

FONDAMENTI DI INFORMATICA

Alma Artis
Francesca Pratesi (ISTI, CNR)

Iterazioni indeterminate

ESERCIZI

- 1) Si scriva una funzione con un parametro n che calcoli e restituisca in output la somma dei numeri pari nell'intervallo $(0, n]$. Ad esempio, se $n = 11$, la funzione deve restituire in output $2 + 4 + 6 + 8 + 10 = 30$.
2) Invocare la funzione definita al punto precedente usando come parametro un numero intero arbitrario (cioè, scelto a piacere). Si stampi quindi il risultato ottenuto.
- Scrivere una funzione con due parametri numerici interi n ed m . La funzione deve verificare se n è divisibile per tutti gli interi tra 1 ed m . In caso positivo la funzione deve restituire *true*, altrimenti *false*. Si invochi la funzione precedentemente definita, passando come parametri attuali i valori: (10, 2), (5, 3), (12, 4). Si stampi quindi il risultato ottenuto nei tre casi.
- Un numero primo è un numero intero maggiore di 1 che sia divisibile solo per 1 e per se stesso. Un modo elementare per verificare che un numero intero n sia un numero primo consiste nel verificare che n (per $n > 1$) non sia divisibile per alcun numero intero nell'intervallo $[2, n-1]$.
Scrivere una funzione *numero_primo*, con un parametro n numerico, che verifichi se n è un numero primo. La funzione deve restituire *true* se l'argomento è un numero primo, *false* altrimenti. Si invochi la funzione precedentemente definita, passando come parametri attuali i valori: 12, 3, 11, 673, 29. Si stampi quindi il risultato ottenuto nei diversi casi.
- Scrivere una funzione che prende un parametro numerico intero n . La funzione deve generare n numeri casuali e restituire *true* se almeno un numero è maggiore di 0,7, *false* altrimenti. Si invochi la funzione precedentemente definita.
- Scrivere una funzione che prende due parametri numerici interi n ed m . La funzione deve chiedere n numeri all'utente e calcolare quanti dei numeri letti sono maggiori di m . Si invochi la funzione precedentemente definita.
- Il fattoriale di un numero intero $n > 1$, indicato con $n!$, è uguale al prodotto dei numeri interi da 1 ad n . Cioè: $n! = 1 * 2 * 3 * \dots * n$.
Ad esempio, se $n = 4$, $n! = 1 * 2 * 3 * 4 = 24$.
Si scriva una funzione con un parametro n che calcoli e restituisca in output il valore del fattoriale di n .
Invocare la funzione definita al punto precedente usando come parametro un numero intero arbitrario. Si stampi quindi il risultato ottenuto.
- Scrivere una funzione che, dati due parametri interi positivi n ed m , calcoli e restituisca il risultato di n^m . Non si utilizzino funzioni di libreria, ma solo funzioni aritmetiche. Si invochi la funzione definita al punto precedente, usando come parametri attuali due numeri interi positivi scelti a piacere.



ESERCIZI 2

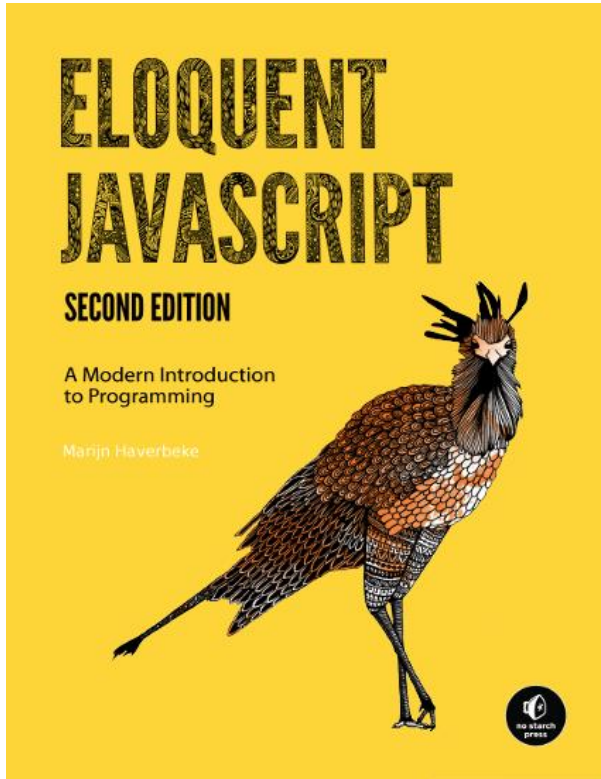
- Si scriva una funzione *random_intero*, con un parametro numerico intero *n*. La funzione deve generare e restituire in output un numero casuale intero compreso tra 1 ed *n*.
Si ricordi che:
Math.random() genera un numero decimale casuale in (0,1);
Math.ceil(x) calcola il numero intero più vicino ad *x* per eccesso.
- Scrivere un programma che chieda in input tre valori che rappresentano le lunghezze di tre lati di un triangolo; decidere se il triangolo è equilatero, isoscele o scaleno e stampare il risultato sulla console
 - *Idea 1: confrontare i lati a coppie fino a quando non si hanno informazioni sufficienti a decidere di che tipo è il triangolo*
 - *Idea 2: contare quante coppie di lati uguali ci sono e in base a questo identificare il tipo di triangolo*





