PROGRAMACION

DAM · Mare Nostrum · 2024

1er Trimestre
Curso 2023 - 2024
Lista de Clases

S E P T I E M B R E

- **- 23.09.14 - Jueves**
  Clase 1:
  - Introducción a los tipos de datos.
- **- 23.09.15 - Viernes**
  Clase 2:
  - Introducción a las funciones de clase.
- **- 23.09.18 - Lunes**
  Clase 3:
  - Continuación funciones de clase.
- **- 23.09.19 - Martes**
  Clase 4:
  - Continuación funciones de clase.
  - Estudio de bucles WHILE / FOR .
- **- 23.09.21 - Jueves**
  Clase 5:
  - Funcion de numeros primos.
- **- 23.09.22 - Viernes**
  Clase 6:
  - Funcion de Collatz
  - Funciones de Objetos Dolphin
- **- 23.09.25 - Lunes**
  Clase 7:
  - Simulación Carrera Delfines
  - Realizar Funciones de Objeto
- **- 23.09.26 - Martes**
  Clase 8:
  - Funciones de Getters y Setters
  - Validar los parámetros de entrada
- **- 23.09.28 - Jueves**
  Clase 9:
  - Funciones de Concatenaciones de strings.
- **- 23.09.29 - Viernes**
  Clase 10:
  - Serie Fibonacci
  - Constructores

O C T U B R E

- **- 23.10.02 - Lunes**
  Clase 11:
  - Contenedores
  - Listas
  - Arrays
- **- 23.10.03 - Martes**
  Clase 12:
  - Declaración de Listas
  - Declaración de Arrays
  - Funciones de contenedores de datos
- **- 23.10.05 - Jueves**
  Clase 13:
  - Binary Search
- **- 23.10.09 - Lunes**
  - Festivo
- **- 23.10.10 - Martes**
  Clase 15:
  - Advertencias y consejos
  - Funcion Sort. Metodos para ordenar una coleccion
- **- 23.10.11 - Miercoles**
  Clase 16:
  - Repaso y aclaracion de dudas
- **- 23.10.13 - Viernes**
  Clase 17:
  -

- **- 23.10.06 - Viernes**
  Clase 14:
- Funciones tipo de examen. Listas y Arrays
- Reverse List, AddValue, PrintList

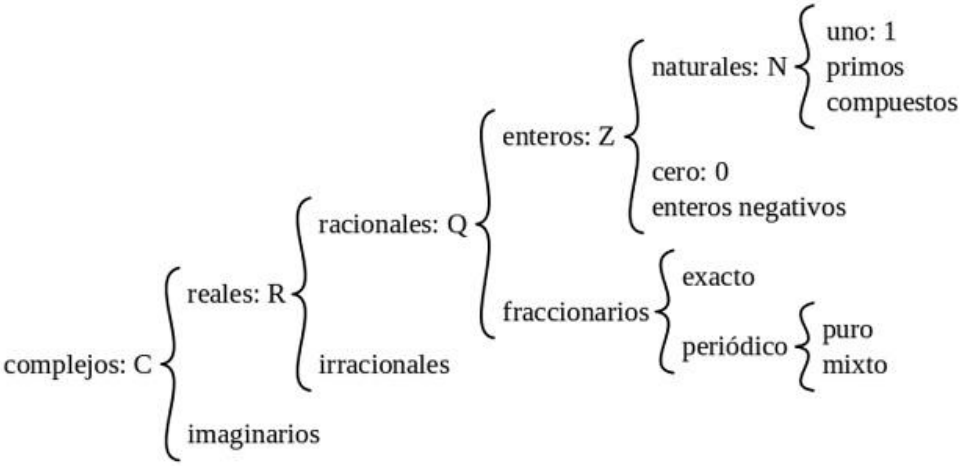
Clase 1

Jueves, 14 Septiembre 2023

Indice

Tipos de numeros

- Los **Números Naturales «N»** son todos los números mayores de cero (algunos autores incluyen también el 0) que sirven para contar. No pueden tener parte decimal, fraccionaria, ni imaginaria. N = [1, 2 , 3, 4, 5...]
- Los **Números Enteros «Z»** incluye al conjunto de los números naturales, al cero y a sus opuestos (los números negativos). Es decir: Z = [...-2, -1, 0, 1, 2...]
- Los **Números Racionales «Q»** son aquellos que pueden expresarse como una fracción de dos números enteros. Por ejemplo: Q = [¼, ¾, etc.]
- Los **Números Reales «R»** se definen como todos los números que pueden expresarse en una línea continua, por tanto incluye a los conjuntos anteriores y además a los números irracionales como el número «∏» y «e».
- Los **Números Complejos «C»** incluye todos los números anteriores más el número imaginario «i». C = [N, Z, Q, R, I].



