

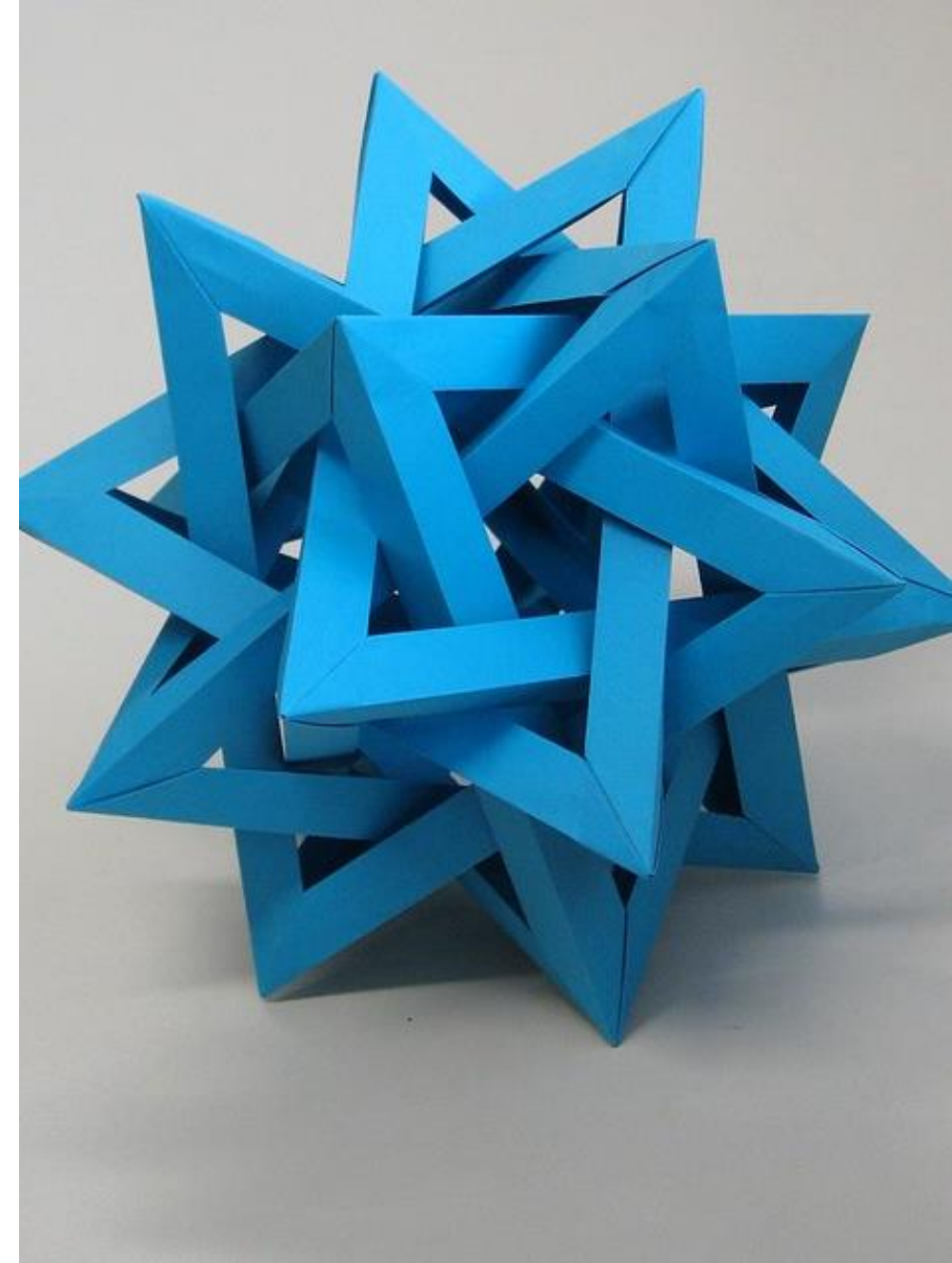


Unità P1: Introduzione alla programmazione

BREVE INTRODUZIONE AD HARDWARE,
SOFTWARE, E SVILUPPO DI ALGORITMI



Capitolo 1



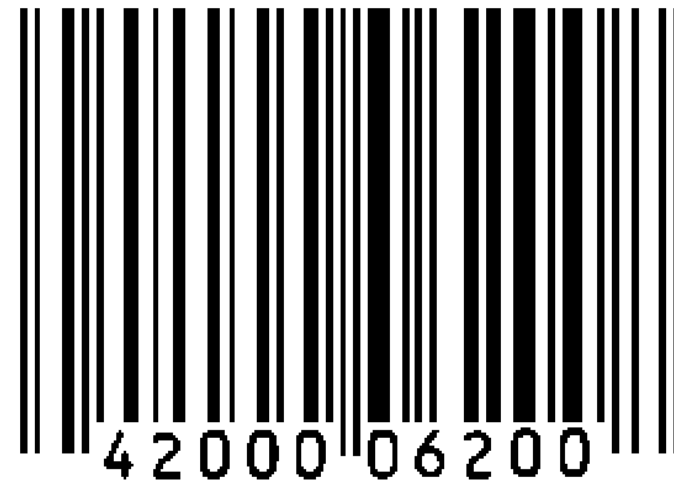
Unità P1: Obiettivi

- Introduzione agli elaboratori elettronici e alla programmazione
 - Hardware e software dei computer e programmazione
 - Scrivere ed eseguire il primo programma in Python
 - Diagnosticare e correggere errori di programmazione
 - Usare pseudo-codice per descrivere un algoritmo
- Diagrammi di Flusso (Flow Chart) come supporto per il Problem Solving
 - Rappresentazione
 - Passaggi progettuali
- Introduzione a Python
 - Strumenti
 - Linguaggio

