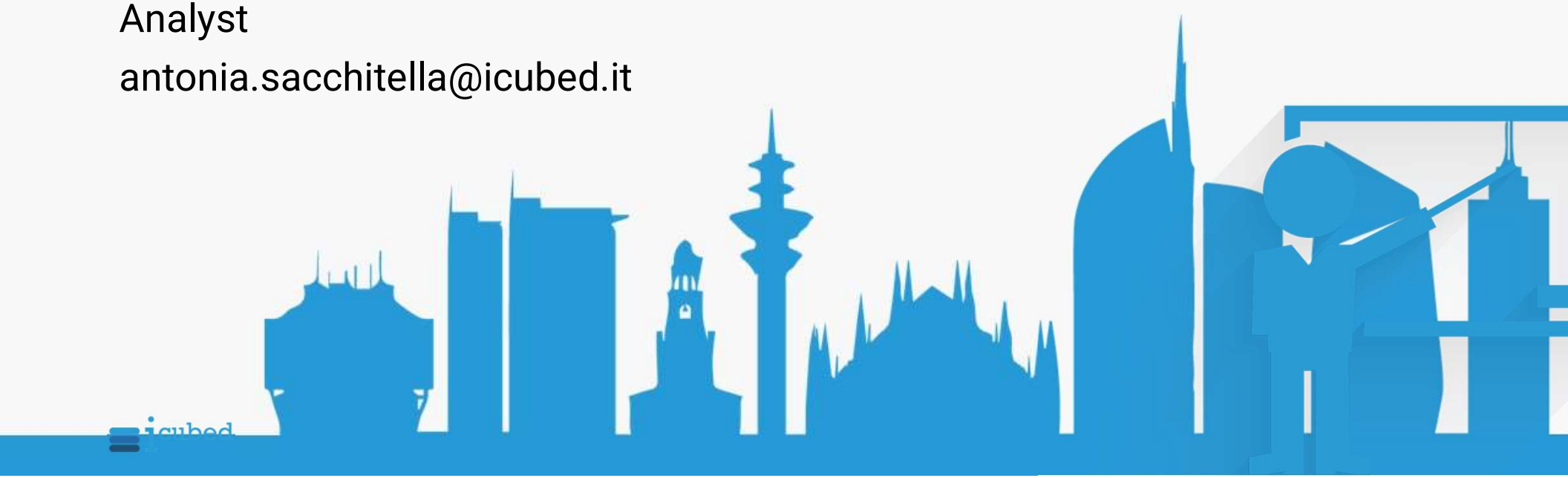
Fondamenti di Programmazione

Antonia Sacchitella
Analyst
antonia.sacchitella@icubed.it



Macchina

Macchina: è un artefatto costruito dall'essere umano per semplificare o sostituirsi all'uomo nel compiere un lavoro.

Nata soprattutto per la necessità di effettuare calcoli numerici.

Computer

Macchina programmabile:

- Conserva l'informazione (tramite una memoria)
- Esegue una serie di istruzioni per elaborare l'informazione
- È possibile fornire le istruzioni da eseguire

L'esecuzione delle istruzioni può avvenire in maniera automatica o semi-automatica

Codifica dell'informazione in base binaria (bit):

1) "Semplice" per la costruzione dei circuiti

2)Rappresenta la minima significativa informazione possibile (true-false on-off …)

icubed

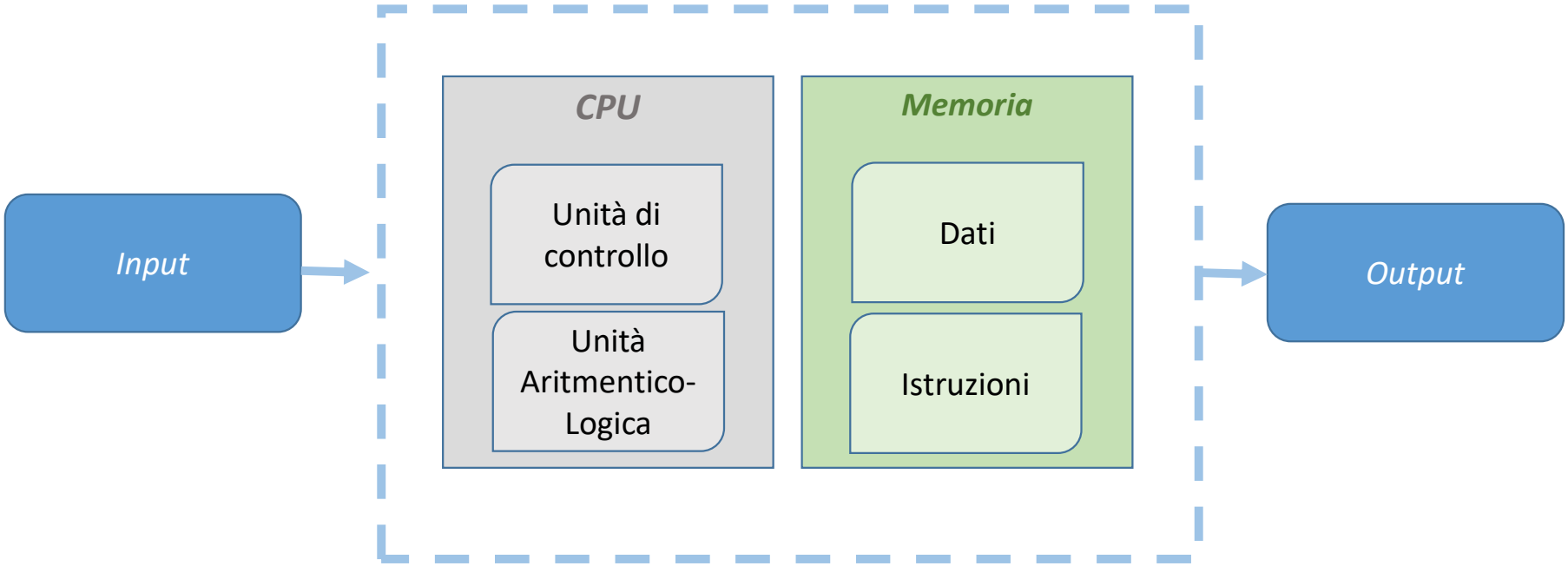
Istruzioni

Una macchina programmabile si distingue da un calcolatore in quanto capace di ricevere ed eseguire delle istruzioni.

Tali istruzioni devono essere:

- In numero finito
- Comprensibili dalla macchina
-Non ambigue (precise)

Modello Von Neumann



CPU

La CPU è composta da:

- Unità di controllo: dispositivo in grado di selezionare e interpretare una istruzione. Coordina di conseguenza l'azione delle varie parti della macchina.

- Unità Aritmetico-Logica: dispositivo che compie le azioni relative all'istruzione.

=> l'implementazione fisica è il processore (o microprocessore)

Memoria

Parte fondamentale del modello (non è importante solo la CPU!)

Una memoria si definisce in base:

- **Capacità**: quantità di informazione che può contenere
- **Velocità**: relativa al tempo necessario per immagazzinare e recuperare l'informazione
- **Volatilità**: capacità di preservare nel tempo l'informazione
- **Costo**: unità minima di informazione (bit o byte)

Memoria Principale

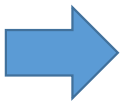
Nella memoria principale (o Memoria centrale) vengono caricati il programma e i dati necessari per eseguirlo

- **Breve-medio termine**: informazione mantenuta per il tempo dell'esecuzione
- **Volatile**: informazione persa se la macchina viene spenta

Memoria Principale

Composta da celle

- <indirizzo,contenuto>
- Identificata univocamente dall'indirizzo
- Dimensioni dipendono dal processore
- Accesso Random: il tempo in cui si accede alle celle è sempre lo stesso



Random Access Memory (RAM)

Indirizzo	Contenuto
0	10000101
1	00000010
2	10001011

Memoria Secondaria

Nella Memoria secondaria vengono caricati programmi e dati in maniera permanente.

- **Lungo termine**: informazione rimane in maniera permanente
- **Non volatile**: contenuto della memoria non viene perso con lo spegnimento del computer

Accesso Sequenziale: il tempo in cui si accede a una cella è dettato dalla sua posizione

Disco Rigido

Esempi di Memoria

Memoria Centrale:

