Nombre y Apellido\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Matricula\_\_\_\_\_\_\_\_\_\_\_ Sección\_\_\_\_\_\_\_\_\_

**Ejercicio 1** Realice un programa de su elección utilizando el lenguaje de programación de su elección donde se incluyan las sentencias “**IF**”, “**Else-IF**”, “**For**” y “**Case**” las variables “**Int**”, “**Decimal**”, “**String**” y “**Booblean**” y se utilice cuando menos 1 función.

**Ejercicio 2** Responda Correctamente.

1. ¿Cuáles son las ventajas de C# sobre C, C++ o Java?
2. ¿Cómo se utilizan los nombres de espacio en C#?
3. ¿Qué es un constructor?
4. ¿Qué es un destructor?
5. ¿Cómo se sobrecargan métodos en C#?
6. ¿Por qué usar la encapsulación?
7. ¿Cuál es la diferencia entre una clase y un registro (struct)?
8. ¿Qué es el GAC?
9. ¿Cómo ayuda .NET a gestionar las DLLs en un sistema?
10. ¿Qué tipos de errores pueden ocurrir en un programa C# (defina los tres principales)?
11. ¿Para que sirve el método Finalize en .NET?
12. ¿Por qué se prefiere no utilizar Finalize para limpiar?
13. ¿Qué es el método Dispose?
14. ¿Cómo fuerzo la llamada al método Dispose ya que los clientes podrían olvidar llamar el método?
15. ¿Qué es una interfaz?
16. ¿Qué es una clase abstracta?
17. ¿Cómo funciona la caché de salida (output caching) en ASP.NET?
18. ¿Qué es connection pooling y cómo haces que una aplicación lo utilice?
19. ¿Cuáles son los diferentes métodos de mantenimiento de sesión en ASP.NET?
20. ¿Qué hace la propiedad “EnableViewState”? ¿Por qué quisiera prenderla o apagarla?
21. ¿Cuál es la diferencia entre Server.Transfer y Response.Redirect?

