Lenguaje C#

En Unity podemos utilizar varios lenguajes de programación para escribir los scripts que le darán vida a nuestro juego. Sin embargo, durante este curso nos enfocaremos únicamente en C#, que es un lenguaje de programación orientado a objetos muy potente.

El proceso en que el código fuente que escribimos se pasa al código de máquina que ejecuta el procesador es un proceso de traducción que suele tener diferentes caractacterísticas según el lenguaje que utilicemos.

En un lenguaje como C++ este proceso es directamente la compilación del código fuente a código de máquina, lo cual hace a este lenguaje más lento al traducirse pero más rápido al ejecutarse.

En lenguajes interpretados el proceso es el de la traducción línea por línea y ejecución inmediata, lo cual hace a la ejecución más lenta pero no hay que esperar a que ninguna compilación termine para ejecutar.

En lenguajes como Java o C# este proceso suele ser híbrido. El código fuente se compila a un código intermedio que luego termina siendo ejecutado sobre una plataforma o máquina virtual que es necesaria que esté instalada en la máquina que se corre el programa y que termina de traducir el código mediante una compilación justo a tiempo, un proceso de traducción dinámica que es más rápido que la interpretación aunque no es exactamente lo mismo que la compilación.

Esto hace a Java y C# lenguajes muy interesantes pues mientras que exista la plataforma o máquina virtual para diferentes plataformas, el programa que escribamos podrá ser ejecutado en todas esas plataformas.

Este es el principio que hace que Unity funcione en tantas plataformas diferentes. Si bien la implementación del Framework .Net de Microsoft funciona únicamente en Windows, existe Mono framework que es una implementación libre y código abierto de este framework de Microsoft. Este nos permite crear aplicaciones con C# que corran sobre esta capa que nos aisla del sistema operativo que está debajo.

Cuando nosotros escribimos el código que define el comportamiento de algún personaje, objeto o característica de nuestro juego el código se compila a un lenguaje intermedio que luego se ejecuta sobre esta plataforma, por eso Mono es necesario.

De todas formas, desde el punto de vista de la creación de nuestro juego y de nuestro rol de programadores, la existencia de Mono y el proceso de traducción y ejecución sobre el framework es invisible y no tenemos que preocuparnos. Tanto en la ejecución mientras lo probamos como al terminar el juego que estemos realizando, Unity se encarga de incluir todos los archivos necesarios, librerías y framework para que podamos ejecutar el juego sin problemas.

Código en C#

- **Comentarios**

```
// Single line

/* Multiple
    line  */

/// XML comments on a single line

/** XML comments on multiple lines */
```

- **Tipos**

Data Type	Description	Example
object	The base of all types	object obj = null;
string	String type – sequence of Unicode characters	string str= "Unity";
sbyte	8-bit signed integral type	sbyte val = 12;
short	16-bit signed integral type	short val = 12;
int	32-bit signed integral type	int val = 12;
long	64-bit signed integral type	long val1 = 12; long val2 = 34L;
bool	Boolean type; a bool value is either true or false	bool val1 = true; bool val2 = false;
char	Character type; a char value is a Unicode character	char val = 'h';
byte	8-bit unsigned integral type	byte val1 = 12; byte val2 = 34U;
ushort	16-bit unsigned integral type	ushort val1 = 12; ushort val2 = 34U;
uint	32-bit unsigned integral type	uint val1 = 12; uint val2 = 24U;

ulong	64-bit unsigned integral type	ulong val1 = 12;
		ulong val2 = 32U;
		ulong val3 = 56L;
		ulong val4 = 78UL;

float	Single-precision floating point type	float val = 1.23F;
double	Double-precision floating point type	double val1 = 1.23; double val2 = 4.56D;
decimal	Precise decimal type with 28 significant digits	decimal val = 1.23M

Value Types

Una variable del tipo de una estructura, es un tipo de dato por valor.

De este modo, la variable contiene el valor o los valores de los miembros de la estructura. No requiere realizar un new para instanciarlo (aunque es factible hacerlo) y su asignación genera otra variable con los valores copiados.

Los tipos de datos por valor NO pueden ser nulos.