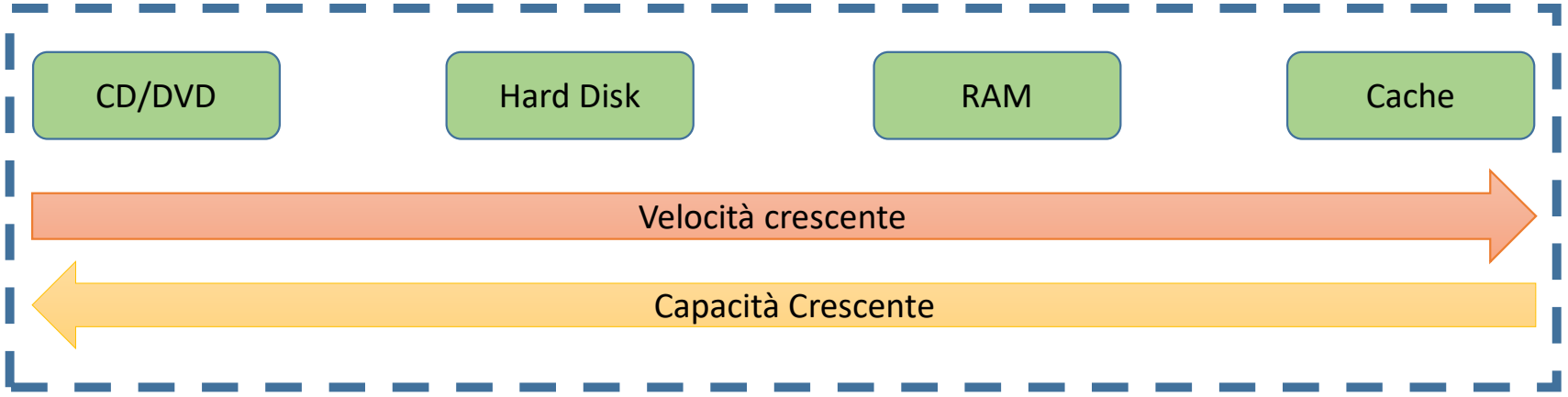
-Cache
-RAM

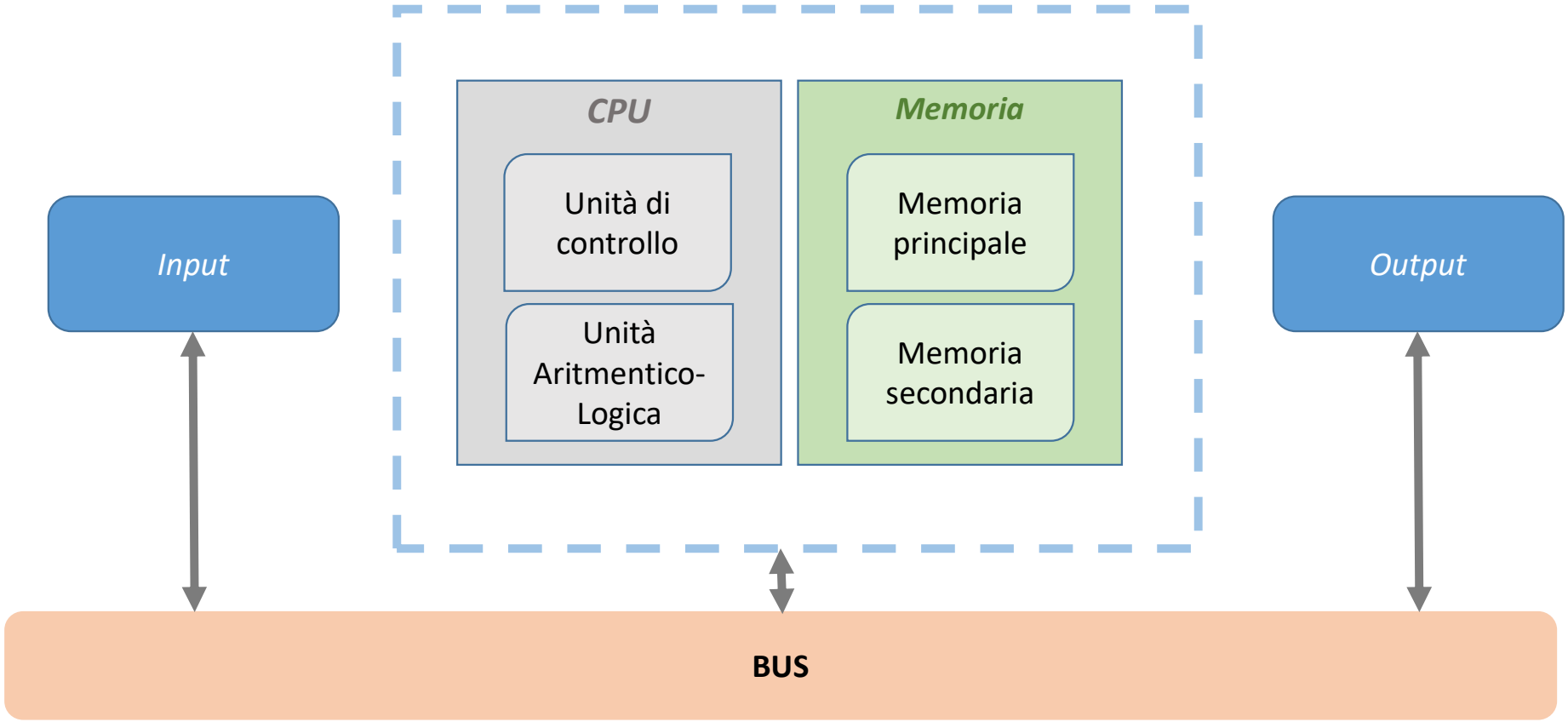
Memoria Secondaria:

-Hard Disk
-CD, DVD

Tipi di Memoria

Tipo	Capacità	Velocità	Volatilità	Costo
Elettronica (Cache, RAM)	Piccola, Media	Alta	Volatile	Medio/Alto
Elettromeccanica(Disco Rigido)	Alta	Media/Bassa	Non volatile	Medio/Basso
Ottica (DVD,CD)	Alta	Bassa	Non volatile	Basso




CPU
Unità di controllo
Unità Aritmentico-Logica
Memoria
Memoria principale
Memoria secondaria
Input
Output
BUS

BUS

Sistema di comunicazione condiviso tra le varie componenti della macchina.

Obiettivo: Consentire lo spostamento delle informazioni da una sezione all'altra (es: dati da dispositivo di input alla memoria, dalla memoria alla CPU...)

Tipologie di computer

Esistono diverse tipologie di computer:

-Laptop

-PC

-WorkStation

-**Server**

Modello Client-Server

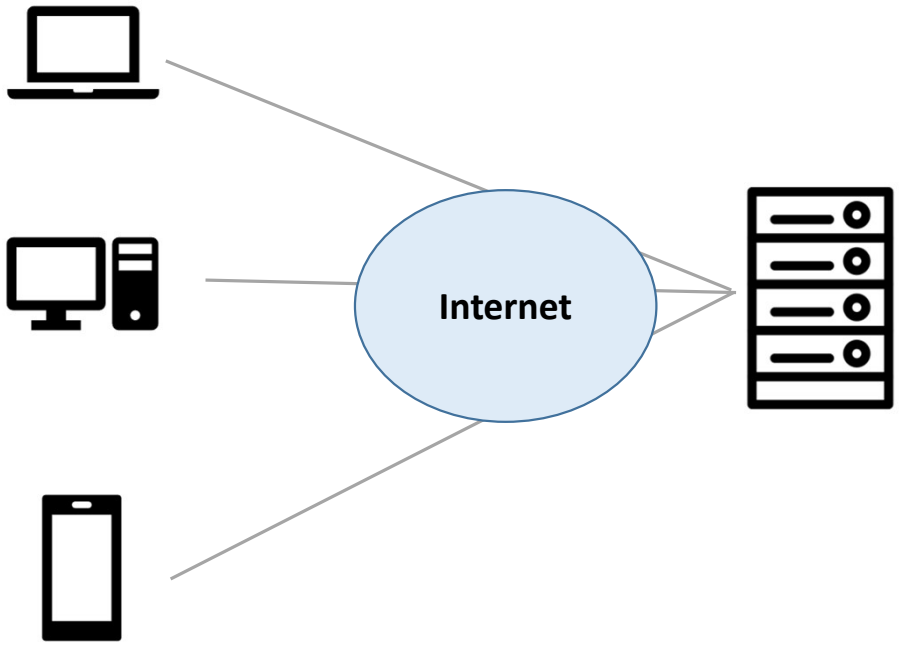
Modello interazione tra calcolatori.

La richiesta dell'utente avviene tramite suddivisione delle azioni tra macchina "personale" e Server.

Client (Macchina personale): formula una richiesta e agevola l'interazione con l'utente (es. tramite periferiche I/O)

Server: risponde alla richiesta dell'utente, elaborando la richiesta, eventualmente attraverso l'accesso a basi di dati. Tipicamente non è dotato di periferiche in quanto l'interazione principale è con altre macchine e non direttamente con l'utente.

Modello Client-Server



Hardware e Software

Una macchina è generalmente composta da due componenti:

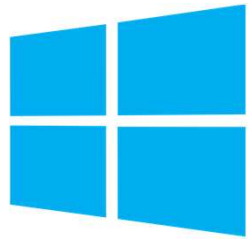
Hardware: componente fisica della macchina
(i negozi "hardware" corrispondono ai ferramenta)

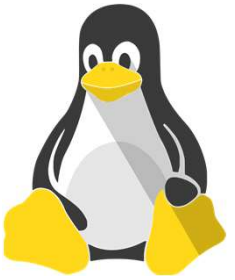
Software: componente formata dalle istruzioni

Sistema Operativo

Insieme di Softwares che serve per accedere alle risorse del computer.

Kernel: nucleo del sistema operativo che gestisce le funzionalità del processore.





Kernel

Gestione diretta del processore nascondendo le sue caratteristiche ad altre applicazioni.

Garantisce:

- Esecuzione sicura dei programmi: un'applicazione non può sovrascrivere una spazio di memoria destinata ad un'altra
- Multitasking: esecuzione contemporanea di più programmi

Funzionalità del Sistema Operativo

- Gestione del processore (Kernel)
- Gestione dell'interfaccia per l'utente (bash, Interfaccia grafica, interfaccia vocale)
- Gestione della memoria
- Gestione I/O

Gestione della Memoria

- Assegna a ciascun processo (programma in esecuzione) una parte della memoria.
- Coordina le memorie della macchina

Principi di località: una parte delle istruzioni e dei dati di un programma sono effettivamente soggetti all'esecuzione del calcolatore. Il processore *tende* a ripetere l'esecuzione di istruzioni che siano allocate in zone di memoria vicine e eseguite da poco.

Memoria Virtuale

Meccanismo del sistema operativo per cui *per l'applicazione* la capacità della RAM è maggiore rispetto alla effettiva capacità fisica.

Memoria secondaria: come "appoggio" per le istruzioni del programma che non servono nell'immediato.

Memoria principale: istruzioni che devono essere eseguite nell'immediato

Processore: esegue le istruzioni presenti in memoria principale

Memoria virtuale

Durante l'esecuzione le istruzioni vengono scambiate tra memoria principale e secondaria in modo che la prima abbia sempre le istruzioni che il processore deve eseguire nell'immediato.

Questo meccanismo è detto **swap** (scambio).

File System

La conservazione delle informazioni da elaborare o elaborate (testi, immagini, audio..) avviene nella memoria secondaria.

File: contenitore di informazioni "generalizzato" (può contenere audio, testi, video...)

Folder/Directory/Cartella : contenitore per la raccolta gerarchica di file.

La struttura gerarchica di cartelle e files viene chiamata file system.

Gestione I/O

Gestione delle operazioni di input e output tramite delle perifiche.

Ogni periferica ha una certa velocità di interazione.

Il sistema operativo deve rendere omogenea l'interazione dell'utente con le diverse periferiche.

=> **buffer**: area di memoria destinate per l'accumolo *temporaneo* delle informazioni

La disponibilità di un buffer potrà consentire di accumulare un certo numero di caratteri prima di trasmetterli al processore

Funzionamento Sincrono e Asincrono

Un programma ha richiesto un'operazione I/O : (inserimento dati da tastiera, visualizzazione dati...), questa può essere correlata con le altre istruzioni in maniera :

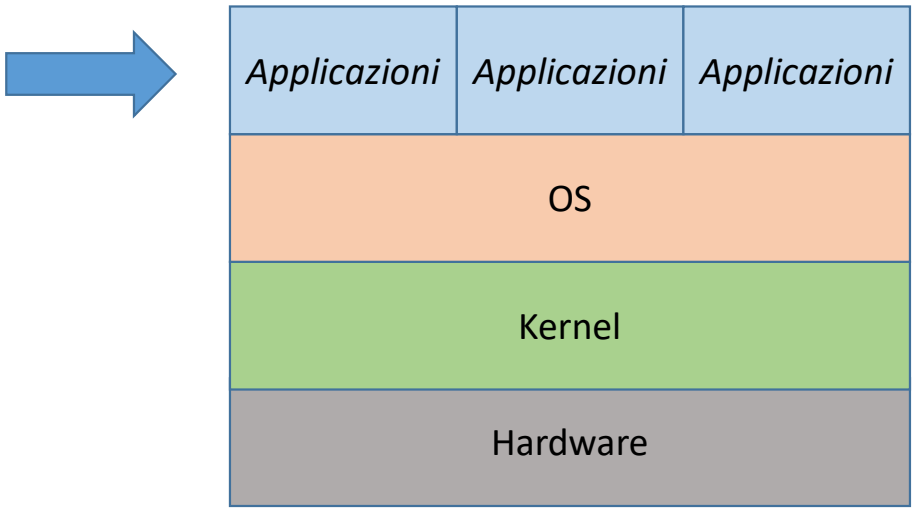
Sincrona: il programma resta in attesa finchè l'operazione richiesta non viene completata.