ISTRUZIONI ITERATIVE

LIBRI E RIFERIMENTI



- Capitolo 2

Eloquent Javascript – Second Edition

Marijn Haverbeke

Licensed under CC license.

Available here: <http://eloquentjavascript.net/>

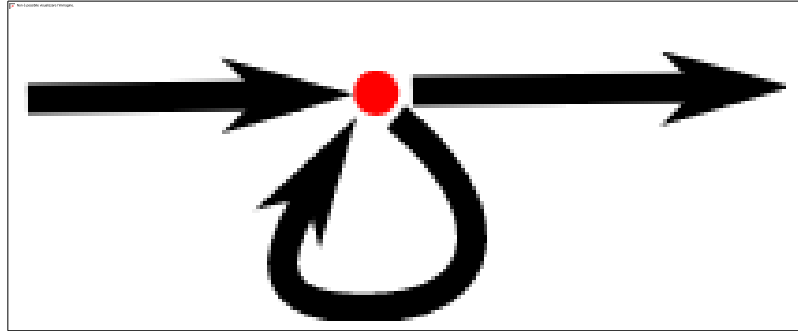
ISTRUZIONI ITERATIVE

- Molti problemi richiedono un **calcolo** che deve essere **ripetuto più volte** per ottenere il risultato finale.
- Le **istruzioni iterative** consentono di indicare la necessità di ripetere un calcolo più volte.



ISTRUZIONI ITERATIVE / CICLI

Un modo per eseguire un pezzo di codice (un insieme di istruzioni) ripetutamente



ciclo
(o loop)

TIPI DI ISTRUZIONI ITERATIVE



DETERMINATE

The diagram consists of two identical rectangular boxes side-by-side. Each box has a dark red background with a lighter red rounded rectangle in the center. The word 'DETERMINATE' is written in black capital letters inside the left box, and 'INDETERMINATE' is written in black capital letters inside the right box. The boxes are slightly offset from each other, with the right box appearing to be in front of the left one.

INDETERMINATE

ITERAZIONE DETERMINATA

Il **calcolo** viene ripetuto **un numero fissato di volte**.

Es.:

- fai 10 giri del parco di corsa
- leggi dall'input k numeri
- genera 5 numeri casuali



ITERAZIONE INDETERMINATA

Il **calcolo** viene ripetuto **finchè una condizione è vera**.

Es.:

- finchè non sei sazio mangia
- leggi un numero dall'input finchè non trovi un numero negativo
- ripeti l'esame di Fondamenti di Informatica fino a che il voto ≥ 18





ISTRUZIONI ITERATIVE: WHILE

ISTRUZIONE WHILE

Sintassi

Il corpo del ciclo può essere un blocco (in questo caso ci sono più istruzioni che vengono ripetute)

```
while (espressione)  
    istruzione_ciclo;
```

ripete istruzione_ciclo finché l'espressione è vera

```
while (espressione){  
    istruzione_blocco1;  
    istruzione_blocco2;  
    // etc...  
    istruzione_bloccoN;  
}
```



ISTRUZIONE WHILE

Sintassi

guardia del ciclo

↑

while (espressione)

istruzione; → corpo del ciclo

Semantica:

- 1) Viene valutata l'espressione
- 2) Se è vera esegue il corpo del ciclo e ritorna al punto 1
- 3) Se è falsa si passa alla prima istruzione dopo il while

ISTRUZIONE FOR: ESEMPIO

```
var condizione = true, numero;  
while (condizione){  
    numero = Number(prompt('inserisci un numero'));  
    if (numero<0) condizione=false;  
}
```



ISTRUZIONE FOR VS WHILE

Con il while è ancora possibile testare quante iterazioni abbiamo fatto, quindi di fatto è possibile usare un while come un for.

