

TECNICO SUPERIORE WEB DEVELOPER FULL STACK

#2 - Ripasso e Funzioni

Le funzioni

Introduciamo le funzioni

- In matematica, una funzione è “una relazione f che ad ogni elemento x del dominio (detto argomento della funzione) associa uno ed un solo elemento del codominio. L'elemento assegnato a x tramite f viene indicato con $f(x)$ ”.
- Le funzioni sono dei mattoncini che possiamo utilizzare per costruire il nostro programma.
- A volte questi mattoncini sono già stati preparati (funzioni predefinite)
- Altre volte siamo noi a doverli preparare (funzioni definite dall'utente/programmatore)

Introduciamo le funzioni

- In matematica, una funzione è “una relazione f che ad ogni elemento x del dominio (detto argomento della funzione) associa uno ed un solo elemento del codominio. L'elemento assegnato a x tramite f viene indicato con $f(x)$ ”.
- Le funzioni sono dei mattoncini che possiamo utilizzare per costruire il nostro programma.
- A volte questi mattoncini sono già stati preparati (funzioni predefinite)
- Altre volte siamo noi a doverli preparare (funzioni definite dall'utente/programmatore)

Perché non possiamo fare tutto nel `main`?

- Lavorando su una piccola parte del problema (la funzione), possiamo focalizzarci solo su quella e ridurre gli errori;
- Ognuno può lavorare su una funzione singolarmente e poi riunire tutto per avere la soluzione;
- Se una funzione serve in più punti del nostro programma, possiamo scriverla una sola volta;
- Riduce la complessità del `main`, migliorando quindi la leggibilità e la manutenibilità del codice.

Funzioni predefinite

- Le funzioni predefinite sono già state scritte da altri programmatori prima di noi.
- Per poterle utilizzare, spesso dobbiamo caricare le librerie (collezioni di funzioni) dove sono definite.
- Funzioni matematiche:
 - `Math.Pow(x, y)`
 - `Math.Sqrt(x)...`
- Funzioni sui caratteri:
 - `ToLower(c)`
 - `ToUpper(c)`
 - `IsLetter(c)`
 - `IsDigit(c)...`

Funzioni definite dallo sviluppatore

Non restituiscono un valore:

- Funzioni che terminano senza restituire un valore
- Non hanno un tipo di ritorno
- void (“nulla, vuoto”) è il tipo da assegnare a queste funzioni

Restituiscono un valore:

- Funzioni che restituiscono un valore attraverso l’istruzione return
- Hanno un tipo di restituzione (detto anche tipo della funzione)
- Il main, finora, ha sempre restituito 0.

Schema di una funzione

```
<modificatore> <tipo> <nome> (<lista parametri>)  
{  
    <Corpo della funzione>  
}
```

- Il tipo della funzione può essere void per le funzioni che non restituiscono o un tipo (int, float, bool...) per le altre
- Il nome della funzione segue le stesse regole delle variabili
- La lista dei cosiddetti parametri formali può anche essere vuota (funzioni senza argomenti)

Chiamare una funzione

- Una volta definita la funzione, possiamo chiamarla dove ne abbiamo bisogno;
- La chiamata di funzione consiste nel nome della funzione, seguito dalla lista dei parametri attuali.

```
stampaCiao();    // non ha parametri
```

```
stampaNCiao(5); // ha un valore come parametro
```

```
stampaNCiao(k); // ha una variabile come parametro
```

Parametri

Parametri attuali

- Usati nella chiamata della funzione;
- Sono gli argomenti reali che vengono forniti alla funzione per svolgere il suo compito;
- Rimangono anche dopo il termine della funzione chiamata, ma il loro valore potrebbe essere cambiato.

Parametri formali

- Usati nella definizione della funzione;
- Sono dei “segnaposto” che possiamo usare nel corpo della funzione in attesa di sapere quale sarà l’argomento reale;
- Sono locali alla funzione, cioè spariscono quando essa termina.


Come avviene il passaggio di parametri?

Nella funzione

```
void somma(int a, int b) {...}
```

Nel main

```
somma ( 5, 7 ) ;
```



- Quando verrà eseguito il corpo di somma, il parametro formale a varrà 5 e b varrà 7

Come avviene il passaggio di parametri?

