

Resumen 1

Parte 1.

1.1 Tipos de datos

Tipo de dato	Tamaño	Rango de valores
byte	1 byte	0 a 255
char	2 byte	U+0000 a U+ffff (caracteres Unicode)
short	2 bytes	-32,768 a 32,767
int	4 bytes	-2,147,483,648 a 2,147,483,647
long	8 bytes	-9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
float	4 bytes	$\pm 1.5e-45$ a $\pm 3.4e38$
double	8 bytes	$\pm 5.0e-324$ a $\pm 1.7e308$
bool	2 bytes	Verdadero o falso
string	-	Cero o más caracteres Unicode

Declaración

String **nombre** = **"Juan Carlos"**; // variable de tipo String

The diagram illustrates the components of the variable declaration `String nombre = "Juan Carlos"; // variable de tipo String`. Four colored arrows point from labels below to specific parts of the code: a red arrow points from **Tipo de Dato** to `String`; a green arrow points from **Nombre de la Variable** to `nombre`; an orange arrow points from **Valor Inicial** to `"Juan Carlos"`; and a yellow-green arrow points from **Comentario** to `// variable de tipo String`.

Tipo de Dato

Nombre de la Variable

Valor Inicial

Comentario

1.2 métodos de variables string

```
int numero = 10;
string mensaje = "hola mundo";

Console.WriteLine(mensaje.Length); //tamaño de espacios del string
Console.WriteLine(numero.ToString()); // convertir a string
Console.WriteLine(numero.Equals(10)); // me devuelve un booleano si es true o false
Console.WriteLine(mensaje.Equals("hola mundo")); // me devuelve un bool se es true o false
Console.WriteLine(mensaje.StartsWith('h')); //comienza con h
Console.WriteLine(mensaje.Remove(7)); //remueve de posición 7 en adelante
Console.WriteLine(mensaje.Replace('o','a')); //reemplaza la o por la a

Console.Read();
```

1.3 conversión de string a int

```
string numero1 = "5";
int numero2 = 7;
int resultado = Int32.Parse(numero1) + numero2;
Console.WriteLine("el resultado es: {0}", resultado); //me convierte el string en int
int res;
bool boolean = Int32.TryParse(numero1, out res); // convierte el string y regresa false o true
Console.WriteLine("se puede convertir {0}", boolean);
```

1.4 convertir de int a string

```
int numero1 = 7;
int numero2 = 77;
string resultado;

Console.WriteLine(numero1.ToString() + numero2.ToString()); // convierte los int en string
//resultado 777
```

1.5 declaración de constantes

```
class Program
{
    const double pi = 3.145278;
    const int meses = 12;

    0 references
    static void Main(string[] args)
    {
    }
```

Parte 2

2.1 métodos

Método sin retorno y sin parámetros

```
0 reference
static void Main(string[] args)
{
    sin_retor_sin_par();
}

//metodo sin retorno y sin parametros
1 reference
public static void sin_retor_sin_par()
{
    Console.WriteLine("metodo sin retorno y sin parametros ");
}
```

Método sin retorno con parámetros

```
static void Main(string[] args)
{
    sin_retor_con_par(7,7);
}

//metodo sin retorno y sin parametros
1 reference
public static void sin_retor_con_par(int a, int b)
{
    Console.WriteLine("metodo sin retorno y con parametros {0}",(a +b));
}
```

Métodos con retorno y sin parámetros

```
static void Main(string[] args)
{
    int resultado = con_retor_sin_par();
    Console.WriteLine("el resultado es {0}", resultado);
}

//metodo sin retorno y sin parametros
1 reference
public static int con_retor_sin_par()
{
    int a = 7;
    int b = 7;
    int r = a + b;
    return r;
}
```

Método con retorno y con parámetros

```
0 references
static void Main(string[] args)
{
    int resultado = con_retor_con_par(7,7);
    Console.WriteLine("el resultado es {0}", resultado);
}

//metodo sin retorno y sin parametros
1 reference
public static int con_retor_con_par(int a, int b)
{
    int r = a + b;
    return r;
}
```