**Ejercicio 3** Responda Falso o Verdadero.

1. En una aplicación de consola para leer los parámetros utilizaremos un **array** de variables de tipo **string** que llamaremos “**args**“.
2. Para cambiar el color de fondo de una aplicación de consola utilizaremos **Console. ColorBackground = System.Color(“Green”);**
3. La siguiente instrucción **Console.WriteLine(“\nChar: {0} y {1}”, Char.MinValue, Char.MaxValue);** nos devolverá una “a” minúscula y una “Z” mayúscula.
4. El tipo “**long**” es el tipo de número entero más alto que podemos representar en C#
5. El tipo “**decimal**” ocupa 128 bits (el doble que un tipo **long**) y es el tipo de número con decimales más grande que se puede representar en C#.
6. El tipo “**double**” ocupa lo mismo que el tipo “**ulong**“.
7. El tipo “**char**” ocupa 2 bytes que es lo mismo que el tipo entero “**short**” que a su vez también ocupa lo mismo que el tipo “**ushort**“, es decir 16 bits.
8. Las variables en C# a diferencia de Java, no distinguen entre mayúsculas y minúsculas.
9. No se pueden utilizar palabras reservadas como nombres de variables.
10. Las variables siempre empiezan por una letra o por el símbolo “**@**“.
11. Para escribir una comilla simple en una variable alfanumérica podemos utilizar la secuencia de escape \’
12. Este es un ejemplo de declaración y asignación de una variable lógica: **Bool EsCierto = true;**
13. Una variable tipo **Object** podemos almacenar cualquier cosa.
14. El tipo **dynamic** representa una dirección de memoria.
15. Mediante la instrucción **“$”** podemos asignar el valor **null** a cualquier variable.
16. Para saber si una variable tiene asignado algún valor distinto de **null** se puede utilizar la propiedad **HasValue**. Por ejemplo if (CodigoPostal.HasValue), siendo CodigoPostal de tipo “**int**” …
17. Para asignar el valor **null** a una variable tipo string podemos asignarla directamente **variable = null;** o bien asignado la cadena vacía **variable = “”**;
18. V1=null; V2=false; V3=(V1&V2)     ; V4=(V1|V2); V5=V3&V4; dará como resultado en V5 “**null**“.
19. La conversión explícita y = (long) x; daría un error de compilación si la variable x fuera de tipo double.
20. Console.WriteLine (**String.Format**(“{0:p}”,0.2)); dará como resultado **20,00%.**
21. Console.WriteLine (**String.Format** (“Importe = {0:e}”,12.35)); dará como resultado **12,35€.**
22. Console.WriteLine (**String.Format**(“{0:T}”,**DateTime.Now**)); dará como resultado la Hora del Sistema hasta los segundos.
23. El método **Parse** se utiliza para convertir una cadena en algún tipo numérico, de igual manera se utilizará el método **ToString** para pasar un tipo numérico a cadena.
24. El método **TryParse** es un método sobrecargado que devuelve un valor booleano true si la conversión se realiza correctamente y false en caso contrario.
25. En caso de no incializar una varibale numérica el valor inicial será **cero**; si es una cadena el carácter vacío, a valor **null** si es del tipo Object, y a true si es del tipo boolean.
26. Las inicializaciones por defecto de las variables no se produce cuando se declaran dentro de una función.
27. El ámbito de las variables puede ser a nivel de bloque, a nivel de función o a nivel de clase, siendo la de mayor ámbito la que se declara a nivel de bloque.
28. El nivel de acceso **public**no se pueden utilizar en el interior de las funciones.
29. El nivel de acceso **protected** restringe el acceso a la variable únicamente al código de la clase donde está definida, así como a las de todas las clases que hereden de esta.
30. El nivel de acceso **internal**no se puede utilizar en el interior de una función.
31. El nivel de acceso **private** no se puede utilizar en el interior de una función.
32. Sino indicamos nada a una variable; esta se considera como **public**.
33. No se puede calcular el valor de una constante a partir de una variable.
34. No se puede calcular el valor de una constante a partir de una constante en tiempo de ejecución, sí se puede en cambio en tiempo de compilación.
35. Todas las enumeraciones comienzan por el índice 0.
36. Las enumeraciones se pueden utilizar como un tipo de variable.
37. Todas las matrices de forma directa o indirecta hay que indicarles el tamaño que tienen sus dimensiones.
38. Para obtener el número de casillas de una matriz utilizaremos la función **GetLenght**
39. Para saber el tamaño de una dimensión de una matriz se utiliza la función **Lenght**(<número dimensión>)
40. Para saber las dimensiones de una matriz se utiliza la función **Ranking**
41. Es posible ordenar una matriz con la función **Array.Sort**
42. Siendo “cadena1” una variable de tipo **string** si queremos buscar la posición del primer carácter “**@**” de la cadena utilizaremos la instrucción **cadena1.IndexOf(“@”);** en caso de existir dos caracteres “**@**” la función devolverá el valor **-1** y en caso de no existir la función devolverá el valor **0**.