Le differenze sono:

- nella sintassi → dobbiamo ricordarci di inizializzare il contatore e di incrementarlo, altrimenti generiamo un loop infinito!
- nell'uso con cui sono stati pensati



ISTRUZIONE FOR VS WHILE

Esempio

```
var i, k = 10;  
for (i=1; i<=k; i++){  
    //istruzioni del ciclo  
}
```

```
var i=1, k = 10;  
while (i<=k){  
    //istruzioni del ciclo  
    i++;  
}
```



ITERAZIONE INDETERMINATA - ESPRESSIONE

Nota bene: l'espressione viene valutata all'**inizio** di ogni iterazione



ITERAZIONE INDETERMINATA – DO WHILE

```
do  
    istruzione;  
while (espressione);
```

Con il do-while l'espressione viene valutata alla **fine** di ogni iterazione

Nota bene: il corpo del while viene eseguito almeno una volta



ITERAZIONE INDETERMINATA – DO WHILE

```
do istruzione;  
while (espressione);
```

Semantica:

- 1) Viene eseguito il corpo del ciclo
- 2) Viene valutata l'espressione
- 3) Se è vera si ritorna al punto 1
- 4) Se è falsa si passa alla prima istruzione dopo il do-while



ESEMPIO

Scrivere una funzione che generi un numero casuale tra -1 e 1 finché non viene generato un numero positivo, restituendo quindi il numero positivo ottenuto.



ESEMPIO

```
function random_positivo(){  
  var numero;  
  do  
    numero = 2*Math.random()-1;  
  while(numero<=0);  
  
  return numero;  
}
```



ESEMPIO 2 (CONTROLLA LE DIFFERENZE)

```
function random_positivo_while(){  
  var numero;  
  numero =2*Math.random()-1;  
  while(numero <=0)  
    numero =2*Math.random()-1;  
  
  return numero;  
}
```



ESEMPIO

Scrivere una funzione che legga in input dei valori numerici, arrestandosi quando incontra il valore di un terminatore (parametro della funzione). La funzione deve restituire quanti numeri sono stati letti prima del terminatore.



ESEMPIO

```
function lunghezza_sequenza(t){  
  var numero;  
  var l =0;//quanti numeri ho letto diversi da t  
  do{  
    numero = Number(prompt());//leggo un numero  
    l++;//aggiorno il contatore  
  }while(numero!=t);//itero finche' non trovo il terminatore  
  
  /*quando esco dal ciclo do-while so che l'ultimo numero letto  
  e' il terminatore, quindi la lunghezza della sequenza va  
  decrementata di 1*/  
  return(l-1);  
}
```


ESEMPIO

con do-while:

```
function lunghezza_sequenza(t){  
  var numero;  
  var l = 0;  
  
  do{  
    numero = readnum();  
    l++;  
  }while (numero!=t);  
  
  return (l-1);  
}
```

con while:

```
function lunghezza_sequenza2(t){  
  var numero;  
  var l = 0;  
  
  numero = readnum();  
  while (numero!=t){  
    l++;  
    numero = readnum();  
  }  
  return l;  
}
```

COME SI ESCE DA UN CICLO

- Normalmente:
 - Un ciclo viene ripetuto finché la guardia non diventa falsa
- Ma c'è un altro modo:
 - L'istruzione break consente di uscire da un ciclo immediatamente



ESEMPIO

- Scrivere una funzione che generi al massimo n numeri casuali fino a che non ne trova uno maggiore di una *soglia*.
- La funzione deve restituire quanti numeri casuali sono stati generati.



ESEMPIO

```
function maggiore_soglia_quanti(n,soglia){  
    var i;  
    var numero;  
    for(i =1;i<=n;i++){  
        numero=Math.random();  
        if(numero>soglia)  
            break;  
    }  
  
    return i;  
}
```



RICORSIONE E FUNZIONI RICORSIVE

FUNZIONI RICORSIVE

- Una funzione ricorsiva è una funzione che contiene una chiamata a se stessa

```
function recursive_function(n){  
    ...  
    ...  
    recursive_function(n-1);  
}
```



ASPETTI FONDAMENTALI

- Una definizione ricorsiva consta di due passi fondamentali:
 - **Passo base:** condizione in cui la funzione può restituire un valore. Non sono necessarie altre chiamate ricorsive. La ricorsione finisce qui.
 - **Passo ricorsivo (induttivo):** la funzione restituisce un valore che dipende da dati semplificati o ridotti. In questo caso c'è la chiamata ricorsiva.



