

# Ricorsione

## Numero triangolare

### Iterativa

```
public class TriangularNumbers {
    // Metodo iterativo per calcolare il numero triangolare
    public static int triangularIterative(int n) {
        int sum = 0;
        for (int i = 1; i ≤ n; i++) {
            sum += i;
        }
        return sum;
    }

    // Metodo main per testare la funzione iterativa
    public static void main(String[] args) {
        int n = 5; // Cambia questo valore per testare con un numero
        diverso
        System.out.println("Il numero triangolare di " + n + " è: " +
        triangularIterative(n));
    }
}
```

### Ricorsiva

```
public class TriangularNumbers {
    // Metodo ricorsivo per calcolare il numero triangolare
    public static int triangularRecursive(int n) {
        if (n == 0)
            return 0;
        else
            return n + triangularRecursive(n - 1);
    }

    // Metodo main per testare la funzione ricorsiva
    public static void main(String[] args) {
```

```

        int n = 5; // Cambia questo valore per testare con un numero
diverso
        System.out.println("Il numero triangolare di " + n + " è: " +
triangularRecursive(n));
    }
}

```

## Parole palindrome

### Iterativa

```

public class Palindrome {
    // Metodo iterativo per verificare se una parola è un palindromo
    public static boolean isPalindromeIterative(String word) {
        int start = 0;
        int end = word.length() - 1;
        // Continua fino a quando start è minore di end
        while (start < end) {
            // Se i caratteri alle posizioni start ed end non
corrispondono, non è un palindromo
            if (word.charAt(start) != word.charAt(end))
                return false;
            // Avanza start e diminuisci end
            start++;
            end--;
        }
        // Se il ciclo è completato senza ritornare false, allora è un
palindromo
        return true;
    }

    // Metodo main per testare la funzione iterativa
    public static void main(String[] args) {
        String word = "radar"; // Cambia questa parola per testare con una
diversa
        if (isPalindromeIterative(word))
            System.out.println(word + " è un palindromo.");
        else
            System.out.println(word + " non è un palindromo.");
    }
}

```

```
}
```

## Ricorsiva

```
public class Palindrome {
    // Metodo ricorsivo per verificare se una parola è un palindromo
    public static boolean isPalindromeRecursive(String word) {
        // Caso base: una stringa vuota o di lunghezza 1 è un palindromo
        if (word.length() ≤ 1)
            return true;
        // Verifica se il primo e l'ultimo carattere sono uguali
        if (word.charAt(0) ≠ word.charAt(word.length() - 1))
            return false;
        // Chiamata ricorsiva con la sottostringa escludendo il primo e
        // l'ultimo carattere
        return isPalindromeRecursive(word.substring(1, word.length() -
1));
    }

    // Metodo main per testare la funzione ricorsiva
    public static void main(String[] args) {
        String word = "radar"; // Cambia questa parola per testare con una
        // diversa
        if (isPalindromeRecursive(word))
            System.out.println(word + " è un palindromo.");
        else
            System.out.println(word + " non è un palindromo.");
    }
}
```

## Conteggio numeri

### Iterativo

```
public class IterativeCounter {
    // Metodo iterativo per contare da 1 a N
    public static void countIterative(int n) {
        for (int i = 1; i ≤ n; i++) {
            System.out.print(i + " ");
        }
    }
}
```

```

    }
}

// Metodo main per testare la funzione iterativa
public static void main(String[] args) {
    int N = 5; // Cambia questo valore per testare con un numero
diverso
    System.out.print("Conteggio iterativo da 1 a " + N + ": ");
    countIterative(N);
}
}

```

## Ricorsivo

```

public class RecursiveCounter {
    // Metodo ricorsivo per contare da 1 a N
    public static void countRecursive(int n) {
        if (n > 0) {
            countRecursive(n - 1);
            System.out.print(n + " ");
        }
    }

    // Metodo main per testare la funzione ricorsiva
    public static void main(String[] args) {
        int N = 5; // Cambia questo valore per testare con un numero
diverso
        System.out.print("Conteggio ricorsivo da 1 a " + N + ": ");
        countRecursive(N);
    }
}