Asincrona: il programma procede con le altre istruzioni senza attendere che la richiesta originale I/O venga completata.

Questo concetto può essere generalizzato non solo a richieste I/O.

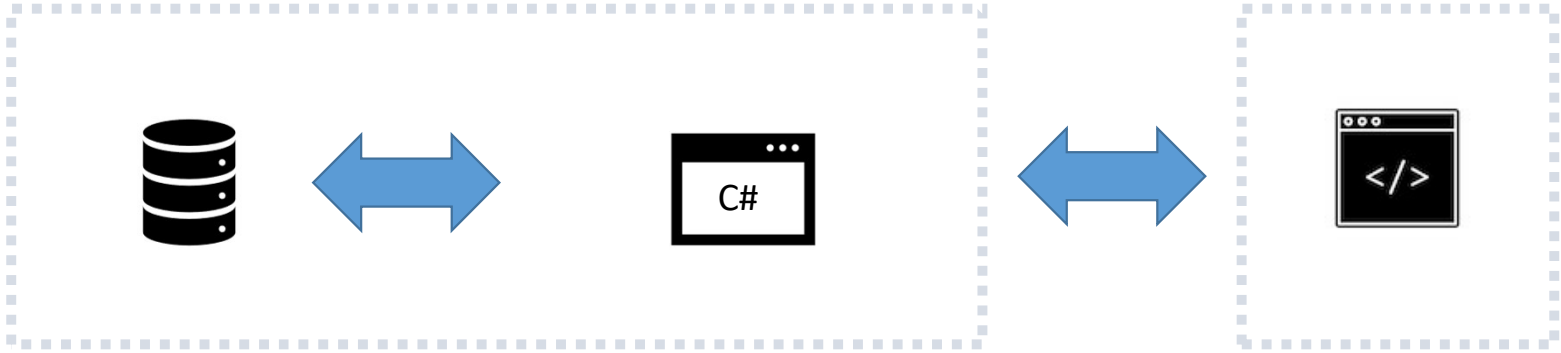
Applicazione

Programmare: Risolvere una classe di problemi in maniera automatica tramite una procedura che sia interpretabile dal calcolatore



icubed

Back-End e Front-End



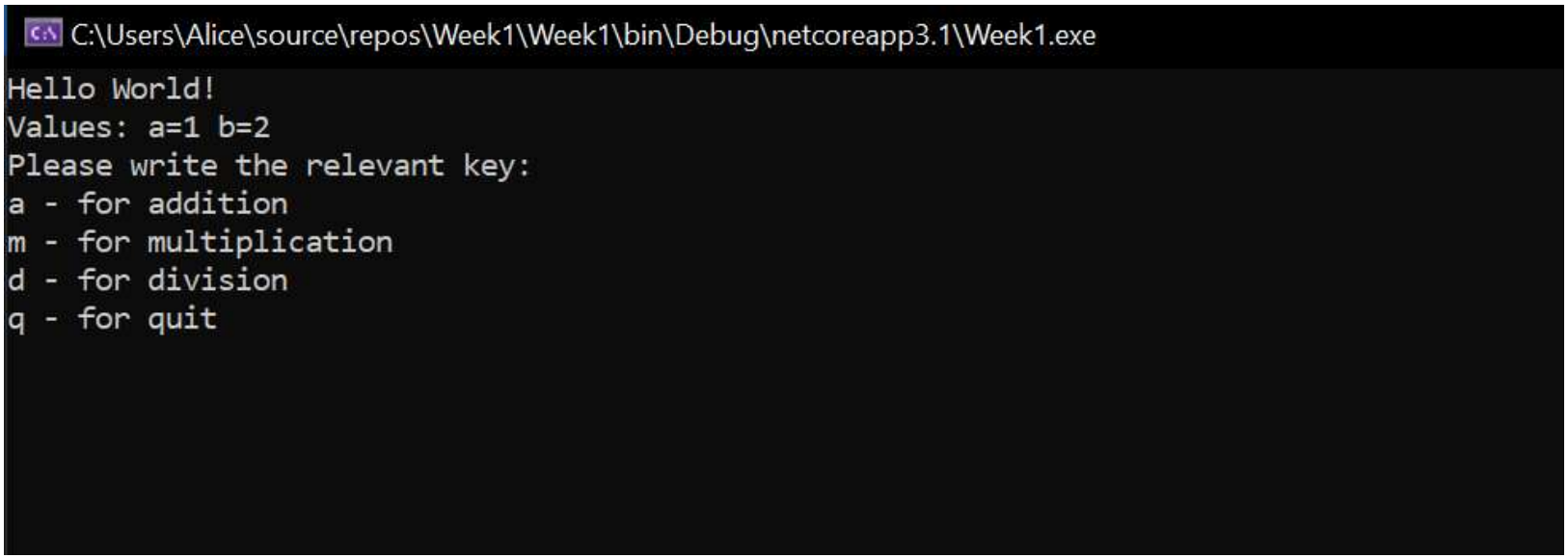
Back-End
Elaborazione del dato

Front-End
Visualizzazione del dato

icubed

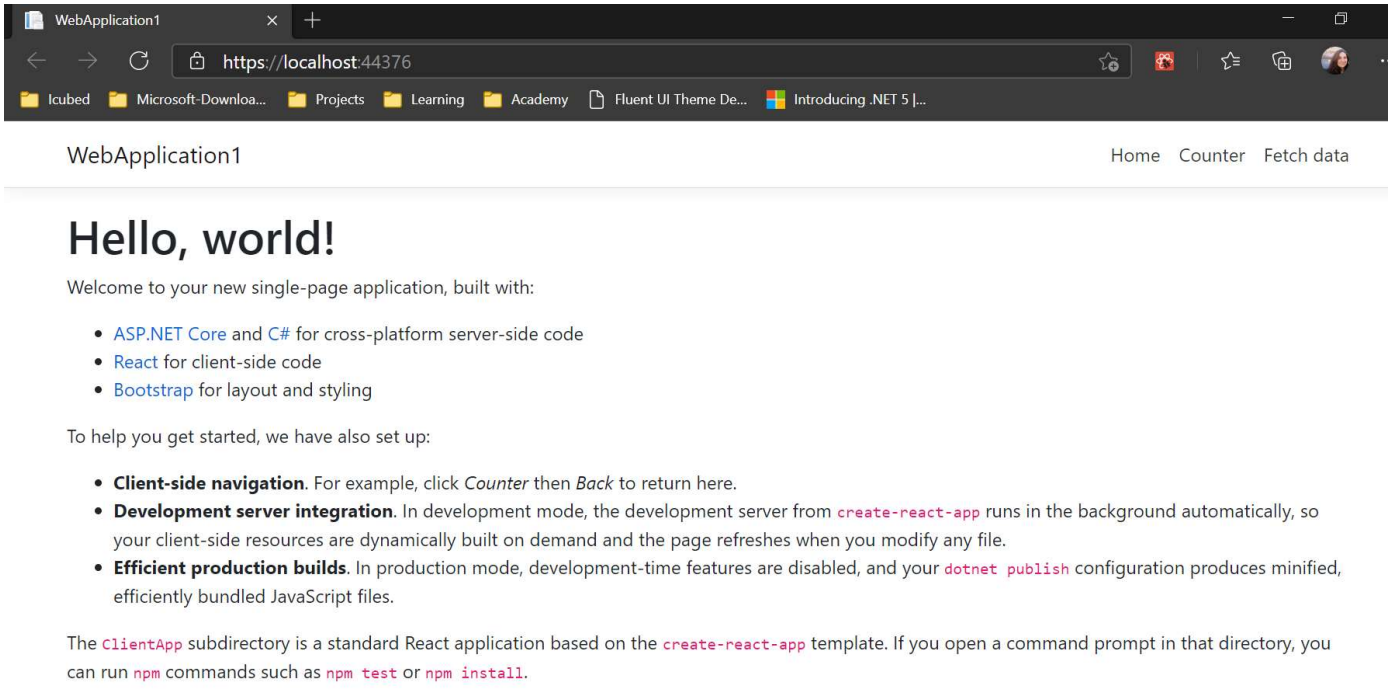
Console Application

Applicazione con interfaccia di command-line



Web Application

Applicazione con interfaccia web



icubed

Windows Application

Applicazione Desktop per sistemi Windows

Possono essere sviluppate tramite le seguenti piattaforme:

- UWP : Universal Windows Platform
- WPF : Windows Presentation Foundation
- Windows Forms

Ognuna ha un set di controlli specifici per la UI (User Interface)

Algoritmo

Processo risolutivo che permette di trasformare una serie di dati iniziali in una serie di risultati finali

Linguaggio di Programmazione

Linguaggio per rappresentare le istruzioni di un algoritmo e l'ordine in cui vengono eseguite