RICORSIONE

- Ad ogni chiamata ricorsiva i dati a cui la funzione viene applicata vengono semplificati (es. si invoca la funzione con parametro $n-1$)
- Procedendo in questo modo si arriverà ad un punto corrispondente ad un caso base
- Una chiamata ricorsiva non è differente da una chiamata di funzione standard:
 - la funzione chiamante sospende la propria esecuzione per eseguire la nuova chiamata
 - l'esecuzione della funzione chiamante riprende quando la chiamata ricorsiva termina
 - c'è solitamente una cascata di chiamate ricorsive (fino al raggiungimento del passo base)
 - si applicano gli stessi concetti già visti a proposito dello stato



ESEMPIO: FATTORIALE

$$fattoriale(n) = \begin{cases} 1 & \text{se } n = 0 \\ n \cdot fattoriale(n - 1) & \text{se } n > 0 \end{cases}$$

← passo base

← passo ricorsivo


↑
i dati vengono semplificati

ESEMPIO: FATTORIALE

```
function fattoriale(n){  
  if (n==1)                ← passo base  
    return 1;  
  else  
    return n*fattoriale(n-1); ← passo induttivo  
}
```


↑
i dati vengono semplificati

ESEMPIO DI ESECUZIONE: FATTORIALE(4)

 n=4

```
function fattoriale(n){  
    if (n==1)  
        return 1;  
    else  
        return n * fattoriale(n-1);  
}
```

ESEMPIO DI ESECUZIONE: FATTORIALE(4)

 n=4

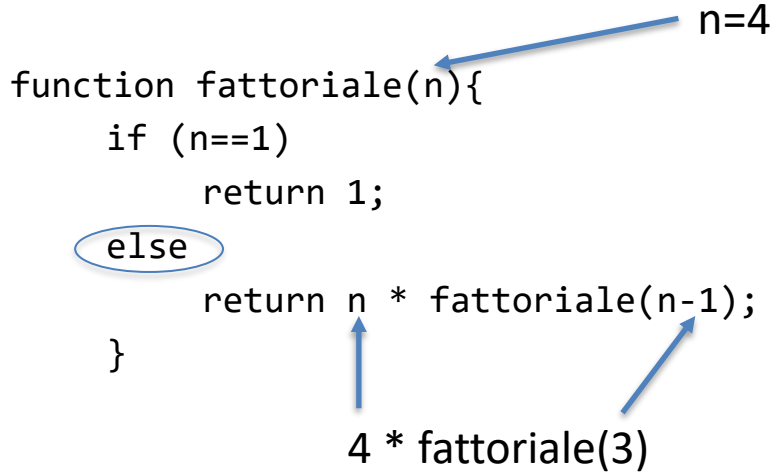
```
function fattoriale(n){  
    if (n==1)  
        return 1;  
    else  
        return n * fattoriale(n-1);  
}
```

ESEMPIO DI ESECUZIONE: FATTORIALE(4)


```
function fattoriale(n){  
    if (n==1)  
        return 1;  
    else  
        return n * fattoriale(n-1);  
}
```

$n=4$


$4 * \text{fattoriale}(3)$



ESEMPIO DI ESECUZIONE: FATTORIALE(4)


 n=4

```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```


 n=3

```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```

ESEMPIO DI ESECUZIONE: FATTORIALE(4)


 n=4


```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```


 n=3

```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```


ESEMPIO DI ESECUZIONE: FATTORIALE(4)


function fattoriale(n){
 if (n==1)
 return 1;
 else
 return n * fattoriale(n-1);
}



function fattoriale(n){
 if (n==1)
 return 1;
 else
 return n * fattoriale(n-1);
}


3 * fattoriale(2)


ESEMPIO DI ESECUZIONE: FATTORIALE(4)

 n=4

```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```


 n=3

```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```


 n=2

```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```


ESEMPIO DI ESECUZIONE: FATTORIALE(4)

 n=4

