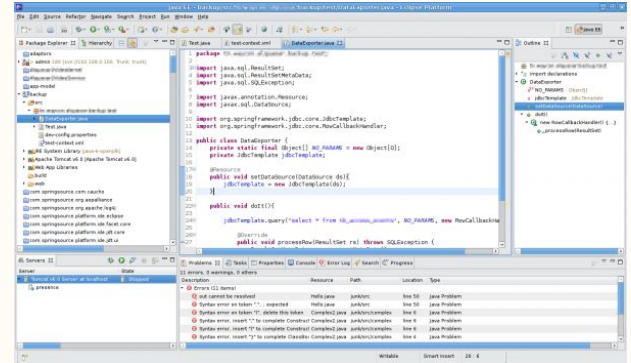
- **Compilar** código fuente desde terminal (command prompt):
  - `javac MiPrograma.java`
  - Se generará `MiPrograma.class`
- También se pueden usar IDEs (Integrated Development Environment)



IntelliJ IDEA



NetBeans



# Cómo compilar y ejecutar Java

- Para **ejecutar** código (en la terminal):

- `java MiPrograma`

↑  
java, sin la c

← Sin .class o .java

(es realmente el nombre de la clase  
cuyo `main()` se ejecuta como  
punto de entrada del programa)

- Hola mundo:

```
public class HolaMundo {  
    public static void main(String[] args) {  
        System.out.println("¡Hola, Mundo!");  
    }  
}
```

# Curso relámpago de Java: tipos de datos primitivos

<i>type</i>	<i>set of values</i>	<i>common operators</i>	<i>sample literal values</i>
int	integers	+ - * / %	99 12 2147483647
double	floating-point numbers	+ - * /	3.14 2.5 6.022e23
boolean	boolean values	&&    !	true false
char	characters		'A' '1' '%' '\n'
String	sequences of characters	+	"AB" "Hello" "2.5"

Recordatorio: Java es *estáticamente tipado*, así que los tipos de datos de todas las variables deben *declararse* antes de que se puedan usar.

# Curso relámpago de Java: operaciones

## Enteros

<i>expression</i>	<i>value</i>	<i>comment</i>
99	99	<i>integer literal</i>
+99	99	<i>positive sign</i>
-99	-99	<i>negative sign</i>
5 + 3	8	<i>addition</i>
5 - 3	2	<i>subtraction</i>
5 * 3	15	<i>multiplication</i>
5 / 3	1	<i>no fractional part</i>
5 % 3	2	<i>remainder</i>
1 / 0		<i>run-time error</i>
3 * 5 - 2	13	<i>* has precedence</i>
3 + 5 / 2	5	<i>/ has precedence</i>
3 - 5 - 2	-4	<i>left associative</i>
( 3 - 5 ) - 2	-4	<i>better style</i>
3 - ( 5 - 2 )	0	<i>unambiguous</i>

## Floats/doubles

<i>expression</i>	<i>value</i>
3.141 + 2.0	5.141
3.141 - 2.0	1.141
3.141 / 2.0	1.5705
5.0 / 3.0	1.6666666666666667
10.0 % 3.141	0.577
1.0 / 0.0	Infinity
Math.sqrt(2.0)	1.4142135623730951
Math.sqrt(-1.0)	NaN

# Curso relámpago de Java: operaciones

## Booleanos

### NOT

<i>a</i>	<i>!a</i>
true	false
false	true

### AND

### OR

<i>a</i>	<i>b</i>	<i>a &amp;&amp; b</i>	<i>a    b</i>
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true



# Curso relámpago de Java: operadores de comparación

<i>op</i>	<i>meaning</i>	<i>true</i>	<i>false</i>
<code>==</code>	<i>equal</i>	<code>2 == 2</code>	<code>2 == 3</code>
<code>!=</code>	<i>not equal</i>	<code>3 != 2</code>	<code>2 != 2</code>
<code>&lt;</code>	<i>less than</i>	<code>2 &lt; 13</code>	<code>2 &lt; 2</code>
<code>&lt;=</code>	<i>less than or equal</i>	<code>2 &lt;= 2</code>	<code>3 &lt;= 2</code>
<code>&gt;</code>	<i>greater than</i>	<code>13 &gt; 2</code>	<code>2 &gt; 13</code>
<code>&gt;=</code>	<i>greater than or equal</i>	<code>3 &gt;= 2</code>	<code>2 &gt;= 3</code>

# Curso relámpago de Java: métodos (funciones) básicos

<code>void System.out.print(String s)</code>	<i>print s</i>
<code>void System.out.println(String s)</code>	<i>print s, followed by a newline</i>
<code>void System.out.println()</code>	<i>print a newline</i>

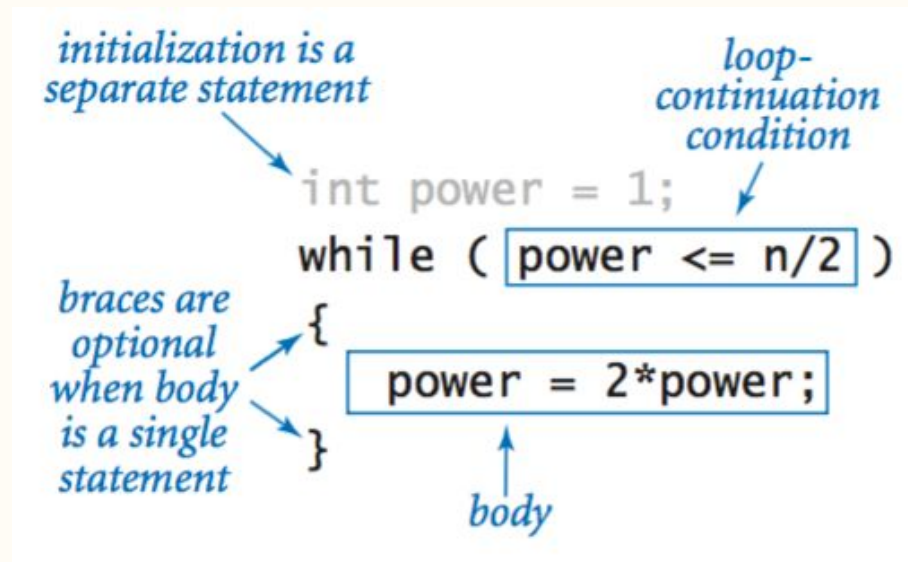
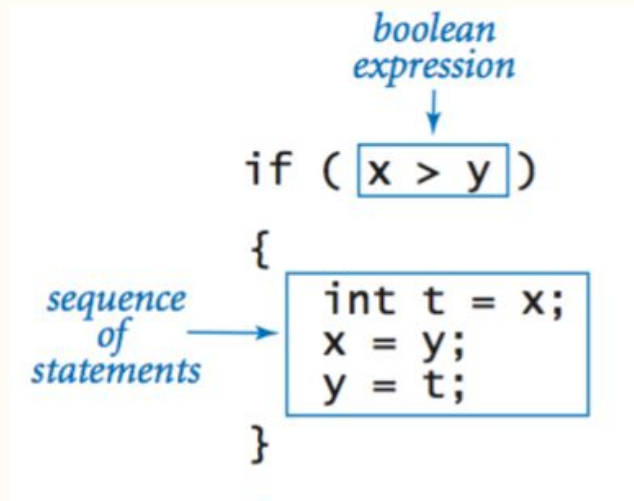
<code>int Integer.parseInt(String s)</code>	<i>convert s to an int value</i>
<code>double Double.parseDouble(String s)</code>	<i>convert s to a double value</i>
<code>long Long.parseLong(String s)</code>	<i>convert s to a long value</i>

