Code kata - demo

Un *code kata* è un esercizio di codifica rivolto a un problema specifico, spesso risolvibile in poche righe di codice. Lo scopo è quello di migliorare le capacità di codifica e acquisire confidenza con gli elementi basilari del linguaggio.

Se non diversamente specificato, ogni *code kata* deve essere risolto mediante un metodo (che può usare altri metodi). Se sono richieste più versioni – varianti – del *code kata*, deve essere prodotto un metodo per ogni versione.

Svolgimento del code kata

Un *code kata* si svolge in due parti: *preparazione* e *soluzione*.

La prima, chiamata anche *setup*, serve a creare la condizioni per necessarie per poter affrontare correttamente la soluzione: un *metodo di test*, che sarà usato per testare la soluzione. Questo dovrà definire i dati di input, quali dati di output ci si aspetta (sotto forma di commento) e il codice di visualizzazione dell'output.

Durante il *setup* deve essere creato lo "scheletro" del metodo che rappresenta la soluzione, *prima di provare a risolvere il problema!* Soltanto dopo che il *setup* è stato completato, (e il codice compila) si potrà procedere alla fase di soluzione.

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**:
    - **Setup**:
        - Realizzazione di un metodo che verifica la correttezza della soluzione realizzata.
        - Creazione dello scheletro del metodo che rappresenterà la soluzione.
    - **Implementazione 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

Setup

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

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

//setup: metodo di test
static void TestInversioneSequenza()
{
    int[] dati = { 5, 3, 7, 9 };
    //-> 9, 7, 3, 5             (scrivo l'output atteso)

    // scrivo la chiamata al metodo che risolve il code kata, PRIMA di cominciare a
    // implementarlo!
    int[] risultato = InvertiSequenza(dati);

    foreach (var item in risultato)  // visualizzo output per verificare la soluzione
    {
        Console.Write("{0}, ", item);
    }
}
```

Creo lo scheletro del metodo risolutivo, il quale deve compilare, anche se non produce il risultato atteso:

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

Nota bene: non esiste input dall'utente.

Soluzione (implementazione con ciclo for, ciclo while e foreach)

L'implementazione che segue 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.