-

Estudios relacionados

- Codigo Ascii de carácteres especiales

Caracteres ASCII de control

00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)
06	ACK	(reconocimiento)
07	BEL	(timbre)
08	BS	(retroceso)
09	HT	(tab horizontal)
10	LF	(nueva línea)
11	VT	(tab vertical)
12	FF	(nueva página)
13	CR	(retorno de carro)
14	SO	(desplaza afuera)
15	SI	(desplaza adentro)
16	DLE	(esc.vínculo datos)
17	DC1	(control disp. 1)
18	DC2	(control disp. 2)
19	DC3	(control disp. 3)
20	DC4	(control disp. 4)
21	NAK	(conf. negativa)
22	SYN	(inactividad sínc)
23	ETB	(fin bloque trans)
24	CAN	(cancelar)
25	EM	(fin del medio)
26	SUB	(sustitución)
27	ESC	(escape)
28	FS	(sep. archivos)
29	GS	(sep. grupos)
30	RS	(sep. registros)
31	US	(sep. unidades)
127	DEL	(suprimir)

Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

ASCII extendido (Página de código 437)

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

-
- Tutoriales W3Schools

Clase 2

Viernes, 15 Septiembre 2023

Indice

Tipos de datos

- Los reales se definen con un 'double'. Tambien es posible usar un float, en caso justificado.
- *'Double'* es un tipo de dato de 64 bits.
  'Float' es un tipo de dato de 32 bits. La GPU trabaja con floats

- Lenguajes y sus generaciones:
- **Primera generación:** lenguaje máquina
  Cada computadora tiene sólo un lenguaje de programación que su procesador puede ejecutar; pues bien, éste es su lenguaje nativo o lenguaje de máquina. Los programas en lenguaje máquina se escriben en el nivel más básico de operación de la computadora. Las instrucciones se codifican como una serie de unos (1) y ceros (0).

- **Segunda generación:** lenguaje ensamblador
  Para evitar que los programadores tuvieran que programar directamente en código binario o máquina, se desarrollaron unos programas para traducir instrucciones a código de máquina. Estos programas se llamaron ensambladores, puesto que leían las instrucciones que las personas podían entender en lenguaje ensamblador y las convertía al lenguaje máquina. El lenguaje ensamblador también es de bajo nivel, ya que cada instrucción de este lenguaje corresponde a una instrucción de lenguaje maquinal.

- **Tercera generación:** lenguaje de alto nivel
  Estos lenguajes son parecidos al inglés y facilitan el trabajo de los desarrolladores de software. Existen muchos lenguajes de tercera generación como, por ejemplo, COBOL, BASIC, FORTRAN, C, PASCAL, etc. Con estos lenguajes, los programadores pueden escribir en una sola instrucción lo equivalente a varias instrucciones complicadas de bajo nivel.

- **Cuarta generación:** lenguaje orientado al usuario (4GL)
  Con los lenguajes 4GL, los usuarios finales escriben sus programas de manera sencilla para consultar una base de datos y para crear sistemas de información personales o departamentales. Muchos de estos lenguajes disponen de una interfaz gráfica y sólo obligan al usuario o programador a usar instrucciones sencillas y fáciles de manejar.

- **Quinta generación:** lenguajes naturales Los lenguajes naturales se asemejan más al lenguaje humano que sus antecesores, los lenguajes 4GL. Aunque estos lenguajes se encuentran en sus inicios, la mayoría de las herramientas de uso y trabajo con el ordenador tenderán a este tipo de lenguajes.

Conocimientos básicos

- **Jerarquía de funciones de clase:**
  Declaracion de variables
  Asignaciones
  Condicionales
  Bucles
  Rupturas
  Retornos
- Declaración de variables:

```
'
int a;
int b;
int a, b;
'
```

- Asignación de variables:

```
'
a = 5;
int b = 12;
'
```

- Condicionales:

```
'
if(condicion)
{
    código ejecutable
}
else
{
    código ejecutable
}
'
```

- Bucles:

```
'
// While, en caso que NO conozcamos el numero de iteraciones

while(condicion)
{
    codigo ejecutable
}

// For, en caso que SI conozcamos el numero de iteraciones

for(int i = 0; i<n; i++)
{
    codigo ejecutable en un bucle de n veces
}
'
```

- Rupturas:

```
'
// Break, hace saltar la linea de compilación hasta el cierre del cuerpo '}'
break; //salta al siguiente '}'

// Continue, hace saltar la linea de compilación hasta el inicio del bucle
continue // salta al inicio del bucle
'
```

- Retornos
  'return', termina el metodo o funcion y devuelve el valor indicado.

- **Jerarquía de funciones de objeto:**
  Enumerations
  Atributos
  Constructores
  Getters and setters
  Metodos

Conocimiento de funciones

- FUNCION: Escribir una función que realiza la suma de dos numeros que se le pasan por parametros:

```
'
public class Functions
{
    public static int CalculateSum(int number1, int number2)
    {
        // Opcion 1
```

```
            return number1 + number2;

            // Opcion 2
            int result;
            result = number1 + number2;
            return result;
        }
    }
```

- Invocar la funcion:
-

```
    '
    public class Program
    {
        public static void Main(args)
        {
            // Declarar variables
            int a, b;

            // Asignar valores
            a = 5;
            b = 10;

            // Invocar la funcion de suma de numeros
            Functions.CalculateSum(a,b);
        }
    }
```