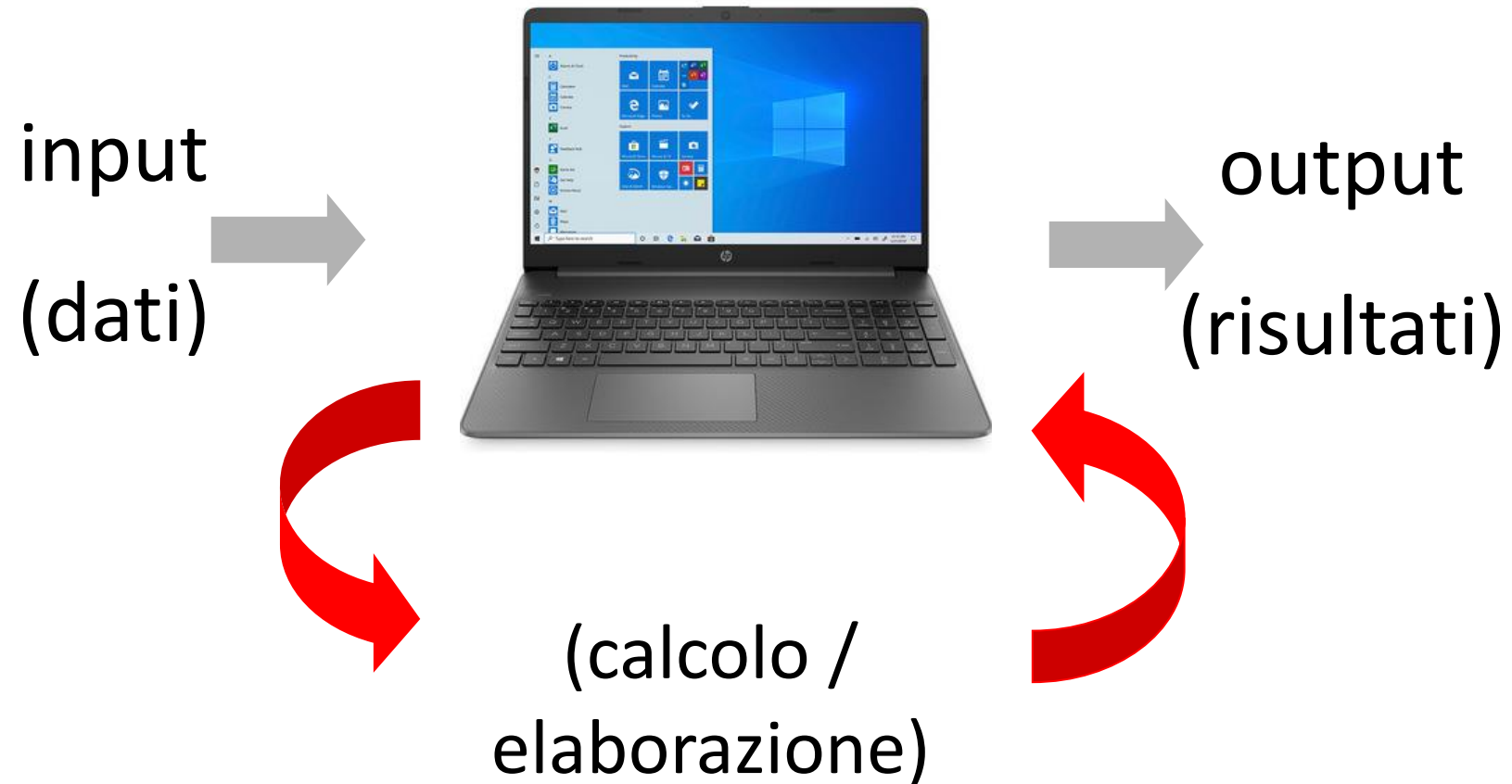
Introduzione ai sistemi di elaborazione

Definizione di Informatica

- L'Informatica (Computer Science) è la scienza che si occupa dello studio della **rappresentazione e manipolazione dell'informazione**