# Curso relámpago de Java: métodos (funciones) básicos

```
public class Math
```

<code>double abs(double a)</code>	<i>absolute value of a</i>
<code>double max(double a, double b)</code>	<i>maximum of a and b</i>
<code>double min(double a, double b)</code>	<i>minimum of a and b</i>
<code>double sin(double theta)</code>	<i>sine of theta</i>
<code>double cos(double theta)</code>	<i>cosine of theta</i>
<code>double tan(double theta)</code>	<i>tangent of theta</i>
<code>double toRadians(double degrees)</code>	<i>convert angle from degrees to radians</i>
<code>double toDegrees(double radians)</code>	<i>convert angle from radians to degrees</i>
<code>double exp(double a)</code>	<i>exponential (<math>e^a</math>)</i>
<code>double log(double a)</code>	<i>natural log (<math>\log_e a</math>, or <math>\ln a</math>)</i>
<code>double pow(double a, double b)</code>	<i>raise a to the bth power (<math>a^b</math>)</i>
<code>long round(double a)</code>	<i>round a to the nearest integer</i>
<code>double random()</code>	<i>random number in <math>[0, 1)</math></i>
<code>double sqrt(double a)</code>	<i>square root of a</i>
<code>double E</code>	<i>value of e (constant)</i>
<code>double PI</code>	<i>value of <math>\pi</math> (constant)</i>

# Curso relámpago de Java: control de flujo



# Curso relámpago de Java: control de flujo

The diagram illustrates the components of a Java `for` loop. It shows the following code snippet with annotations:

```
int power = 1;
for (int i = 0; i <= n; i++) {
    System.out.println(i + " " + power);
    power = 2*power;
}
```

Annotations and their corresponding code parts:

- initialize another variable in a separate statement* points to `int power = 1;`
- declare and initialize a loop control variable* points to `int i = 0` in the `for` header.
- loop-continuation condition* points to `i <= n` in the `for` header.
- increment* points to `i++` in the `for` header.
- body* points to the block of code inside the loop: `System.out.println(i + " " + power);` and `power = 2*power;`.

# Curso relámpago de Java: arreglos

```
double[] a;           // declare the array
a = new double[n];    // create the array
for (int i = 0; i < n; i++) // elements are indexed from 0 to n-1
    a[i] = 0.0;       // initialize all elements to 0.0
```

```
String[] SUITS = {
    "Clubs", "Diamonds", "Hearts", "Spades"
};

String[] RANKS = {
    "2", "3", "4", "5", "6", "7", "8", "9", "10",
    "Jack", "Queen", "King", "Ace"
};
```

# Curso relámpago de Java: arreglos 2D

```
double[][] a = new double[m][n];
```

```
double[][] a;  
a = new double[m][n];  
for (int i = 0; i < m; i++)  
    for (int j = 0; j < n; j++)  
        a[i][j] = 0;
```

# Curso relámpago de Java: entrada/salida (terminal)

```
void System.out.print(String s)    print s  
void System.out.println(String s) print s, followed by a newline  
void System.out.println()         print a newline
```

```
import java.util.Scanner;  
  
public class ReadInputScanner {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in); // Create a Scanner object linked to  
        standard input  
  
        System.out.print("Enter your name: ");  
        String name = scanner.nextLine(); // Read a full line of text  
  
        System.out.print("Enter your age: ");  
        int age = scanner.nextInt(); // Read an integer  
  
        System.out.println("Hello, " + name + "! You are " + age + " years old.");  
  
        scanner.close(); // Close the scanner to release resources  
    }  
}
```



# Curso relámpago de Java: entrada/salida (archivo)

## Leer de archivo con `java.io.File` y `java.util.Scanner`

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class ReadExample {
    public static void main(String[] args) {
        try {
            File file = new File("output.txt");
            Scanner scanner = new Scanner(file);

            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                System.out.println(line);
            }

            scanner.close();
        } catch (FileNotFoundException e) {
            System.out.println("File not found.");
            e.printStackTrace();
        }
    }
}
```

# Curso relámpago de Java: entrada/salida (archivo)

## Escribir a archivo con `java.io.FileWriter`

```
import java.io.FileWriter;
import java.io.IOException;