```

```

1  class Ricorsione {
2      public void contatore(int n) {
3          System.out.println(n);
4
5          if(n>0)
6              contatore(n-1);
7
8          System.out.println(n);
9      }
10
11     static int fatt(int n) {
12         int f;
13         if (n==0)
14             f=1;                /* condizione di chiusura */
15         else
16             f=n*fatt(n-1);        /* passo induttivo */
17
18         return(f);
19     }
20
21     static int minimo(int a[],int i,int min,int n){
22         if(i == n)
23             return min;
24         if(a[i] < min)
25             return minimo(a, i+1, a[i], n);
26
27         return minimo(a, i+1, min, n);
28     }
29 }
30
31
32 public class Main{
33     // ESERCIZIO CHE ISTANZIA LA CLASSE Ricorsione
34
35     // Metodo main con il parametro formale (String[] args)
36     public static void main(String[] args) {
37         // Istanza della classe Ricorsione
38         Ricorsione Conta = new Ricorsione();
39
40         int n = 15;
41         System.out.println("Contatore");
42         Conta.contatore(n);
43         System.out.println();
44
45
46         //Dichiaro le variabili necessarie iniziali
47         final int NMAX = 5;
48         int[] a = new int[NMAX];
49
50         //POPOLOAMENTO
51         System.out.println("Popolamento array");
52         a[0] = 10;
53         a[1] = 3;
54         a[2] = 1;
55         a[3] = 9;
56         a[4] = 2;
57
58         n = NMAX;
59         int i =0;
60         int min=a[0];
61         min = Conta.minimo(a, i, min, n);
62         System.out.println("Il minimo trovato è " + min);
63         System.out.println();
64
65         n = NMAX;
66         int ris = Conta.fatt(n);
67         System.out.println("Il fattoriale di " + n + " risulta " + ris);
68
69     }
70 }

```

# La ricorsione

---

La **ricorsione** è una tecnica di programmazione molto potente, che sfrutta l'idea di **suddividere un problema da risolvere in sottoproblemi simili a quello originale**, ma più semplici.

Un metodo si dice **ricorsivo** quando all'interno della propria definizione compare una **chiamata direttamente al metodo stesso**.

Un **algoritmo ricorsivo** per la risoluzione di un dato problema deve essere definito nel modo seguente:

1. prima si definisce come risolvere dei problemi analoghi a quello di partenza, ma che hanno "dimensione piccola" e possono essere risolti in maniera estremamente semplice (detti **casi base**);
  2. poi si definisce come ottenere la soluzione del problema di partenza combinando la soluzione di uno o più problemi analoghi, ma di "dimensione inferiore" (ricorsione).
- Si parla di **ricorsione indiretta** quando nella definizione di un metodo compare la chiamata ad un altro metodo il quale direttamente o indirettamente chiama il metodo iniziale.
  - Un metodo implementa una **ricorsione multipla** quando all'interno della propria definizione compare la **chiamata direttamente al metodo stesso almeno due volte**.
  - In una **ricorsione infinita** vengono attivate infinite **istanze di un metodo**. Come abbiamo già detto, ciò si verifica perché i valori del parametro non si semplificano, o perché **manca la clausola di chiusura** per terminare.

## Ogni algoritmo ricorsivo è iterativo

Un algoritmo ricorsivo: è un algoritmo che è definito in termini di se stesso; è composto

- da un passo induttivo e
- da una condizione di chiusura (base).

Il passo induttivo definisce una funzione per un generico valore di una variabile  $n$  in termini della funzione di  $n-1$ .

La condizione di chiusura definisce la funzione in  $n=0$ . E allora chiamando la funzione per un valore si possono calcolare in cascata tutti i risultati intermedi fino a  $n=0$ .

Un tipico algoritmo che è facilmente espresso in forma ricorsiva è la definizione di **fattoriale**:

```
integer f=fatt(n)
{
  if n==0
    f=1;           /* condizione di chiusura */
  else
    f=n*fatt(n-1); /* passo induttivo */
  end
  return(f);
}
```

Sviluppando passo per passo ad esempio il fattoriale di 5 si ha:

```
fatt(5) =5*fatt(4)
        =5*4*fatt(3)
        =5*4*3*fatt(2)
        =5*4*3*2*fatt(1)
        =5*4*3*2*1*fatt(0)
        =5*4*3*2*1*1=5!
```

La ricorsione pur permettendo codici molto compatti ed eleganti è di solito poco efficiente perché ogni funzione chiamata alloca memoria nello stack. Quindi è possibile, se non consigliabile, trasformare l'algoritmo in forma iterativa. Nel caso del fattoriale si ha

```
integer f=fattoriale(n)
{
  f=1;
  for i=1:n
    f=f*i;
  end
  return(f)
}
```

## Ogni algoritmo iterativo è ricorsivo

Prendiamo un qualunque algoritmo iterativo:

```
for i=1:N
{
  blocco istruzioni;
}
end
```

posso definire una funzione ricorsiva

```
void ciclo(integer i)
{
  if i>N
    /* condizione di uscita */
    return;
  else
    blocco istruzioni;
    ciclo(i+1);
  end
}
```

Naturalmente a seconda del ciclo dovrò scegliere opportunamente la condizione di chiusura.