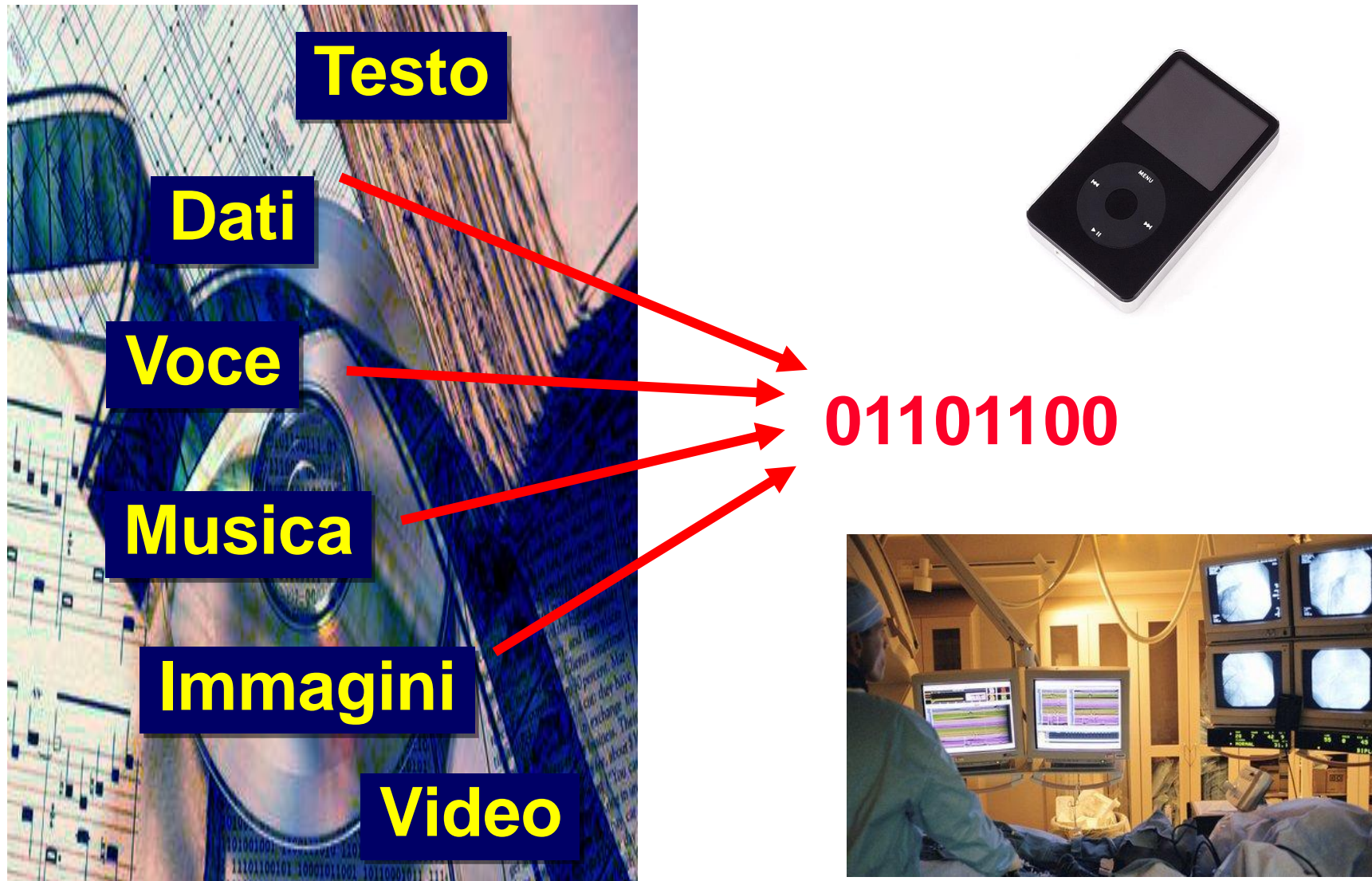
Elaboratore elettronico



Problemi

- Come codificare i dati in un formato che sia comprensibile dall'elaboratore
- Come codificare gli ordini come sequenza di operazioni che produce l'elaborazione desiderata
- Come decodificare il risultati in un formato che possa essere compreso dall'utente umano

Informazione digitale: tutto diventa «bit»