public class WriteExample {
    public static void main(String[] args) {
        try {
            FileWriter writer = new FileWriter("output.txt");
            writer.write("Hello, world!\n");
            writer.write("This is a second line.\n");
            writer.close();
            System.out.println("Successfully wrote to the file.");
        } catch (IOException e) {
            System.out.println("An error occurred.");
            e.printStackTrace();
        }
    }
}
```

# Curso relámpago de Java: librería estándar

<https://docs.oracle.com/javase/8/docs/api/>

Java™ Platform  
Standard Ed. 8

All Classes All Profiles

Packages

java.applet  
java.awt  
java.awt.color  
java.awt.datatransfer  
java.awt.dnd  
java.awt.event  
java.awt.font  
java.awt.geom  
java.awt.im  
java.awt.im.spi  
java.awt.image  
java.awt.image.renderable  
java.awt.print  
java.beans  
java.beans.beancontext  
java.io  
java.lang  
java.lang.annotation  
java.lang.instrument  
java.lang.invoke  
java.lang.management  
java.lang.ref  
java.lang.reflect  
java.math  
java.net  
java.nio  
java.nio.channels  
java.nio.channels.spi  
java.nio.charset  
java.nio.charset.spi

OVERVIEW PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV NEXT FRAMES NO FRAMES

## Java™ Platform, Standard Edition 8 API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

### Profiles

- compact1
- compact2
- compact3

Packages

Package	Description
java.applet	Provides the classes necessary to c
java.awt	Contains all of the classes for crea
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for
java.awt.dnd	Drag and Drop is a direct manipul

# Curso relámpago de Java: todo es un objeto

Java es orientado a objetos desde el núcleo: "todo es un objeto"

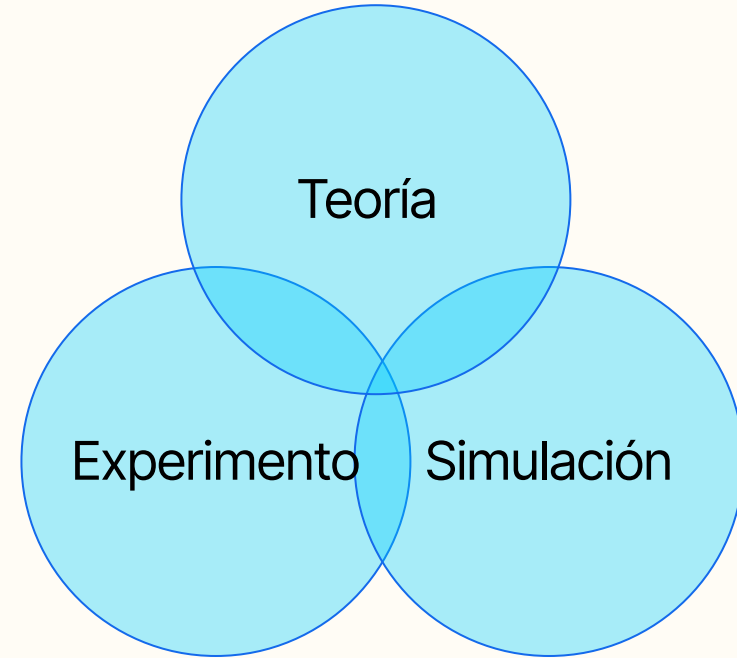
```
public class HolaMundo {  
    public static void main(String[] args) {  
        System.out.println("¡Hola, Mundo!");  
    }  
}
```

Primitive type	Wrapper class	Constructor arguments
byte	<a href="#">Byte</a>	byte or String
short	<a href="#">Short</a>	short or String
int	<a href="#">Integer</a>	int or String
long	<a href="#">Long</a>	long or String
float	<a href="#">Float</a>	float, double or String
double	<a href="#">Double</a>	double or String
char	<a href="#">Character</a>	char
boolean	<a href="#">Boolean</a>	boolean or String

# Simulación computacional

¿Qué es la simulación computacional?

- **Modelos matemáticos y algorítmicos** que representan un sistema (físico, biológico, económico, etc.) y que se llevan a cabo mediante la ejecución de **programas de computadora**.
- **Entender y predecir** las **propiedades** y **comportamiento** del sistema bajo distintas condiciones.



# Simulación computacional

¿Por qué hacer simulación computacional?

- Hacer experimentos es muy **difícil, caro o imposible**
  - p. ej. simular reacciones nucleares o la formación de galaxias
  - Mejor simular el avión antes de construirlo