Consejos y advertencias

- Existe una guía de estilo para dar formato al código y para nombrar a las distintas funciones. Cada grupo de trabajo tiene su propia guía de estilo. Habitualmente las funciones se nombran con 'Verbo+Sustantivo' ej:'CalculateSum' o 'GetIndexAt'.
- Cuando dentro de un *cuerpo* (codigo fuente que se escribe entre {}) solo hay una línea de instrucción, es posible eliminar los indicadores{}.
- Una función debe ocupar como máximo una pantalla del ordenador, y debemos evitar el scroll vertical.
- **JAMÁS** ejecutaremos un 'Console.ReadLine()" dentro del funcionamiento de un método.
- **JAMÁS** se debe escribir una sentencia 'return' dentro de un bucle IF

Clase 3

Lunes, 18 Septiembre 2023

Indice

Funciones de clase

- **FUNCION:** Averiguar si un número es mayor que otro.

```
'
public class Function   // nombre de la clase
{
    public static bool IsMajor(int number1, int number2)  // Opción completa
    {
        if(number1 > number2)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    public static bool IsMajor(int number1, int number2)  // Opción mas sencilla
    {
        if(number1 > number2)
            return true;
        return false
    }

    public static bool IsMajor(int number1, int number2)  // Opción perfecta
    {
        return (number1 > number2);
        // Devuelve true si el primer valor es mayor.
        // En caso contrario develve 'false'.
    }
}
'
```

- Los comentarios ayudan a entender el funcionamiento de nuestro código.
  Un **comentario en linea** empieza con '//' y un **comentario en bloque** se define dentro de un '/* */'
- El nombre de una función debe ser un nombre ***descriptivo*** y a la vez fácil de entender. Es muy útil pensar en **'Verbo + Sustantivo'** a la hora de poner nombres.

- **FUNCION:** Devuelve un booleano para comprobar que un numero es par.

```
'
public class Function
{
    public static bool IsEven(int number)
    {
        if(number % 2 == 0)
        {
            return true;
        }
        else
        {
            return false;
        }
    }

    public static bool IsMajor(int number)
    {
        if(number % 2 == 0)
            return true;
        return false
    }

    public static bool IsMajor(int number)
    {
        return (number % 2 == 0);
    }
}
'
```

- Operadores:
  '+ - / *'
  Comparadores:
  '< > ≤ ≥'

Booleanos:
'== != && ||'
Módulo o resto
'%'

- Toda función de clase empieza con un 'public static'
- Si abrimos una llave '{}' hay que cerrarla inmediatamente.
- Una función que devuelve un tipo de dato, siempre se cierra con un 'return'

- **FUNCION:** Devuelve el mayor de tres números

```
'
public class Function
{
    public static bool GetMajor(int number1, int number2, int number3)
    {
        if(number1 > number2 && number1 > number3)
            return number1;
        else if(number2 > number3)
            return number2;
        else
            return number3;
    }
}
'
```

- ***&&***
  Todas las condiciones deben ser ciertas. Puerta lógica **'AND'**
- ||
  Al menos una condición debe ser cierta. Puerta lógica **'OR'**
- Funcionamiento de un **ELSE/IF**
  Añadimos un 'else if' en caso que tengamos varios 'if', pero la condición 'else' se aplique en todos los casos

```
'
public static int Function(int number1, int number2, int number3)
{
    if(number1 > number2 && number1 > number3)
        return number1;
    else if(number2 > number3)
        return number2;
    else
        return number3;
}

// Es lo mismo que el código siguiente:

public static int Function(int number1, int number2, int number3)
{
    if(number1 > number2 && number1 > number3)
    {
        return number1;
    }
    else
    {
        if(number2 > number3)
        {
            return number2;
        }

        else
        {
            return number3;
        }
    }
}
'
```

- Es muy aconsejable utilizar funciones que hemos diseñado con anterioridad. En este caso se podria *llamar* a la función 'IsMajor'.

Clase 4

Martes, 19 Septiembre 2023

Indice

Continuación de estudio Funciones de clase

- Función Sumatorio: Calcula el sumatorio de un número

```
'
public class Functions
{
    public static bool GetSummatory(int number)
    {
        int count = 1;
        int result = 0;
        while (count &le number)
        {
            result += count;
            count ++;
        }
        return result;
    }
}
'
```

- Función Serie: Imprime por consola una serie de números pares positivos y negativos.

```
'
public class Functions
{
    public static void CalculateSerie(int number)
    {
        int result = 0;
        Console.WriteLine(result);

        for(int i = 1; i < number/2; i++)
        {
            if(IsEven(i))
            {
                result = result * 1 * -2;
            }
            else
            {
                result = result * 1 * 2;
            }
            Console.WriteLine(result);
        }
    }
}
'
```

Notas Importantes

- Un bucle **WHILE** se utiliza cuando NO conocemos el número exacto de iteraciones.

```
while(condicion)
{
    sentencia;
}
```

Un bucle **FOR** se utiliza cuando SI conocemos el número de iteraciones.

```
if(inicio de la iteracion; condicion que debe cumplirse; final de la iteración)
{
    sentencia;
}
```

- Un **'Code Snippet'** es un trozo de código que funciona por sí mismo y ejecuta ciertas tareas, pero sí mismo no se considera función.
- es **MUY IMPORTANTE** utilizar todas las funciones que hemos escrito en nuestro código. Las funciones que se escriben dentro del Programa pueden ser utilizadas en cualquier localización de nuestra solución. Es buena praxis llamar a otras funciones que **SIMPLIFICAN** nuestro codigo fuente.