Value Types	
Primitives	Signed • sbyte (signed byte) • short • int • long Unsigned • byte • ushort • uint • ulong Character • char Floating-point • float • double • decimal Logical • bool
Enums	enum GameState : int { Unknown, Off, On }
Structs	struct Point { int x, y; }

Reference Types

Un objeto, instancia de una clase, es un tipo de dato referenciado. De este modo la "variable" objeto posee realmente una referencia al objeto en memoria y su "copia" por asignación copiará dicha referencia.

Deben ser creados explícitamente (con un new)

Pueden ser nulos.

Reference Types	
Classes	public class Circle { }
Interfaces	interface IShape { }
Arrays	int[] a = new int[10];
Delegates	delegate void MyFunction();

`object` (superclass de todas las clases)
`string`
arrays, clases, interfaces, delegates

- **Conversiones**

```
// int to string
int x = 123;
string y = x.ToString();  // y is "123"

// string to int
y = "456";
x = int.Parse(y);    // or x = Convert.ToInt32(y);

// double to int
double z = 3.5;
x = (int) z;    // x is 3  (trunca decimal)
```

Implícitas

No requieren especificar nada. Son seguras y no se pierde precisión en la operación.

From	To
sbyte	short, int, long, float, double, or decimal
byte	short, ushort, int, uint, long, ulong, float, double, decimal
short	int, long, float, double, or decimal
ushort	int, uint, long, ulong, float, double, or decimal
int	long, float, double, or decimal
uint	long, ulong, float, double, or decimal
long	float, double, or decimal
char	ushort, int, uint, long, ulong, float, double, or decimal
float	double
ulong	float, double, or decimal

Explícitas

Requieren de un type cast operator. Los tipos de datos son compatibles pero existe una posible pérdida de información en la operación.

From	To
sbyte	byte, ushort, uint, ulong, or char
byte	sbyte or char
short	sbyte, byte, ushort, uint, ulong, or char
ushort	sbyte, byte, short, or char
int	sbyte, byte, short, ushort, uint, ulong,or char
uint	sbyte, byte, short, ushort, int, or char
long	sbyte, byte, short, ushort, int, uint, ulong, or char
char	sbyte, byte, or short
float	sbyte, byte, short, ushort, int, uint, long, ulong, char,or decimal
ulong	sbyte, byte, short, ushort, int, uint, long, or char
double	sbyte, byte, short, ushort, int, uint, long, ulong, char, float,or decimal
decimal	sbyte, byte, short, ushort, int, uint, long, ulong, char, float, or double

- **Operadores**

Comparación

```
==   <   >   <=   >=   !=
```

Aritméticos

```
+   -   *   /
%  (módulo)
/  (división entera si ambos operandos son enteros)
Math.Pow(x, y)
```

Asignación

```
=   +=   -=   *=   /=    %=   &=   |=   ^=   <<=   >>=   ++   --
```

Lógicos

```
&&   ||   &   |    ^    !
```

Concatenación de string

```
+
```

- **Variables**

```
int num = 3;
float numWithComma = 5.5f;
string text = "Hello World";
bool isValid = true;

~~int 3_var_starts_with_number = 3;~~
~~int var with spaces = 3;~~
```

- **Constantes**

```
const double PI = 3.14;

// Can be set to a const or a variable.
// May be initialized in a constructor.
readonly int MAX_HEIGHT = 9;
```

- **Enumeradores**

```
enum Action {Start, Stop, Rewind, Forward};

enum Status {Flunk = 50, Pass = 70, Excel = 90};
```

- **Arrays**

```
char[] letters = { 'a', 'b', 'c' };
int[] numbers = new int[5] { 1, 10, 5, 23, 43 };
int[] another_array_default_values = new int[10];

string[] names = new string[5];
names[0] = "David";
int namesSize = names.Length;

float[,] twoD = new float[rows, cols];
twoD[2,0] = 4.5f;

int[][] jagged = new int[3][] { new int[5], new int[2], new
int[3] };
jagged[0][4] = 5;
```

Existen collections que contienen ArrayList, List, etc. en el namespace System.Collections.