- Il numero ed il tipo dei parametri attuali (cioè della chiamata) deve corrispondere al numero ed al tipo dei parametri formali (cioè della definizione).
- In caso contrario ottengo errori come:
 - error: too many arguments to function
 - error: too few arguments to function

Il chiamante

- Il chiamante di una funzione f è la funzione g che contiene nel suo corpo la chiamata alla funzione f .
- Può essere un'altra funzione (a volte la funzione può anche chiamare se stessa) oppure il `main`.
- Visto che il `main` è una funzione, chi è il suo chiamante?

Funzioni che restituiscono un valore

Funzioni che restituiscono un valore

- Sono funzioni il cui risultato viene restituito al chiamante.
- Restituiscono un solo valore.
- Vengono chiamate all'interno di espressioni

Il chiamante può:

- Salvare il risultato in una variabile

```
float x = f(2);
```

- Stampare il risultato

```
Console.WriteLine(f(2));
```

- Usarlo nei calcoli

```
float area = Math.PI*Math.Pow(raggio,2.0)
```

- Usarle come parametri di altre funzioni

```
float x = Math.Pow(Math.Abs(base),exp)
```


Il comando `return`

- Le funzioni che restituiscono un valore terminano con una istruzione nella forma

`return expr;`

- `expr` può essere:
 - un valore costante (es. 7, 3.14, 'g', true, ...),
 - una variabile (paperino, somma, lanci,...)
 - una espressione composta (3+8, costo-sconto, prezzo*0.2, k >= 0...)
 - una chiamata a funzione (che restituisca)

Il comando `return`

- `expr` viene prima valutata e poi restituita.
- Quando viene eseguita l'istruzione `return`, tutte le successive vengono ignorate.
- Vi sono scuole di pensiero diverse in merito all'avere un'unica istruzione `return` che conclude la funzione (single exit-point)

Esercizio 1

- Scrivete una funzione che, presi come parametri due numeri reali, restituisca il maggiore dei due.

Prima possibile soluzione

```
static float maggiore(float a, float b) {  
    float m;  
    if(a >= b) {  
        m = a;  
    } else {  
        m = b;  
    }  
    return m;  
}
```

Seconda possibile soluzione

```
static float maggiore(float a, float b) {  
    if(a >= b) {  
        return a;  
    }  
    else {  
        return b;  
    }  
}
```

Terza possibile soluzione

```
static float maggiore(float a, float b) {  
    if(a >= b) {  
        return a;  
    }  
    return b;  
}
```

Esercizi

- Scrivete un programma che calcoli attraverso opportune funzioni l'area ed il perimetro di un rettangolo dati i suoi lati.
- Scrivete una funzione che abbia come parametri il prezzo di vendita e la percentuale di sconto e restituisca il prezzo scontato.
- Scrivete una funzione che prenda come parametro il voto di una verifica e restituisca un valore booleano che dica se è sufficiente.

Funzioni che non restituiscono un valore

Restituire VS Non restituire

Cosa hanno in comune

- Hanno una struttura simile
 - Hanno una firma
 - Hanno un corpo
- Possono essere definite prima o dopo il main
- Possono avere o non avere parametri formali

Cosa hanno di diverso

- Nella firma, il tipo della funzione è sostituito da void
- Il comando return può essere usato, ma senza alcun argomento (return;)
- Non vengono chiamate in espressioni, ma come istruzioni autonome

Esercizio 2

Scrivere una funzione che prenda come parametro un punteggio (compreso tra 0 e 100) e stampi il corrispettivo grado di superamento, secondo la tabella:

Punteggio	Grado
≥ 95	Super
≥ 90	A
≥ 80	B
≥ 70	C
≥ 60	D
Altrimenti	F

Passaggio di parametri

Diversi passaggi di parametri?