```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```

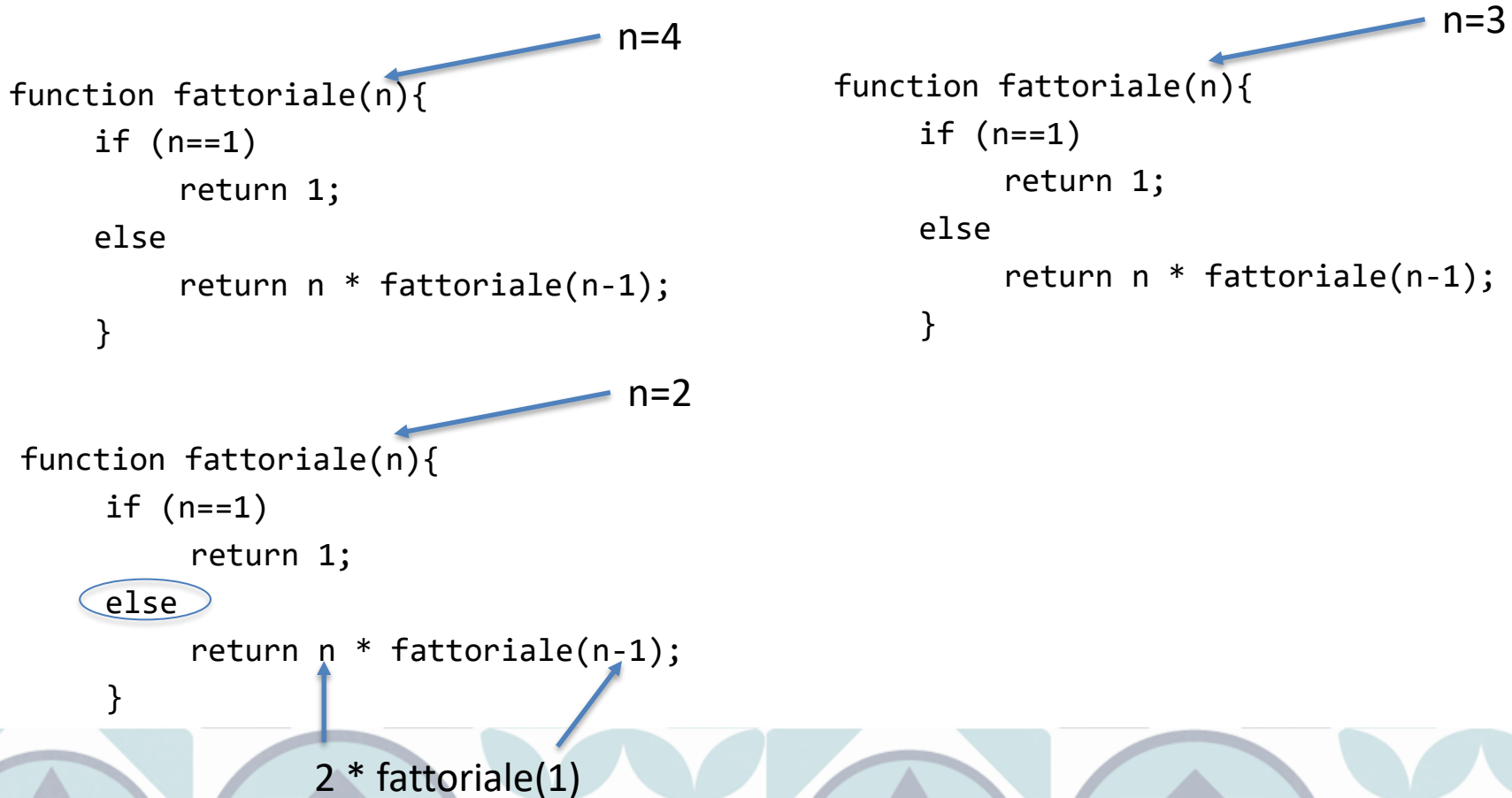
 n=3

```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```

 n=2

```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```

ESEMPIO DI ESECUZIONE: FATTORIALE(4)



function fattoriale(n){
 if (n==1)
 return 1;
 else
 return n * fattoriale(n-1);
}

function fattoriale(n){
 if (n==1)
 return 1;
 else
 return n * fattoriale(n-1);
}


function fattoriale(n){
 if (n==1)
 return 1;
 else
 return n * fattoriale(n-1);
}

2 * fattoriale(1)


