# Test di valutazione – Modulo 1

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Nome | Serena |
|  |  | Cognome | Curti |
|  |  | Data | 11/10/2019 |

Leggete attentamente ogni domanda e argomentare quanto più possibile fornendo anche degli esempi.  
ATTENZIONE: Le domande a risposta multipla possono contenere più risposte corrette.

1. *Quali tra i seguenti è un IDE? Per quelli che non sono IDE, spiegare per quale motivo non è ritenuto tale*

Notepad - \_\_\_\_\_Non ha le funzionalità che permettono la scrittura intelligente del codice\_\_\_\_\_\_

Visual Studio Code - \_Rispetto a Visual Studio ha delle funzionalità limitate per questo è definito editor \_\_

.NET - \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Visual Studio - \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Cos’è una funzione?

Un blocco di codice che esegue determinate istruzioni

È un’eccezione non gestita

È una lista di elementi

È una struct con determinati livelli di accesso

1. *Cosa sono le proprietà, le costanti e le variabili? Dare una descrizione dettagliata di ciascun elemento.*

Le proprietà, nella programmazione orientata agli oggetti, rappresentano le informazioni degli oggetti stessi, dei dati sui quali poter far eseguire delle elaborazioni.

Variabili e costanti fanno parte degli attributi di una classe, entrambi sono contenitori che occupano dello spazio nella memoria e al loro interno possono essere inseriti dei valori. La differenza tra le due è che nelle variabili i valori inseriti possono essere modificati nel corso dell’esecuzione del programma, al contrario i valori delle costanti non possono essere modificati.

1. *Cosa sono gli operatori logici e a cosa servono? Fornire una serie di esempi di utilizzo in codice C#.*

Gli operatori logici permettono di eseguire delle operazioni tra gli elementi A e B che hanno una relazione. Ne esistono diversi in base al bisogno, per esempio di uguaglianza == , disuguaglianza != , condizionali && (e), || (oppure), di incremento ++ , decremento -- , aritmetici + , - , \* , / .

Per esempio se in un ciclo *for* vogliamo che ogni volta venga incrementato il valore i:

for (int i = 0 ; i < rubrica.Length ; i ++ )

Possiamo creare una moltiplicazione tra due interi A e B:

public static int Mult(int A , int B)

{

return A + B;

}

1. *Elencare quante più differenze possibili da .NET Core e .NET Framework?*

.NET Framework è il più vecchio, viene preinstallato su Windows gira solo su questo sistema operativo mentre .NET Core gira su Windows, Linux e Mac, è quindi multipiattaforma e open source.

Esistono però delle tecnologie su .NET Framework che non sono disponibili su .NET Core come per esempio Web Form ASP.NET. Entrambi devono sottostare e implementare i contratti presenti in .NET Standard Library.

1. *Qual è la differenza tra una proprietà, un metodo o un campo “private” e “public” in una classe?*

Ogni classe gode di ereditarietà, ovvero tutto ciò che sta all’interno della classe *veicoli a motore* per ereditarietà passera nella sua sottoclasse *automobile.* Grazie a questa caratteristica è possibile quindi specificare se l’oggetto all’interno di una classe sarà pubblico o privato. Se sarà pubblico potrà essere letto e richiamato anche nelle sottoclassi o sopraclassi, al contrario se sarà privato non potrà essere visto dalle altre classi neanche per ereditarietà.

1. *Quali sono le differenze tra Do-While, While, For, For-Each e Switch? Fornire una descrizione ed un esempio di ciascuno di essi*

Il *Do-While* crea un ciclo che viene eseguito uno o più volte, diverso dal *While* che crea un ciclo che può essere eseguito zero o più volte. Viene inserita in entrambi una condizione che se viene restituita come vera (true) l’esecuzione continua con la prima istruzione del ciclo, se invece viene restituita come falsa (false) l’esecuzione continua con la prima istruzione dopo il ciclo.

do

{

readLine = reader.ReadLine();

if (readLine != null)

datiDiUscita.Add(readLine);

}

while (readLine != null);