2.2 try catch

```
Console.WriteLine("ingrese un numero");
string numero = Console.ReadLine();

try
{
    int numero2 = Int32.Parse(numero);
}
catch (FormatException)
{
    Console.WriteLine("el formato es incorrecto");
}
catch (OverflowException)
{
    Console.WriteLine("excedio el numero de que puede contener");
}
catch (ArgumentNullException)
{
    Console.WriteLine("el valor ingresado fue nulo");
}
catch (Exception)//agarra todos los errores posibles
{
    Console.WriteLine("ocurrio un error y no se pudo especificar");
}
finally
{
    Console.WriteLine("lo que esta aqui se va aplicar sea no no que falle ");
}
```

2.3 operadores

Categoría	Operadores
Aritméticos	+ - * / %
Lógicos	! &&
A nivel de bits	& ^ ~
Concatenación	+
Incremento, decremento	++ --
Desplazamiento	<< >>
Relacional	== != < > <= >=
Asignación	= ^= <<= >>=
Acceso a miembro	.
Indexación	[]
Conversión	()
Condicional	? : ??
Creación de objeto	new
Información de tipo	as is sizeof typeof

3toma de decisiones

3.1 if

```
int numero = 11;
if (numero > 10)
{
    Console.WriteLine("el numero es mayor a 10");
}
else if (numero == 10)
{
    Console.WriteLine("el numero es igual a 10");
}
else
{
    Console.WriteLine("el numero es menor a 10");
}
```

3.2 switch

```
int mes = 1;
switch (mes)
{
    case 1:
        Console.WriteLine("enero");
        break;
    case 2 :
        Console.WriteLine("febrero");
        break;
}
Console.ReadLine();
```

3.2 if mejorado

```
int temperatura = -17;
string estadoAgua;

estadoAgua = temperatura > 10 ? "gas" : temperatura > 0 ? "liquido" : "solido";
Console.WriteLine("estado del agua es: {0}", estadoAgua);
```

4ciclos

4.1For

```
//repetir el mensaje 3 veces sumando 10
int num = 0;
for (int i = 0; i < 3; i++)
{
    Console.WriteLine("el numero es: {0}", num);
    num = num + 10;
}
```

4.2 Do while

```
//do while para menu
bool salir = false;
string respuesta;
bool opcion_incorrecta = false;
do
{
    if (opcion_incorrecta == false)
    {
        Console.WriteLine("aplico codigo");
    }
    Console.WriteLine("desea salir si o no");
    respuesta = Console.ReadLine();
    if (respuesta.Equals("si"))
    {
        salir = true;
    }
    else if (!respuesta.Equals("no"))
    {
        salir = false;
        opcion_incorrecta = true;
        Console.WriteLine("opcion incorrecta");
    }
    else
    {
        opcion_incorrecta = false;
    }
} while (salir == false);
```

4.3 While

```
//contador de numeros
Console.WriteLine("escriba un numero:");
int numero = int.Parse(Console.ReadLine());
int x = 0;
while ( x < numero)
{
    Console.WriteLine("numero {0}" , x+1);
    x++;
}
```

5 programación orientada a objetos

Miembros de Clases:

Variables públicas y privadas

Propiedades públicas o privadas

Constructores

Métodos públicos o privados

5.1 Clases: tiene propiedades, variables nombre y métodos presentarse

```
0 references
static void Main(string[] args)
{
    humano hum = new humano();
    hum.nombre = "allan";
    hum.presentarse();
}
```



```

class humano
{
    public string nombre;

    1 reference
    public void presentarse()
    {
        Console.WriteLine("hola soy {0}", nombre);
    }
}

```

5.2 constructores: sirven para inicializar las variables

```

0 references
static void Main(string[] args)
{
    humano hum = new humano("allan");
    humano hum2 = new humano("diego");
    hum.presentarse();
    hum2.presentarse();
}

```

```

0 references
class humano
{
    public string nombre;

    2 references
    public humano(string nombre)
    {
        this.nombre = nombre;
    }

    2 references
    public void presentarse()
    {
        Console.WriteLine("hola soy {0}", nombre);
    }
}

```

5.3 sobrecarga de constructores se llaman igual, pero tienen diferente número o tipo de parámetros de entrada

```
0 references
static void Main(string[] args)
{
    humano hum = new humano("allan");
    humano hum2 = new humano();
    hum.presentarse();
    hum2.presentarse();
}
}
```

```

class humano
{
    public string nombre;
    1 reference
    public humano()
    {
        nombre = "leiton";    }
    1 reference
    public humano(string nombre)
    {
        this.nombre = nombre;
    }
    2 references
    public void presentarse()
    {
        Console.WriteLine("hola soy {0}", nombre);
    }
}