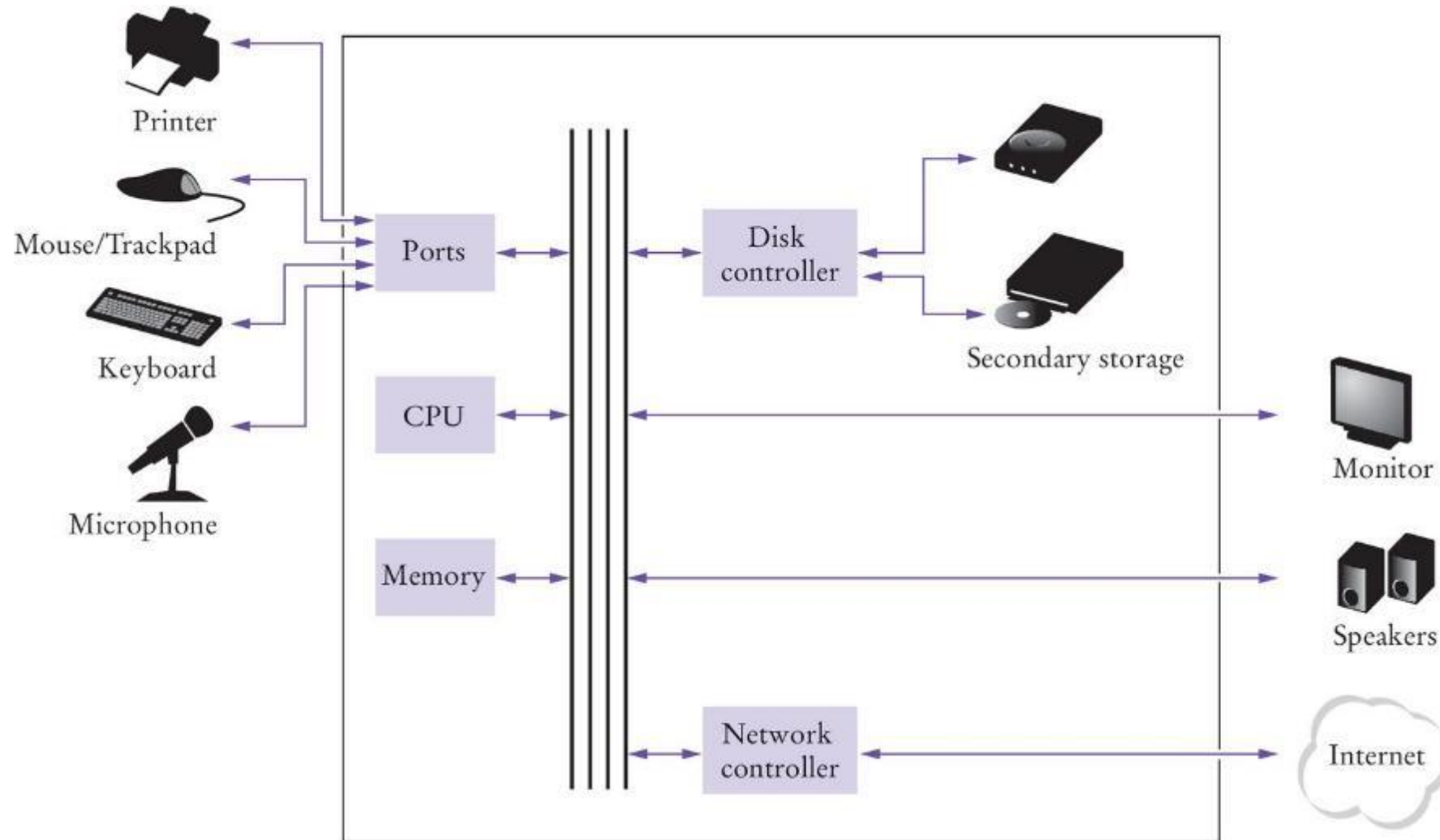
Hardware e Software

- Un calcolatore elettronico è composto da due parti:
 - Hardware: la componente fisica, consiste di dispositivi elettronici e parti meccaniche, magnetiche ed ottiche
 - Software: la parte «intangibile», consiste di:
 - Programmi: le «istruzioni» per l'hardware
 - Dati: le informazioni su cui lavorano i programmi