ESEMPIO DI ESECUZIONE: FATTORIALE(4)

 n=4


```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```

 n=3

```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```


 n=2

```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```


 n=1

```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```


ESEMPIO DI ESECUZIONE: FATTORIALE(4)

 n=4


```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```

 n=3

```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```


 n=2


```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```


 n=1

```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```

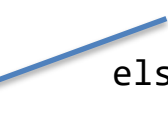
ESEMPIO DI ESECUZIONE: FATTORIALE(4)


function fattoriale(n){
 if (n==1)
 return 1;
 else
 return n * fattoriale(n-1);
}



function fattoriale(n){
 if (n==1)
 return 1;
 else
 return n * fattoriale(n-1);
}



function fattoriale(n){
 if (n==1)
 return 1;
 else
 return n * fattoriale(n-1);
}


function fattoriale(n){
 if (n==1)
 return 1;
 else
 return n * fattoriale(n-1);
}



ESEMPIO DI ESECUZIONE: FATTORIALE(4)

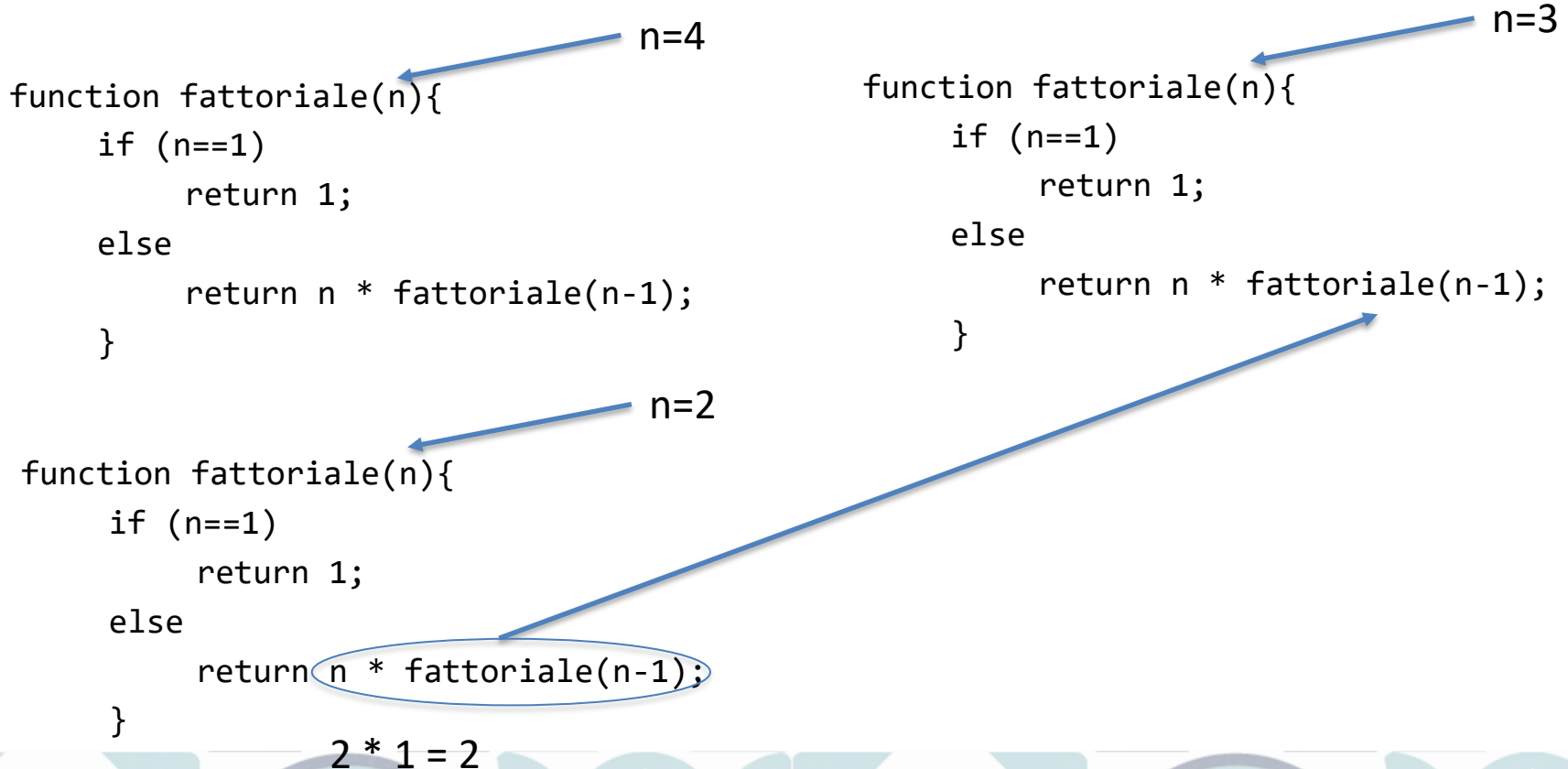

function fattoriale(n){
 if (n==1)
 return 1;
 else
 return n * fattoriale(n-1);
}