- **Strings**

```
// String concatenation
string school = "Harding ";
school = school + "University";   // school is "Harding
University"

// String comparison
string mascot = "Bisons";
if (mascot == "Bisons")     // true
if (mascot.Equals("Bisons"))    // true
if (mascot.ToUpper().Equals("BISONS"))    // true
if (mascot.CompareTo("Bisons") == 0)     // true

Console.WriteLine(mascot.Substring(2, 3));     // Prints "son"

// My birthday: Oct 12, 1973
DateTime dt = new DateTime(1973, 10, 12);
string s = "My birthday: " + dt.ToString("MMM dd, yyyy");

// Mutable string
System.Text.StringBuilder buffer = new
System.Text.StringBuilder("two ");
buffer.Append("three ");
buffer.Insert(0, "one ");
buffer.Replace("two", "TWO");
Console.WriteLine(buffer);     // "one TWO three"
```

- **Control de flujo de ejecución**

```
string greeting = age < 20 ? "What's up?" : "Hello"; // operador
ternario

if (x < y)
  Console.WriteLine("greater");

if (x != 100) {
  x *= 5;
  y *= 2;
}
else
  z *= 6;

string color = "red";
switch (color) {               // Can be any predefined type
  case "red":   r++;   break; // break is mandatory; no fall-
through
  case "blue":  b++;   break;
  case "green": g++;   break;
  default:  other++;   break; // break necessary on default
}

while (i < 10)
  i++;

for (i = 2; i <= 10; i += 2)
  Console.WriteLine(i);

do
  i++;
while (i < 10);

foreach (int i in numArray)
  sum += i;

// foreach puede ser usado para iterar cualquier colección
using System.Collections;
ArrayList list = new ArrayList();
list.Add(10);
list.Add("Bisons");
list.Add(2.3);

foreach (Object o in list)
  Console.WriteLine(o);
```

- **Funciones, Métodos, Clases, Objetos**

```
// Return single value
int Add(int x, int y) {
   return x + y;
}

int sum = Add(2, 3);  // Return no value
void PrintSum(int x, int y) {
   Console.WriteLine(x + y);
}

PrintSum(2, 3);


// Pass by value (default), in/out-reference (ref), and out-
reference (out)
void TestFunc(int x, ref int y, out int z, Point p1, ref Point
p2) {
   x++;  y++;  z = 5;
   p1.x++;       // Modifying property of the object
   p1 = null;    // Remove local reference to object
   p2 = null;   // Free the object
}

class Point {
   public int x, y;
}

Point p1 = new Point();
Point p2 = new Point();
p1.x = 2;
int a = 1, b = 1, c;   // Output param doesn't need initializing
TestFunc(a, ref b, out c, p1, ref p2);
Console.WriteLine("{0} {1} {2} {3} {4}",
   a, b, c, p1.x, p2 == null);   // 1 2 5 3 True

// Accept variable number of arguments
int Sum(params int[] nums) {
  int sum = 0;
  foreach (int i in nums)
    sum += i;
  return sum;
}

int total = Sum(4, 3, 2, 1);   // returns 10


SuperHero hero = new SuperHero();

hero.Name = "SpamMan";
```

```
hero.PowerLevel = 3;

hero.Defend("Laura Jones");
SuperHero.Rest();   // Calling static method

SuperHero hero2 = hero;   // Both refer to same object
hero2.Name = "WormWoman";
Console.WriteLine(hero.Name);   // Prints WormWoman

hero = null ;   // Free the object

if (hero == null)
  hero = new SuperHero();

Object obj = new SuperHero();
Console.WriteLine("object's type: " + obj.GetType().ToString());
if (obj is SuperHero)
  Console.WriteLine("Is a SuperHero object.");
```

- **Modificadores de acceso**

```
public
private
internal
protected
protected internal
static
```

- **Constructores, Destructores**

```
class SuperHero {
  private int mPowerLevel;

  public SuperHero() {
     mPowerLevel = 0;
  }

  public SuperHero(int powerLevel) {
    this.mPowerLevel= powerLevel;
  }

  ~SuperHero() {
    // Destructor code to free unmanaged resources.
    // Implicitly creates a Finalize method.
  }
}
```