Hardware

- **Hardware** comprende gli elementi fisici di un sistema di elaborazione
 - Esempi: monitor, mouse, memoria esterna, tastiera,
- La **central processing unit** (CPU) controlla l'esecuzione del programma e l'elaborazione di dati
- **I dispositivi di memoria** comprendono la memoria interna (RAM) e la memoria secondaria
 - Hard disk
 - Dischi Flash
 - CD/DVD
- **I dispositivi di Ingresso / Uscita (Input / output)** permettono all'utente di interagire con il computer
 - Mouse, tastiera, stampante, schermo, ...

Vista semplificata dell'hardware di un PC



Software

- **Software** viene tipicamente sviluppato sotto forma di «programma applicativo» (App)
 - Microsoft Word è un esempio di software
 - I Giochi elettronici sono software
 - I sistemi operativi ed i driver dei dispositivi sono anch'essi software
- **Software**
 - Il Software è una sequenza di istruzioni e decisioni implementata in qualche linguaggio e tradotta in una forma che possa essere eseguita nel computer
 - Il Software gestisce i dati utilizzati dalle varie istruzioni
- I computer eseguono istruzioni molto semplici in rapida successione
 - Le istruzioni più semplici vengono raggruppate per eseguire compiti più complessi
- La programmazione è l'atto di progettare ed implementare i programmi software

Programmi

- Un programma per computer indica al computer la **sequenza di passi** necessaria a completare un determinato compito
 - Il programma consiste di un (elevatissimo) numero di istruzioni primitive (semplicissime)
- I computer possono eseguire un ampio spettro di compiti perché possono **eseguire** diversi programmi
 - Ciascun programma è progettato per indirizzare il computer affinché lavori su un compito specifico
- **Programmazione:**
 - È l'atto (e anche l'**arte**) di progettare, implementare e verificare (testare) i programmi

Eseguire un programma

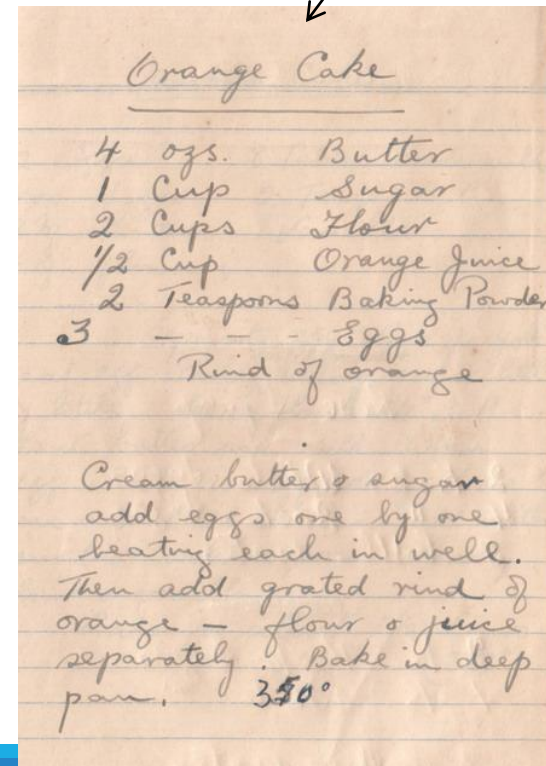
- Le istruzioni di un programma ed i relativi dati (come testi, numeri, audio o video) sono memorizzati in formato digitale
- Per eseguire un programma, questo deve essere portato in memoria, dove la CPU lo possa leggere
- La CPU esegue il programma un'istruzione per volta
 - Il programma può anche reagire agli input provenienti dall'utente
- Le istruzioni e gli input dell'utente determinano l'esecuzione del programma
 - La CPU legge i dati (compreso l'input utente), li modifica e li riscrive nuovamente in memoria, sullo schermo, o sulla memoria di massa