- Muchos sistemas no se pueden reducir bien a un **modelo teórico** matemático que se pueda resolver

# Simulación computacional

## Componentes de una simulación computacional

- **Modelo** del sistema
  - Una representación matemática/concreta del sistema usando ecuaciones, reglas, algoritmos, etc.
- **Parámetros y condiciones iniciales**
  - Valores que definen el problema y su estado inicial
- **Métodos numéricos**
  - Cómo se lleva a cabo computacionalmente

# Simulación computacional

## Tipos de simulaciones computacionales

- **Determinista vs estocástica**
  - Si incluimos o no el efecto del “azar”, a veces simplemente lo que desconocemos
- **Discreta vs continua**
  - Simulamos componentes discretos (e.g. agentes) o resolvemos ecuaciones diferenciales continuas



# Simulación computacional

Muchísimos ejemplos.

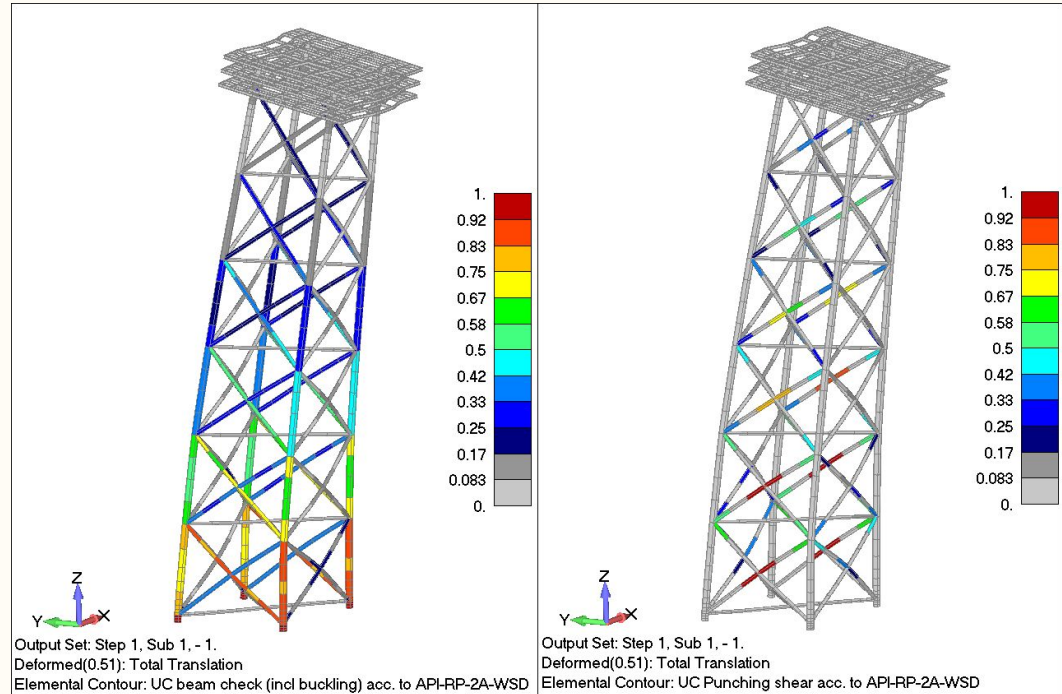
- Modelación de clima



# Simulación computacional

Muchísimos ejemplos.

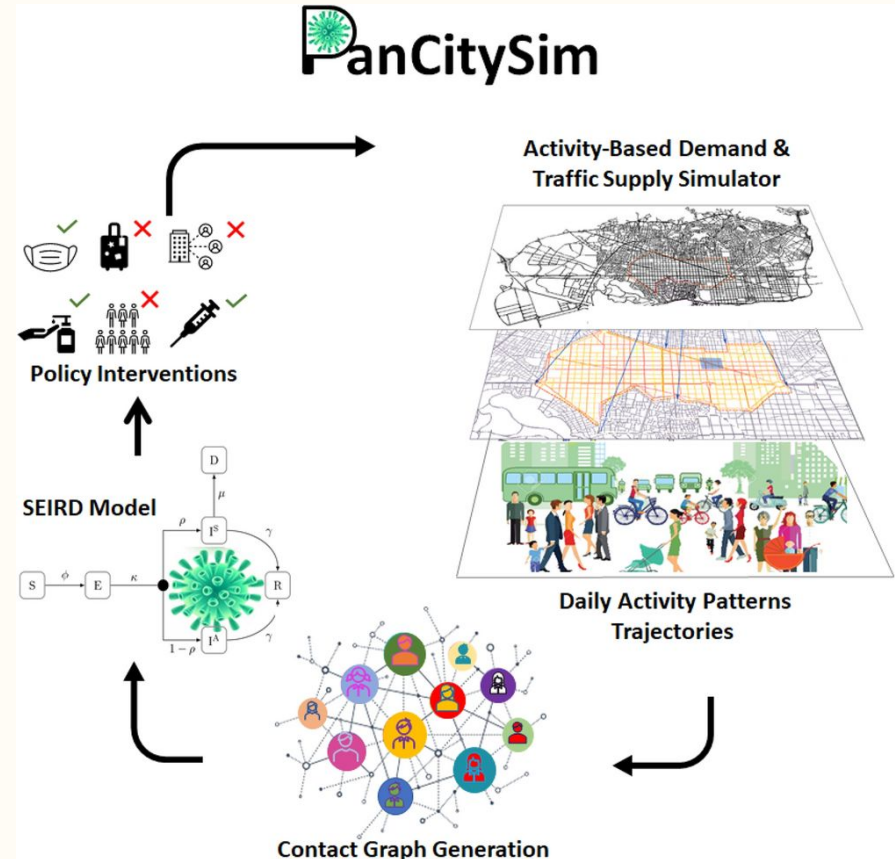
- Modelado estructural para ingeniería



# Simulación computacional

Muchísimos ejemplos.

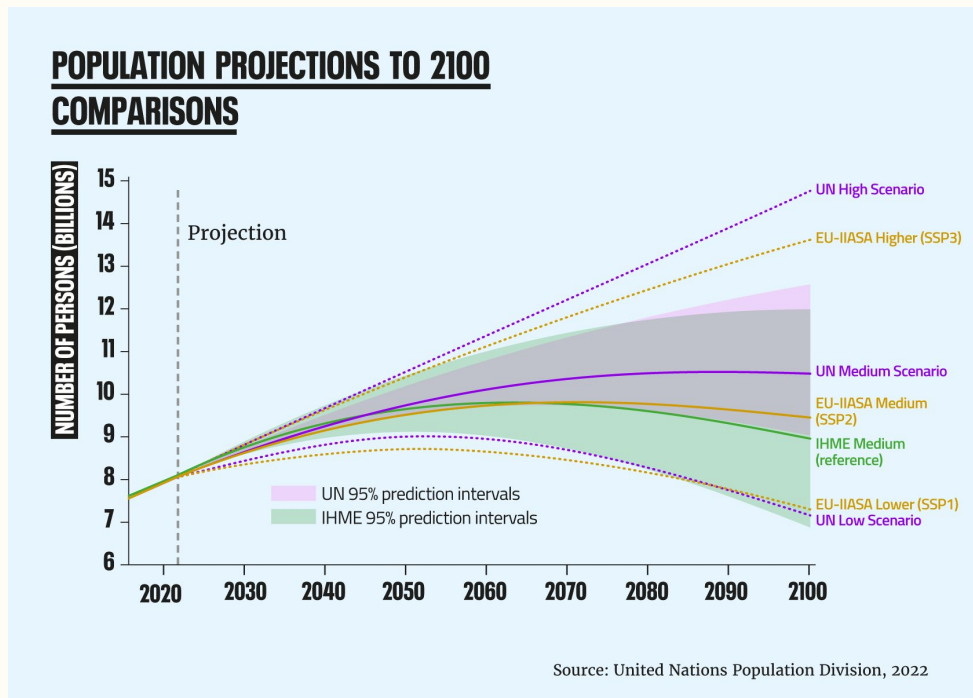
- Propagación de epidemias



# Simulación computacional

Muchísimos ejemplos.

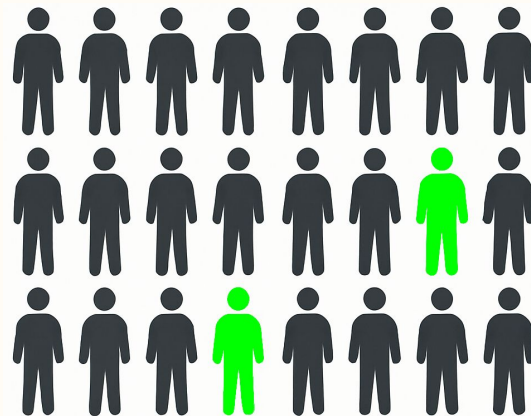
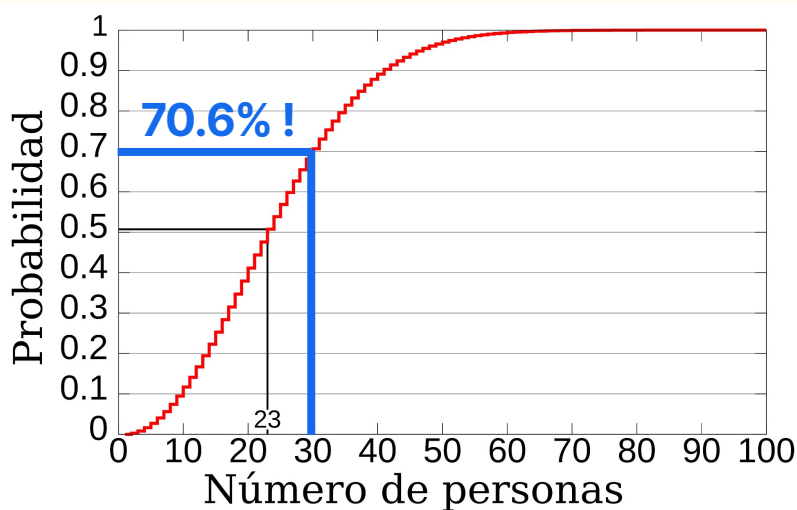
- Crecimiento demográfico y económico



# La Paradoja del Cumpleaños

Imaginemos un grupo de 30 personas.

¿Cuál es la probabilidad de que dos de ellas tengan **el mismo cumpleaños**?

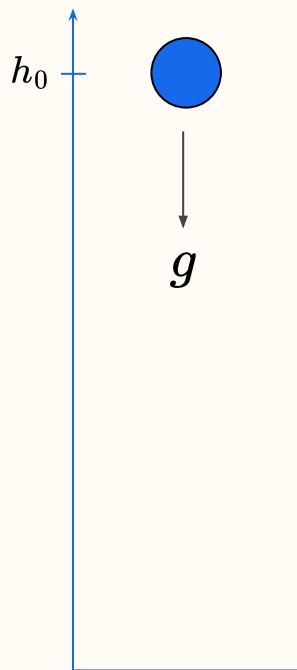


¿Será??

→ Vamos a simularlo con Java

# Caída libre

Simulemos el problema de la caída vertical de un objeto.



Debemos ahora hacer una **simulación temporal**.

## Parámetros

- Altura inicial
- Gravedad
- Paso de tiempo

## Estado de la simulación

- Tiempo actual
- Altura del objeto
- Velocidad del objeto

## Avance de la simulación

En cada paso de tiempo, actualizamos la velocidad, altura y el tiempo:

$$v \leftarrow v + g \Delta t$$

$$h \leftarrow h - v \Delta t$$

$$t \leftarrow t + \Delta t$$

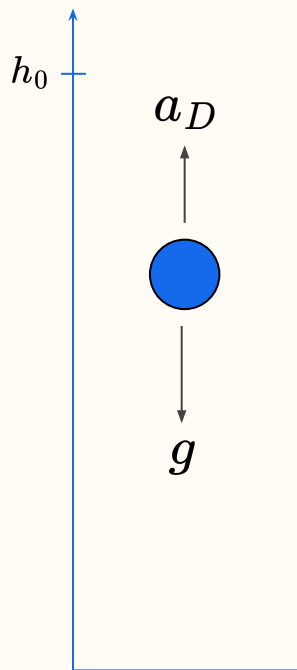
Continuamos hasta que el objeto llega al suelo,  $h = 0$

¿Cuánto tiempo tarda en llegar al suelo?

¿Cuál es su velocidad?

# Caída libre con resistencia del aire

Incluycamos ahora la **resistencia del aire**.



La resistencia del aire produce una **fuerza adicional** que **frena** la caída del objeto:

$$a_D = \frac{k}{m} v^2$$

A mayor velocidad, más resistencia del aire

$k$  coeficiente de arrastre (forma, tamaño)

$m$  masa del objeto

Nueva regla de actualización de la velocidad:

$$v \leftarrow v + g\Delta t - a_D\Delta t$$

$$k = 0.5, m = 10$$

¿Cuánto tiempo tarda ahora en llegar al suelo?

¿Cuál es su velocidad final?

¿Hay una velocidad máxima?

De hecho, ¡ya no hay una fórmula exacta para el tiempo de caída!

# Caída libre con resistencia del aire

Escribamos los resultados en un archivo de texto que podemos graficar usando otro programa.