43. Para reemplazar en una cadena se puede utilizar el método **Replace.**
44. Las estructuras “**struct**” se construyen únicamente a partir de la combinación de otros tipos básicos del lenguaje C#.
45. Las estructuras “**struct**” pueden incluir funciones;
46. La clase **StringBuilder** permite entre otras funciones la concatenación de cadenas, es ligeramente más lenta que la concatenación directa con el operador “+” pero evita las excepciones en caso de concatenar tipos de datos **hetereogéneos**.
47. El operador “**Is**” devuelve un valor booleano indicando si la variable es del tipo que se indica.
48. La comparación **If (test1)&(test2)** es menos o igual de eficiente que **If (test1)&&(test2)**.
49. El operador “**%**” devuelve el cociente de una división entera.
50. La comparación **(test1)^(test2)** será cierta sólo si **test1** es **true** y **test2** es **true**.
51. Esta estructura con **if** sería equivalente a una estructura **switch**de 3 opciones y un default, **if** (condición) { … } **else**{ …**if** (condición) {…} **else** {…} }
52. En un **switch** si alguna opción no tiene la instrucción **break** el programa continuará evaluando las opciones inferiores del **switch**.
53. Una estructura **while (true) { Console.Write(“a”); }** provoca un error de compilación ya que detecta que estamos en un bucle infinito y no dejará compilar el programa.
54. Una estructura: **do** { … } **foreach** (<enumeración>); se ejecutará al menos una vez.
55. El siguiente bucle se ejecutará 8 veces **for (k1 = 23; k1 > 0; k1=(k1–)-3) { … }**
56. El siguiente código daría error de compilación porque se están comparando diferentes tipos de números: **double num\_pi = Math.PI; for (k1 = 23+(long) num\_pi; k1 > -2.56; k1=(k1–)-5/2) {…}**
57. El siguiente código intercambiaría el valor de las variables x e y sin utilizar una tercera **variable Int x=-10, y=+10; x = x + y + y; y = x – y – y;**
58. La visibilidad de un procedimiento de forma predeterminada es “**private**” a no ser que indiquemos otra opción en su declaración.
59. La ejecución de un **return** dentro de una función hace que la función asigne un valor a la instrucción llamante de la función y además termine la ejecución de dicha función aunque haya más cógido fuente por debajo de ella, de hecho el compilador nos dará una advertencia de que el código siguiente e **return** es inaccesible.
60. Los procedimientos de propiedades, también llamadas encapsuladores y utilizan las palabras reservadas get y set para acceder a la propiedad.
61. Existe la posibilidad de definir el comportamiento de un operador al usar como operando variables a priori no compatibles con el operador. Por ejemplo: public static Cliente **operator**  
    **+** (Cliente cl1, Cliente cl2) {… return …}
62. Las variables por referencia siempre deben ir precedidas de la palabra reservada “**ref**” o “**out**“.
63. Las palabras reservadas “**ref**” y “**out**” deben ser utilizadas tanto en la llamada a la función como en la declaración de parámetros del procedimiento o función.
64. La diferencia entre “**ref**” y “**out**” es que mientras las variables que se pasan por “**ref**” no necesitan ser inicializadas, las variables que se pasan por “**out**” necesitan obligatoriamente ser pasadas inicializadas.
65. Cuando creamos un procedimiento o función que no está sobrecargado ni tiene parámetros opcionales, debemos saber a priori el número exacto de parámetros que le vamos a pasar, es decir el número de parámetros está determinado por la propia declaración de la función o procedimiento.
66. Es posible crear parámetros opcionales, asignándole un valor por defecto en la declaración, además hay que tener la precaución de inicializar a partir del primer parámetro inicializado, el resto de parámetros de la derecha para que ocurra ningún error de compilación.
67. Cuando se hace la llamada a una función o procedimiento con parámetros opcionales habría que dejar los huecos en la llamada para que el compilador sepa a qué parámetro nos estamos refiriendo, por ejemplo en una llamada a una función con 3 parámetros donde el primero es obligatorio y los dos últimos opcionales una posible llamada a esa función sería **resultado = FuncioLoQue Sea (25, , ref variable3);**
68. Los parámetros siempre deben llevar el orden en que fueron declarados excepto el caso en que sean nominados, en cuyo caso siempre podemos ponerlos como queramos, por ejemplo **FuncionLoQueSea(25, parametro2: 33, “€”);** siendo los parámetros del a la función Int parametro1,Int parametro2 y String parametro3.
69. Esta declaración de función **public**  