Clase 5

Jueves, 21 Septiembre 2023

Indice

Continuación de estudio Funciones de clase

- Funciones Ejemplo: Devolver 'true' si un número es primo.

```
'
public class Functions
{
    public static bool IsPrime(int number)
    {
        for(int i = 2; i < number; i++)
        {
            if(number % i == 0)
                return false;
        }
        return true;
    }
}
'
```

Notas Importantes

- Es importante recordar que NUNCA debemos introducir dentro de una condicional for, una sentencia return. No tiene sentido devolver una variable si ANTES NO HEMOS acabado todas las iteraciones del bucle FOR

Clase 6

Viernes, 22 Septiembre 2023

Indice

Continuación de estudio Funciones de clase

- Funciones Ejemplo: Imprimir por pantalla la Serie Collatz

```
'
public class Functions
{
    public static void Collatz(int number)
    {
        int result = number;
        Console.WriteLine(result);

        while(result != 1)
        {
            if(IsEven(result))
                result /= 2;
            else
                result = (result * 3) + 1;

            Console.WriteLine(result);
        }
    }
}
'
```

Funciones ejemplos

- 1.- Devolver el menor de 2 números.
- 2.- Devolver verdadero si un número es par.
- 3.- Devolver el menor de 3 números.
- 4.- Imprimir la siguiente serie por consola: 0, -2, 4, -6, 8, -10.
- 5.- Devolver verdadero si un número es primo.
- 6.- Imprimir la serie de COLLATZ por pantalla.
- 7.- Función Sumatorio.
- 8.- Función Productorio.

Clase 7

Lunes, 25 Septiembre 2023

Indice

Funciones de objeto

- Funciones de Objeto de la clase Dolphin

```
public class Dolphin
{
    public double life;

    public double GetLife()
    {
        return life/10;
    }
}

public class Main
{
    Dolphin d1 = new Dolphin();
    double resultLife = d1.GetLife();
}
```

- Las funciones de objeto explican el comportamiento de las instancias de una clase.
- Se diseñan en el interior de la clase. Se caracterizan por no llevar 'static'. Igualmente es raro que necesiten parámetros para funcionar.
- Para invocarlas, debemos utilizar la notación por punto: d1.GetLife(). Esto significa que la función es invocada por el objeto 'd1'
- Podemos almacenar los valores resultantes de las funciones en variables que creamos a propósito en la clase Main de nuestro programa.

Ejemplos de porcentajes

- El porcentaje de un número sigue la siguiente regla:
  Parte / Todo * (tipo de tanto)
- Para aumentar un número un 20 por ciento utilizamos la siguiente fórmula:
  número * (1 + 20 / 100)
- Para disminuir un número un 20 por ciento utilizamos la siguiente fórmula:
  número * (1 - 20 / 100)

Clase tipo

-

```
// FUNCIONES DE CLASE

public class Game
{
    public static
}

// FUNCIONES DE OBJETO

public class Student
{
    public string name;
}
```

- Funciones de Clase
  Clase = Funciones = static = retorno = void = Categoria = Enseñanzas
- Funciones de Objeto
  Objeto = (no)static = (sin)retorno = Instancias = Objetos = registro = parametros

Clase 8

Martes, 26 Septiembre 2023

Indice

Getters

- Funciones creadas explicitamente para 'recoger' o conseguir el valor de un atributo. Esto sucede porque es muy posible que otros programadores quieran utilizar los valores que hemos creado en nuestras funciones, a las cuales habitualmente no tendran acceso directo, sino a través de los getters.

```
'
public class Dolphin
{
    private double life;

    public double GetLife()
    {
        return life;
    }
}
'
```

Habitualmente *no reciben ningún parámetro y siempre devuelven un valor*

Setters

- Funciones creadas explicitamente para 'establecer' el valor de un atributo Se debe realizar la *validación* del parámetro, para que realice un correcto funcionamiento de nuestro programa.

```
'
public class Dolphin
{
    private double life;

    public void SetLife(double value)
    {
        this.life = value;
    }
}
'
```

Habitualmente *no devuelven nada y siempre reciben un parámetro*

- La validación de los setters puede implicar tres sistemas diferentes:
  1.- "Clampear" o "Saturar" los parametros de entrada. Esto significa llevar los parámetros a los niveles máximos y mínimos(Clampear) o simplemente a los niveles máximos (Saturar).
  2.- Comprobar que los valores son correctos y en este caso, el programa funciona normalmente
  3.- Darse cuenta que los valores son incorrectos y en este caso, lanzar un error de advertencia.