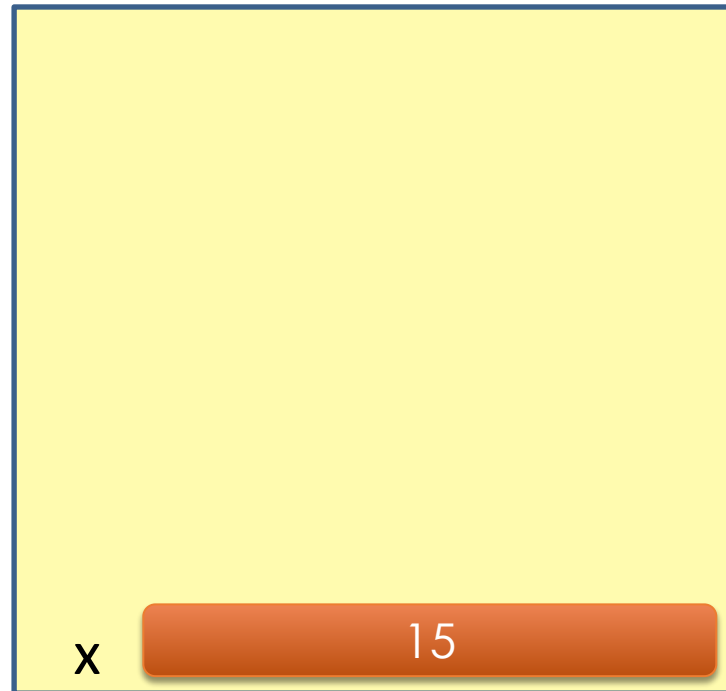
- Abbiamo visto che, quando chiamiamo una funzione, i parametri attuali della chiamata vengono associati ai parametri formali. Ci sono vari modi di associare i parametri attuali a quelli formali:
 - Valore
 - Valore-risultato
 - Risultato
 - Riferimento/indirizzo
 - Costante
 - Nome

Il passaggio per Valore

- Il passaggio per valore è quello che abbiamo usato finora, senza sapere che si chiama così.
- Con questo sistema il VALORE del parametro attuale viene COPIATO nella zona di memoria del parametro formale.

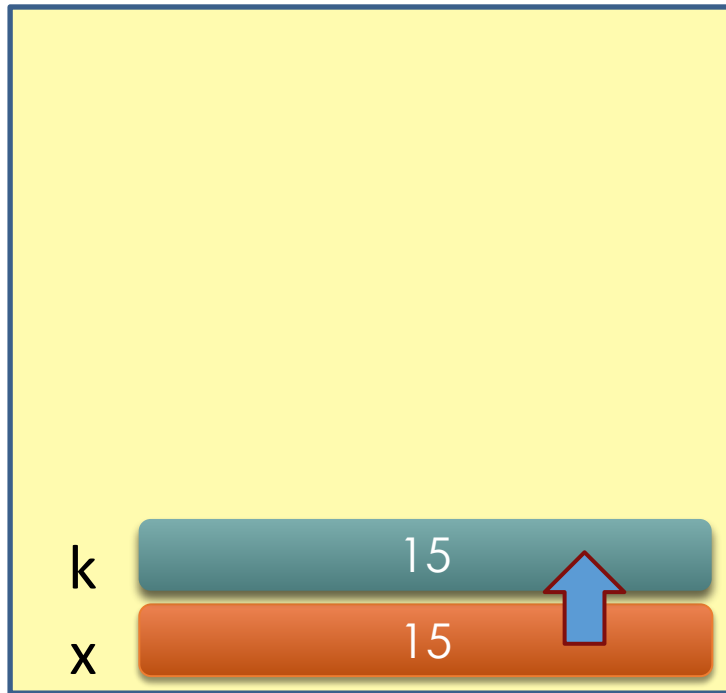
Cosa succede in memoria

```
void myfunction(int k) {  
    k += 4;  
    Console.WriteLine(k);  
}  
  
int main() {  
    int x = 15;  
    myfunction(x);  
    Console.WriteLine(x);  
}
```