Programma

Algoritmo tradotto in un linguaggio di programmazione

Approccio alla programmazione

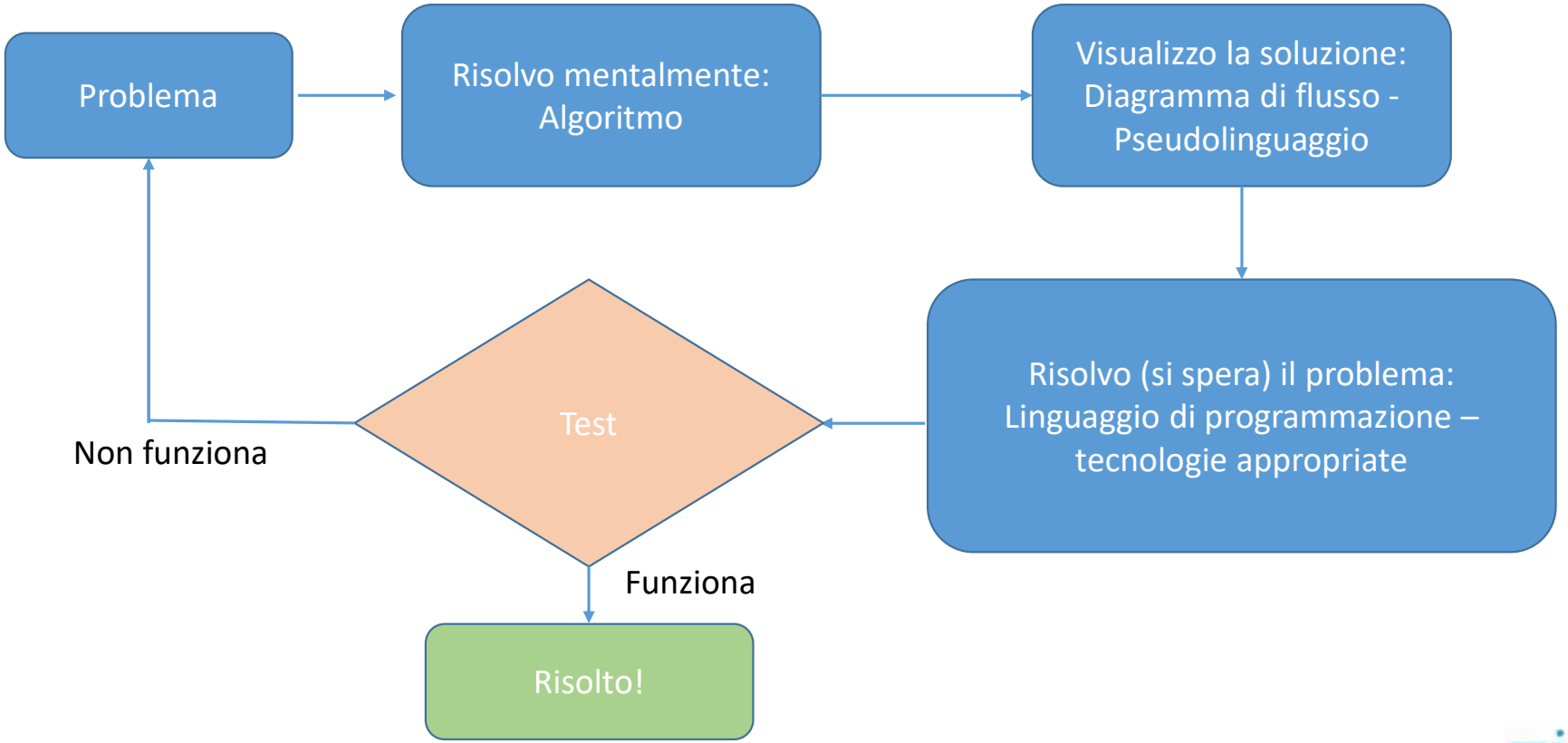
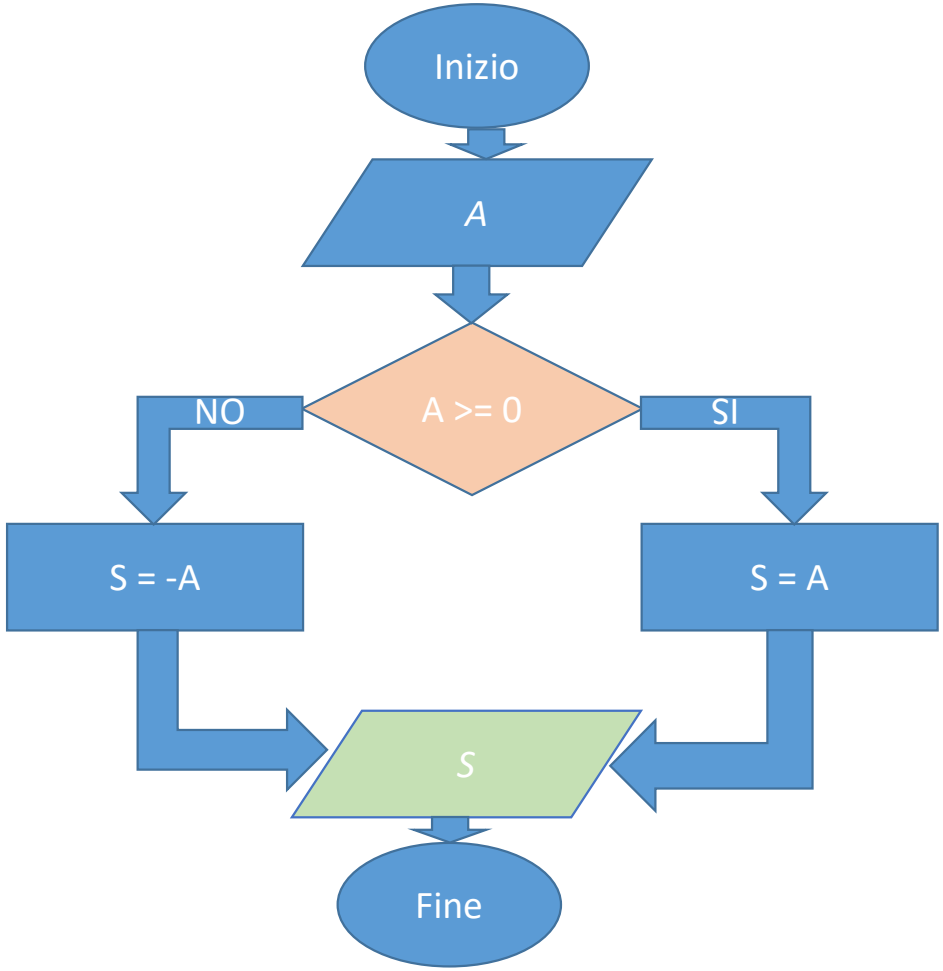


Diagramma di flusso



Descrizione visuale di un algoritmo

Valore Assoluto

icubed

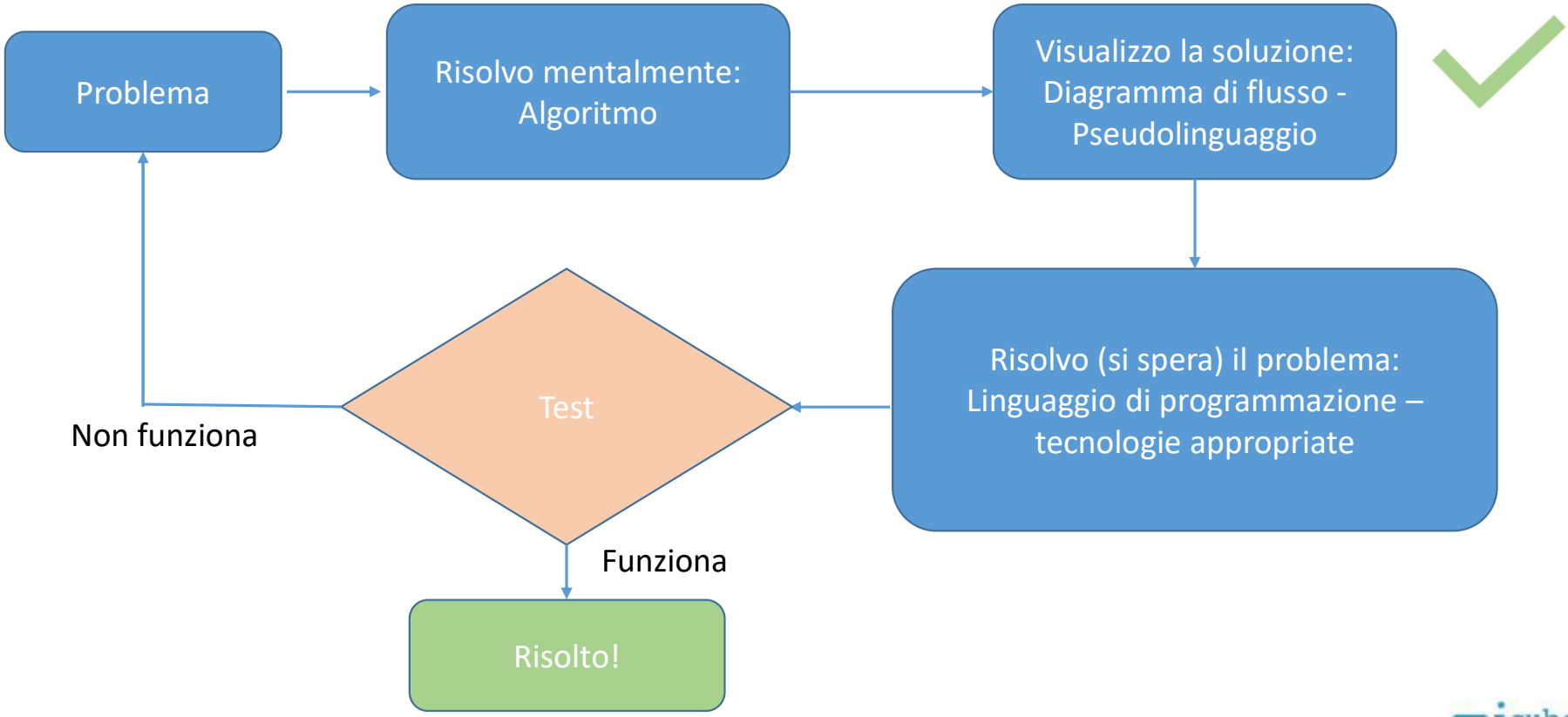
Pseudolinguaggio

```
Start Modulo
Input A
If A>=0
      Then S:=A
      Else S:=-A
Output S
End
```

- Descrizione dell' algoritmo tramite linguaggio
- **Non** è un linguaggio di programmazione!

icubed

Approccio alla programmazione



Codici

Linguaggio di programmazione → Codice sorgente

Linguaggio macchina → Codice Eseguibile

icubed

Editor

- Permette di scrivere il codice sorgente (tramite un determinato linguaggio di programmazione)

Compilatore

- Controllo della sintassi del linguaggio del programmazione
- Traduce il codice sorgente in codice macchina
- Un codice che "compila" non è detto che sia corretto: il compilatore controlla solo la sintassi, non la logica!

E Molti Altri....