```

5.4 modificadores de acceso



Modificadores de Acceso

Modificador	Comentario
<code>public</code>	Sin restricciones. Visible desde cualquier método de cualquier clase
<code>private</code>	Sólo es visible desde los métodos de la misma clase
<code>protected</code>	Visible desde los métodos de una clase y todas sus sub-clases
<code>internal</code>	Visible desde los métodos de todas las clases dentro de un mismo ensamblaje
<code>protected internal</code>	Visible desde los métodos de una clase, sus sub-clases, y todas las clases en su mismo ensamblaje

OALP-2004 All Rights Reserved

5.5 propiedades de una clase que sirven para que las variables de una clase sean privadas y puedan ser modificadas o obtenidas solo por set y get

```
0 references
static void Main(string[] args)
{
    humano hum2 = new humano();
    string nombre = hum2.Nombre; // llama al metodo get
    Console.WriteLine("el nombre es : {0}", nombre);
    hum2.Nombre = "allan"; //ingresa el nombre set solo si el nombre es allan
    hum2.presentarse();
}
```

```

class humano
{
    //variable privada
    private string nombre;
    //propiedad de esa variable
    // public string Nombre { get; set; } asi se crea propiedad sin restriccion
    2 references
    public string Nombre //propiedad nombre con restriccion en el set solo sea allan
    {
        get { return nombre; }
        set {
            if (value.Equals("allan"))
            {
                nombre = value;
            }
            else {
                Console.WriteLine("no se pudo ingresar ese valor ");
            }
        }
    }
    1 reference
    public humano()
    {
        nombre = "leiton";
    }

    1 reference
    public void presentarse()
    {
        Console.WriteLine("hola soy {0}", nombre);
    }
}

```

5.xAcceder a métodos privados

```

0 references
static void Main(string[] args)
{
    humano hum = new humano();
    hum.validar_aceder_inf(true);
}

```

```

2 references
class humano
{
    private int salario = 500;

    1 reference
    private void informacionPrivada()
    {
        Console.WriteLine("su salario es:{0}", salario);
    }

    1 reference
    public void validar_aceder_inf(bool accede)
    {
        if (accede)
        {
            informacionPrivada();
        }
        else
        {
            Console.WriteLine("no tiene permisos!!!");
        }
    }
}

```

7 arrays

7.1 declaración de array y uso de length

```

string[] nombres = new string[7]; // forma de declarar 1
nombres[0] = "allan";
string[] nombres2 = new string[] { "allan2" }; // forma de declarar 2
string[] nombres3 = { "allan3" }; // forma de declarar 3

Console.WriteLine("el tamaño de array nombres es " + nombres.Length); // métodos de propiedad de
array

```

7.2 uso de foreach: el cual recorre un array uno x uno

```
string[] nombre = new string[] { "allan", "leiton", "vargas" }; // forma de declarar

foreach (string nom in nombre)
{
    Console.WriteLine( nom );
}
```

7.2 vectores o arrays multidimensionales

Ejemplo 1 de declaración array

```
string[,] nombre = new string[,]; // primero fila y despues columna
{
    // columna 0   columna 1
/*fila 0 */ { "allan", "leiton" },
/*fila 1 */ { "rebeca", "vargas" },
/*fila 2 */ { "pedro", "suarez" }
};
Console.WriteLine("vargas. " + nombre[1,1]);

foreach (string nomb in nombre)
{
    Console.WriteLine(nomb);
}
```

Ejemplo 2 de declaración array

```

int fila;
int columna;
Console.WriteLine("ingrese fila");
fila = Int32.Parse(Console.ReadLine());
Console.WriteLine("ingrese columna");
columna = Int32.Parse(Console.ReadLine());
string[,] nombre = new string[3,2]; //declaracion de array

    for (int i = 0; i < fila; i++) //for recorre array y llena los espacios
    {
        for (int x = 0; x < columna; x++)
        {
            Console.WriteLine("ingrese un nombre en la posicion {0} {1}: ", i, x);
            nombre[i, x] = Console.ReadLine();
        }
    }
foreach (string nom in nombre) //foreach recorre el array
{
    Console.WriteLine("nombre: "+ nom);
}