- **Herencia**

```
class FootballGame : Competition {
  ...
}
```

- **Polimorfismo, Métodos virtuales, Hiding/Overriding**

```
class ClaseBase
{
   public void doHide() { Console.WriteLine("Hide Base"); }
   virtual public void doOver() { Console.WriteLine("Over
Base"); }
}

class ClaseHija : ClaseBase
{
   new public void doHide() { Console.WriteLine("Hide Hija"); }
   override public void doOver() { Console.WriteLine("Over
Hija"); }
}

class Program
{
   static void Main(string[] args)
   {
      Console.WriteLine("---B---");
      ClaseBase b = new ClaseBase();
      b.doHide();
      b.doOver();

      Console.WriteLine("---H---");
      ClaseHija h = new ClaseHija();
      h.doHide();
      h.doOver();

      Console.WriteLine("---HB---");
      ClaseBase hb = new ClaseHija();
      hb.doHide();
      hb.doOver();
    }
}
```

- **Interfaces**

```
// Interface definition
interface IAlarmClock {
  ...
}

// Extending an interface
interface IAlarmClock : IClock {
  ...
}

// Interface implementation
class WristWatch : IAlarmClock, ITimer {
   ...
}
```

- **Namespaces**

```
namespace Harding.Compsci.Graphics {
  ...
}

or

namespace Harding {
  namespace Compsci {
    namespace Graphics {
      ...
    }
  }
}

// Import single class
using Rectangle = Harding.CompSci.Graphics.Rectangle;

// Import all class
using Harding.Compsci.Graphics;
```

- **Manejo de excepciones**

```
Exception up = new Exception("Something is really wrong.");
throw up;

try {
  y = 0;
  x = 10 / y;
} catch (Exception ex) {      // Variable "ex" is optional
  Console.WriteLine(ex.Message);
} finally {
  // Code that always gets executed
}
```

- **Números aleatorios**

```
Random r;// = new Random(DateTime.Now.Millisecond);
r = new Random();
for (int i = 0; i < 20; i++)
{
   Console.WriteLine( r.Next(0,10) );
}
```

Ejercicios

Los siguientes ejercicios se platean para realizar utilizando MonoDevelop únicamente (sin necesidad de Unity), con el objetivo de familiarizarse con C# en programas simples que involucren únicamente una consola (línea de comandos).

Para ello es necesario crear un nuevo proyecto en Monodevelop (crear uno por ejercicio dentro de una misma solución) que sea de tipo ConsoleApplication.

Tenga en cuenta que con Console.WriteLine() se escribe en pantalla y que con Console.ReadLine() se lee un texto ingresado por el usuario en la consola.

- **Ejercicios básicos**

1. Mostrar en pantalla la suma de dos números ingresados por teclado.

2. Dados dos números n1 y n2, indicar en pantalla cual es el mayor y cuál es el menor o igual.

3. Dados dos números n1 y n2, mostrar en pantalla:
si n1 > n2: La suma de ambos números.
si n1 = n2: El producto de ambos números y la constante k = 5.
si n1 < n2: La diferencia de ambos números.

4. Solicitar al usuario el ingreso de un número por teclado. Indicar en pantalla si el mismo es par o no.

Ej:
Ingrese número: **3**
Es impar

5. Solicitar al usuario el ingreso de un número n por teclado, mostrarlo en pantalla sólo si el número ingresado es menor a k = 5 y volver a solicitar el ingreso del número hasta que el mismo sea igual a cero.

Ej:
Ingrese número: **3**
3
Ingrese número: **7**
Ingrese número: **8**
Ingrese número: **1**
1
Ingrese número: **0**
0

6. Solicitar al usuario el ingreso de un número. Si es menor a 1 o mayor a 5 solicitar nuevamente el ingreso. Mostrar el nombre del número en texto.

7. Solicitar al usuario el ingreso de un carácter y un número n. Mostrar en pantalla el texto ingresado n veces.

Ej:
Ingrese el texto: **H**
Ingrese el número: **3**
H
H
H

8. Programar la función void escribeArrayInt(int[] numeros) , que imprime en consola un array de números enteros.

9. Programar una función que reciba como parámetro un array y lo imprima en orden inverso (es decir desde el último elemento hacia el primero).

10. Programar una función que reciba como parámetro un array e imprima primero los elementos de posición par y después los de posición impar.

11. Programar una función que reciba como parámetro un array e imprima primero los elementos pares (independientemente de que ocupen posición par o impar) y después los impares.

12. Dado un array de 10 elementos con números arbitrarios dentro de el. Crear un función con el siguiente prototipo:

int GetIndex(int[] nums, int val);

Que retorne el índice del primer elemento que posea el valor val. Si el mismo no es encontrado deberá retornar -1.