IDE

Integrated Development Envirronment

- Editor
- Compilatore
- Strumento per debug

Linguaggi di Programmazione

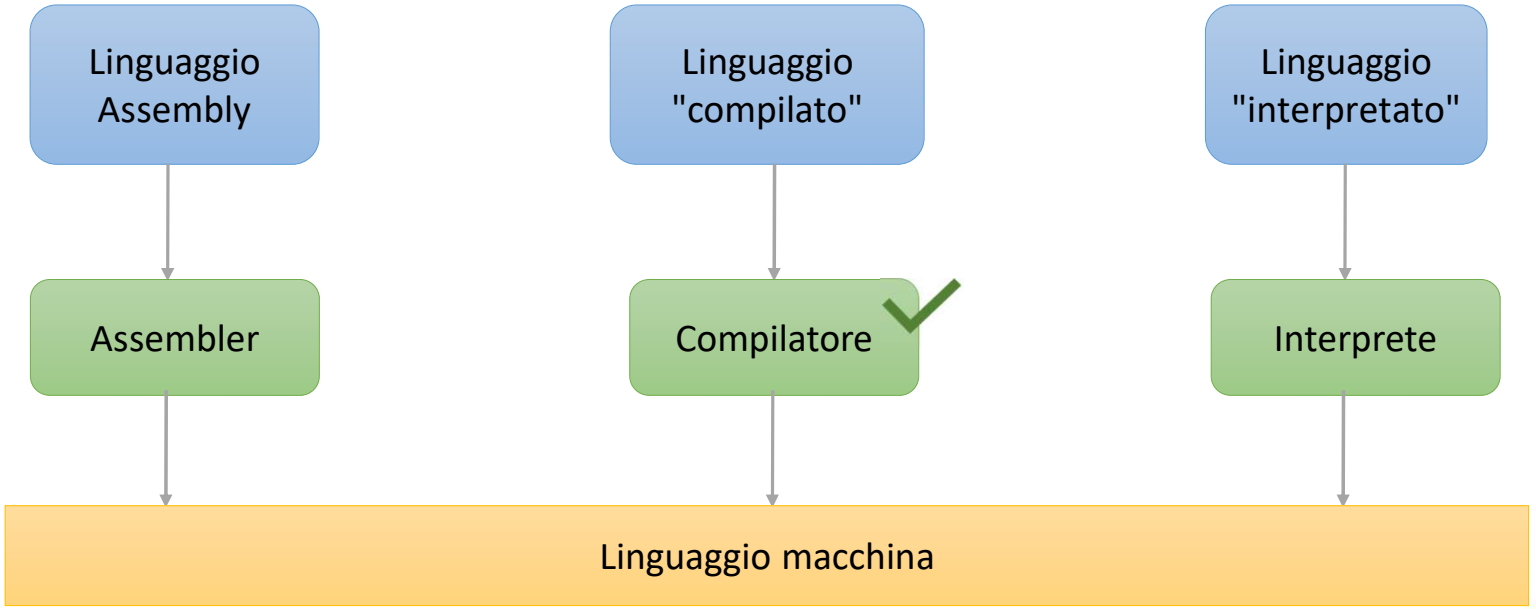
Worldwide, Jan 2021 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	30.44 %	+1.2 %
2		Java	16.76 %	-2.0 %
3		JavaScript	8.44 %	+0.3 %
4		C#	6.53 %	-0.7 %
5	↑	C/C++	6.33 %	+0.3 %
6	↓	PHP	6.05 %	-0.2 %
7		R	3.87 %	+0.1 %
8		Objective-C	3.71 %	+1.2 %
9		Swift	2.14 %	-0.3 %
10		TypeScript	1.78 %	-0.0 %
11		Matlab	1.74 %	-0.1 %
12		Kotlin	1.7 %	+0.0 %
13	↑↑	Go	1.33 %	+0.1 %
14	↓	VBA	1.2 %	-0.2 %
15	↓	Ruby	1.12 %	-0.2 %
16	↑↑	Rust	1.03 %	+0.3 %
17	↓	Scala	0.72 %	-0.3 %
18	↓	Visual Basic	0.69 %	-0.2 %
19	↑↑↑↑↑↑	Lua	0.64 %	+0.3 %

icubed

Linguaggi di Programmazione

Esistono diversi linguaggi di programmazione che possono essere divisi per il tipo di "traduttore" in codice macchina.



Interprete vs Compilatore

Interprete

- per ogni riga di codice letta, viene tradotta l'istruzione direttamente in codice macchina.
- Non esiste una vera distinzione tra esecuzione e traduzione del programma

Compilatore

- Lettura integrale del codice in modo da produrre un codice eseguibile (traduzione)
- Una volta prodotto il codice eseguibile, il compilatore non serve più.

Paradigmi di programmazione

Si dividono i linguaggi di programmazione in due paradigmi:

1. Imperativo

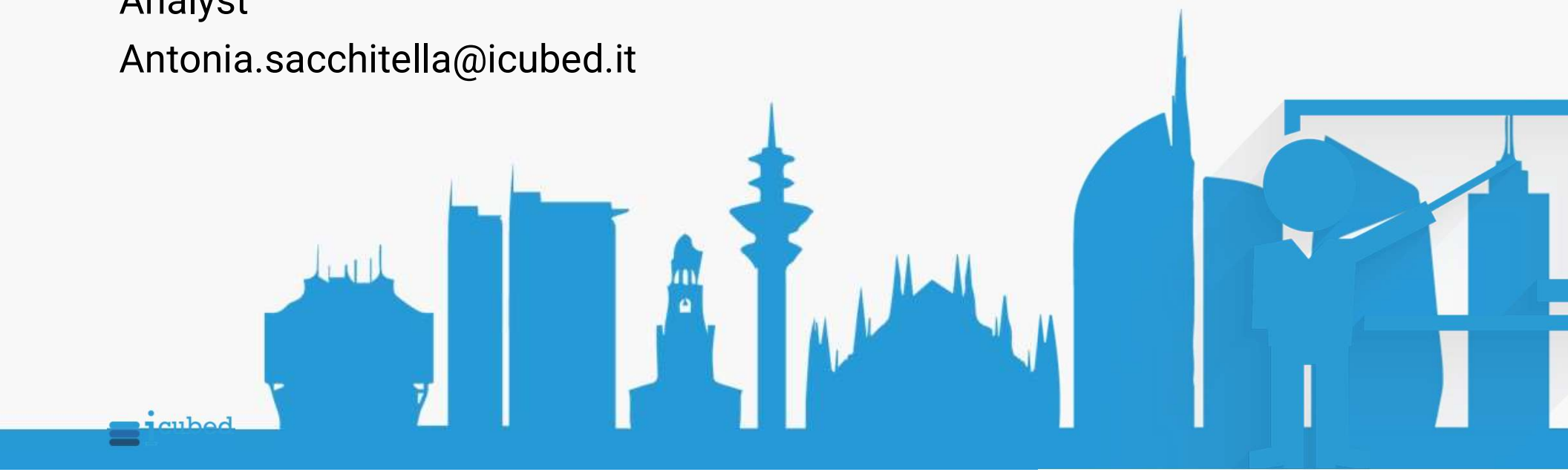
Il programmatore scrive le istruzioni per la macchina (C#)

2. Dichiarativo

Il programmatore specifica il risultato finale, ma non istruisce la macchina sul come raggiungerlo (SQL)

.Net e C#

Antonia Sacchitella
Analyst
Antonia.sacchitella@icubed.it



.NET = un ambiente multi-forma

- .NET permette di sviluppare qualsiasi tipo di applicazione:
  - Web e cloud
  - Native per Windows
  - client/server
  - mobile
  - servizi
- Come sviluppatore, certamente rappresenta un vantaggio notevole
- Imparo una volta C#, poi lo posso usare per qualsiasi utilizzo, sia la parte client di un progetto, sia quella server, etc.
- In passato, invece, la prassi vuole che per ognuna di queste singole implementazioni, si utilizzi un linguaggio o una sua variante particolare.

.NET Framework e .NET Core

- .NET Framework
  - Solo Windows, dal 2000
- .NET Core
  - Windows, Linux, macos, dal 2016
- Sono entrambe implementazioni di .NET

Dal Novembre 2020: **.Net 5**

icubed

.NET Standard Library

Specifiche per set API

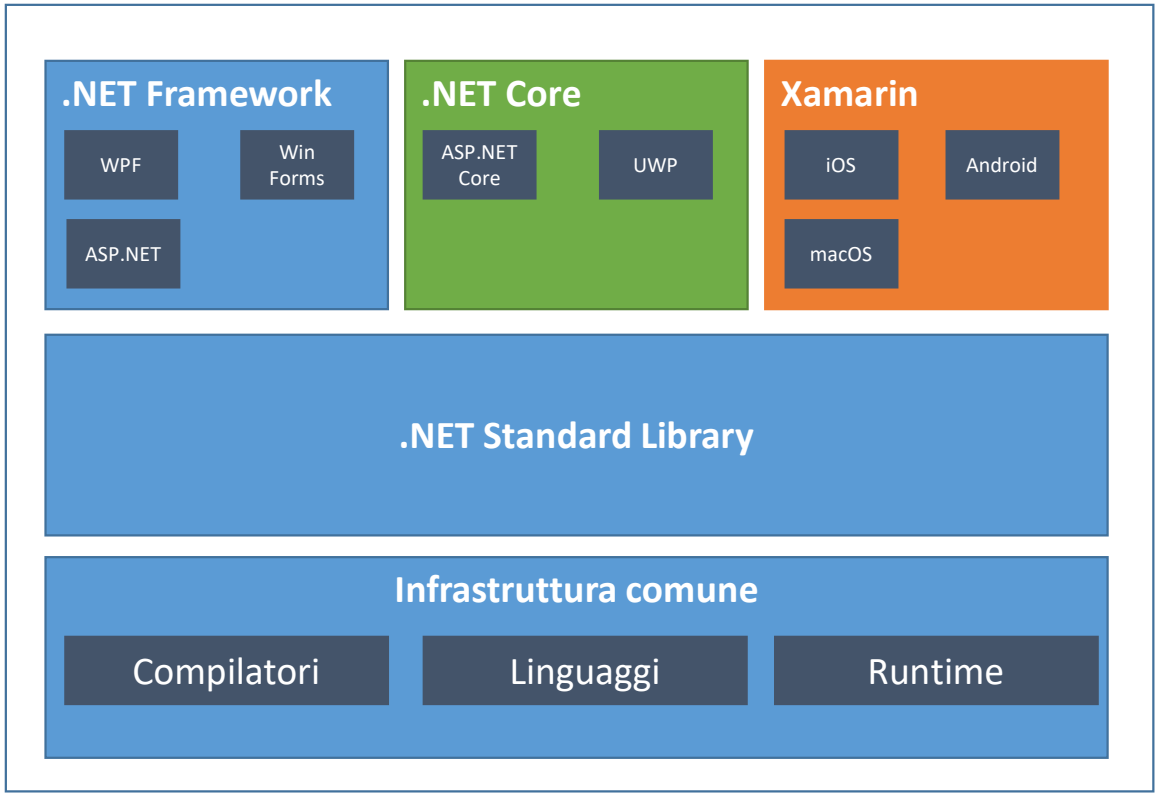
- Base Class Library indipendente dalla piattaforme

Comune a tutti i runtime

- .NET Core, .NET Framework

Garanzia di compatibilità

- Anche all'indietro grazie alle versioni



icubed

.NET Standard Library

Ogni versione è retro-compatibile con le precedenti

1.6: .NET Core 1

2.0: .NET Core 2

.NET Standard	1.01.0	1.11.1	1.21.2	1.31.3	1.41.4	1.51.5	1.61.6	2.02.0
.NET Core	1.01.0	1.01.0	1.01.0	1.01.0	1.01.0	1.01.0	1.01.0	2.02.0
.NET Framework (con .NET Core 1.x SDK))	4.5	4.5	4.5.1	4.64.6	4.6.1	4.6.2		
.NET Framework (con .NET Core 2.0 SDK)	4.5	4.5	4.5.1	4.64.6	4.6.1	4.6.1	4.6.1	4.6.1
Mono	4.64.6	4.64.6	4.64.6	4.64.6	4.64.6	4.64.6	4.64.6	5.45.4
Xamarin.iOS	10.010.0	10.010.0	10.010.0	10.010.0	10.010.0	10.010.0	10.010.0	10.14
Xamarin.Mac	3.03.0	3.03.0	3.03.0	3.03.0	3.03.0	3.03.0	3.03.0	3.83.8
Xamarin.Android	7.07.0	7.07.0	7.07.0	7.07.0	7.07.0	7.07.0	7.07.0	8.08.0
UWP	10.010.0	10.010.0	10.010.0	10.010.0	10.010.0	10.0.1629910.0.16299	10.0.1629910.0.16299	10.0.1629910.0.16299
Windows	8.08.0	8.08.0	8.18.1					
Windows Phone	8.18.1	8.18.1	8.18.1					
Silverlight per Windows Phone	8.08.0							