```

7.3 array uni,bi,y tridimensional


```

//array unidimensional
string[] nombres = new string[] {"allan", "leiton" };//ejemplo 1
string[] nombre2 = new string[2];//ejemplo 2
nombre2[0] = "allan";
nombre2[1] = "leiton";
//array bidimensional
string[,] nombres3 = new string[,];//ejemplo 1
{
    //columna 0 columna 1
/*fila 0*/ {"allan","leiton"},
/*fila 1*/ {"jose","vargas"},
/*fila 2*/ {"brandon","quiros"}
};
// fila columna
string[,] nombres4 = new string[3, 2];//ejemplo 2
nombres4[0,0] = "allan";
nombres4[0, 1] = "leiton";
//array tridimensional
string[,,] nombres5 = new string[,,];//ejemplo 1
{
/*tabla 0*/{ //columna 0 columna
    /*fila 0*/{"allan","leiton" },
    /*fila 1*/{"diego","vargas" }
},
/*tabla 1*/{ //columna 0 columna 1
    /*fila 0*/{"diana","rojas"},
    /*fila 1*/{"rebeca","umana" }
},
/*tabla 2*/{ //columna 0 columna 1
    /*fila 0*/{"carlos","perez" },
    /*fila 0*/{"rosa","maria" }
}
};
string[,,] nombres7 = new string[3, 3, 2];//ejemplo 2
nombres7[1, 0, 0] = "juan";

```

7.4 pasar array por parámetro

```
0 references
static void Main(string[] args)
{
    string[] nombre = new string[7];
    nombre[5] = "allan";
    metodo_array(nombre);
}

1 reference
public static void metodo_array(string [] p_array)
{
    string prueba = p_array[5];
    Console.WriteLine("la prueba es "+ prueba);
}
```

7.5 arraylist se pueden guardar diferentes tipos de datos no solo de un tipo y pueden tener tamaño definido o no

```
//arraylist indeterminado no tiene un tamaño definido
ArrayList miarraylist = new ArrayList();
//arraylist determinado tiene un tamaño definido
ArrayList miarraylist2 = new ArrayList(17);
//Agregar elementos al arrays
miarraylist.Add(7);
miarraylist.Add("allan");
miarraylist.Add(77.7);
//borrar elementos
miarraylist.Remove(7); //borra el primer valor que tenga 7 en el arraylist
miarraylist.RemoveAt(1); //elimina el elemento por posición
//contar los elementos en el arraylist
Console.WriteLine(miarraylist[0]); //muestra el valor de posición 0
Console.WriteLine(miarraylist.Count); //muestra la cantidad
foreach (var arr in miarraylist) //recorre el array list
{
    Console.WriteLine("mi array: {0}", arr);
}
```

7.x listas que tienen que ser declaradas de un solo tipo de dato y no tiene tamaño definido

```
List<int> numeros = new List<int> {7,77};
numeros.Add(777);
Console.WriteLine("cantidad: {0} ",numeros.Count);
Console.WriteLine("numero especifico {0}", numeros[2]);

foreach (int lis in numeros)
{
    Console.WriteLine("numero {0}", lis);
}
```

8 HERENCIA

8.1 Hereda las propiedades y métodos de sus clases hijas ejemplo

Main

```
animal ani = new animal();

perro per = new perro();//instancia objeto de clase perro
per.Tamano = "grande";//agrega valores a las propiedades de la clase animal
per.Color = "cafe";
per.Raza = "american";//agrega valor a la propiedades de la clase perro
Console.WriteLine("tamano {0} color {1} y raza {2} ", per.Tamano,per.Color,per.Raza);
per.comer();//llama el metodo comer sobre escrito de la clase perro
per.ladrrar();//llam el metod de la clase perro
ave av = new ave();//instancia un objeto de la clase ave
av.Tamano = "pequeno";//agrega valores a las propiedades de la clase animal
av.Color = "negro";
av.ColorPico = "rojo";//agrega valores a las propiedades de la clase ave
Console.WriteLine("tamano {0} color {1} y raza {2} ", av.Tamano, av.Color, av.ColorPico);
av.comer();//llama el metodo herdado de la clase animal
av.volar();//llama el metodo heredado de la clse ave
```

