

The background is white and decorated with various tech-related icons in shades of purple, blue, and pink. These include a code editor window with a green arrow icon, a cloud, a USB drive, a Wi-Fi symbol, a smartphone, a laptop, a mouse with a green cord, a monitor, a camera, a folder, a keyboard, and a mouse with a green cord.

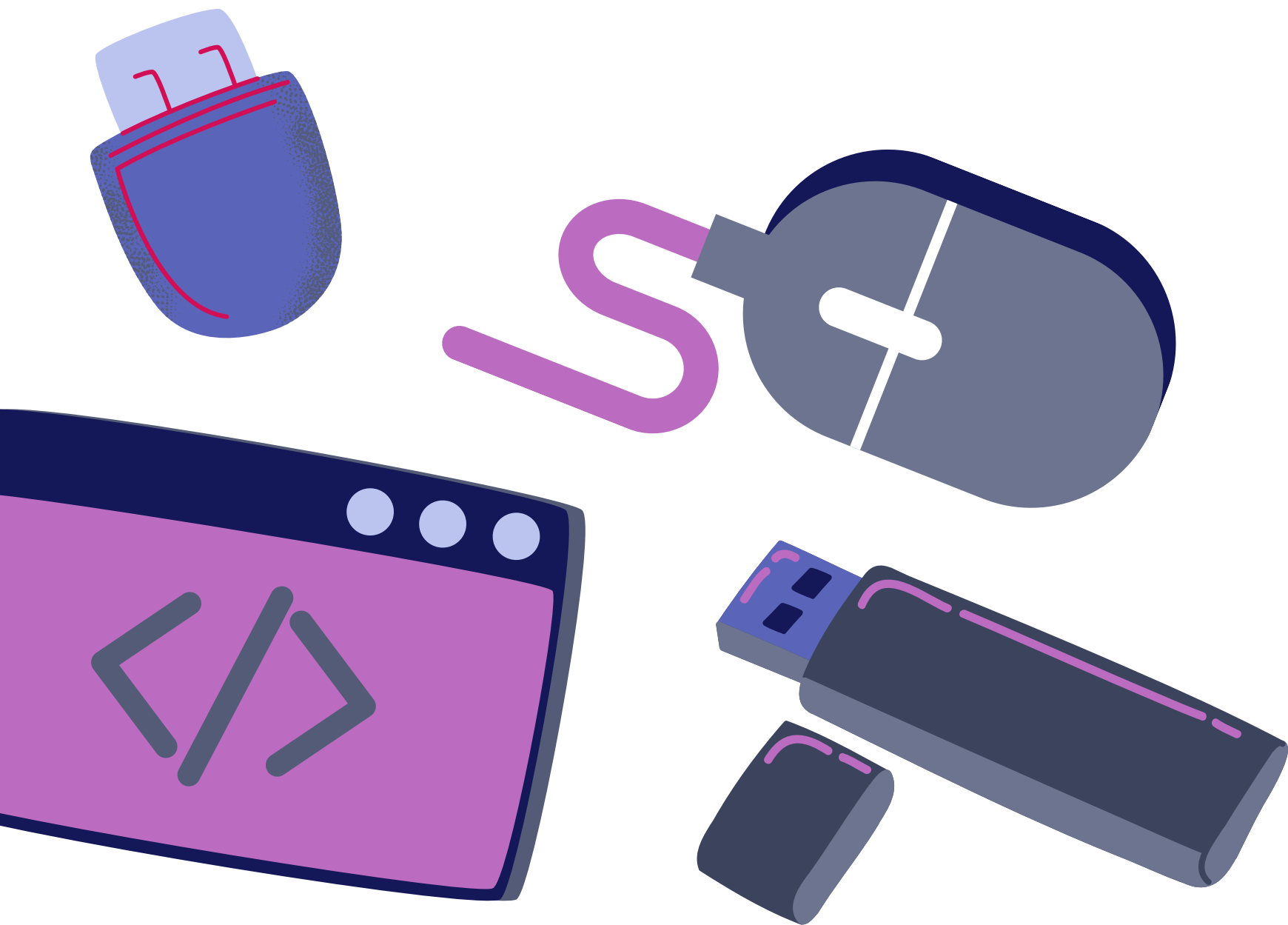
# **TRABAJO PRÁCTICO N°2**

## ***Sintaxis y Semántica de los Lenguajes K2055***

# GRUPO 1

Pablo Federico Gimenez

Fabian Montes Solis



# Lenguajes

## C#



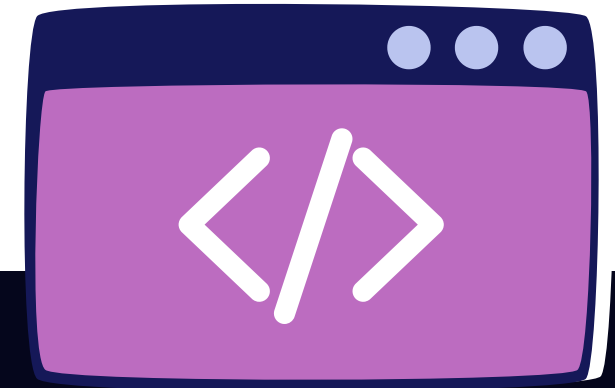
C# (leído en inglés “C sharp”) es un lenguaje orientado a objetos de propósito general diseñado por Microsoft para su plataforma .NET

## Javascript



JavaScript es un lenguaje de programación que los desarrolladores utilizan para hacer páginas web interactivas.

# C# breve historia



- **Andrés Hejlsberg** conocido por su creación del lenguaje de programación Turbo Pascal y por ser el arquitecto principal de Delphi, decidió formar un equipo de trabajo en 1999 para crear un nuevo lenguaje de programación, que hoy conocemos como C#. En sus inicios el nombre inicial que se barajó fue Cool (C Object Oriented Language).
- C# se presentó al público en el año 2000 en una conferencia. El lenguaje fue diseñado para ser simple, moderno y versátil y su primera versión oficial lanzada en el año 2002, se parecía mucho a Java. De hecho, se creó con el fin de ser algo exclusivo de Microsoft y que compita con Java.