```
import java.io.FileWriter;  
import java.io.IOException;  
import java.io.PrintWriter;
```

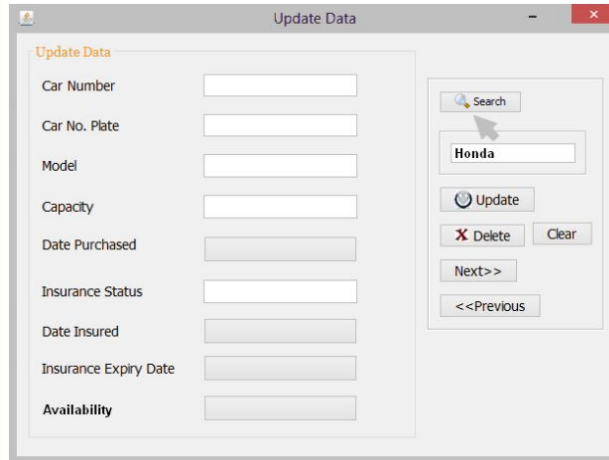
Para escribir con formato como `printf`:

```
try (PrintWriter writer = new PrintWriter(new FileWriter(archivo))) {  
    ...  
    writer.printf("  %-12.2f %-18.3f %-18.3f\n", tiempo, altura, velocidad);  
    ...  
} catch (IOException e) {  
    System.err.println("Error: " + e.getMessage());  
}
```



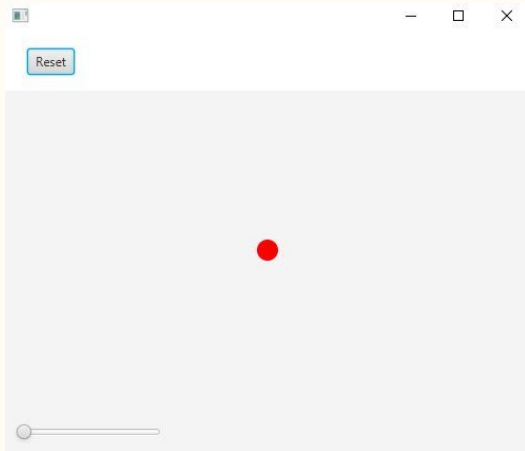
# Visualización en tiempo real en Java

- Java tiene varias librerías para aprovechar el entorno gráfico directamente desde el código:
  - **Swing:** diseñado para crear aplicaciones de escritorio (ventanas, cuadros de texto, botones, tablas, etc.)



# Visualización en tiempo real en Java

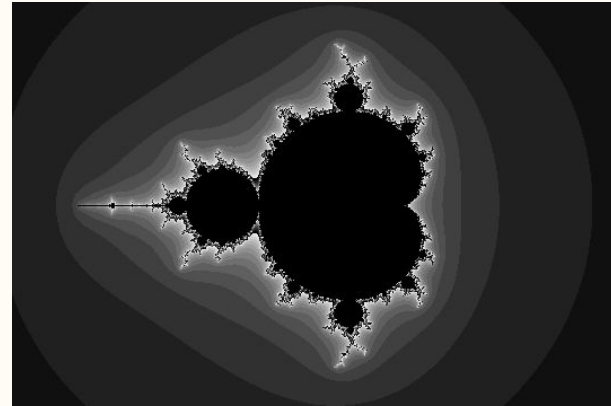
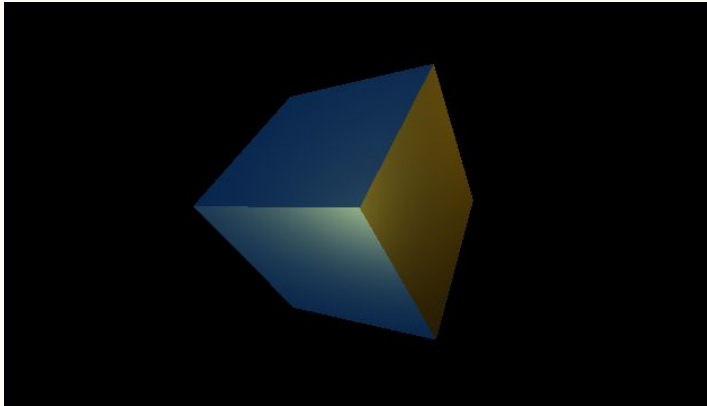
- Java tiene varias librerías para aprovechar el entorno gráfico directamente desde el código:
  - **JavaFX:** versión más moderna, más poderosa



Sin embargo, en 2018 Oracle dejó de empaquetar JavaFX con el SDK estándar; ahora debe instalarse separadamente.

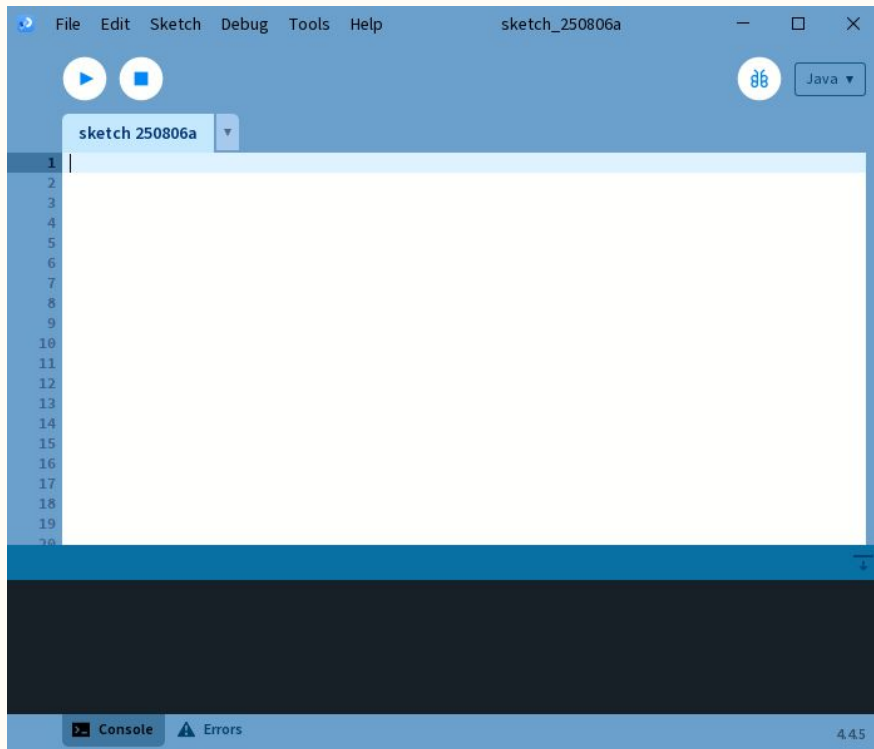
# Visualización en tiempo real en Java

- Hay también entornos gráficos externos para Java:
  - **Processing**: de alto nivel, orientada a simulación y visualización con poco código → **Usaremos Processing**

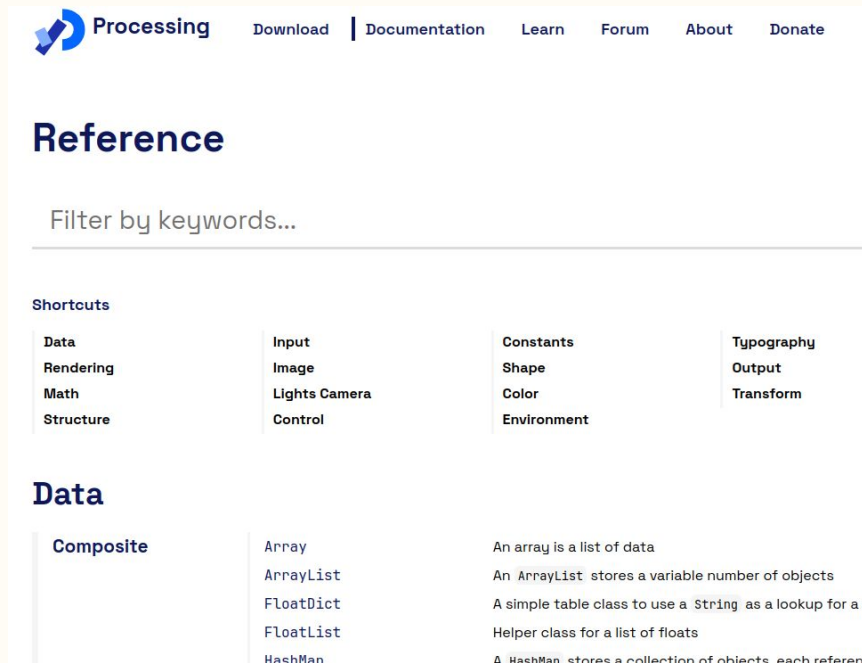


# Instalación de Processing

<https://processing.org/download>



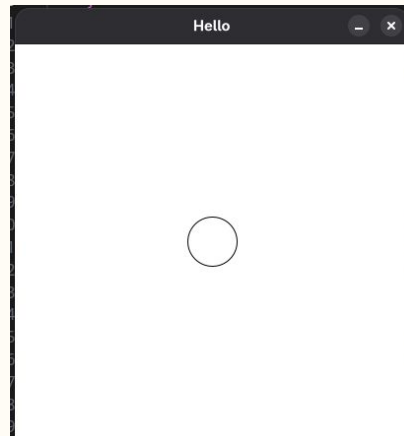
<https://processing.org/reference>



# Primeros pasos en Processing

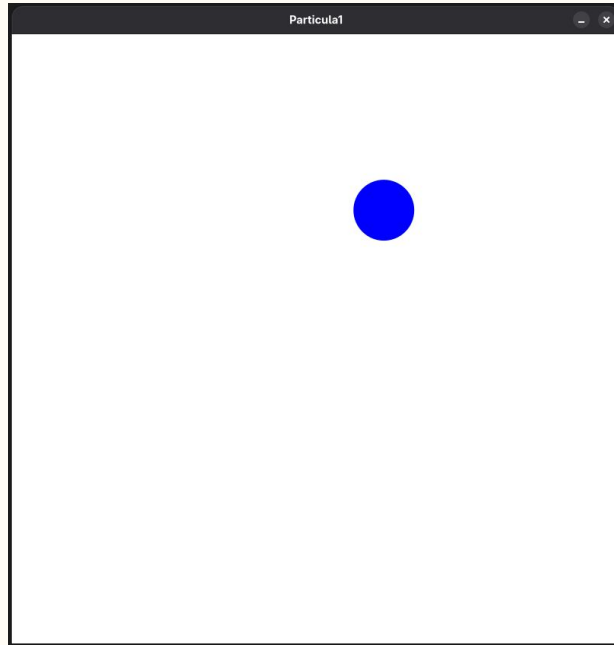
- Processing se encarga de la mayor parte del código de Java por nosotros
- Un sketch de Processing tiene al menos dos partes básicas:
  - `setup()`: se ejecuta una sola vez al inicio
  - `draw()`: se ejecuta continuamente (es como nuestro `while`)
- Un "hola mundo" en Processing:

```
void setup() {  
  size(800, 800);  
}  
  
void draw() {  
  background(255);  
  circle(width/2, height/2, 50);  
}
```



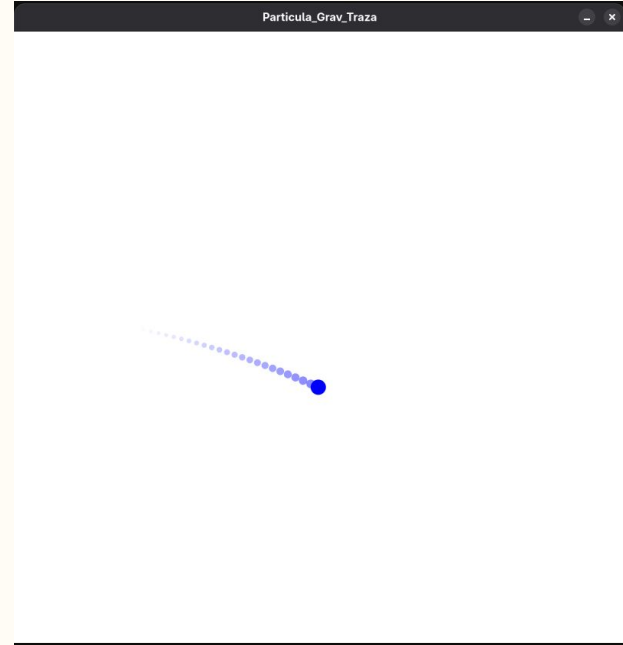
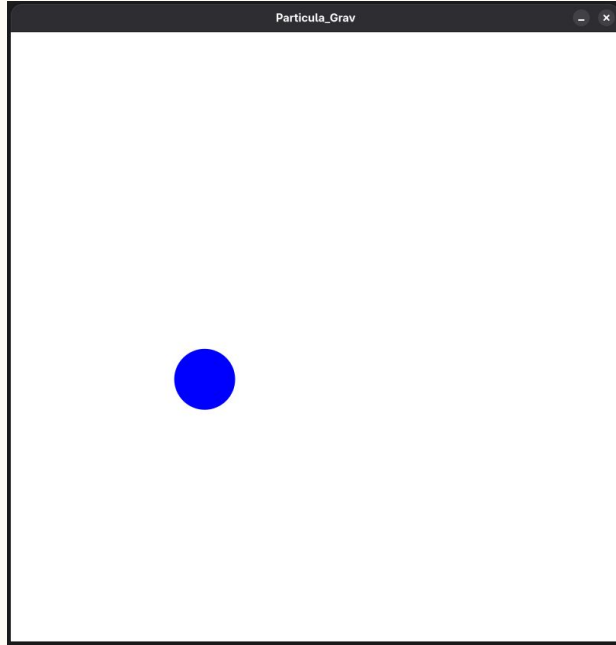
# Partículas en una caja

**Simulemos** y **visualicemos** partículas rebotando en una caja, usando Processing.



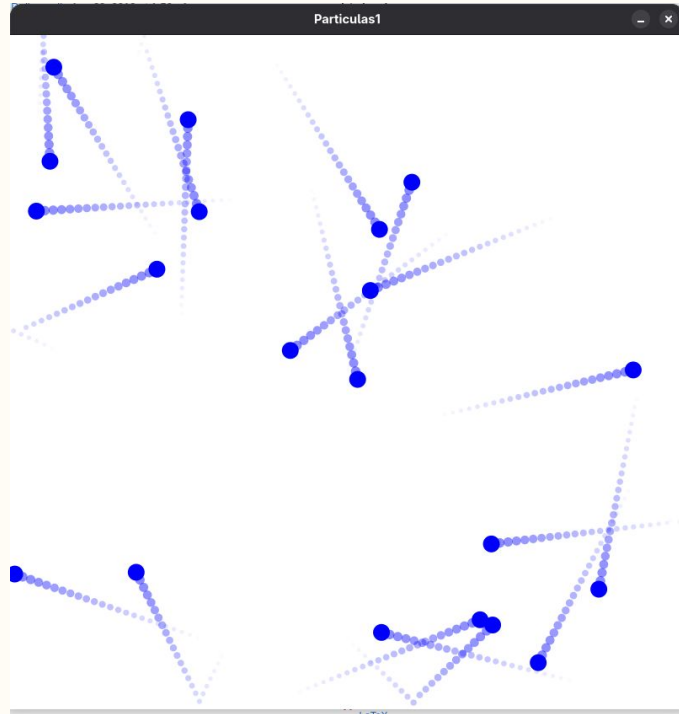
# Partículas en una caja

Agregar **gravedad** y **traza**.



# Partículas en una caja

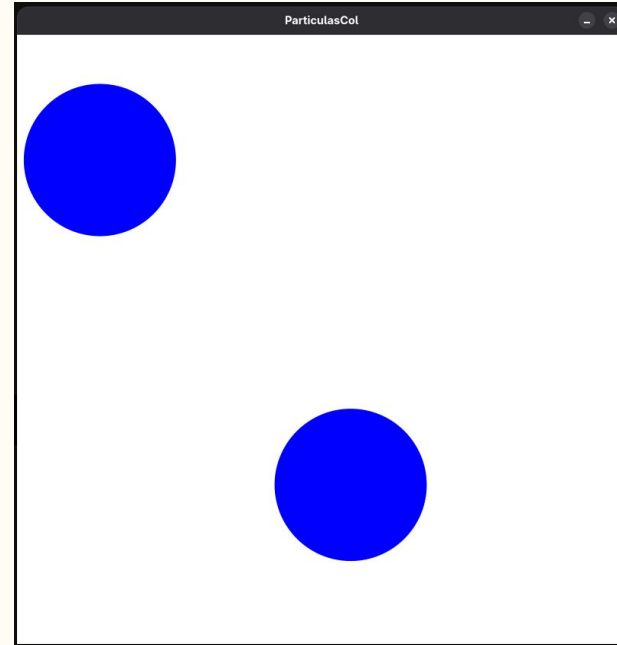
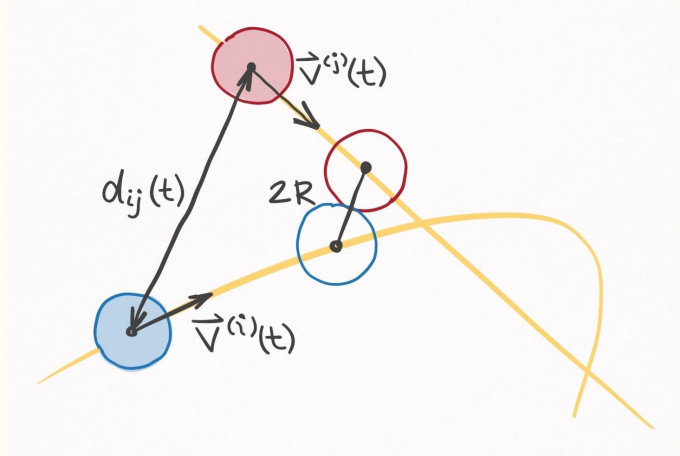
Muchas partículas (ya sin gravedad).





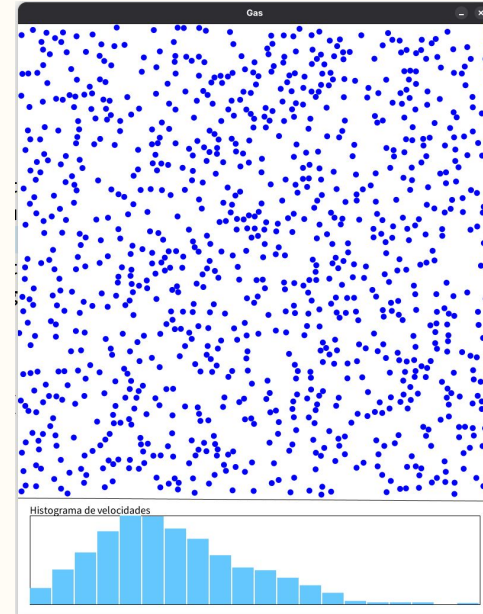
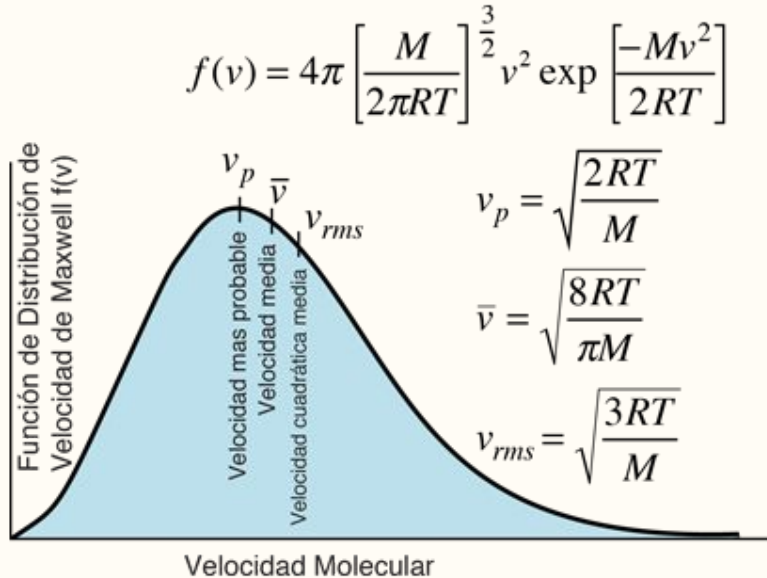
# Partículas en una caja

Implementamos colisiones entre partículas.



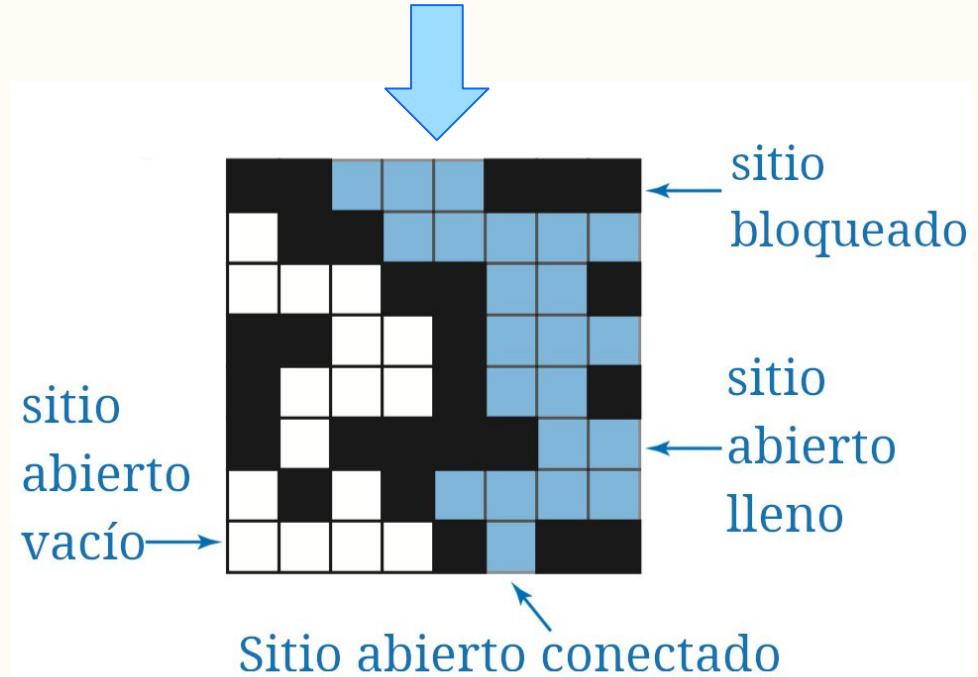
# Simulación de un gas

- Finalmente, tenemos la simulación de un gas ideal, y calculamos el **histograma de velocidades**.
- La simulación reproduce un resultado clásico de la **teoría cinética de gases**.



# Modelo de percolación

- ¿Puede un fluido atravesar una matriz porosa?



# Modelo de percolación

## Algoritmo

- Inicializamos los sitios al azar, con una **probabilidad** de que un sitio esté **abierto**.