Il *For* esegue un’istruzione nel quale è necessario indicare una condizione nel quale indichiamo anche per quante volte deve essere eseguita l’istruzione. Nel caso in esempio l’istruzione verrà eseguita incrementando sempre di uno, per tutta la lunghezza dell’elemento.

for (var index = 0; index < rubrica.Count; index++)

{

Console.WriteLine($" => {rubrica[index].FirstName}, {rubrica[index].LastName}");

//Console.WriteLine(" => " + rubrica[index].FirstName + ", " + rubrica[index].LastName);

}

Lo *Switch* esegue invece solo una tra le istruzioni che ha al suo interno. Nell’esempio sarà l’utente a dare il valore che che permetterà al programma di capire quale istruzione dovrà eseguire. Dopo ogni Case è infatti necessario inserire anche il Break che permette l’uscita dall’istruzione. In tutti gli altri casi precedenti il break può essere usato per l’uscita forzata dal ciclo in riferimento.

switch (selezione)

{

case 1:

FunzioniMatematiche.RecuperaDivisioneEDividendoEDividi();

break;

case 2:

FunzioniRubrica.InserisciPersoneEMostraRubrica();

break;

case 3:

FunzioniRubrica.InserisciNumeroArbitrarioPersoneInRubrica();

break;

case 4:

FunzioniFileSystem.CreaStrutturaPerConservazioneDati();

break;

case 5:

FunzioniArray.RiempiArrayECopia();

break;

case 0:

Console.WriteLine("Uscita....");

break;

default:

Console.WriteLine("Selezione non valida");

break;

}

1. *Cosa sono le eccezioni, a cosa servono e quando non dovrebbero essere mai utilizzate?*

Le eccezioni sono degli imprevisti, degli errori che possono verificarsi durante l’esecuzione del programma. Per gestire le eccezioni vengono solitamente usati *try*  e *catch* così da provare ad eseguire delle azioni. Inserire le eccezioni è importante perchè nel momento in cui si presenta il problema, se non ci fosse la prova il programma finirebbe con un errore. Inserendo il *try* possiamo quindi individuare cosa ha creato il nostro problema.

Non vanno utilizzate in qualsiasi contento altrimenti non si riesce a capire da dove deriva il nostro errore.

1. *Qual è la differenza tra un Array e una Lista? Fornire un esempio di ciascuno di essi, come si esegue la creazione e la popolazione di queste strutture.*

Un Array è un insieme di elementi che devono necessariamente essere dello stesso tipo, inoltre la grandezza dell’Arrey deve essere determinata nel momento in cui lo si crea.

Al contrario, una Lista è un insieme di elementi generici dei quali può essere determinato il tipo, mentre la grandezza della Lista può essere modificata durante l’esecuzione del programma.

Per creare un Array dobbiamo quindi specificare il tipo, poi il nome dell’Array e la sua grandezza:

int CreoArray[ ] = new int[ 10 ]; 🡨 abbiamo così dato il comando di creare in nuovo Arrey

Per riempire un Array è possibile creare un ciclo *for* che legga da *Console* e per ogni *input* inserisca, per tutta la lunghezza dell’Aarray, progredendo ogni volta di una posizione il dato che è stato letto.

for (var i = 0; i < 10; i++)

{

var valore = Console.ReadLine();

CreoArray[i] = valore;

}

Per creare una lista invece possiamo dichiarare il nome e il tipo di variabile che verrà inserita al suo interno: List<string> = new List<string>

1. *Esercitazione:*

*Si chiede di creare un app console in .NET Framework Core 3.0 che in fase di esecuzione richieda l’inserimento di un numero compreso da 1 e 10; dopo la lettura del numero, dovranno essere creati un pari numero di oggetti “prodotto” (classe Product), ciascuno dei quali caratterizzato da codice (alfanumerico, campo “Code”) e nome (campo “Name”). Una volta terminato l’inserimento dei prodotti, gli stessi devono essere stampati a video e scritti all’interno di un file di testo in modo tale da poter essere facilmente riletti (la funzione di rilettura non è richiesta).*

[Tot: \_\_/10]