.NET Standard Library

Pacchetti NuGet

- Estrema modularizzazione
- Assembly/Package per namespace

https://www.nuget.org/packages/NETStandard.Library

- Metapackage: insieme di tutti i pacchetti .NET Standard

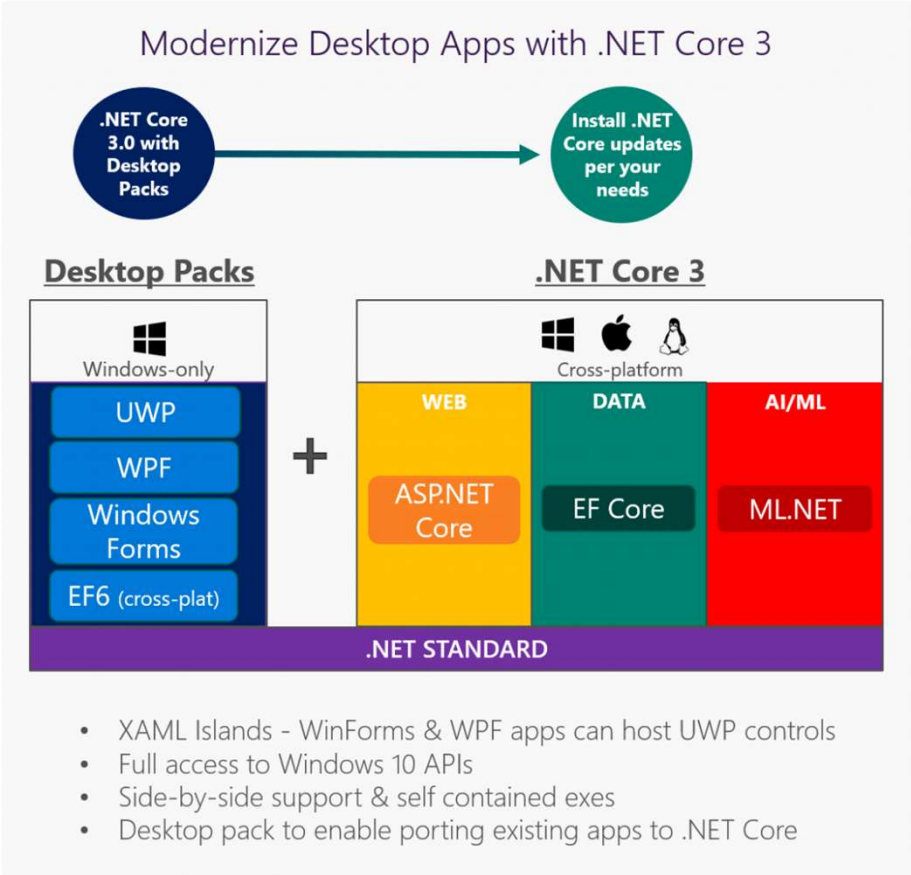
https://www.nuget.org/packages/Microsoft.NETCore.App

- Metapackage: insieme di tutti i pacchetti .NET Core

.NET Core and beyond

- .NET Framework = «obsoleto»
- .NET Core = recente
- .Net = più che attuale

- .NET Core con Desktop Pack
  - Portabilità del codice migliorata
  - EF 6 funzionerà su EF Core
  - Performance!
  - Accesso alle API di Windows 10
  - XAML Island
  - Side by side e full framework deployment



icubed

.Net Core vs .NET Framework

App model

- .NET Core supporta solo console e ASP.NET
- WPF*, WinForms* non supportati perché specifici per Windows
  - Tecnicamente possono esserci pacchetti NuGet per WPF, per esempio, compatibili .NET Standard, ma funzionanti solo su Windows

API

- La maggior parte in comune, ma in assembly diversi (riorganizzati)
- Non API specifiche di Windows
- No Code Access Security

.NET Core è multi piattaforma e open source

- .NET Framework è read only per un sottoinsieme

Cos’è .NET Core

Piattaforma di sviluppo cross-platform
- Windows, macOS, Linux

Open source su GitHub
- Supportato da Microsoft

Eredita Base Class Library di .NET Framework 4.x
- Rivisto: no dipendenze Windows

Compatibile .NET Framework, Xamarin and Mono

icubed

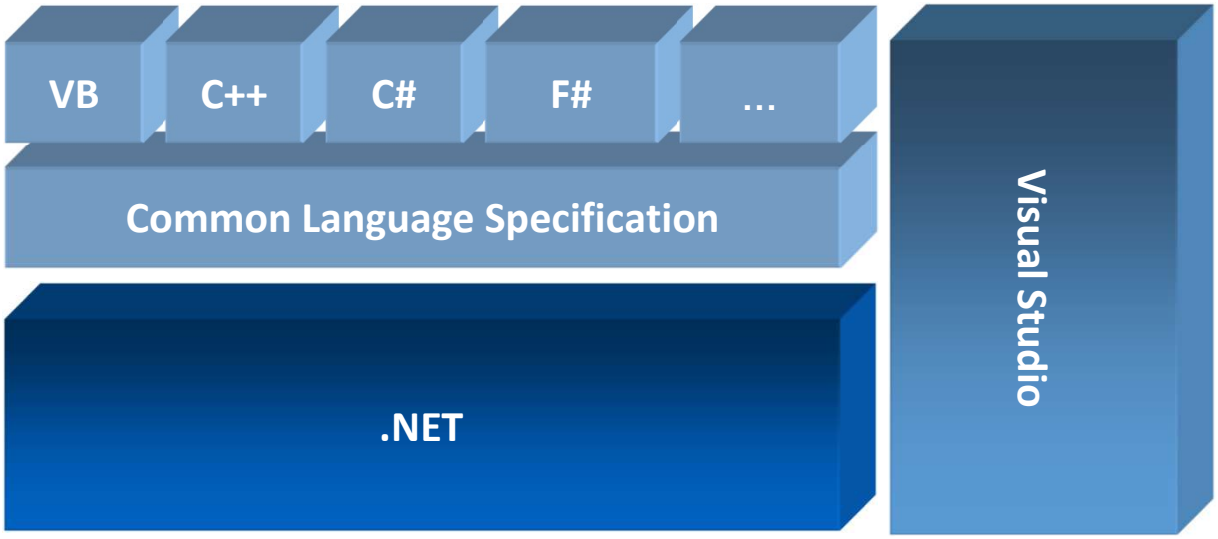
.Net Core

Essendo multipiattaforma, .Net Core può essere utilizzato da più sistemi operativi.

Visual Studio è disponibile solo su Windows e macOS.

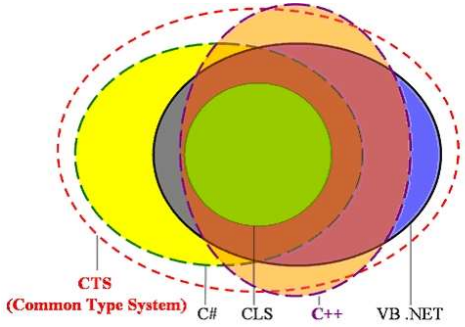
L'interfaccia bash è "comune" a tutti i sistemi operativi => si sono creati dei comandi da terminal in modo che un programma sia sviluppabile anche solo sfruttando la bash.

Linguaggi di .NET



CLS: Common Language Specification

- Le CLS definiscono le specifiche comuni a tutti i linguaggi .NET in modo che gli assembly prodotti siano usabili da tutti i linguaggi.
- Una conseguenza (vantaggio) ovvia rispetto ai linguaggi tradizionali va in direzione della compatibilità binaria.
- Il CTS definisce come devono essere costruiti e definiti i tipi (visibilità, contenuto in termini di proprietà, metodi, ecc.).



icubed

Alcune precisazioni

- C# non è più potente di VB.NET.
- C# non produce codice più “veloce” di VB.NET.
- C# non è il linguaggio “ufficiale” di .NET.
- C# non è la copia di Java.
- C# non è il successore di C++.

Caratteristiche di C#

- C# è un linguaggio nato col .NET Framework.
- C# è un linguaggio **object-oriented**
- C# è un linguaggio **case sensitive**.

Consiglio:

- Rispettare le regole di nomenclatura del codice.
- Ricordare sempre che la nomenclatura in C# è diversa rispetto a quella utilizzata in altri linguaggi come Java.

Struttura di un’applicazione .NET

Una o più classi e/o strutture (tipi)

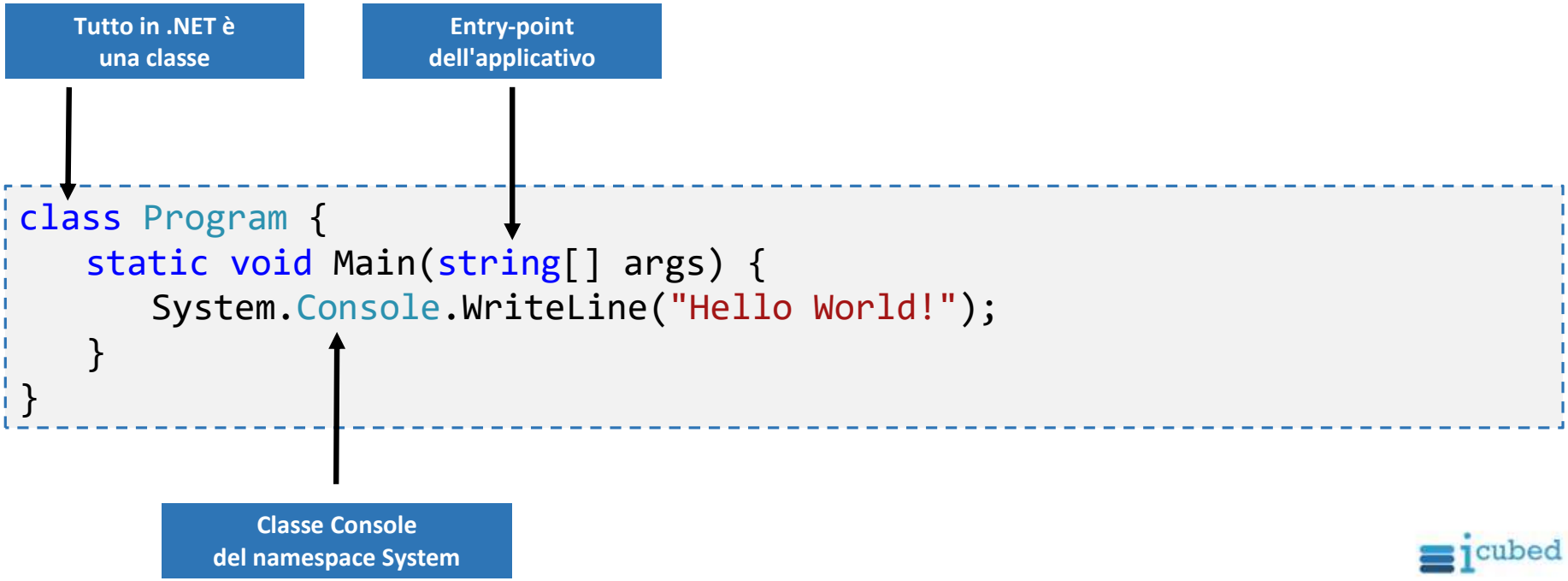
- Classi e strutture sono tipi costituiti da campi, proprietà, metodi ed eventi (membri).