    **double Media (int[] notas)** está mal le faltaría la palabra reservada **params** ya que estamos pasando como parámetro un array.
70. Esta declaración de función **public**  
    **double Media (int[] notas)** está bien si lo que le pasamos a la función es una variable por referencia que apunte a un array de enteros.
71. Esta declaración de función **public**  
    **static**  
    **double Media (param int[] notas)** está bien si le pasamos por valor una serie cualquiera de números enteros, por ejemplo: **Media(4\*2, 2+3, 7-7)**.
72. Los **objetos** son la representación simbólica de las **clases**.
73. Las tres principales propiedades de los objetos son: **la herencia, el polimorfismo y la instanciación**.
74. El **UML** es una clase abstracta propia del lenguaje C#.
75. Tenemos 7 posibles combinaciones para determinar la visibilidad de una clase**: public, internal, private, private internal, protected, abstract, sealed.**
76. **Protected**: no permite crear instancias de esta clase, sólo sirve para ser heredada como clase base. Suelen tener los métodos definidos pero sin ninguna operatividad con lo que se suele escribir estos métodos en las clases derivadas.
77. **Internal**: cuando una clase es la última de una jerarquía, por lo que no podrá ser utilizada como base de otra clase.
78. **Private internal:**la clase tendrá las mismas propiedades que private e internal.
79. **Private:** la clase sólo puede usarse en el módulo en la que está definida.
80. Cuando declaramos una clase como parcial (**partial**) le indicamos al compilador que vamos a definir una clase en diferentes declaraciones, incluso con la posibilidad de reescribir algún método de la clase en una segunda o tercera declaración parcial de la clase.
81. Cuando declaramos alguna propiedad como **private** la única forma de acceder a ella será a través de los métodos o de la encapsulación utilizando las funciones “**get**” y “**set**“.
82. Un ejemplo de encapsulación sería: public **String Apellido { get{ elApellido = value.ToUpper(); } set {return elApellido;}}**donde elApellido sería la propiedad **private**.
83. Se pueden restringir los accesos a una propiedad, si sólo incluimos la opción “**get**” lo que estamos haciendo es dando permisos de lectura.
84. Si incluimos la opción “**set**” lo que hacemos es dar permisos de lectura y escritura.
85. No es posible crear una propiedad que haga referencia (es decir sea del mismo tipo) a la propia clase que estamos creando, ya que produciría un error de compilación. Es decir: **public class Persona {… private Persona Pepe = new Persona(); … }** daría error al hacer una referencia circular.
86. Si es posible crear una propiedad que sea un array de objetos de la misma clase que la clase que estamos creando . Es decir: **public class Persona {… private Persona[] losHijos = new Persona[10]; ….}**
87. Todos los métodos de una clase pueden manejar campos de la clase incluso si son privados.
88. La **sobrecarga** es la creación dentro de la clase, de un grupo de métodos que tienen el mismo nombre pero con un número de parámetros distinto aunque los parámetros coincidentes tienen que ser del mismo tipos de datos.
89. Las clases derivadas siempre heredan las propiedades y métodos de su clase base.
90. Las propiedades privadas no son heredadas por las clases hijas, simplemente heredarán los métodos que acceden a esas propiedades privadas.
91. Para poder sobrescribir un método en una clase hija, se utilizará la palabra reservada **over**en la clase hija y **virtual**en la clase heredada.
92. El siguiente método que se sobrescribe: **public sealed void visualización () { base.visualizacion();}** haría lo mismo que el método heredado, pero con la diferencia de que ese método ya no podría reescribirse en otras clases que lo hereden ya que incluye la directiva **sealed**.
93. Cuando declaramos en una clase un método como abstracto (**abstract**) estamos indicándole al compilador que el método no tiene implementación sólo definición.
94. Cuando un método es marcado como abstracto entonces la clase a la que pertenece también tiene que ser marcada como abstracto (**abstract**).
95. Además de poder hacer clases parciales (**partial**) también es posible hacer métodos parciales.
96. Para poder acceder a las propiedades de un objeto utilizaremos la siguiente sintaxis: **<nombre del objeto>.<propiedad>**.
97. Si estamos realizando una aplicación en Windows Form el objeto más aconsejable para la introducción de datos por parte del usuario es el objeto tipo **Label**
98. Un **ListBox** nos muestra un campo donde sólo se visualiza un registro pero puede desplegarse para ver otros valores.
99. Utilizaremos un **TextBlock** para mostrar el contenido de texto de varias líneas.
100. Uno de los métodos del objeto **Button** es el **OnClickMouse**