Clase animal

```
class animal
{
    private string color; //variables
    private string tamano;
    //propiedades
    4 references
    public string Color { get { return color; } set { color = value; } }
    4 references
    public string Tamano { get { return tamano; } set { tamano = value; } }
    //metodo virtual para que pueda ser sobrescrito por sus clases hijas
    3 references
    public virtual void comer()
    {
        Console.WriteLine("esta comiendo");
    }
}
```

Clase perro que hereda de clase animal

```
class perro:animal //hereda de la clase animal
{
    private string raza; //variable
    //propiedad con get a ser de la variable raza
    2 references
    public string Raza { get { return raza; } set { raza = value; } }
    //metodo ladrar solo de la clase perro
    1 reference
    public void ladrar()
    {
        Console.WriteLine("esta ladrando");
    }
    //metodo comer para sobrescribir el metod de la clase animal llamado comer igual
    3 references
    public override void comer()
    {
        Console.WriteLine("esta comiendo el perro");
    }
}
```

Clase ave que hereda de la clase animal

```
class ave:animal//hereda de la clase animal
{
    private string colorPico;//variable
    //propiedad de retorno y asignacion de la variable colorPico
    2 references
    public string ColorPico { get { return colorPico; } set { colorPico = value; } }
    //metodo volar de la clase ave
    1 reference
    public void volar()
    {
        Console.WriteLine("esta volando");
    }
}
```

8.2 interface el cual tiene propiedades métodos o eventos nadamas declarados sin contenido que tienen que estar por obligación en las clases que hereden de el

```
static void Main(string[] args)
{
    notificacion not = new notificacion();
    not.mostrarMensaje();
    not.Fecha = "17";
    Console.WriteLine("la fecha es {0}", not.verfecha());
}
```

```
1 reference
interface Interface1
{
    1 reference
    public string Nombre { get; set; }//propiedad Nombre de la interface
    2 references
    void mostrarMensaje();//metodo de la interface
    2 references
    string verfecha();//metodo de la interface
}
```

```

2 references
class notificacion:Interface1//hereda de la interface
{
    1 reference
    public string Nombre { get; set; } //llama al metodo de la interface por obligacion
    private string fecha;
    1 reference
    public string Fecha { get {return fecha; } set {fecha = value; } } //propiedad para acceder a fecha
    2 references
    public void mostrarMensaje() //metodo creado por obligacion de la interface
    {
        Console.WriteLine("muestra el mensaje");
    }
    2 references
    public string verfecha() //metodo creado por obligacion por la interface
    {
        return this.fecha;
    }
}

```

9 polimorfismo: varias formas, virtual en padre y override en hijos para sobre escribir los métodos

```

0 references
static void Main(string[] args)
{
    cuadrado cua = new cuadrado(); //declara un objeto de cuadrado
    cua.Largo = 7; //ingresa 7 al largo de cua
    cua.area(); //llama al metod de area del cuadrado sobre escrito por override
    triangulo tri = new triangulo(); //declara un objeto de triangulo
    tri.Largo = 7; //asigna 7 en el largo de triangulo
    tri.area(); //llama al metodo area de figuras
}

```

```

class figuras
{
    private int largo;
    private int ancho;
    4 references
    public int Largo { get {return largo; } set {largo = value; } }
    0 references
    public int Ancho { get {return ancho; } set { ancho = value; } }

    3 references
    public virtual void area() //usa virtual para que pueda sere sobreescrito por sus hijos
    {
        int area = this.largo * this.largo;
        Console.WriteLine("el area de figuras es {0}", area);
    }
}

```

```

class cuadrado:figuras//hereda de figuras sus prop largo y ancho pero
                        //sobreescribe el metodo de area
{
    3 references
    public override void area()
    {
        int area = this.Largo * this.Largo;
        Console.WriteLine("el area de cuadrado es {0}", area);
    }
}

```

```

2 references
class triangulo:figuras//hereda de figuras sus metodo area y largo y ancho
{
}