# Javascript breve historia

- JavaScript fue creado por **Brendan Eich**, se introdujo en 1995 como una forma de agregar programas a páginas web en el navegador Netscape Navigator. En esa época, empezaban a desarrollarse las primeras aplicaciones web y por tanto, las páginas web comenzaban a incluir formularios complejos, surgió la necesidad de un lenguaje de programación que se ejecutara en el navegador del usuario.
- A lo largo de los primeros años fue mejorando y incluyendo tecnologías como elemento XML HttpRequest por parte de Microsoft en 1998. En el 2000, Douglas Crockford inventó el documento JSON. En 2004, Gmail empezó a utilizar JavaScript e incorporó AJAX masivamente para hacer sus procesos más eficaces.



# BNF Javascript

JS

*PrivateIdentifier* ::

*# IdentifierName*

*IdentifierName* ::

*IdentifierStart*

*IdentifierName IdentifierPart*

*IdentifierStart* ::

*IdentifierStartChar*

*\ UnicodeEscapeSequence*

*IdentifierPart* ::

*IdentifierPartChar*

*\ UnicodeEscapeSequence*

*NumericLiteralSeparator* ::

*-*

*BooleanLiteral* ::

**true**

**false**

*DecimalDigits*<sub>[Sep]</sub> ::

*DecimalDigit*

*DecimalDigits*<sub>[?Sep]</sub> *DecimalDigit*

*[+Sep] DecimalDigits*<sub>[+Sep]</sub> *NumericLiteralSeparator* *DecimalDigit*

*DecimalDigit* :: one of

**0 1 2 3 4 5 6 7 8 9**

*NonZeroDigit* :: one of

**1 2 3 4 5 6 7 8 9**

*ExponentPart*<sub>[Sep]</sub> ::

*ExponentIndicator* *SignedInteger*<sub>[?Sep]</sub>

*ExponentIndicator* :: one of

**e E**

*AsciiLetter* :: one of

**a b c d e f g h i j k l m n o p q r s t u v w x y z A B C D E F G H I J K L M**

**N O P Q R S T U V W X Y Z**



# BNF Javascript



*DecimalLiteral* ::