1.- "Clampear" o "Saturar" los parametros de entrada.

```
'
public class Dolphin
{
    private double life;

    public void SetLife(double value)
    {
        if (value < 0)
            this.life = 0;
        else if ( value > 100)
            this.life = 100;
        else
            this.life = value;
```

```
    }
}
'
```

2.- Comprobar que los valores son correctos y en este caso, el programa funciona normalmente

```
'
public class Dolphin
{
    private double life;

    public void SetLife(double value)
    {
        if (value > 0 && value < 100)
            this.life = value;
        // else el programa no realiza ninguna acción
    }
}
'
```

3.- Darse cuenta que los valores son incorrectos y en este caso, lanzar un error de advertencia.

```
'
public class Dolphin
{
    private double life;

    public void SetLife(double value)
    {
        if (value < 0 || value > 100)
            throw new Exception("Error de validacion de parametros");
        this.life = value
    }
}
'
```

Clase 9

Jueves, 28 Septiembre 2023

Indice

Concatenación de strings

- Ejemplo 1: Función que devuelve la concatenación de dos strings.

```
'
public static string Concat(string a, string b)
{
    return a + b;
}
'
```

- Ejemplo 2: Función que devuelve una serie de números.

```
'
public static string Concat(int number)
{
    string result = "0";
    for(int i = 0; i < number; i++)
    {
        result += "," + i;
    }
    return result;
}
'
```

- Ejemplo 3: Función que devuelve una serie de números.

```
'
public static string Concat(int number)
{
    string result = "0";
    int multiplicador = 1;
    for(int i = 0; i < number; i++)
    {
        multiplicador *= 2;
        result += "," + multiplicador;
    }
    return result;
}
'
```

Clase 10

Viernes, 29 Septiembre 2023

Indice

Serie de Fibonacci

- Funcion que se le pasa un número y devuelve ese número de elementos de la serie Fibonacci

```
'
public static string Fibonacci1(int number)
{
    int result = "0,1";
                                    int n1 = 0;
                                    int n2 = 1;
                                    int sumResult = 0;

                                    for(int i = 0; i < number - 1; i++)
                                    {
                                            sumResult = n1 + n2;
                                            result += "," + sumResult;
                                            n1 = n2;
                                            n2 = sumResult;
                                    }
                                    return result;
}
'
```

- Funcion que se le pasa un número y devuelve el número posterior de la serie Fibonacci

```
'
public static string Fibonacci2(int number)
{
    int result = "0,1";
                                    int n1 = 0;
                                    int n2 = 1;
                                    int sumResult = 0;

                                    while(number < sumResult)
                                    {
                                            sumResult = n1 + n2;
                                            result += "," + sumResult;
                                            n1 = n2;
                                            n2 = sumResult;
                                    }
                                    return result;
}
'
```

Constructores

- Funciones creadas explicitamente para 'establecer' los atributos del objeto que estamos creando.

```
'
public Dolphin (double actualLife, double maxValueLife)
{
    life = actualLife;
    maxLife = maxValueLife
}
'
```

- Es una función porque recibe la llamada de los parámetros "( )" y además tiene un cuerpo de función "{ }"
- La función del constructor tiene el **mismo nombre** que el nombre de la clase
- Esta función no devuelve ningún valor
- Ejemplo de uso

```
'
public static void Main()
{
    Dolphin d1;
    d1 = new Dolphin(100.0, 250.0);
}
'
```

Clase 11

Lunes, 2 Octubre 2023

Indice

Contenedores

- Los contenedores son colecciones de elementos. Pueden ser de cualquiera de os tipos elementales.
- Los tipos de contenedores son: Arrays, Listas, Diccionarios, Árboles
- Es importante recordar que los **contenedores se almacena en una variable que apunta a una direccion de memoria,** donde existe un contenedor que se ha creado mediante la funcion new.

Listas

- Ejemplo de lista

```
'
public static void Main()
{
    List l;            // se crea una variable llamada 'l' de tipo Lista de enteros
    l = new List();    // se crea la lista en una direccion de la memoria
    l = null;               // l apunta a la nada, se elimina la lista del programa
}
'
```

- Metodos que hacen funcionar una lista

```
'
public static void Main()
{
    Add(element)     // añade 'element' al final de la lista

    l.Add(40);
    l.Add(-10);
    l.Add(3);
                            // [40,-10,3]

    Remove(element)  // elimina 'element' al final de la lista
    RemoveAt(index)  // elimina 'element' en la posicion 'index'

    l.RemoveAt(2);
                            // [40,-10]

    l[i] = value     // Actualiza el valor del elemento 'i' de una lista

    l[0] = 60;
    int i = 1;
    l[i + 0] = 3;
                            // [60,3]

    l.Count          // Hace el conteo del número de elementos que hay en una lista

    int n = l.Count;
                            // n = 2

    l.Insert(index, element)   // Inserta 'element' en la posicion 'index'

    l.Insert(1, -20);
                            // [60,-20,3]

    l[1] = l[2]      // el elemento de la posicion 1 es igual al elemento de la posicion 2

                            // [60,3,3]

    l.Clear()        // Se eliminan todos los elementos de la lista. Vaciar la lista

                            // [0]

}
'
```

Arrays

- Colecciones de elementos con la característica que no permite cambiar el tamaño del array original

```
'
public static void Main()
{
    int [] a;               // Se crea la variable 'a' que es del tipo array de int
    a = new int[4];         // Se crea un array en una posición de la memoria donde apunta el puntero 'a'
                            // Es Imprescindible especificar el número de celdas que contiene
                            // el array. Este número no se puede cambiar
    a
                            // [0,0,0,0]
    a[3] = -10;
    a[0] = a[3];
```

```
                                // [-10,0,0,-10]

    int n = a.Length;
                                // n = 4
    int [] b = a;
                                // b = [-10,0,0,-10]
                                // b, del tipo array de int, apunta a la misma dirección que a.
}
'
```