**Ejercicio 4** Seleccione la Respuesta Correcta

* 1. **Cuántos bits ocupan los siguientes tipos numéricos:**

1. byte: 4 bits, long: 32 bits, short: 16 bits y decimal: 132 bits.
2. byte: 8 bits, long: 64 bits, short: 16 bits y decimal: 128 bits.
3. byte: 4 bits, long: 32 bits, short: 24 bits y decimal: 128 bits.
4. byte: 8 bits, long: 64 bits, short: 16 bits y decimal: 132 bits.

**2. Según el código que salida se obtendría y de que tipo?:**

class Test

{

staticvoid Main()

{

double x = 1243.73;

int c;

int a;

a = (int) x;

    c = 1234.73;

System.Console.WriteLine(a);

}

}

1. “1243.73” de tipo int.
2. “1243” de tipo int.
3. “1234” de tipo double.
4. “1243.73” de tipo double.

**3. Qué significa este ejemplo “If (test1)&&(test2)” en relación al operador “&&”?**

1. Solo evalua test2 si test1 es cierto.
2. Cierto si ambos son ciertos.
3. Cierto si alguno de los dos es cierto.
4. Ninguna respuesta es correcta.

**4. En caso de que no se le de valor a un tipo bolean el valor por defecto es ? :**a) True.  
b) False.  
c) Null.  
d) Hay que darle un valor obligatoriamente.  
e) Todas son falsas.

**5. Cuanto es un byte?**  
a) 16 bits  
b) 9 bits  
c) 16 kb  
d) 9 kb  
e) a y c son correctas  
f) b y d son correctas  
g) Ninguna es correcta

**6. ¿Como podemos hacer un cambio de variable de tipo double a string?**

* 1. Double y = 55;

String x = y.ToString();

* 1. Double x = 55;

String y = Double.x.ToString();

* 1. Double x;

x = Double.parseDouble(textBox.Text);

* 1. Ninguna es correcta.

**7. ¿Cual de las siguientes afirmaciones es correcta?**

1. La palabra this hace referencia a la instancia actual de la clase.
2. Se utiliza como modificador del primer parámetro de un método de extensión.
3. Se utiliza para obtener acceso a miembros con el fin de evitar ambigüedades con nombres similares.
4. Todas son correctas.

**8. La forma del resultado de la conversion String.format(“{0:f}”,0.2); es:**

a) Resultado: 20,00%  
b) Resultado: 2,458136e+005  
c) Resultado: 0,20  
d) Ninga es cierta  
**9. Crear un OBJETO a partir de un una clase es lo que se llama \_\_\_\_\_\_\_\_\_.**  
a) INSTANCIAR  
b) ENCAPSULACIÓN  
c) CLASES DERIVADAS  
d) Todas son cierta  
**10. Las tres propiedades de la ORIENTACIÓN A OBJETOS son:**

a) Private, set, get  
b) String, int, double  
c) Encapsulacion, herencia, polimorfismo  
d) Ninguna es cierta  
**11) Un Objeto es:**

a) . Una entidad con características similares a otro objeto

b). Una entidad con características diferentes a otro objeto

c). Una entidad con características propias

d). Ninguna de las anteriores

**12) Que objeto utilizaremos para receptar un dato detro de un formulario**

a). Button

b). Radiobutton

c). Label

d). Textbox

**13) La aplicación Windows forms me permite**

1. Crear un formulario en blanco para agregar objetos

2. Crear un contenedor de objetos

3. Crear una forma con varios objetos

4. Ninguna de las anteriores

**14. ¿ Cuál de ellos no es una estructura de control?**

a) if

b) try

c) while

d) switch

**15. ¿Que mostrará el siguiente código?**

class Ejercicio {

static void Main(String[] args) {

int var1 = 4;

String var2 = “La variable 1 vale “;

if (var1 < 4) {

int var3 = 2;

int var4 = var1 + var3;

Console.WriteLine(“La varible 4 vale: ” + var4);