## Quando utilizzarla

### PRO

Spesso la ricorsione permette di risolvere un problema anche molto complesso con poche linee di codice

### CONTRO

La ricorsione è poco efficiente perché richiama molte volte una funzione e questo:

- richiede tempo per la gestione dello stack (allocare e passare i parametri, salvare l'indirizzo di ritorno, e i valori di alcuni registri della CPU)
- consuma molta memoria (alloca un nuovo stack frame ad ogni chiamata, definendo una nuova ulteriore istanza delle variabili locali non static e dei parametri ogni volta)

## CONSIDERAZIONE

Qualsiasi problema ricorsivo può essere risolto in modo non ricorsivo (ossia iterativo), ma la soluzione iterativa potrebbe non essere facile da individuare oppure essere molto più complessa

## CONCLUSIONE

Quando non ci sono particolari problemi di efficienza e/o memoria, l'approccio ricorsivo è in genere da preferire se:

- è più intuitivo di quello iterativo
- la soluzione iterativa non è evidente o agevole

## Esempio

Supponiamo di voler calcolare l'*area* di una forma triangolare di dimensione  $n$  come quella riportata sotto, nell'ipotesi che ciascun quadrato `[]` abbia area unitaria. Il valore dell'area corrispondente viene a volte chiamato *numero triangolare  $n$ -esimo*.

Ad esempio, per  $n=4$  il triangolo è fatto come segue:

```
[]  
[] []  
[] [] []  
[] [] [] []
```

Osservando questo schema possiamo affermare che il quarto numero triangolare è 10.

Supponiamo di voler scrivere un metodo che dato  $n$ , calcola il *numero triangolare  $n$ -esimo*.

```
public static int numeroTriangolare(int n) {  
    int result;  
  
    ....  
  
    return result;  
}
```

Ragioniamo per approssimazioni successive.

Se l'ampiezza del triangolo è 1, il triangolo consiste di un unico quadrato `[]` e la sua area è uguale a 1.

```
public static int numeroTriangolare(int n) {  
    int result;  
    if (n == 1)  
        result = 1; // caso base  
  
    .....  
  
    return result;  
}
```

Per trattare il caso più generale, consideriamo questa figura:

```
[]  
[] []  
[] [] []  
[] [] [] []
```

Supponiamo adesso di conoscere l'area del triangolo più piccolo, formato dalle prime tre righe: il *terzo numero triangolare*. In questo caso possiamo calcolare facilmente l'area del triangolo grande: il *quarto numero triangolare*:



**risultato** = **n** + **numero\_triangolare\_di\_n-1**;

Come possiamo calcolare il valore di **numero\_triangolare\_di\_n-1**?

Facciamolo calcolare a (una diversa istanza de) lla funzione **numeroTriangolare()** che stiamo definendo.

**risultato** = **n** + **numeroTriangolare(n-1)**;

```
public static int numeroTriangolare(int n) {  
    int result;  
    if (n == 1)  
        result = 1; // caso base  
    else  
        result = n + numeroTriangolare(n - 1); // ricorsione  
  
    return result;  
}
```

## Esercizio

Scrivere una funzione ricorsiva che controlla se una stringa è palindroma (ovvero se “rigirandola” non cambia, es. “ossesso” è palindroma).

Esempi di frasi palindrome:

- I verbi brevi (“iverbibrevi”)
- Aceto nell'enoteca (“acetonellenoteca”)
- I topi non avevano nipoti (“itopinonavevanonipoti”)

## Soluzione

Definizione ricorsiva di palindromicità:

- Una stringa nulla è palindroma. Esempio: “”.
- Una stringa di un carattere è palindroma. Esempio: “a”.
- Una stringa tale che il primo e l'ultimo carattere sono uguali e la sottostringa nel mezzo è palindroma, è palindroma. Esempio: “ossesso”.

## Algoritmo

```
bool palindroma(string s, int start, int len)  
{  
    if (len <= 1) return true;  
    if (s[start] == s[start+len-1] && palindroma(s, start+1, len-2))  
        return true;  
    else  
        return false;  
}
```

## Esercizio

Scrivere una funzione ricorsiva che, avendo in input un array di n interi di interi positivi, dia in output TRUE se tutti gli elementi sono maggiori di 10, FALSE altrimenti.

Soluzione:

```
boolean tutti(int A[], int n)  
    if (n=0) return TRUE  
    else if (A[n] <= 10 ) return FALSE  
    else return tutti(A, n-1)  
end
```