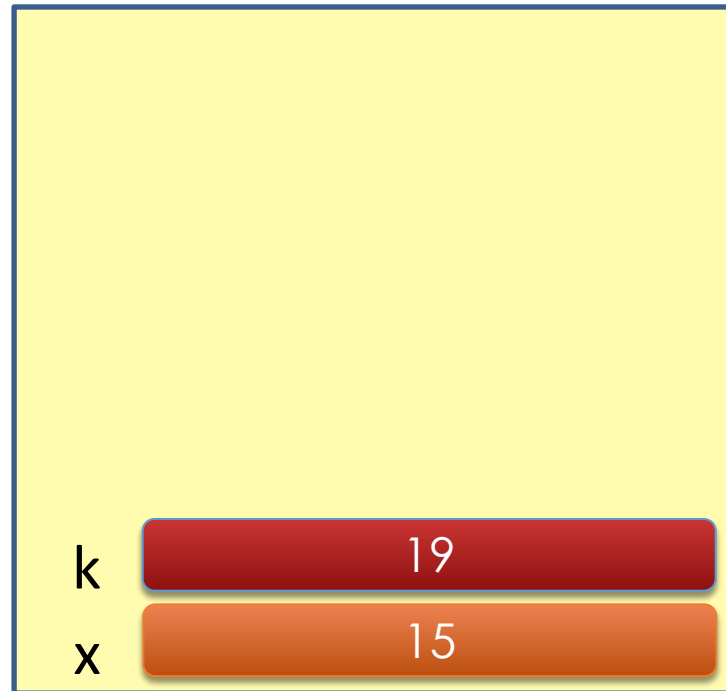
Cosa succede in memoria

```
void myfunction(int k) {  
    k += 4;  
    Console.WriteLine(k);  
}  
  
int main() {  
    int x = 15;  
    myfunction(x);  
    Console.WriteLine(x);  
}
```



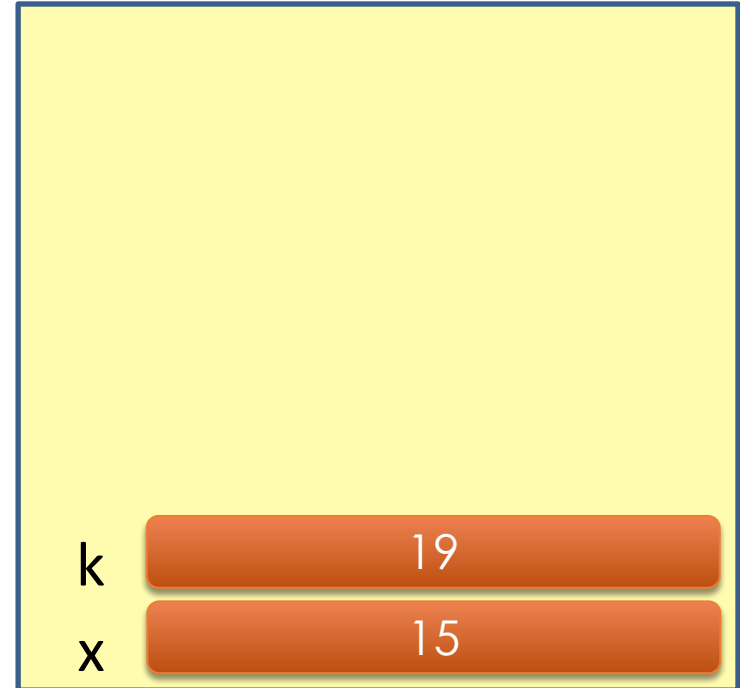
Cosa succede in memoria

```
void myfunction(int k) {  
    k += 4;  
    Console.WriteLine(k);  
}  
  
int main() {  
    int x = 15;  
    myfunction(x);  
    Console.WriteLine(x);  
}
```