*DecimalIntegerLiteral* . *DecimalDigits*<sub>[+Sep]</sub> *opt* *ExponentPart*<sub>[+Sep]</sub> *opt*  
.  
*DecimalDigits*<sub>[+Sep]</sub> *ExponentPart*<sub>[+Sep]</sub> *opt*  
*DecimalIntegerLiteral* *ExponentPart*<sub>[+Sep]</sub> *opt*

*DecimalIntegerLiteral* ::

0  
*NonZeroDigit*  
*NonZeroDigit* *NumericLiteralSeparator*<sub>opt</sub> *DecimalDigits*<sub>[+Sep]</sub>  
*NonOctalDecimalIntegerLiteral*

*HexIntegerLiteral*<sub>[Sep]</sub> ::

0x *HexDigits*<sub>[?Sep]</sub>  
0X *HexDigits*<sub>[?Sep]</sub>

*HexDigits*<sub>[Sep]</sub> ::

*HexDigit*  
*HexDigits*<sub>[?Sep]</sub> *HexDigit*  
<sub>[+Sep]</sub> *HexDigits*<sub>[+Sep]</sub> *NumericLiteralSeparator* *HexDigit*

*HexDigit* :: one of

0 1 2 3 4 5 6 7 8 9 a b c d e f A B C D E F

*OctalDigit* :: one of

0 1 2 3 4 5 6 7

*NonOctalDigit* :: one of

8 9

*HexIntegerLiteral*<sub>[Sep]</sub> ::

0x *HexDigits*<sub>[?Sep]</sub>

0X *HexDigits*<sub>[?Sep]</sub>

*SourceCharacter* ::

any Unicode code point

# Javascript - Ordenamiento

## Funcion “sort()” Ejemplo

```
const array = [5, 3, 8, 4, 1, 9, 6, 7, 2, 0];  
  
console.time('sort');  
array.sort((a, b) => a - b);  
console.timeEnd('sort');  
  
console.log('array ordenado:', array);
```

sort: 0.031005859375 ms

array ordenado: ▶ (10) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

sort: 0.02392578125 ms

array ordenado: ▶ (10) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

sort: 0.026123046875 ms

array ordenado: ▶ (10) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

En Google Chrome se utiliza el motor V8 para ejecutar Javascript tambien lo utiliza Node.js. Depende del navegador Firefox tiene SpiderMonkey y Safari JavaScriptCore .



```

const array = [5, 3, 8, 4, 1, 9, 6, 7, 2, 0];
const bubbleSort = (arr) => {
  let wall = arr.length - 1;
  while (wall > 0){
    let index = 0;
    while (index < wall) {
      if (arr[index] > arr[index + 1]) {
        let aux = arr[index];
        arr[index] = arr[index + 1];
        arr[index + 1] = aux;
      }
      index++;
    }
    wall--;
  }
  return arr;
};
console.time('bubbleSort');
bubbleSort(array);
console.timeEnd('bubbleSort');

console.log('array ordenado:', array);

```

# ALGORITMO DE ORDENAMIENTO “bubble-sort”

Intercambia repetidamente elementos adyacentes para organizarlos de forma ascendente.

---

```

bubbleSort: 0.051025390625 ms

```

```

array ordenado: ▶ (10) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```

---

```

bubbleSort: 0.04296875 ms

```

```

array ordenado: ▶ (10) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```

# ALGORITMO DE ORDENAMIENTO

## Insertion

Permite ordenar con Insertion implica pasar por una pila, tomar un elemento, compararlo con el primero, intercambiar lugares si un elemento es más grande que otro y continuar este proceso hasta que el elemento mínimo se encuentre en la ubicación correcta.

---

insertionSort: 0.046142578125 ms

---

array ordenado: ▶ (10) [5, 3, 8, 4, 1, 9, 6, 7, 2, 0]

---

insertionSort: 0.0498046875 ms

---

array ordenado: ▶ (10) [5, 3, 8, 4, 1, 9, 6, 7, 2, 0]

