Code kata - demo

Un *code kata* è un esercizio di codifica rivolto a un problema ben definito, di norma risolvibile in poche righe di codice con un metodo, o al più un piccolo programma. Lo scopo è quello di migliorare le proprie capacità di codifica e prendere confidenza con gli elementi basilari del linguaggio.

Requisiti

Se non diversamente specificato, ogni *code kata* deve produrre un metodo (questo può usare altri metodi). Se sono richieste più versioni deve essere prodotto un metodo per ogni versione.

Prima di eseguire il *code kata* deve essere scritto il codice necessario per verificarne il corretto funzionamento.

Dimostrazione

Di seguito viene mostrato un *code kata*. L'esercizio viene proposto e risolto nel seguente modo:

- **Problema**: una semplice descrizione del problema da risolvere.
- **Esempio**: un esempio di dati di input e del risultato atteso.
- **Variazioni**: indicazioni su soluzioni alternative, o che si basino su vincoli aggiuntivi.
- **Soluzione dimostrativa**:
    - **Test**: Un metodo che verifica la correttezza della soluzione realizzata.
    - **Implementazione (o implementazioni) della soluzione.**

1 Invertire una sequenza di numeri

Problema

Dato un vettore di interi, restituire un nuovo vettore con gli stessi elementi in posizione inversa.

Esempio:

```
int[] dati = {5, 3, 7, 9};
// -> 9, 7, 3, 5
```

Variazioni

Fornire versioni distinte, usando i cicli `for`, `while` e `foreach`.

Soluzione dimostrativa

Scrivo prima il metodo che serve a testare il corretto funzionamento del codice:

```
static void Main(string[] args)
{
    TestInversioneSequenza();
}

//metodo di test
static void TestInversioneSequenza()
{
    int[] dati = { 5, 3, 7, 9 };
    //-> 9, 7, 3, 5
    int[] risultato = InvertiSequenza(dati);

    foreach (var item in risultato)
    {
        Console.Write("{0}, ", item);
    }
}
```

Implemento una versione del metodo che compila, anche se non produce ancora il risultato atteso:

```
private static int[] InvertiSequenza(int[] dati)
{
    return dati;
}
```

Fin qui l'obiettivo è stato quello di preparare le condizioni per implementare la soluzione nel modo più semplice e rapido possibile. Nota bene: non scrivo istruzioni per l'input dei dati, poiché fanno perdere tempo e rendono difficile verificare il funzionamento del metodo.

Implementazione con ciclo for

Segue un'implementazione che usa due indici; uno per scorrere i dati, l'altro per scorrere il vettore `risultato` in ordine inverso.

```
static int[] InvertiSequenza(int[] dati)
{
    int[] risultato = new int[dati.Length];
    int j = dati.Length-1;
    for (int i = 0; i < dati.Length; i++)
    {
        risultato[j] = dati[i];
        j--;
    }
    return risultato;
}
```

Segue una versione con ciclo `while` e un solo indice:

```
static int[] InvertiSequenza2(int[] dati)
{
    int[] risultato = new int[dati.Length];
    int i = 0;
    while (i < dati.Length)
    {
        risultato[dati.Length-i-1] = dati[i];
        i++;
    }
    return risultato;
}
```

Infine con ciclo `foreach`:

```
static int[] InvertiSequenzaConForeach(int[] dati)
{
    int[] risultato = new int[dati.Length];
    int i = dati.Length - 1;
    foreach (var item in dati)
    {
        risultato[i] = item;
        i--;
    }
    return risultato;
}
```

La realizzazione di più varianti ha un triplice scopo:

- Acquisire una solida comprensione del problema.
- Acquisire confidenza con i fondamentali del linguaggio.
- Provare a migliorare il codice, cercando soluzioni più semplici ed eleganti.