13. Dado un array de 20 elementos con números arbitrarios dentro de el. Crear un función que retorne cuantas veces se encuentra un determinado número dentro de él.

- ## Ejercicios intermedios

1. Crear un programa que solicite una cadena de caracteres y retorne la cantidad de letras que posee la misma.

a. Utilizando la propiedad **Length** del string.
b. Utilizando una función que haga una reimplementación propia la propiedad **Length** del string.

2. Crear un programa que lea una cadena de caracteres y la compare contra otra definida por código dentro de la aplicación:

a. Utilizando el operador **== entre string**.
b. Utilizando una función que haga una reimplementación propia del operador **== entre string**.

Ej:
Ingrese la palabra clave: loquesea
> Esa no es la palabra clave.

3. Crear un programa que solicite dos cadenas de caracteres y retorne a ambas concatenadas.

a. Utilizando el método **Concat()**.
b. Utilizando una función que haga una reimplementación propia del método **Concat()**.

4. Crear un programa que solicite una cadena de caracteres y retorne la misma convirtiendo una letra a mayúsculas y otra a minúsculas de modo alternado.

5. Crear un programa que solicite una cadena de caracteres y retorne la misma invertida.

6. Crear un programa que solicite una cadena de caracteres y retorne la suma del valor del valor ascii de cada carácter.

7. Crear un programa que solicite una cadena de caracteres y retorne la misma sin vocales.

8. Crear un programa que solicite tres cadena de caracteres de retorne las mismas concatenadas por orden alfabético de la primer letra de cada cadena (separar las palabras por medio de algún caracter arbitrario).

Ej:
cadena1: boca
cadena2: racing
cadena3: river
Resultado: boca-racing-river

9. Crear un programa que solicite dos cadenas de caracteres e informe si la primera se encuentra contenida dentro de la segunda.

a. Utilizando el método **Contains()**.
b. Utilizando una función que haga una reimplementación propia del método **Contains()**.

Ej:
cadena1: "La inmensa casa se encontraba sobre la ladera"
cadena2: "casa"
Resultado: sí.

10. Escribir una función que reciba como parámetro un array de int y determine cuales son los elementos de mayor y menor valor de la matriz.

11. Escribir una función que reciba como parámetro un array de int y devuelva el promedio de todos los elementos del array.

12. Escribir una función que reciba como parámetro un array y lo devuelva invertido.

13. Un palíndromo es una cadena que dice lo mismo si se lee hacia adelante que si se lee hacia atrás. Algunos ejemplos de palíndromos son "radar", "ojo", "oso". Escriba una función que al recibir un array que contiene una cadena devuelva verdadero si dicha cadena es un palíndromo. (Se considera que la cadena no tiene espacios ni signos de puntuación).

14. Escribir una función que al recibir dos arrays del mismo tipo devuelva un tercer array resultado de intercalar posición a posición ambos arrays recibidos. En caso de que un array fuera mas grande que otro se intercalaran todas las posiciones posibles y luego se concatenarán atrás las restantes del array mas grande.

Ejemplo:
Array 1: **[1 | 3 | 4 | 67 | 2 | 1 | 45 | 98]**
Array 2: **[3 | 4 | 68 | 23 | 5 | 1]**
Array Resultante: **[1 | 3 | 3 | 4 | 4 | 68 | 67 | 23 | 2 | 5 | 1 | 1 | 45 | 98]**

15. Escribir una función que reciba por parámetro un array de caracteres y determine, en caso de haber paréntesis, si están ubicados correctamente.
Es decir:
- Si hay un paréntesis de apertura debe existir su correspondiente paréntesis de cierre.
- No debe haber paréntesis de cierre sin que lo anteceda su respectivo paréntesis de apertura.

Ejemplos:
• "**Hls (gstr) k ( )**" : Los paréntesis SI están ubicados correctamente.
• "**jkl) (d**" : Los paréntesis NO están ubicados correctamente.
• "**greee (ml**": Los paréntesis NO están ubicados correctamente.

- ## Ejercicios avanzados

1. Escribir una función que reciba una matriz de int por parámetro y determine cuales son los elementos de mayor y menor valor de la matriz.

2. Escribir una función que reciba una matriz de int (de igual cantidad de filas y columnas)

y devuelva la suma de los valores de la diagonal que va desde [0][0] a [n] [n].