Funciones de ejemplo

- Funcion 1: Se le pasa una lista de strings y devuelve el número de elementos que hay en su interior.

```
'
public class ListExample
{
    public static int GetListItems(List<string> list)
    {
        return list.Count;
    }
}
'
```

- Funcion 2: Se le pasa una lista de dobles y devuelve el número de elementos que son positivos.

```
'
public class ListExample
{
    public static int GetPositiveListItems(List<double> list)
    {
        int result = 0;
        for(int i = 0; i < list.Count; i++)
        {
            if(list[i] > 0)
                result++;
        }
        return result;
    }
}
'
```

- Funcion 3: Se le pasa un array de dobles y devuelve el número de elementos que son positivos.

```
'
public class ListExample
{
    public static int GetPositiveListItems(double[] list)
    {
        int result = 0;
        for(int i = 0; i < list.Length; i++)
        {
            if(list[i] > 0)
                result++;
        }
        return result;
    }
}
'
```

Clase 12

Martes, 3 Octubre 2023

Indice

Inicializadores de Listas

- Podemos declarar e inicializar las Listas/Arrays de diferentes maneras:
-

```
1.-
'
public static void Main()
{
    List<int> list;
    list = new List<int>();

    list.Add(1);
    list.Add(5);
    list.Add(10);
}
'
```

-

```
2.-
'
public static void Main()
{
    List<int> list = new List<int>();

    list.Add(1);
    list.Add(5);
    list.Add(10);
}
'
```

-

```
3.-
'
public static void Main()
{
    List<int> list = new List<int>{1, 5, 10};
}
'
```

-

```
4.-
'
public static void Main()
{
    List<int> list = new List<int>
    {
        1,
        5,
        10
    };
}
'
```

Inicializadores de Arrays

-

```
1.-
'
public static void Main()
{
    int[] list;
    list = new int[3];
```

```
    list[0] = 1;
    list[1] = 5;
    list[2] = 10;
}
'
```

- ```
  2.-
  '
  public static void Main()
  {
      int[] list = new int[3];

      list[0] = 1;
      list[1] = 5;
      list[2] = 10;
  }
  '
  ```

- ```
  3-
  '
  public static void Main()
  {
      int[] list = new int[]{1, 5, 10};
  }
  '
  ```

- ```
  4.-
  '
  public static void Main()
  {
      int[] list = new int[]
      {
      1,
      5,
      10
      };
  }
  '
  ```

Funciones que se utilizan habitualmente en Listas y Arrays

- **Función 1:** Crear una función que se le pase una lista de enteros y un valor. Devuelve verdadero o falso si el valor está dentro de la lista.
- Es buena idea utilizar el nombre de *'Contains'* para evaluar si una lista contiene un elemento.
- ```
  '
  public class Functions
  {
      public static bool ContainsNumber(List<int> list, int number)
      {
          if( list == null || list.Count == 0)
              return false;

          for(int i = 0; i < list.Count; i++)
          {
              if (list[i] == number)
                  return true;
          }
          return false;
      }
  }
  '
  ```

- Es aconsejable **validar los parámetros de entrada** de la función. Hay que comprobar que la lista no apunte a null y que no esté vacía.

- Queda reservado el uso de la función Remove(). Será motivo de suspenso su uso.
- Podemos hacer 'folding' con snippets de código siempre que queramos simplificar el código escrito. Para ello empleamos #region y #endregion

- **Funcion 2:** Se le pasa una lista de enteros y te devuelve el valor mayor.

```
'
public class ListExample
{
    public static int GetMajor(List<int> list)
    {
        if( list == null || list.Count == 0)
            return int.MinValue;

        int result = list[0];
        for(int i = 0; i < list.Count; i++)
        {
            if(list[i] > result)
                result = list[i];
        }
        return result;
    }
}
'
```

- **Funcion 3:** Función que devuelve la posición del valor mayor.

```
'
public class ListExample
{
    public static int GetMajor(List<int> list)
    {
        if( list == null || list.Count == 0)
            return -1;

        int aux = list[0];
        int index = 0;
        for(int i = 0; i <& list.Count; i++)
        {
            if(list[i] > aux)
            {
                index = i;
                aux = list[i];
            }
        }
        return index;
    }
}
'
```

- Los valores de índice se establecen por convención como *index*.
- Podemos llamar a la función anterior para buscar el valor mayor, pero tendríamos el problema de recorrer dos *for* distintos, con el doble de carga para el programa. Es preferible crear un único for que resuelva este problema.