Cosa succede in memoria

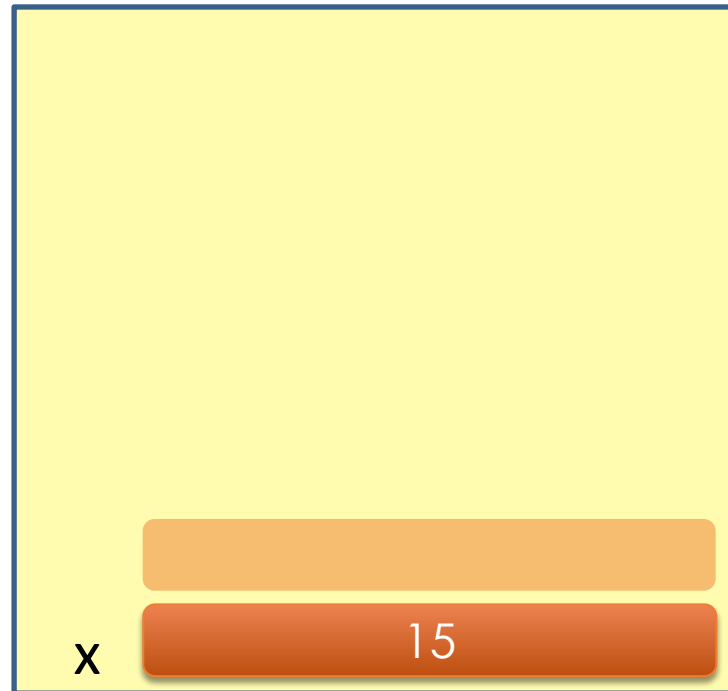
```
void myfunction(int k) {  
    k += 4;  
    Console.WriteLine(k);  
}  
  
int main() {  
    int x = 15;  
    myfunction(x);  
    Console.WriteLine(x);  
}
```



19

Cosa succede in memoria

```
void myfunction(int k) {  
    k += 4;  
    Console.WriteLine(k);  
}  
  
int main() {  
    int x = 15;  
    myfunction(x);  
    Console.WriteLine(x);  
}
```



19 15

Il passaggio per Riferimento

- Prima di introdurre il concetto di passaggio per riferimento (o per indirizzo), bisogna prima capire come si costruisce una variabile in memoria.
- Quando il programma esegue una istruzione di dichiarazione, la variabile dichiarata viene allocata, cioè viene riservata una zona di memoria che servirà a contenere il valore della variabile.
- Ogni zona della memoria ha un indirizzo, rappresentato come un valore esadecimale tipo 0xABCD1234

Variabili ed Indirizzi

- Ogni variabile, quindi, possiede un proprio indirizzo, diverso da tutte le altre variabili dichiarate.
- Il programma memorizza quindi l'abbinamento tra il nome della variabile ed il suo indirizzo in memoria

- Quando faccio riferimento ad una variabile, ad es. `Console.WriteLine(x)`, il computer la interpreta così:

E' usata la variabile `x`

L'indirizzo di `x` è

`0xABCD1234`

Stampa il contenuto della cella di memoria

`0xABCD1234`

I modificatori per il passaggio per riferimento

C#, a differenza di altri linguaggi di programmazione, prevede 3 diverse “modalità” di passaggio per riferimento, che grossolanamente possiamo associare a

“Read Only”

“Write Only”

“Read/Write”

I modificatori possibili sono `in`, `out` e `ref`.

Come si effettua un passaggio per riferimento

Nella definizione

- Si aggiunge un modificatore al tipo del parametro

```
void myfunction(mod int k)
{
    k += 4;
    Console.WriteLine(k);
}
```

Nel chiamante

- Nel caso del modificatore `in` non cambia nulla

```
myfunction(x);
```

- Negli altri casi bisogna specificare il modificatore utilizzato

```
myfunction(mod x);
```

Il modificatore `in` (“Read Only”)

La parola chiave `in` fa sì che gli argomenti vengono passati per riferimento.

Imposta il parametro formale come alias dell'argomento, che deve essere una variabile.

In altre parole, qualsiasi operazione sul parametro viene eseguita sull'argomento, ma gli argomenti `in` non possono essere modificati dal metodo chiamato.

La motivazione è per migliorare le performance dichiarando che il valore non sarà modificato dal metodo.

Il modificatore `out` (“Write Only”)

La parola chiave `out` fa sì che gli argomenti vengono passati per riferimento.

Imposta il parametro formale come alias dell'argomento, che deve essere una variabile. In altre parole, qualsiasi operazione sul parametro viene eseguita sull'argomento.

Le variabili passate come argomenti `out` non devono essere inizializzate prima di essere passate in una chiamata al metodo. È necessario tuttavia che il metodo chiamato assegni un valore prima della restituzione del metodo.