Cucinare vs Programmare

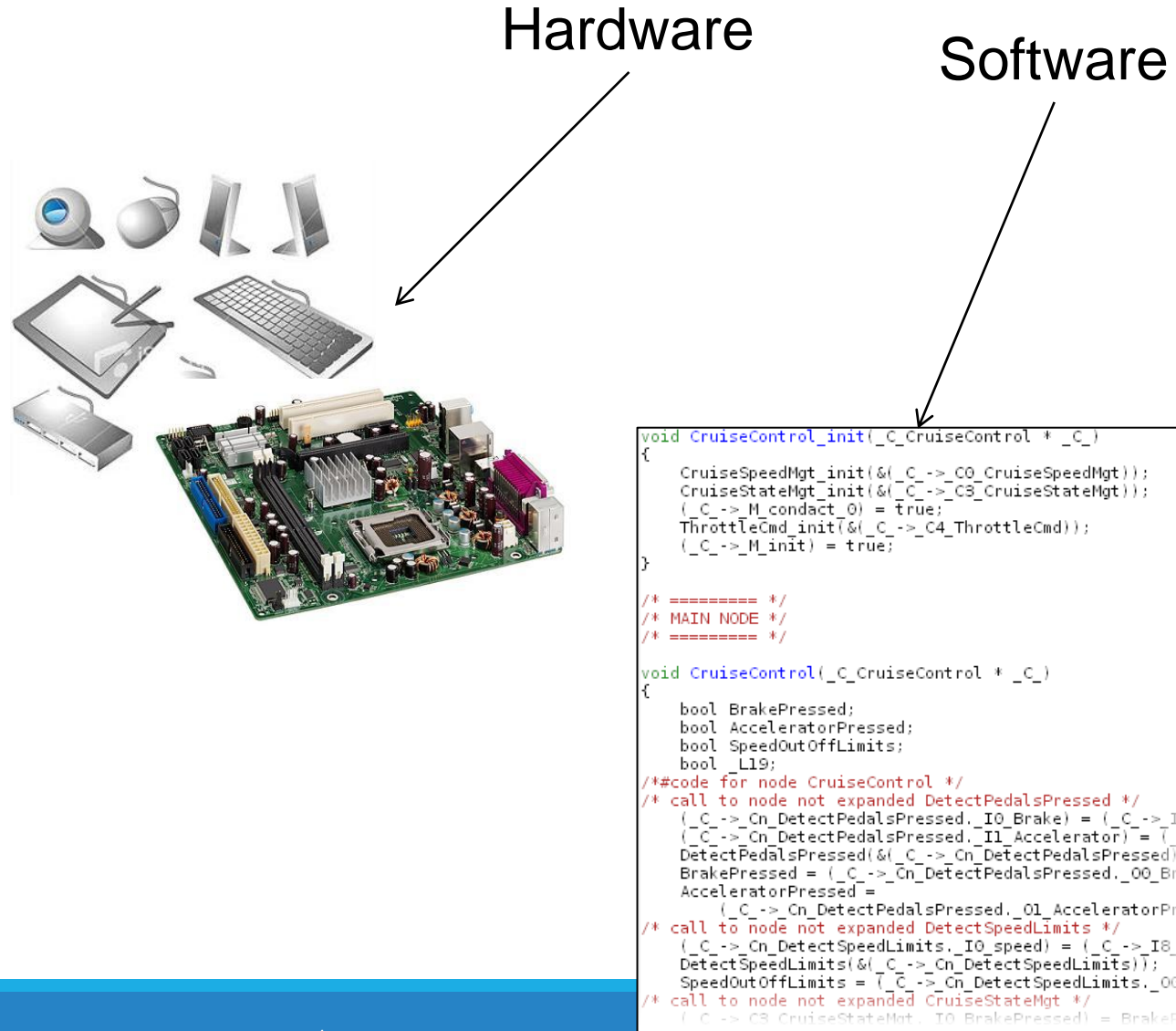


Hardware

Software



Cucinare vs Programmare

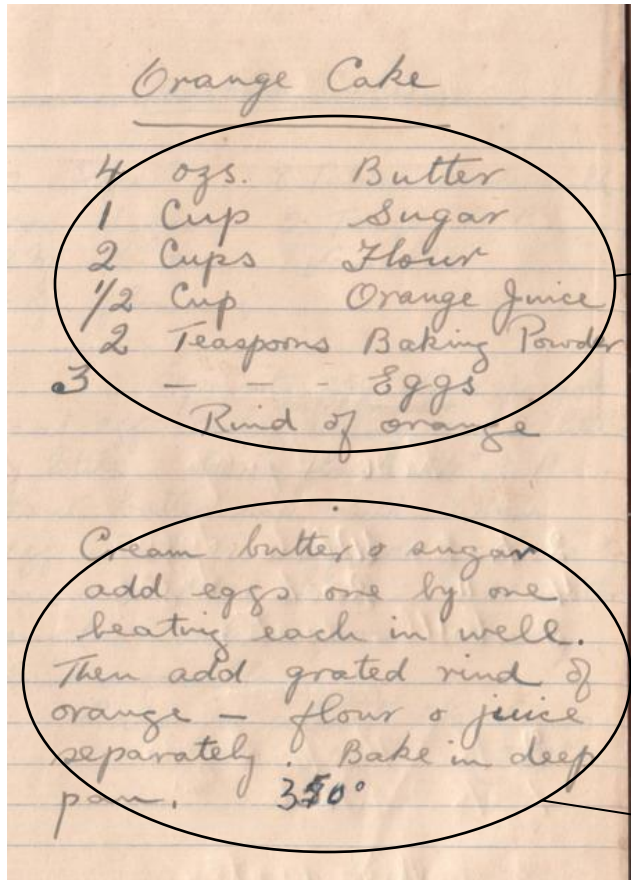


```
void CruiseControl_init(_C_CruiseControl * _C_)
{
    CruiseSpeedMgt_init(&(_C->_C0_CruiseSpeedMgt));
    CruiseStateMgt_init(&(_C->_C3_CruiseStateMgt));
    (_C->_M_conduct 0) = true;
    ThrottleCmd_init(&(_C->_C4_ThrottleCmd));
    (_C->_M_init) = true;
}

/* ===== */
/* MAIN NODE */
/* ===== */

void CruiseControl(_C_CruiseControl * _C_)
{
    bool BrakePressed;
    bool AcceleratorPressed;
    bool SpeedOutOffLimits;
    bool _L19;
    /*code for node CruiseControl */
    /* call to node not expanded DetectPedalsPressed */
    (_C->_Cn_DetectPedalsPressed._I0_Brake) = (_C->_I1
    (_C->_Cn_DetectPedalsPressed._I1_Accelerator) = T
    DetectPedalsPressed(&(_C->_Cn_DetectPedalsPressed))
    BrakePressed = (_C->_Cn_DetectPedalsPressed._00_Br
    AcceleratorPressed =
        (_C->_Cn_DetectPedalsPressed._01_AcceleratorPr
    /* call to node not expanded DetectSpeedLimits */
    (_C->_Cn_DetectSpeedLimits._I0_speed) = (_C->_I8_
    DetectSpeedLimits(&(_C->_Cn_DetectSpeedLimits));
    SpeedOutOffLimits = T_C->_Cn_DetectSpeedLimits._00
    /* call to node not expanded CruiseStateMgt */
    (_C->_C3_CruiseStateMgt._I0_BrakePressed) = BrakeP