Uno o più file sorgenti

- In Visual Studio non si possono creare progetti utilizzando diversi linguaggi.
- È possibile usare DLL (assembly) create utilizzando un qualsiasi linguaggio .NET (CLS).

La prima applicazione: Hello World!

L'*entry-point* è il punto di ingresso di una applicazione che nelle applicazioni WinForm e Console è la funzione **Main**.



icubed

Variabili

Le variabili sono delle entità che contengono o puntano ai dati utilizzati nell'ambito di un blocco di codice.

La validità della variabile si esaurisce con il termine al termine del blocco di codice in cui viene definita e usata (scope)
Il valore di una variabile può essere sovrascritto in fase di esecuzione del programma.

Dichiarazione: tipo di istruzione che permette di creare una nuova variabile

Assegnazione: tipo di istruzione che permette di modificare il valore di una variabile

Inizializzazione: tipo di istruzione che permette di creare e assegnare un valore a una nuova variabile

Costanti

Una costante è un tipo di variabile che non può essere modificata in fase di esecuzione dell'applicativo.

Viene contrassegnata con la keyword **const** .

Deve essere **obbligatoriamente inizializzata** in fase di compilazione e non può essere in seguito modificata tramite un'operazione di assegnamento.

Esempio

```
//Inizializzazione
int i = 1, j = 2, z = 3;

//Dichiarazione
string x;

//Assegnazione
x = "a";
```

```
//Costanti

const double PI = 3.1415;

const string hello = "Hello World";
```

icubed

Keyword - C#

abstract	event	Namespace	static	as	explicit	new	string
base	extern	null	struct	bool	false	object	switch
break	finally	operator	this	byte	fixed	out	throw
case	float	override	true	catch	for	params	try
char	foreach	private	typeof	checked	get	protected	unit
class	goto	public	ulong	const	if	readonly	ref
continue	implicit	uncheched	unsafe	decimal	in	return	ushort
default	int	sbyte	using	delegate	interface	sealed	value
do	internal	set	virtual	double	is	short	volatile
else	lock	sizeof	void	enum	long	stackaloc	while

Allocazione della memoria

Allocazione statica:

La quantità di memoria da allocare è determinate e fissata a tempo di compilazione.

=>zona di memoria corrispondente: *stack*

Allocazione dinamica:

La quantità di memoria da allocare può variare in fase di esecuzione del programma.

=>zona di memoria corrispondente: *heap*

Tipo

C# è un linguaggio detto «tipizzato»

Un tipo serve per descrivere il ruolo di una variabile nell'ambito del codice e ne caratterizza le funzionalità

Il linguaggio garantisce che il valore inserito nella variabile sia utilizzato in maniera coerente rispetto al tipo della variabile.

I tipi di base

- Dichiarati nel namespace **System** (mscorlib.dll):
  - Boolean, Byte, SByte, Char
  - Int16, Int32, Int64, UInt16, UInt32, UInt64
  - Decimal, Single, Double
  - DateTime, TimeSpan
  - String
- Il linguaggio dichiara alcuni alias per questi tipi:
  - bool (Boolean), Byte (Byte), SByte (SByte), Char (Char)
  - int (Int32)
  - decimal (Decimal), float (Single), double (Double)
  - DateTime per Date e Time, TimeSpan
  - string (String)
- In blu i tipi che non sono conformi alle specifiche CLS.

Value Type

- Contengono direttamente il dato nell'ambito dello stack del thread.
- Una copia di un Value Type implica la copia dei dati in esso contenuti.
- Le modifiche hanno effetto solo sull'istanza corrente.
- Contengono sempre un valore (null **non è** direttamente ammesso).
- I Value Type comprendono:
  - i tipi primitivi come int, byte, bool, ecc.
  - **enum**, **struct** (definiti dall'utente).

```
int i = 1;
bool b = true;
double d = 0.123;
DateTime a = DateTime.Now;
```

Reference Type

- Contengono solo un riferimento ad un oggetto nell'ambito dell'heap.
- La copia di un Reference Type implica la duplicazione del solo reference.
- Il reference che non referenzia nessuna istanza vale **null**.
- **Tutte le classi sono Reference Type!**

```
string s = "C#";
DataSet ds = New DataSet();
Person p = New Person();
```

Tipi di valore vs Tipi di riferimento

Tipo di valore	Tipo di riferimento
Contengono direttamente il valore dei dati	Contiene il riferimento alla locazione in memoria della variabile
Non devono essere instanziati	Deve essere instanziato
Non possono essere nulli	Può essere nullo

System.Object

- **System.Object** è il reference type “root” della gerarchia di tutti i tipi del .NET Framework.
- **System.Object** e con esso tutti i reference type espongono questi metodi di base:

Equals – Comparazione di oggetti

Finalize – Distruzione delle risorse unmanaged

GetHashCode – Generazione di un numero utile in un algoritmo di hashing

ToString – Generazione di una stringa che descrive l'istanza del tipo

Boxing e Unboxing

Vantaggi dei Value Type:

- non sono allocati nel managed heap, ma sullo stack del thread
- non sono soggetti al Garbage Collector
- non sono referenziati da puntatori

È possibile contenere un Value Type in un Reference Type: questa operazione si chiama **Boxing**.

- È necessaria quando si vuole associare ad un tipo di valore un riferimento.
- L'operazione inversa si chiama **Unboxing**.

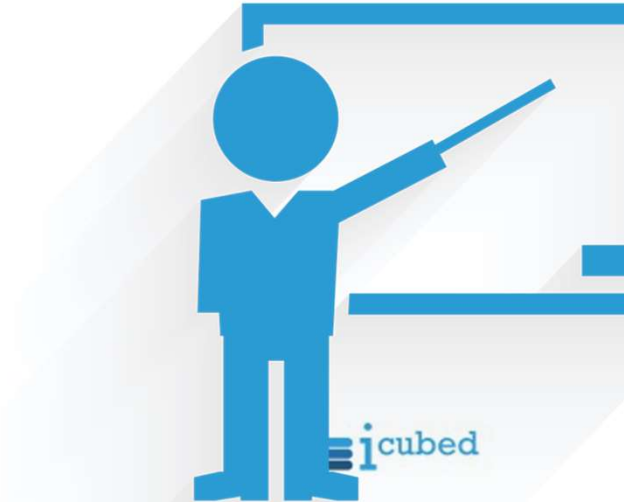
```
int i = 123;      // a value type
object o = i;     // boxing
int j = (int)o;   // unboxing
```

icubed

Demo

Tipi di variabile

Esercizio: Scambio tra valori



icubed

Gli operatori

• Uguaglianza	== !=
• Condizionali	&& ||
• Relazione	< <= > >= is
• Incremento	++ diventa +=1
• Decremento	-- diventa -= 1
• Aritmetici	+ - * /
• Assegnazione	= += -= *= /= %=

icubed

Tabelle di verità

Per l’operatore &&:

A	B	$A \wedge B$
F	F	
F	V	
V	F	
V	V	

Tabelle di verità

Per l’operatore ||:

A	B	$A \vee B$
F	F	
F	V	
V	F	
V	V	

Gli statement

- **Statement condizionali**
  - if, switch case
- **Statement iterativi**
  - while, do, for, foreach
- **Statement di salto**
  - goto, exit, continue, return, throw
- **Altri statement**
  - vuoto, dichiarativi, espressioni, label, try-catch-finally (gestione eccezioni), lock (sincronizzazione multi-thread), using (pattern IDisposable)

Statement condizionali

Espressione booleana (true/false)

```
if (espressione) {
    //...
} else {
    //...
}
```

Valori multipli

Sono ammesse anche le stringhe

```
switch (espressione) {
    case valore1:
        //...
    case valore2:
    case valore3:
        //...
    default:
        //...
        break;
}
```

Demo

Statement Condizionali

icubed

Esercitazione n. 1

Scrivi un algoritmo che inseriti due numeri interi da tastiera esegua :

- La loro somma se uno dei due è pari.
- Il loro prodotto se tutti e due sono pari.
- La loro divisione se nessuno dei due è pari.

icubed

Statement iterativi

```
do {
   //...
} while (espressione)
```

Espressione booleana (true/false)

```
while (espressione) {
   //...
}
```

Inizializzazione

Controllo ad ogni ciclo

Incremento ad ogni ciclo

```
for (int i = 0; i < 100; i++) {
   //...
}
```

icubed

Demo

Statement Iterativi

icubed

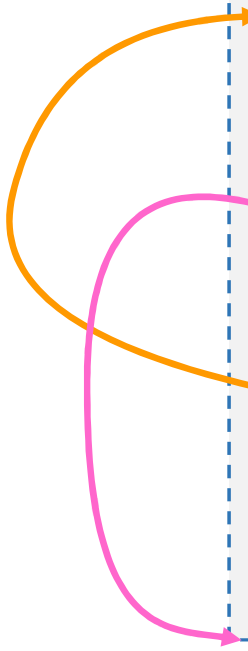
Statement di salto

```
goto OK

//...

OK:
```

Da evitare!
Scarsa leggibilità



```
for (int i = 0; i < 10; i++) {

    Console.Write(i);

    if (i == 5) {
        break;
    }

    if (i > 2) {
        continue;
    }

    Console.WriteLine("*");
}
```

Output

```
0*
1*
2*

345
Stop!
```

icubed

Demo

Statement di salto

icubed

Debugger (VS)

- Componente che consente di agganciarsi al processo che esegue il codice e di “mostrarne” il flusso.
- Permette di intervenire durante l’esecuzione dell’applicazione
- Per controllare il flusso dell’applicazione sfruttando due funzionalità:

Breakpoint

Watch

icubed

Breakpoint

Il debugger esegue l’applicazione ininterrottamente finchè non incontra un’eccezione.

I Breakpoint permettono di fissare un punto preciso dell’applicazione in maniera che il debugger si possa fermare.