}

Console.WriteLine(var2 + var1);

a) Se produce error

b) La variable 4 vale: 6

c) La variable 4 vale: 6 La variable 1 vale: 4

d) La variable 1 vale: 4

**16. ¿Si ponemos a una variable un nombre reservado, que cáracter debemos poner para que no lo sea?**

a) /

b) ?

c) @

d) %

**17. Los tipos de variables enteras :**  
a)char , String  
b) byte ,  ushort , uint , ulong    
c) float ,double , decimal.   
d) b y c son correctas

**18. En este fragmento de codigo, cual es la salida por pantalla:**

class Program  
{  
    static void Main(String[]args)  
    {

        int x=5, y=5;  
          
        for (int i=0;  i<y;  i–){  
        Console.write(i+”,”);  
    }

}}

* 1. imprimira 0,2,3,4  
     b) imprimira 0,-1,-2,-3,-4 y se parara  
     c) sera un bucle infinito que empiece con el 0 y siga decrementando infinitamente   
     d) ninguna es correcta

**19. Elige la salida correcta por pantalla**

class Program  
{  
    static void Main(String[]args)  
    {

    int [] array = new int [10];  
    int [] vector = new int[5];  
        for (int i=0; i<array.Length ; i++) {  
          
        array[i]=99;  
}

Console.WriteLine(array[0]);  
    }

}}

a) 0  
b) 10  
c) 99  
d) 5

**20. Elige la correcta:**

a) \n= salto de pagina   
b) \n= salto de línea   
c) \n= tabulacion horizontal  
d) \n= retorno de carro

**21. ¿Cuáles son las propiedades de la orientación a objetos?**

a) Encapsulación

b) Herencia

c) Polimorfismo

d) Todas son ciertas

**22. ¿Qué secuencia de escape utilizaremos para un salto de línea?**

a) \n

b) \’

c) \\

d) \b

**23. ¿Que visualizará el siguiente código?**

**string** cadena =“123”;

Console.writeLine  
(cadena + 456);

a) 123  
b) 456  
c) 123456  
d) 579

**24. Tipos de variables numéricas decimales**

a) Short – Float – Long  
b) Double – Float – Int  
***c)*** Double – Float – Decimal  
d) Ningunas de las anteriores

**25. Cual de las siguientes, es una definición correcta de un array de objetos:**  
***a)***private Persona[] p = new Persona[10];  
b)private persona p [10];  
c)Persona[] = new Persona ;  
d)Todas son correctas

**26. Cual sería la salida del siguiente código:**

* 1. double x;
  2. long y;
  3. x = 21.234323;
  4. y = (long) x;

**27. Console.WriteLine (“valor de y:” + y);**

a)valor de y: 21.234323  
b)valor de y: 21.23  
***c)***valor de y: 21

**28. La visibilidad de un procedimiento viene determinada por la declaración:**

a) Private, public o internal.

 b) Return.

 c) Sólo public.

 d) Todas son falsas.

**29. La ejecución de***return***provoca:**

 a) El inicio del proceso.

 b) La salida de la función.

 c) Una nueva sentencia.

 d) Todas son ciertas.

**30. La variable instancia:**

 a) Es el objeto de la clase.

 b) Representa un atributo de un objeto.

 c) Es el método de una clase.

 d) Ninguna de las anteriores es cierta.

**31. Teniendo esta instrucción: Qué procedimiento visible, por defecto se indica:**

void CambiarValor() {  
  Console.WriteLine(“¡¡¡Aprobado!!!”);  
}  
a) Private.  
b) Public  
c) Protected  
d) Ninguno