3. Escribir una función que reciba como parámetro un array de cadenas de caracteres e invierta su orden.

4. Escribir una función que reciba como parámetro una matriz de números enteros y retorne el resultado de sumar cada elemento cuyo indice de columna sea múltiplo de 2, y restar cada elemento cuyo indice de columna no lo sea.

Ejemplo: Dada la matriz:

```
    0  1  2  3  4  5
0 [ 2 | 3 | 4 | 1 | 2 | 5 ]
1 [ 5 | 8 | 9 | 0 | 2 | 3 ]
2 [ 1 | 1 | 4 | 1 | 9 | 5 ]
3 [ 6 | 1 | 7 | 5 | 2 | 1 ]
4 [ 7 | 0 | 4 | 1 | 8 | 5 ]
5 [ 8 | 2 | 3 | 1 | 2 | 3 ]
```

Se muestra **(+)** si el numero debería sumarse y **(-)** si debería restarse.

```
     0       1       2       3       4       5
0 [ (-) 2 | (-) 3 | (+) 4 | (-) 1 | (+) 2 | (-) 5 ]
1 [ (-) 5 | (-) 8 | (+) 9 | (-) 0 | (+) 2 | (-) 3 ]
2 [ (-) 1 | (-) 1 | (+) 4 | (-) 1 | (+) 9 | (-) 5 ]
3 [ (-) 6 | (-) 1 | (+) 7 | (-) 5 | (+) 2 | (-) 1 ]
4 [ (-) 7 | (-) 0 | (+) 4 | (-) 1 | (+) 8 | (-) 5 ]
5 [ (-) 8 | (-) 2 | (+) 3 | (-) 1 | (+) 2 | (-) 3 ]
```

El resultado en este caso seria:

Resultado = - 2 – 5 – 1 – 6 – 7 – 8 – 3 – 8 – 1 – 1 – 0 – 2 + 4 + 9 + 4 + 7 + 4 + 3 – 1 – 0 – 1 – 5 – 1 – 1 + 2 + 2 + 9 + 2 + 8 + 2 – 5 – 3 – 5 – 1 – 5 – 3 = **-19**

5. Escribir una función que reciba por parámetro dos matrices de igual cantidad de filas y columnas y devuelva "true" en caso de que ambas matrices tengan los mismos elementos, no necesariamente ubicados en las mismas posiciones.

Ejemplo:

Matriz 1: Matriz 2:

```
[ 1 | 2 | 4 ] [ 5 | 2 | 1 ]
[ 2 | 3 | 5 ] [ 3 | 2 | 4 ]
```

Ambas matrices tienen los elementos **{ 1, 2, 2, 3, 4, 5 }**, aunque no están ubicados en las mismas posiciones.

6. Escribir una función que reciba una matriz por parámetro y devuelva un array con el resultado que se obtiene al recorrer la matriz en forma de espiral.

Ejemplo:
Dada la matriz:

```
    0   1   2   3   4   5
0 [ 2 | 3 | 4 | 1 | 2 | 5 ]
1 [ 5 | 8 | 9 | 0 | 2 | 3 ]
2 [ 1 | 1 | 4 | 1 | 9 | 5 ]
3 [ 6 | 1 | 7 | 5 | 2 | 1 ]
4 [ 7 | 0 | 4 | 1 | 8 | 5 ]
5 [ 8 | 2 | 3 | 1 | 2 | 3 ]
```

El vector resultante seria:

[2 | 3 | 4 | 1 | 2 | 5 | 3 | 5 | 1 | 5 | 3 | 2 | 1 | …. | 4 | 1 | 5 | 7]

7. Escribir una función que reciba una matriz de int por parámetro y devuelva el índice de la fila que mayor suma.

Ejemplo:
Dada la matriz:

0 [3 | 5 | 1 | 0] à 3 + 5 + 1 + 0 = **9**
1 [4 | 6 | 0 | 9] à 4 + 6 + 0 + 9 = **19**
2 [7 | 8 | 3 | 2] à 7 + 8 + 3 + 2 = **20**

La fila que mas suma es la 2.

7. Creando las clases y los objetos que crea necesarios, hacer:

a. Un juego de TaTeTi
b. Un juego de Ahorcado
c. Un juego de BlackJack
d. Un juego de Batalla Naval