```

9.2 lectura y escritura de un archivo txt

```

static void Main(string[] args)
{
    //LEER
    //texto a leer en la siguiente ubicacion
    TextReader leer = new StreamReader(@"C:\Users\ALLAN\Desktop\prueba.txt");
    Console.WriteLine(leer.ReadToEnd());//lee todo el archivo de texto
    Console.WriteLine(leer.ReadLine());//lee la primera linea de texto
    leer.Close();//cerrar
    //guarda el texto en un string y mostrarlo
    string mensaje_string = File.ReadAllText(@"C:\Users\ALLAN\Desktop\prueba.txt");
    Console.WriteLine(mensaje_string);
    //guarda las lineas del texto en las posiciones de un array
    string[] mensaje_array = File.ReadAllLines(@"C:\Users\ALLAN\Desktop\prueba.txt");
    foreach (string men in mensaje_array)
    {
        Console.WriteLine(men);
    }
    //ESCRIBIR
    //agrega valores a un archivo ya creado y sino esta esta creado lo crea
    StreamWriter escribir = File.AppendText(@"C:\Users\ALLAN\Desktop\prueba2.txt");
    escribir.WriteLine("uno");//agrega uno
    escribir.WriteLine("dos");//agrega dos mas abajo de uno
    escribir.WriteLine("tres");
    escribir.Close();
    //sobreescribe datos en un archivo ya creado sino esta lo crea
    TextWriter escribir2 = new StreamWriter(@"C:\Users\ALLAN\Desktop\prueba3.txt");
    escribir2.WriteLine("uno");//agrega uno
    escribir2.WriteLine("dos");//agrega dos mas abajo de uno
    escribir2.WriteLine("tres");
    escribir2.Close();
}

```

Temas avanzados

10 clase math

```
Referencias
static void Main(string[] args)
{
    //redondeo hacia arriba
    Console.WriteLine("redondeo 17.7 es {0}", Math.Ceiling(17.7));
    //redondeo hacia abajo
    Console.WriteLine("redondeo 17.7 es {0}", Math.Floor(17.7));
    int a = 7;
    int b = 17;
    Console.WriteLine("el mayor entre a y b es {0}", Math.Max(a,b));
    Console.WriteLine("el menor entre a y b es {0}", Math.Min(a, b));
    Console.WriteLine("3 elevado a la 5 es {0}", Math.Pow(3,5));
    Console.WriteLine("pi es {0}", Math.PI); ;
    Console.WriteLine("la raiz cuadrada de 7 es {0}", Math.Sqrt(7));
    Console.WriteLine("coseno de 1 es {0}", Math.Cos(1));
}
```

10.2 clase random

```
bool salir = false;
do
{
    Console.WriteLine("ingrese una pregunta ");
    Console.ReadLine();
    Random ram = new Random();//creo la clase random
    int respuesta = ram.Next(0, 4);//guardo el numero al azar mayor a 0 y menor a 4
    if (respuesta == 1)
    {
        Console.WriteLine("si");
    }
    else if (respuesta == 2)
    {
        Console.WriteLine("quizas");
    }
    else
    {
        Console.WriteLine("no");
    }
    Console.WriteLine("desea salir si o no");
    string res = Console.ReadLine();
    if (res == "si")
    {
        salir = true;
    }
    else { salir = false; }
} while (salir == false);
```