**32. Da error este código al compilar?:**  
int a = 5;               
int b = a + 2;   
bool test = true;  
int c = a + test;

a) No, no hay error y compilará  
b) Hay  error en la segunda línea.  
c) Hay  error en la tercera línea.  
d) Hay error en la cuarta línea.  
**34. ¿Que aparecerá al ejecutar este código?**  
Double myDouble = 42.72;  
 int myInt = Convert.ToInt32(myDouble);  
   Console.WriteLine(myInt);

a) 42.72  
b) 42  
c) No se ejecutará, producirá error  
d) 43

**35 Un Evento es...**

1. El resultado de la acción de un usuario.
2. El código que fuerza una acción de usuario

## 36

**static void Test(out int x, out int y)**

**{**

**x = 42;**

**y = 123;**

**Console.WriteLine(x == y);**

**}**

1. Falso
2. Cierto
3. Puede ser cierto
4. No compila

## 37

**static void Main(string[] args)**

**{**

**float value = 100000000;**

**while(value > 0)**

**{**

**–value;**

**}**

**Console.WriteLine(“Value es {0}”, value);**

**}**

1. Value es 0
2. Value es 0.00000001
3. Nunca se imprime el valor de Value
4. No compila

## 38

**int x = 1;**

**x = x++;**

1. x==1
2. x==2
3. No existe el operador ++
4. No compila

## 39

**static void Main(string[] args)**

**{**

**double result1 = 3.65d + 0.05d;**

**float result2 = 3.65f + 0.05f;**

**Console.WriteLine(result1 == 3.7d);**

**Console.WriteLine(result2 == 3.7f);**

**Console.WriteLine(result1 == result2);**

**}**

1. TRUE, TRUE, TRUE
2. TRUE, FALSE, TRUE
3. FALSE, FALSE, FALSE
4. TRUE, TRUE, FALSE
5. FALSE, TRUE, TRUE
6. FALSE, TRUE, FALSE

**static void Main(string[] args)**

**{**

**double delta = 0.1;**

**int i = 0;**

**double value = 0d;**

**for (; i < 100; i++)**

**{**

**value += delta;**

**}**

**Console.WriteLine(value);**

**Console.ReadLine();**

**}**

1. Evidentemente no compila
2. 10
3. Casi 10
4. No termina
5. 1

**Ejercicio 5** Defina y explique con sus palabras y de un ejemplo de los siguientes elementos:

Propiedades métodos y eventos de los controles más importantes

El formulario

Control etiqueta

Control cuadro de texto

Control Botón

Control cuadro combinado

Control lista

Control imagen

Control temporizador

Control menú

Acceso a datos con ADO.NET desde C#

Control DataGridView

Depuración de aplicaciones

Creación de un programa de instalación para la aplicación

Declaración de una clase

Propiedades o atributos de una clase

Métodos de una clase

Utilización de las clases.

Clases instanciadas y no

Constructores y destructores

Herencia

Polimorfismo

Funciones para el manejo de arrays

Funciones para el manejo de cadenas

Funciones para la conversión de tipos

El entorno de trabajo

Controles.

Dibujar los controles en el formulario

Manipulación de los controles visuales

Propiedades de los controles

Definir los controladores de eventos

Manipulación de los controles desde código

Tipos de datos y trabajo con variables

Tipos de datos básicos

Las variables en C#

Declaración de una variable

Alcance de las variables

Literales

Operadores y expresiones

Constantes

Arrays

Estructuras de control

Sentencias de control: ¿Para qué sirven?

Condiciones. Operaciones de comparación

Condiciones complejas.

Operadores booleanos

Sentencia If

Sentencias If más complejas

Sentencia switch

Estructuras de repetición o bucles

Sentencia for

Sentencia foreach

Sentencia while

Common Language Runtime (CLR)

Microsoft Intermediate Language (MSIL)

NameSpaces

Explorador de soluciones

Panel de propiedades

¿Qué es un proyecto?

Tipos de proyectos

**Ejercicio 6** Crear el programa en C# que resuelva los siguientes problemas.

1. Dado un monto ingresado desde teclado. Calcular el descuento considerando que si el monto está por encima de 100, el descuento es del 10%, y por debajo de 100, el descuento es del orden del 2%.

2. Calcular mediante un algoritmo repetitivo la suma de los primeros N numeros Naturales. Siendo N un dato ingresado desde teclado.

3. Dadas N notas de un estudiante, calcular:

* Cantidad de notas desaprobadas
* Cantidad de notas aprobadas
* Promedio de notas
* Promedio de notas aprobadas y Promedio de notas desaprobadas