function fattoriale(n){
 if (n==1)
 return 1;
 else
 return n * fattoriale(n-1);
}



function fattoriale(n){
 if (n==1)
 return 1;
 else
 return n * fattoriale(n-1);
}


 $2 * 1 = 2$

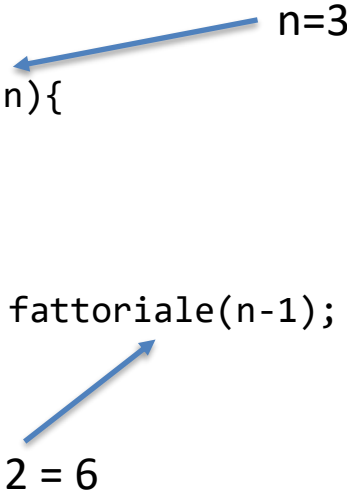
ESEMPIO DI ESECUZIONE: FATTORIALE(4)



ESEMPIO DI ESECUZIONE: FATTORIALE(4)

 n=4

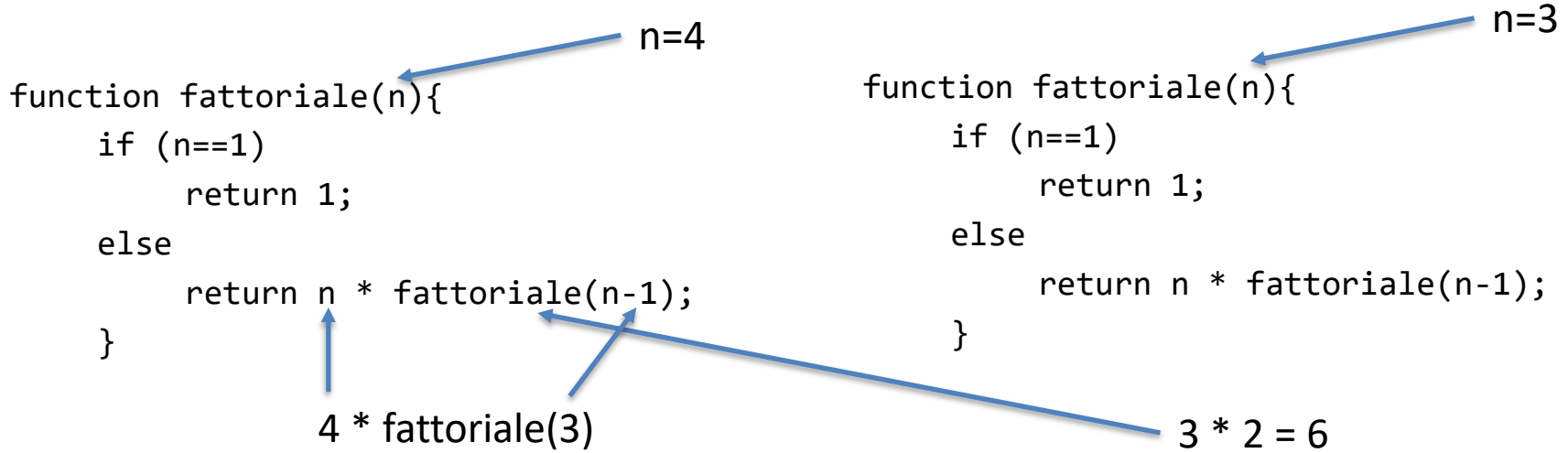
```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```

 n=3

```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```

$3 * 2 = 6$

ESEMPIO DI ESECUZIONE: FATTORIALE(4)



ESEMPIO DI ESECUZIONE: FATTORIALE(4)

```
function fattoriale(n){  
  if (n==1)  
    return 1;  
  else  
    return n * fattoriale(n-1);  
}
```

$n=4$

$4 * 6 = 24$