10.3 expresiones regulares

```
Archivo Edición Formato Ver Ayuda
CARACTER DE ESCAPE
\t - Coincide con un tab
\n - Coincide con una nueva línea

CLASES DE CARACTERES
. - Carácter comodín: coincide con cualquier carácter excepto con \n
\d - Coincide con cualquier dígito decimal. (0-9)
\D - Coincide con cualquier carácter que no sea un dígito decimal. (0-9)
\w - Coincide con cualquier carácter de una palabra. (a-z, A-Z, 0-9, _)
\W - Coincide con cualquier carácter que no pertenezca a una palabra.
\s - Coincide con cualquier carácter que sea un espacio en blanco. (space, tab, newline)
\S - Coincide con cualquier carácter que no sea un espacio en blanco. (space, tab, newline)
[character_group] - Coincide con cualquier carácter de grupo_caracteres. Por defecto distingue mayúsculas y minúsculas.
[^character_group] - Negativo: Coincide con cualquier carácter que no sea de grupo_caracteres. Por defecto distingue mayúsculas

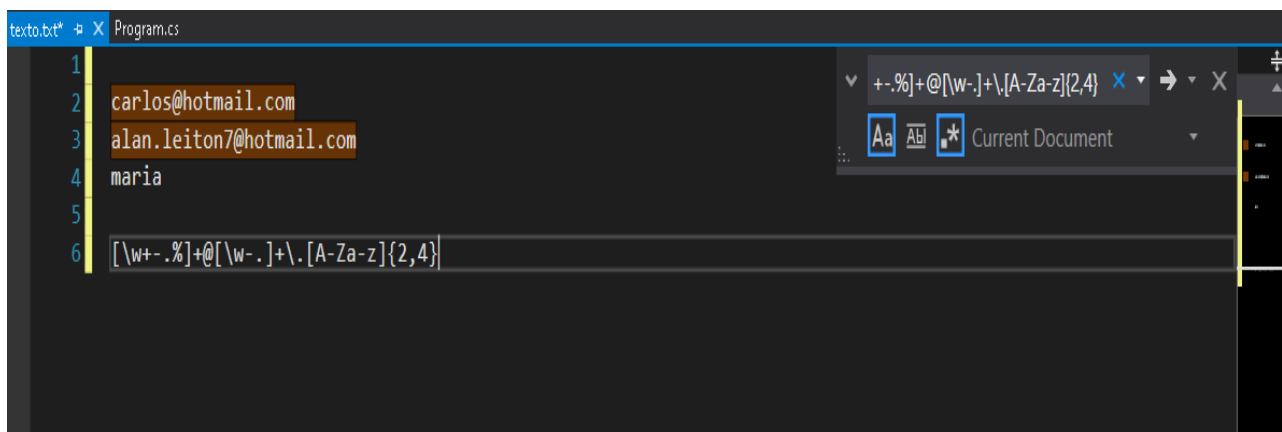
DELIMITADORES
^ - La coincidencia debe comenzar al principio de la cadena.
$ - Por defecto, la coincidencia se debe producir al final de la cadena o antes de \n al final de la cadena
\A - La coincidencia se debe producir al principio de la cadena.
\Z - La coincidencia se debe producir al final de la cadena o antes de \n al final de la cadena.
\b - La coincidencia se debe producir en un límite entre un carácter \w (alfanumérico) y un carácter \W (no alfanumérico).
\B - La coincidencia no se debe producir en un límite \b.

CONSTRUCTORES DE ALTERNANCIA
| - Ya sea lo que está antes de | o después.

CONSTRUCTORES DE AGRUPAMIENTO
( ) - Grupo

CUANTIFICADORES
* - Coincide con el elemento anterior cero o más veces.
+ - Coincide con el elemento anterior una o más veces.
? - Coincide con el elemento anterior cero o una vez.
{n} - Coincide con el elemento anterior exactamente n veces.
{n,m} - Coincide con el elemento anterior al menos n veces, pero no más de m veces.
```

En c# creo un archive de texto ctrl + f para sacar el ejecutador de expresiones y las comparo para ver si me sirven



The screenshot shows a text editor with a file named 'texto.txt' open. The editor contains the following text:

```
1
2 carlos@hotmail.com
3 alan.leiton7@hotmail.com
4 maria
5
6 [\w+.%]+@[ \w-. ]+ \. [A-Za-z]{2,4}
```

On the right side, there is a search bar with the regular expression `+.%.+@[\w-.]+ \. [A-Za-z]{2,4}` entered. Below the search bar, there are icons for 'Aa' (case sensitivity), 'Ab' (whole word), and a search icon. The text 'Current Document' is also visible.

Validar una expresión con false o true

```
References
static void Main(string[] args)
{
    string numero = "2278 - 05 - 93";
    string correo = "alan.leiton7@hotmail.com";
    //clase reg con la expresion regular
    Regex reg = new Regex(@"^([\w+-.%]+@[\w-.] +\.[A-Za-z]{2,4},?)+$");
    if (reg.IsMatch(correo))// me dice en bool si es falso o verdadero la expresion
    {
        Console.WriteLine("si coincide con la expresion ");
    }
    else
    {
        Console.WriteLine("no coincide con la expresion ");
    }
}
```