---

insertionSort: 0.038818359375 ms

---

array ordenado: ▶ (10) [5, 3, 8, 4, 1, 9, 6, 7, 2, 0]

```
const array = [5, 3, 8, 4, 1, 9, 6, 7, 2, 0];

const insertionSort = (arr) => {
  for (let i = 1; i < arr.length; i++) {
    let key = arr[i];
    let j = i - 1;
    while (j >= 0 && arr[j].num > key.num) {
      arr[j + 1] = arr[j];
      j--;
    }
    arr[j + 1] = key;
  }
  return arr;
};

console.time('insertionSort');
insertionSort(array);
console.timeEnd('insertionSort');

console.log('array ordenado:', array);
```

```
JS archivo.js > partition
```

```
1  const array = [5, 3, 8, 4, 1, 9, 6, 7, 2, 0 ];
2
3  function partition(arr, low, high) {
4      let pivot = arr[high];
5      let i = (low - 1);
6
7      for (let j = low; j < high; j++) {
8
9          if (arr[j] <= pivot) {
10             i++;
11
12             [arr[i], arr[j]] = [arr[j], arr[i]];
13         }
14     }
15
16     [arr[i + 1], arr[high]] = [arr[high], arr[i + 1]];
17     return (i + 1);
18 }
19 // Función principal de QuickSort
20 function quickSort(arr, low, high) {
21     if (low < high) {
22         let pi = partition(arr, low, high);
23
24         quickSort(arr, low, pi - 1);
25         quickSort(arr, pi + 1, high);
26     }
27 }
28 console.time('quickSort');
29 var n = array.length;
30 quickSort(array, 0, n-1);
31 console.timeEnd('quickSort');
32
```

# ALGORITMO DE ORDENAMIENTO

## “QuickSort”

El algoritmo quick sort se basa en la estrategia divide-y-vencerás, porque divide el problema en dos subproblemas, que se resuelven de manera individual e independiente. Los resultados se unen después.

---

```
quickSort: 0.02978515625 ms
```

```
array ordenado: ► (10) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

---

```
quickSort: 0.030029296875 ms
```

```
array ordenado: ► (10) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

# BNF C#



`<program> ::= {<namespace_declaration>}`

`<namespace_declaration> ::= "namespace" <identifier> "{" {<type_declaration>} "}"`

`<type_declaration> ::= <class_declaration> | <interface_declaration> | <enum_declaration>`

`<class_declaration> ::= "class" <identifier> "{" {<class_member_declaration>} "}"`

`<class_member_declaration> ::= <field_declaration> | <method_declaration> | <property_declaration> |  
<constructor_declaration>`

`<field_declaration> ::= <type> <identifier> ";"`

`<method_declaration> ::= <type> <identifier> "(" [<parameter_list>] ")" "{" <statement_list> "}"`

`<property_declaration> ::= <type> <identifier> "{" <accessor_declarations> "}"`

`<constructor_declaration> ::= <identifier> "(" [<parameter_list>] ")" "{" <statement_list> "}"`

`<statement_list> ::= {<statement>}`

# BNF C#



`<statement> ::= <local_variable_declaration> | <expression_statement> | <o> | <o> | <o>`

`<local_variable_declaration> ::= <type> <identifier> ["=" <expression>] ";"`

`<expression> ::= <identifier> | <literal> | <binary_expression>`

`<binary_expression> ::= <expression> <binary_operator> <expression>`

`<type> ::= "int" | "float" | "bool" | "string" | <identifier>`

`<identifier> ::= <letter> {<letter_or_digit>}`

`<literal> ::= <integer_literal> | <string_literal>`

`<binary_operator> ::= "+" | "-" | "*" | "/" | "=="`

`<integer_literal> ::= <digit> {<digit>}`

`<string_literal> ::= "\"" {<character>} "\""`

`<digit> ::= "0" | "1" | ... | "9"`

# BNF/C#



## Explicación y Componentes Clave

- **<program>**: Un programa en C# consta de una o más declaraciones de espacios de nombres.
- **<namespace\_declaration>**: Un espacio de nombres que puede contener declaraciones de tipos como clases o interfaces.
- **<type\_declaration>**: Define las estructuras básicas, como clases, interfaces y enumeraciones.
- **<class\_declaration>**: Define una clase, que puede contener campos, métodos, propiedades y constructores.
- **<statement\_list>**: Es una secuencia de declaraciones dentro de un método o bloque de código.
- **<expression>**: Define cómo se pueden estructurar las expresiones en C#, incluyendo identificadores, literales y operaciones binarias.
- **<type>**: Representa los tipos básicos disponibles en C# y los definidos por el usuario.