- **Funcion 4:** Función que devuelve verdadero o falso si una lista está ordenada

```
'
public class ListExample
{
    public static bool IsOrdered(List<int> list)
    {
        if (list == null || list.Count == 0)
            return false;

        bool result = true;

        for(int i = 0; i < list.Count - 1; i++)
        {
```

```
            if (list[i] > list[i + 1])
                result = false;
        }
        if(list[list.Count - 1] > list[list.Count])
            result = false;
        return result;
    }
}
'
```

- **Funcion 5:** Función que ordena los valores de una lista de manera *ascendente*.

```
'
public class ListExample
{
    public static List<int> Sort(List<int> list)
    {
        if (list == null || list.Count == 0)
            throw new Exception("Error de validación de parámetros");

        List <int> result = new List <int>();
        int aux;

        for(int i = 0; i < list.Count - 1; i++)
        {
            for(int j = list.Count - 1; j > 0; j--)
            {
                if (list[j] < list[j - 1])
                    {
                        aux = list[j - 1];
                        list[j - 1] = list[j];
                        list[j] = aux;
                    }
            }
        list.Add(list[i]);
        }
        return result;
    }
}
'
```

Clase 13

Viernes, 6 Octubre 2023

Indice

Funciones para listas/arrays

- Crear una función que calcule la media de los valores de un array
-
```
'
public static double GetMedian(double [] array)
{
    double aux = 0.0;

    for(int i = 0; i < array.Length; i++)
    {
        aux += array[i];
    }

    return aux / array.Length;
}
'
```

- Es MUY IMPORTANTE validar los parámetros de entrada de la función

- **FUNCION:** Crear una función que calcule la media de los valores de un array, solo si superan el valor dado llamado 'Threshold'
-
```
'
public static double GetThresholdMedian(double [] array, double Threshold)
{
    double aux = 0.0;
    int count = 0;


    for(int i = 0; i < array.Length; i++)
    {
        if(array[i] > Threshold)
        {
            aux += array[i];
            count ++;
        }
    }
    return aux / count;
}
'
```

- **FUNCION:** Crear una función que devuelva el número de veces que se repite el número de mayor valor.
-
```
'
public static int GetMaxNumberRepeated(List<int> list)
{
    int result = 0;
    int maxNumber;
    maxNumber = GetMajor(list);

    /*
    for(int i = 0; i < list.Count; i++)
    {
        if(list[i] > maxNumber)
            maxNumber = list[i];
    }
    */

    for (int i = 0; i < list.Count; i++)
    {
        if (list[i] == maxNumber)
            result++;
    }
    return result;
}
```

```
'
```

- **FUNCION:** Crear una función que devuelva la lista que se pasa por parametro en orden inverso.
-

```
'
public static List GetReverseList(List<int%gt list)
    {
        if (list == null || list.Count < 0)
            throw new Exception("Error");

        List result = new List();

        for (int i = list.Count; i &ge 0; i--)
        {
            result.Add(i);
        }
        return result;
    }
'
```

Clase 14

Viernes, 6 Octubre 2023

Indice

Funciones para listas/arrays

- **FUNCIÓN:**Crear una función que calcule la media de los valores de un array
-
```
'
public static double GetMedian(double [] array)
{
    double aux = 0.0;

    for(int i = 0; i<array.Length; i++)
    {
        aux += array[i];
    }

    return aux / array.Length;
}
'
```

- Es MUY IMPORTANTE validar los parámetros de entrada de la función

- **FUNCION:** Crear una función que calcule la media de los valores de un array, solo si superan el valor dado llamado 'Threshold'
-
```
'
public static double GetThresholdMedian(double[] array, double Threshold)
{
    double aux = 0.0;
    int count = 0;

    for(int i = 0; i<array.Length; i++)
    {
        if(array[i] > Threshold)
        {
            aux += array[i];
            count ++;
        }
    }
    return aux / count;
}
'
```

- **FUNCION:** Crear una función que devuelva el número de veces que se repite el número de mayor valor.
-
```
'
public static int GetMaxNumberRepeated(List<int> list)
{
    int result = 0;
    int maxNumber;
    maxNumber = GetMajor(list);

    for (int i = 0; i<list.Count; i++)
    {
        if (list[i] == maxNumber)
            result++;
    }
    return result;
}
'
```

- **FUNCION:** Crear una función que devuelva la lista que se pasa por parametro en orden inverso.
-
```
'
```

```
public static List GetReverseList(List<int%gt list)
    {
        if (list == null || list.Count < 0)
            throw new Exception("Error");

        List result = new List();

        for (int i = list.Count; i &ge 0; i--)
        {
            result.Add(i);
        }
        return result;
    }
'
```

- **FUNCION:** Crear una función que imprima los valores de una lista de enteros en pantalla
-

```
'
public static void PrintList(List<int> list)
    {
        for(int i = 0; i <list.Count; i++)
        {
            Console.WriteLine(list[i]);
        }
    }
'
```

- **FUNCION:** Crear una función que imprima los valores de un array de enteros en pantalla
-

```
'
public static void PrintList(int[] array)
    {
        for(int i = 0; i <array.Length; i++)
        {
            Console.WriteLine(array[i]);
        }
    }
'
```

- **FUNCION:** Crear una función que añada un valor dado por parametros al final de un array de enteros.
-

```
'
public static void AddValueToArray(int[] array, int number)
    {
        if(array == null || array.Length == 0)
            throw new Exception("Error");

        int[] result = new int[array.Length + 1];

        for(int i = 0; i <array.Length; i++)
        {
            result[i] = array[i];
        }

        result[array.Length] = number;
    }
'
```

Clase 15

Martes, 10 Octubre 2023

Indice

Sugerencias y Advertencias

- Si mostramos una informacion por pantalla, hay dos opciones:
  Devolver 'void' o devolver 'string'. Es mejor la última opción.
- Si tenemos que mostrar informacióno por pantalla y se trata de una serie, lo mejor es devolver una Lista.
- En caso que tengamos una función 'void' y la validación sea negativa, podemos simplemente devolver 'return'.

```
'
public static void Function(List<int> list)
{
    if (list == null)
        return ;

}
'
```

- 'Console.WriteLine(string text)'

```
// Escribe cada sentencia en lineas distintas
text1
text2
text3
```

'Console.Write(string text)'

```
// Escribe cada sentencia en la misma linea
text1text2text3
```

'Console.BackgroundColor()'
'Console.ForegroundColor()'
Cambia el color de la fuente y del fondo de la consola.