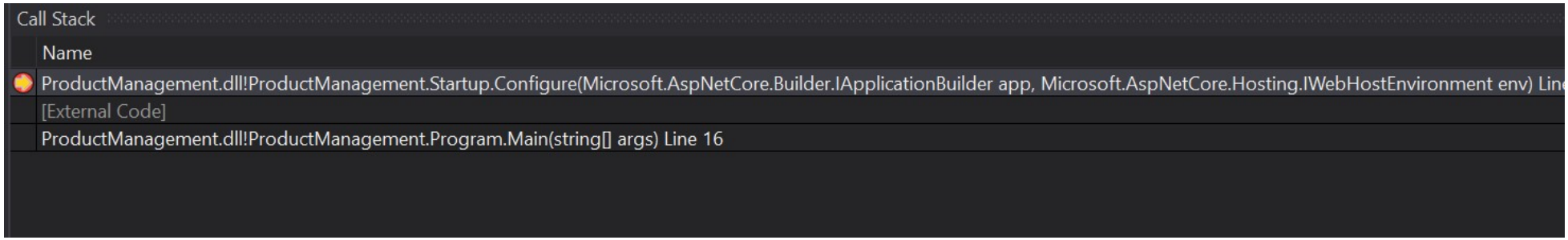
In questo modo lo sviluppatore può verificare lo stato di una variabile all’interno del codice.

Watch

Una volta che il debugger ha fermato l’esecuzione dell’applicazione, è possible analizzare il contenuto di una variabile tramite un processo chiamato **quick watch**.

Call Stack

Una volta che il debugger ha fermato l'esecuzione dell'applicazione, è possible vedere lo *storico delle chiamate* che hanno portato all'attuale codice in esecuzione in una finestra **Call Stack**.



icubed

Esercitazione n. 2

Utilizza un ciclo for realizzare un algoritmo che esegua la somma e la media di n numeri.

Sia i valori da sommare che il numero n di elementi da inserire vengono acquisiti dall'utente da tastiera.

icubed

Cast

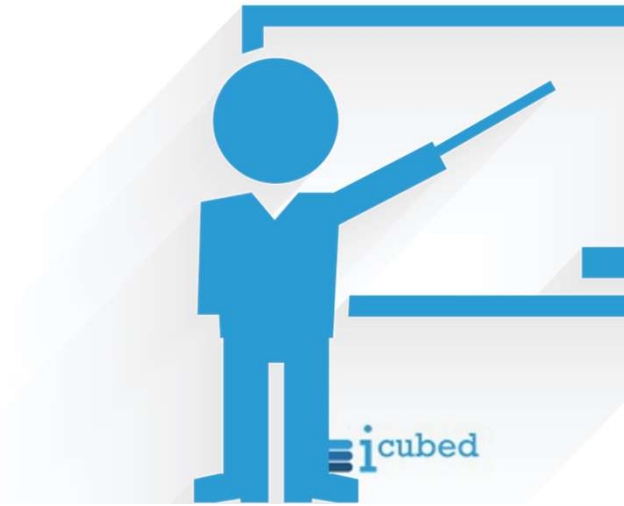
- Il **cast** è l'operazione di conversione di un tipo ad un altro **«affine».**

- Conversione Implicita
- senza rischio di perdita di informazioni
- Sviluppatore non se ne deve preoccupare più di tanto

- Conversione Esplicita
- presuppone perdita di informazioni
- tipo di destinazione non può rappresentare l'interno dominio del tipo di partenza

Demo

Cast

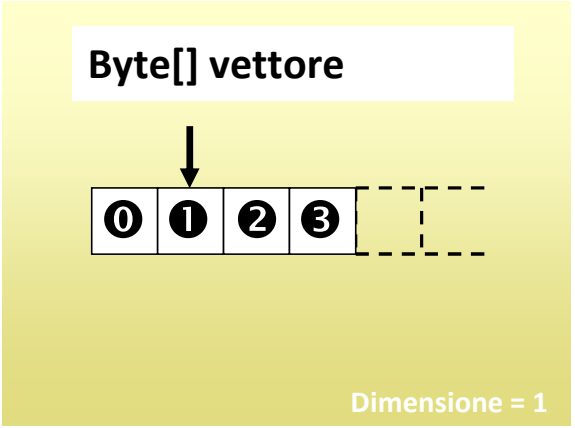


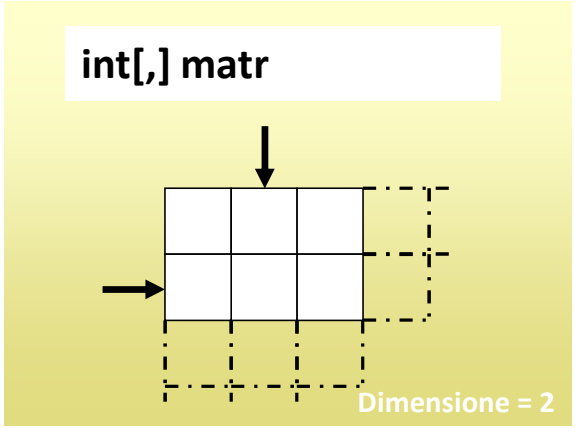
Array

- Gli *array* memorizzano delle **sequenze finite** di elementi, la cui lunghezza deve essere nota a priori
- Gli elementi vengono richiamati in base ad un indice numerico che parte sempre da **zero**
- Gli array non sono direttamente ridimensionabili
- La dimensione di un array può essere calcolata a runtime

Array multidimensionali

- Il metodo **GetLength** permette di conoscere il rank della n-esima dimensione dell'array.






```
Byte[] vettore = new Byte[5];
Byte[] vettore = new Byte[] {1, 2, 3, 4, 5}
```

```
int[,] matrice = new int[2, 3]
int[,] matrice = new int[,] {{1, 2, 3}, {4, 5, 6}}
```

icubed

Collection

- Le *collection* mantengono una lista dinamica di oggetti, la cui lunghezza non ha limiti intrinseci
- Gli elementi possono essere di diverso tipo e possono essere read-only
- A seconda delle interfacce implementate, gli elementi possono essere enumerati e listati secondo modalità diverse
- Sono meno performanti degli array

Collection

- Gli elementi di una lista, o più in generale di una collection, possono essere scansionati utilizzando un opportuno statement che è il foreach

```
int[] array = { 1, 2, 3, 4, 5, 6, };
foreach (int el in array)
{
    Console.WriteLine("Valore: {0}", el);
}
```

icubed

Namespace System.Collections

ArrayList

BitArray

Hashtable

Queue

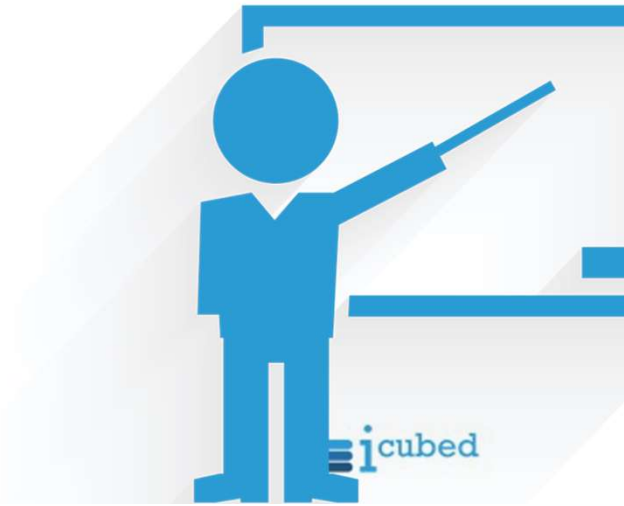
Stack

StringCollection

Ecc.

Demo

Arrays



icubed

Routine

Insieme di istruzioni racchiuse in un unico blocco di codice che formano un'entità richiamabile in altri punti del codice.

L'elaborazione data dalla routine può avere:

- Parametri di input
- Valore di ritorno (risultato)

Le routine si suddividono in:

- **Funzioni** (hanno valore di ritorno)
- **Procedure** (non hanno valore di ritorno)

Dichiarazione Routine

- In sostanza la dichiarazione di un metodo è composta di:
  - zero o più keyword;
  - il tipo di ritorno del metodo oppure **void**;
  - il nome del metodo;
  - l'elenco dei parametri tra parentesi tonde.

- La *firma* (*signature*) di un metodo è rappresentata dal nome, dal numero dei parametri e dal loro tipo; il valore ritornato **non** fa parte della firma.

```
void MyMethod(string str) {
    // ...
}
```

```
int MyMethod(string str) {
    int a = int.Parse(str);
    return a;
}
```

Passaggio dei parametri

- I parametri di un metodo possono essere passati:
  - *By Value* (default)
  - *By Reference* (keyword **ref**)

```
int x = 0;
MyClass.MyMethod(2, 3, ref x);


public static void MyMethod(int a, int b, ref int c) {
    c = a + b;
}
```

Passaggio dei parametri

È possibile:

- passare un Value Type *By Value*;
- passare un Value Type *By Reference*;
- passare un Reference Type *By Value*;
- passare un Reference Type *By Reference*.

Di default i parametri vengono passati *By Value*.

icubed

Passaggio parametri by reference

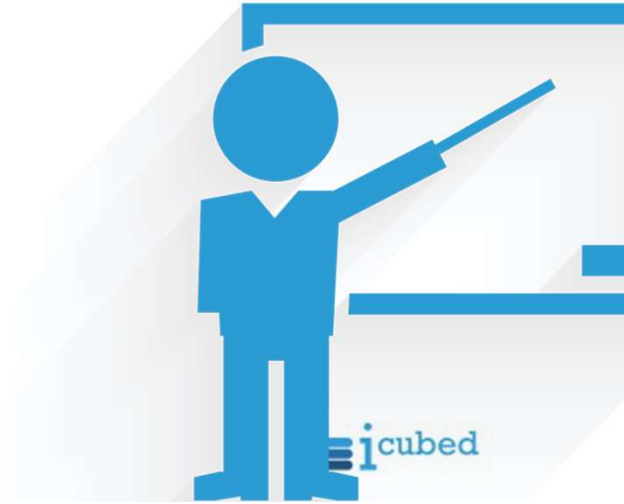
```
void Method(ref int refArgument)
{
    refArgument = refArgument + 44;
}

int number = 1;
Method(ref number);
Console.WriteLine(number);
// Output: 45
```

Demo

Routine

Passaggio dei parametri



icubed

Esercitazione n.3

Realizzare un'applicazione console che consenta di eseguire il calcolo della bolletta della luce.

Si richiede di sviluppare un menù attraverso cui l'utente può scegliere di:

- inserire i propri dati (nome, cognome e kwH consumati);
- richiedere il calcolo del costo della bolletta che è costituito da una quota fissa di 40€ più il prodotto tra I kwH e 10
- stampare a video i valori della bolletta, inclusi nome, cognome e costo pagato.

Ciascuna delle operazioni descritte sopra dovrà essere implementata attraverso una opportune routine.