```


Cucinare vs Programmare



Ingredients/Data

Instructions/Code

```
/* ===== */  
/* MAIN NODE */  
/* ===== */  
  
void CruiseControl(_C_CruiseControl * _C_)  
{  
    bool BrakePressed;  
    bool AcceleratorPressed;  
    bool SpeedOutOffLimits;  
    bool _L19;  
  
    /*code for node CruiseControl */  
    /* call to node not expanded DetectPedalsPressed */  
    (_C->Cn_DetectPedalsPressed._I0_Brake) = (_C->I0_Brake);  
    (_C->Cn_DetectPedalsPressed._I1_Accelerator) = (_C->I1_Accelerator);  
    DetectPedalsPressed(&(_C->Cn_DetectPedalsPressed));  
    BrakePressed = (_C->Cn_DetectPedalsPressed._O0_BrakePressed);  
    AcceleratorPressed = (_C->Cn_DetectPedalsPressed._O1_AcceleratorPressed);  
  
    /* call to node not expanded DetectSpeedLimits */  
    (_C->Cn_DetectSpeedLimits._I0_speed) = (_C->I0_speed);  
    DetectSpeedLimits(&(_C->Cn_DetectSpeedLimits));  
    SpeedOutOffLimits = (_C->Cn_DetectSpeedLimits._O0_SpeedOutOffLimits);  
  
    /* call to node not expanded CruiseStateMgt */  
    (_C->Cn_CruiseStateMgt._I0_BrakePressed) = BrakePressed;  
    (_C->Cn_CruiseStateMgt._I1_AcceleratorPressed) = AcceleratorPressed;  
    (_C->Cn_CruiseStateMgt._I2_SpeedOutOffLimits) = SpeedOutOffLimits;  
    CruiseStateMgt(_C->Cn_CruiseStateMgt);  
}
```