# C#- Ordenamiento

## Funcion

## “sort()” Ejemplo

```
$ dotnet run
Array.Sort() tiempo: 6546 ticks
List<T>.Sort() tiempo: 666 ticks
LINQ OrderBy() tiempo: 43529 ticks
```

Array.Sort()

0,2835 ms

List<T>.Sort()

0,2743 ms

OrderBy()

9,7681 ms

```
1  using System;
2  using System.Collections.Generic;
3  using System.Diagnostics;
4  using System.Linq;
5
6  0 referencias
7  class Program
8  {
9
10     0 referencias
11     static void Main()
12     {
13         int[] array = { 5, 3, 8, 4, 1, 9, 6, 7, 2, 0 };
14         List<int> list = new List<int>(array);
15
16         // Medir rendimiento de Array.Sort()
17         Stopwatch stopwatch = Stopwatch.StartNew();
18         Array.Sort(array);
19         stopwatch.Stop();
20         Console.WriteLine("Array.Sort() tiempo: " + stopwatch.ElapsedTicks + " ticks");
21
22         // Medir rendimiento de List<T>.Sort()
23         stopwatch.Restart();
24         list.Sort();
25         stopwatch.Stop();
26         Console.WriteLine("List<T>.Sort() tiempo: " + stopwatch.ElapsedTicks + " ticks");
27
28         // Medir rendimiento de LINQ OrderBy()
29         stopwatch.Restart();
30         var sortedList = list.OrderBy(x => x).ToList();
31         stopwatch.Stop();
32         Console.WriteLine("LINQ OrderBy() tiempo: " + stopwatch.ElapsedTicks + " ticks");
33     }
34 }
```



# Ordenamiento en objeto

## Personas C#

```
static void Main(string[] args)
{
    List<Person> persons = new List<Person>();
    for (int i = 0; i < 10000; i++)
    {
        persons.Add(new Person("P" + i.ToString(), "Janson" + i.ToString()));
    }

    Stopwatch watch = Stopwatch.StartNew();
    Sort(persons);
    watch.Stop();

    Console.WriteLine("Sort: {0}ms", watch.ElapsedMilliseconds);

    List<Person> personsOrderBy = new List<Person>();
    for (int i = 0; i < 10000; i++)
    {
        personsOrderBy.Add(new Person("P" + i.ToString(), "Janson" + i.ToString()));
    }

    watch = Stopwatch.StartNew();
    OrderBy(personsOrderBy);
    watch.Stop();
    Console.WriteLine("OrderBy: {0}ms", watch.ElapsedMilliseconds);
}
```

```
Sort: 703ms
OrderBy: 23ms
ArraySort: 168ms
```

```
Sort: 216ms
OrderBy: 198ms
ArraySort: 176ms
```

```
Sort: 545ms
OrderBy: 471ms
```

Como podemos notar en situaciones donde se trabaja con objetos y estos tienen un cierto volumen de datos pareceria indicar que orderBy rinde de mejor forma que Sort y que ArraySort

# CUADRO RESUMEN

Característica	C#	JavaScript
Tipado	Estático	Dinámico
Sintaxis	Más formal, orientada a objetos	Más flexible, basada en prototipos
Funciones de ordenamiento	Métodos de instancia de Array y List<T>	Método sort de los arreglos
Comparadores	Delegados o expresiones lambda	Funciones de comparación
Rendimiento	Generalmente más rápido debido a la compilación y optimización del JIT	Puede variar dependiendo del motor JavaScript y la optimización del código

# Conclusiones

En general, no hay un "mejor" algoritmo universalmente ya que depende del contexto. Ambos lenguajes utilizan algoritmos optimizados que se ajustan a una amplia gama de situaciones. Javascript es mas utilizado para las paginas Web y C# es mas utilizado para proyectos mucho mas grandes y amplios .Los dos son eficientes y utiles.



# PREGUNTAS ¿?

- Para cada uno de los navegadores se ejecuta Javascript con el motor V8
- ¿Los algoritmos de ordenamiento de C# son mas rapidos que los de javascript?
- ¿Las BNF de Javascript son mejores en rendimiento que las de C#?
- ¿Con que proposito se creo C# y Javascript?