Me guarda el valor si cumple con la exp regular

```
static void Main(string[] args)
{
    string numero = "2278 - 05 - 93";
    string correo = "alan.leiton7@hotmail.com";
    // clase reg con el valor de la expre regular
    Regex reg = new Regex(@"^([\w+-.%]+@[\w-.] +\.[A-Za-z]{2,4},?)+$");
    // match me guarda en acierto solo la exp reg validas
    Match acierto = reg.Match(correo);
    Console.WriteLine("acierto es {0}", acierto);
}
```

Clase datetime fechas y horas

```
static void Main(string[] args)
{
    //clase datetime llamada fecha y se le asigno una fecha
    DateTime fecha = new DateTime(1975,5,3);
    Console.WriteLine("muestra la fecha asignada {0}", fecha);
    Console.WriteLine("el dia de la fecha 1975 5 3 que seria 3 {0}", fecha.Day);
    Console.WriteLine("fecha hoy sin hora {0}", DateTime.Today);
    Console.WriteLine("fecha de hoy con hora {0}", DateTime.Now);
    Console.WriteLine("manana es {0}", DateTime.Today.AddDays(1));
    Console.WriteLine("dias del mes de diciembre en el ano 1989 {0}", DateTime.DaysInMonth(1989,12));

    Console.WriteLine("hora {0} minutos {1} segundos {2} actual",
        DateTime.Now.Hour,DateTime.Now.Minute,DateTime.Now.Second);
}
```

11 delegate

```
//definimos el delegado con las características que nos interesan
public delegate void midelegado(string mensaje);
0 references
static void Main(string[] args)
{
    radio ra = new radio();//instanciamos obj de clase radio
    pastel pa = new pastel();//instanciamos obj de clase pastel
    //creamos obj de midelegado y lo referenciamos a metodoradio de la clase radio
    midelegado delegado1 = new midelegado(ra.MetodoRadio);//
    //hacemos uso de el metodo
    delegado1("hola a todos");
    delegado1 = new midelegado(pa.MostrarPastel);
    delegado1("pastel");

    //el delegado es para pasar funciones como parametros
    //y los delegados hacen referencia a metodos especificos que cumplen
    //con sus características
    //cuando se llama al delegate el hace referencia a un metodo y le asigna el trabajo
    //expresion lambda es el metodo sin modificador sin, nombre y si
```

2 references

```
class radio
{
    1 reference
    public void MetodoRadio(string p_mensaje)
    {
        Console.WriteLine("estamos en la clase radio");
        Console.WriteLine("el mensaje es {0}", p_mensaje);
    }
}
```

2 references

```
class pastel
{
    1 reference
    public void MostrarPastel(string p_anuncio)
    {
        Console.WriteLine("el pastel llevara el mensaje de {0}", p_anuncio);
    }
}
```

Delegado con expresiones lambda resumidas con func<> con retorno y action<> sin retorno

```
static void Main(string[] args)
{
    //delegado action que no devuelve nada, pasa string un parametro y se le asigna
    //una expresion lambda que envia un string men y muestra mensaje
    Action<string> delegado = men => Console.WriteLine("el mensaje es {0}", men);
    delegado("hola");
    //ejemplo de action que no devuelve valor y pasa 2 parametros
    Action<int, int> d_suma = (num1, num2) => Console.WriteLine("suma {0}",(num1 + num2) );
    d_suma(10, 7);
    //func con retorno string que suma 2 numeros el ultimo es el valor de retorno
    //delegdo 2 parametro y retorno expresion regular lambda
    Func<int, int, string> sumar = (nume1, nume2) => (nume1 + nume2).ToString() ;
    Console.WriteLine("suma {0}", sumar(7, 7));
}
```

Pasando delegados o métodos por parámetros

```
0 references
static void Main(string[] args)
{
    //delegado void principal1 que muestra mensaje enviado
    Action<string> principal1 = men => Console.WriteLine(" mensaje delegado 1 {0}", men);
    Action<string> principal2 = men => Console.WriteLine("mensaje delegado 2 {0}", men);
    lambda lam = new lambda(); //instacia de la clase lambda
    //corre el metodo de la clase lam que tiene por parametro el metodo lambda del delegado
    lam.metodo(principal1, "allan");
    lam.metodo(principal2, "diego");
}
```

```
2 references
class lambda
{
    public void metodo(Action<string> prin, string nombre)
    {
        prin(nombre);
    }
}
```

Resumen 2

WPF.