Il modificatore `ref` (“Read/Write”)

La parola chiave `ref` indica che un argomento viene passato per riferimento, non per valore.

Imposta il parametro formale come alias dell'argomento, che deve essere una variabile. In altre parole, qualsiasi operazione sul parametro viene eseguita sull'argomento.

Un argomento passato a un parametro `ref` deve essere inizializzato prima di essere passato.

Ricapitolando...

- `ref` è usato per affermare che il parametro passato potrebbe essere modificato dal metodo.
- `in` è usato per affermare che il parametro passato non può essere modificato dal metodo.
- `out` è usato per affermare che il parametro passato sarà modificato dal metodo.

Ricapitolando...

Sia `ref` che `in` richiedono che il parametro sia inizializzato prima di essere passato al metodo.

Il modificatore `out` non richiede questo e tipicamente non viene inizializzato prima dell'utilizzo in un metodo.

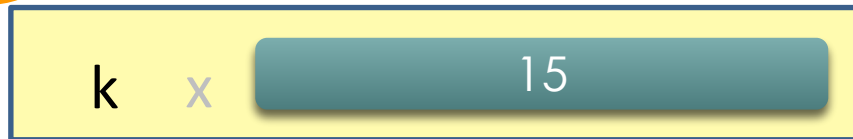
Cosa succede (genericamente) in memoria

```
void myfunction(mod int k) {  
    k += 4;  
    Console.WriteLine(k);  
}  
  
int main() {  
    int x = 15;  
    myfunction(x);  
    Console.WriteLine(x);  
}
```



Cosa succede (genericamente) in memoria

```
void myfunction(mod int k) {  
    k += 4;  
    Console.WriteLine(k);  
}  
  
int main() {  
    int x = 15;  
    myfunction(x);  
    Console.WriteLine(x);  
}
```



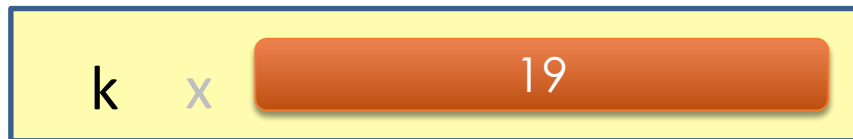
Cosa succede (genericamente) in memoria

```
void myfunction(mod int k) {  
    k += 4;  
    Console.WriteLine(k);  
}  
  
int main() {  
    int x = 15;  
    myfunction(x);  
    Console.WriteLine(x);  
}
```



Cosa succede (genericamente) in memoria

```
void myfunction(mod int k) {  
    k += 4;  
    Console.WriteLine(k);  
}  
  
int main() {  
    int x = 15;  
    myfunction(x);  
    Console.WriteLine(x);  
}
```



A black rectangular box representing a console output, containing the number '19' in red text.

Cosa succede (genericamente) in memoria

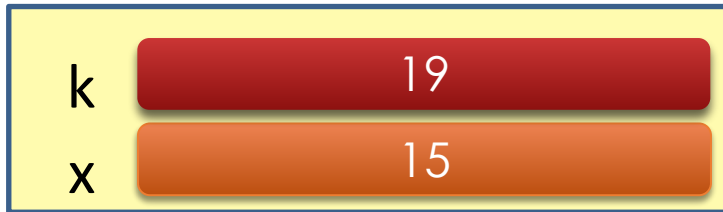
```
void myfunction(mod int k) {  
    k += 4;  
    Console.WriteLine(k);  
}  
  
int main() {  
    int x = 15;  
    myfunction(x);  
    Console.WriteLine(x);  
}
```



19 19

Cosa succede (genericamente) in memoria

void myfunction(int k)



19 15

void myfunction(mod int k)



19 19

L'antico (ma sempre attuale) esercizio delle cifre

- Scrivete un programma che usi una funzione per calcolare la somma delle cifre di un numero dato in ingresso.

Poi scrivere una funzione che:

- Restituisca se quel numero è divisibile per 3;
- Contemporaneamente, assegni il valore ai parametri `quoziente` e `resto`.