- Al principio, llenamos los sitios de la fila de arriba.
- La simulación realiza un *flood fill*:
  - Mantenemos una lista de sitios llenos a checar.
  - Removemos un sitio de la lista, y checamos si tiene un sitio *abierto vacío* al lado; si sí, agregamos ese sitio vecino a la lista a checar.
  - Paramos cuando la lista está vacía
- Nos fijamos si alguno de los sitios de la fila inferior fue llenado.

# Modelo de percolación

Probabilidad sitio abierto: **50%**



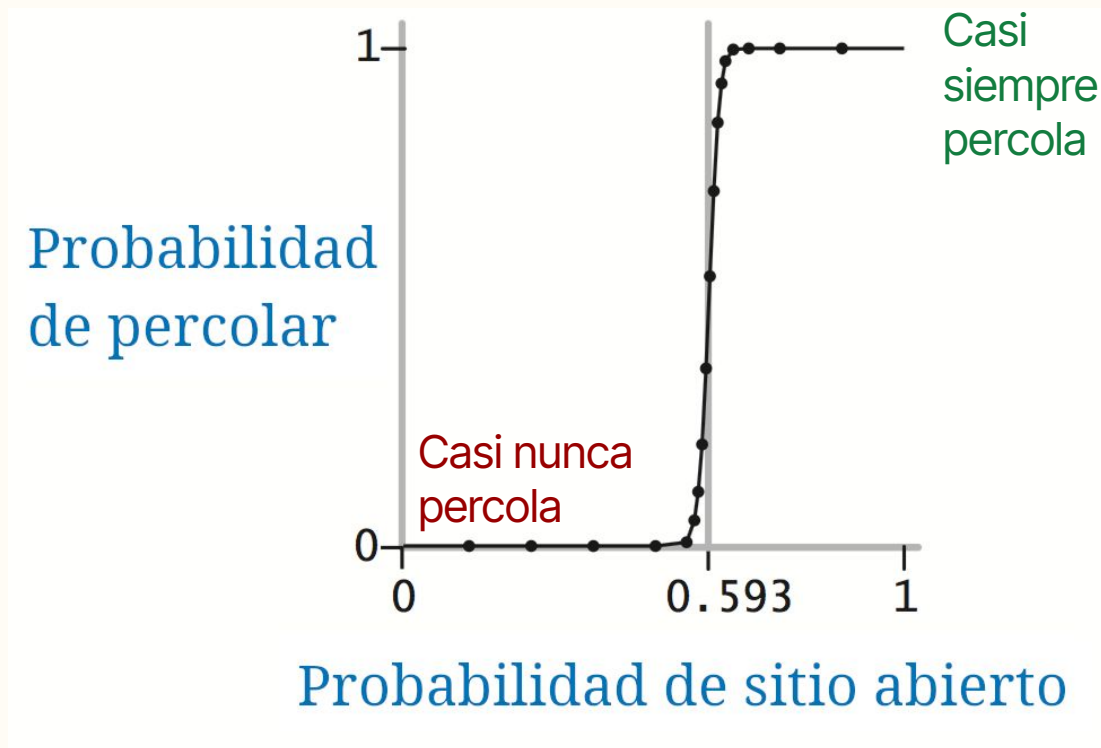
# Modelo de percolación

Probabilidad sitio abierto: **65%**



# Modelo de percolación

Hay una **transición de fase** cerca de 59.3%.



# Modelo de percolación: incendios forestales

- Ligera modificación del modelo de percolación
- Consideramos 4 estados para cada sitio:



Tierra (suelo baldío)



Incendio



Bosque



Bosque quemado

- En cada iteración, checamos aquellos sitios que están quemándose:
  - Cualquier vecino que es bosque no-quemado empieza a quemarse
  - El sitio se convierte en bosque quemado
- Iteramos hasta que no queda ningún sitio incendiándose.
- ¿Cómo depende el área quemada de la densidad de bosque?



# Modelo de percolación: incendios forestales

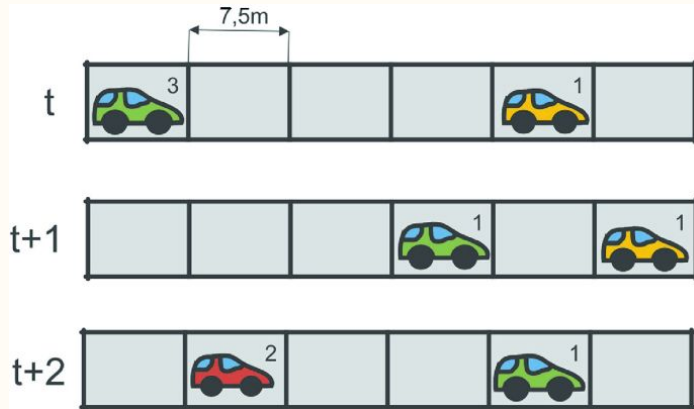