- **Es obligatorio validar todos los parámetros de entrada de una función.**
- No es necesario validar que un array o lista entren con 0 elementos. Esto solo es necesario en caso que queramos acceder al elemento[0]

```
'
public static int[] Function(int[] array)
{
    // Caso normal
    if(array == null)
        return null;

    // Caso excepcional, si queremos acceder al elemento [0]
    if(array == null || array.Length == 0)
        return null;
}
'
```

- Las variables de una función STATIC deben declararse **DENTRO** de la propia función.
- No se deben poner contadores **DENTRO** de un bucle FOR
- Siempre validamos las variables que implican una creación de objeto mediante NEW

Funcion *SWAP*

-

```
'
public static void Swap(List<int> list)
{
    int aux;
    aux = list[i];
    list[i] = list[j];
    list[j] = aux;
```

```
}
'
```

Funcion *SORT*

- ```
  '
  public static void Sort(List<int> list)
  {
      if (list == null)
          return null;

      int aux;
      int n1 = list.Count - 1;
      int n2 = list.Count;

      for(int i = 0; i<n1; i++)
      {
          for(int j = i + 1; j<n2; j++)
          {
              aux = l[i];
              l[i] = l[j];
              l[j] = aux;
          }
      }
  }
  '
  ```
  ```

Clase 16

Miercoles, 11 Octubre 2023

Indice

Clase 17

Viernes, 13 Octubre 2023

Indice

Funciones complejas de contenedores

- **FUNCION** le paso una lista de enteros y una posicion *'index'*. La funcion elimina el valor que se encuentra en la posicion 'index'

```
'
public static void RemoveElement(List<int> list, int index)
{
    list.RemoveAt(index);
}

// Elimina el elemento que esta en la posicion 'index'

public static void RemoveElement(List<int> list, int index)
{
    list.Remove(index);
}
'
// Elimina el elemento cuyo valor es 'index'
```

- La función **'Remove()'** esta **PROHIBIDA**. Siempre usaremos **'RemoveAt()**
- **Funcion** que le paso una lista y un valor que quiero borrar.

```
'
public static void RemoveValue(List<int> list, int value)
{
    if (list == null)
        return;

    for (int i = 0; i < list.Count; i++)
    {
        if (list[i] == value)
        {
            list.RemoveAt(i);
            i--;
        }
    }
}
'
```

- Si le introduzco un punto de ruptura **'break'** inmediatamente sale del bucle donde está anidado.
  Un 'IF' no es un bucle, solo 'WHILE' Y 'FOR'
- **Funcion** con un punto de ruptura 'BREAK'

```
'
public static void RemoveValue(List<int> list, int value)
{
    for(int i = 0; i < list.Count; i++)
    {
        list.RemoveAt(i);
        i--;
        break;          // Rompe el bucle anidado 'FOR'
    }
}
'
```

- IMPORTANTE: Si utilizamos i++ ó i-- como una **expresion** su funcionamiento es distinto al habitual:
  i++
  Primero la variable asignada es **IGUAL** a 'i' y cuando se evalua toda la expresión se **INCREMENTA** la 'i'.
  ++i
  Primero la variable asignada es el **INCREMENTO** de 'i' y cuando se evalua toda la expresión se **INCREMENTA** la 'i'.

- **FUNCION:**Se le pasa una lista y queremos borrar de esta lista otra lista de valores que se le pasan.
-
```
'
public static void RemoveValues(List<int> list, List<int> listValues)
{
    for(int j = 0; j < listValues; j++)
    {
        for(int i = 0; i < list; i++)
        {
            if(list[i] == list[j])
                list.RemoveAt(i--);
        }
    }
}

public static void RemoveValues(List<int> list, List<int> listValues)
{
    for(int i = 0; i < listValues; i++)
    {
        for(int j = 0; j < list; j++)
        {
            if(list[i] == list[j])
                list.RemoveAt(i--);
        }
    }
}
'
```

Consejos y Sugerencias

- **CONSEJO:** No es adecuado contener un 'FOR' dentro de otro 'FOR'. Es mejor solución llamar a las funciones desde el interior del primer 'FOR'
- Si trabajas con 'Listas' es mejor no devolver nada. Por otro lado si trabajas con 'Arrays' lo normal es devolver un 'Array'
- **WARNING:** Cuando declaramos un tipo de dato, el compilador no espera que pueda devolver 'null'.
  Si añadimos **?** al final del tipo de dato, le indicamos al compilador que este datos **SI PUEDE SER NULL**.
  int[]? --> PUEDE SER NULL
  int[] --> NO DEBERÍA SER NULL