# Modelo de tráfico vehicular

- ¿Por qué en hora pico de pronto hay embotellamientos, aun cuando no hay nada "fuera de lo común" (accidente, carril cerrado, etc)?
- Modelo de Nagel & Schreckenberg (1992)

## Autómata celular

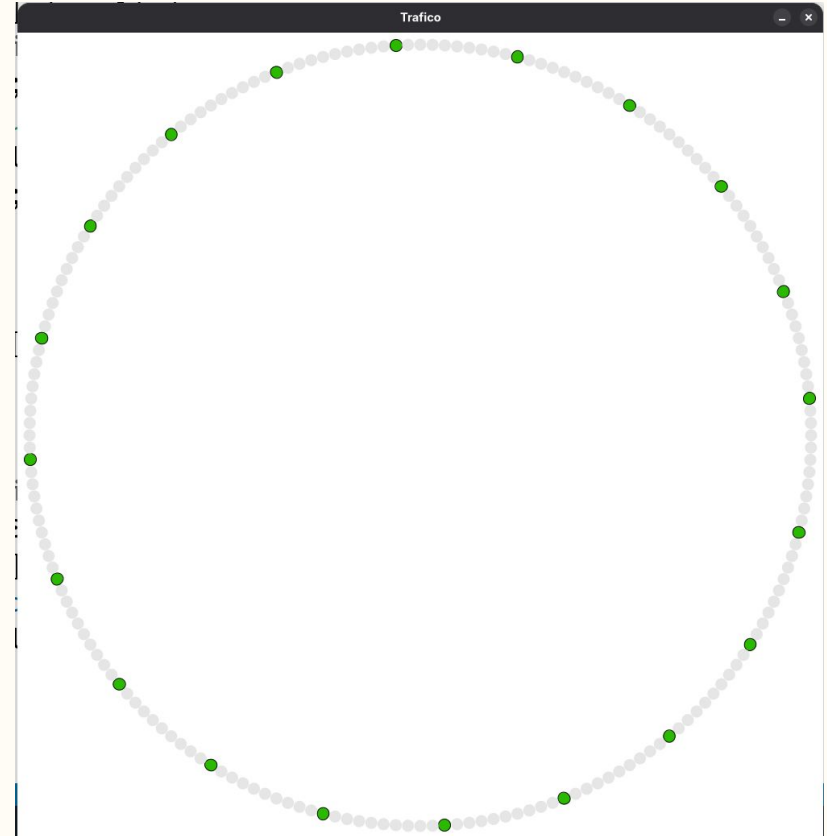
Cada coche ocupa una casilla, tiene una velocidad entre 0 y 3



- En cada iteración, cada coche realice 4 acciones:
  - **Intenta acelerar:** si su  $v < \text{max\_speed}$ ,  $v += 1$
  - **Evita colisión:** si su nueva  $v$  lo haría chocar con el coche adelante, la ajusta
  - **Azar:** con cierta (baja) prob  $v -= 1$
  - **Movimiento:** el coche avanza  $v$  posiciones

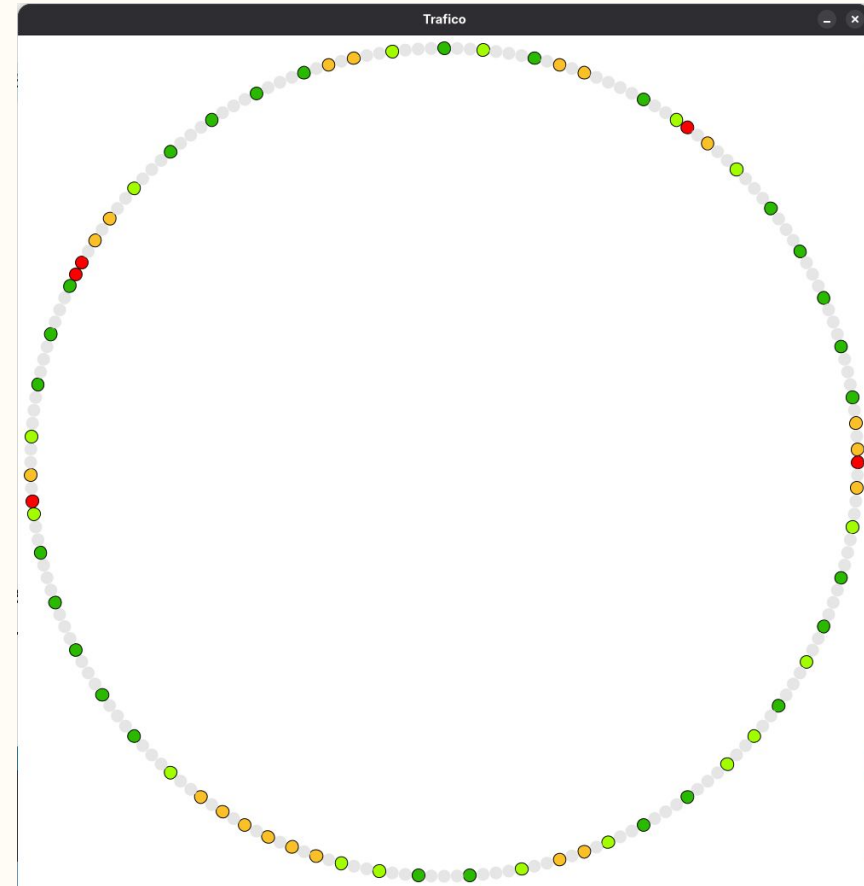
# Modelo de tráfico vehicular

- Si la densidad de coches es baja (para la carretera), todos los coches aceleran a su velocidad máxima y se mantienen espaciados
- No hay embotellamientos, incluso con frenados ocasionales



# Modelo de tráfico vehicular

- Pero si la densidad rebasa un cierto umbral, cualquier frenado leve ocasiona una reacción en cadena que hace que los coches se detengan momentáneamente.
- Es decir, un embotellamiento. Y es espontáneo y duradero.



# Modelo de tráfico vehicular