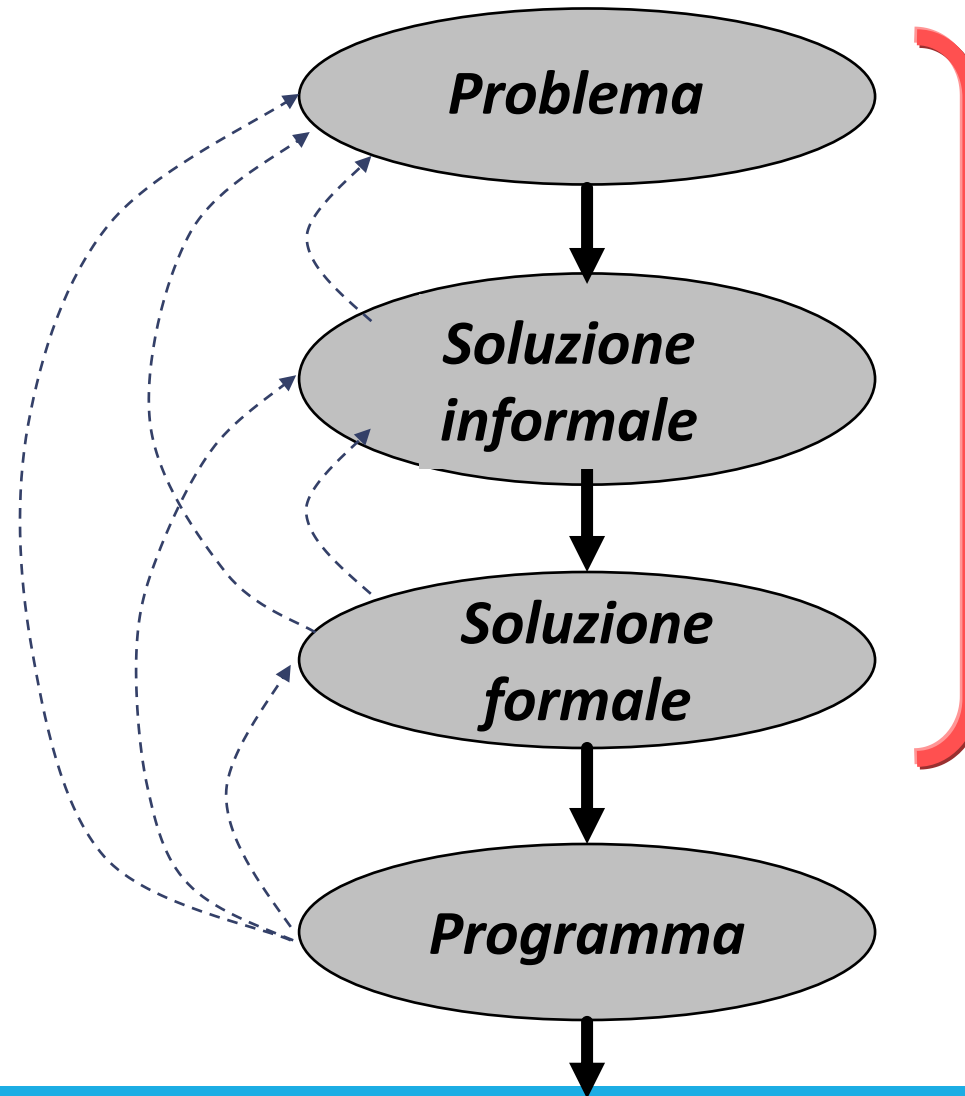
Cosa vuol dire “programmare”?

- La programmazione consiste nello scrivere un «documento» (**file sorgente**) che descrive la soluzione al problema considerato
- In generale, “**la**” soluzione a un dato problema **non** esiste
 - La programmazione consiste nel trovare la soluzione più efficiente ed efficace per il problema, secondo opportune metriche

Cosa vuol dire “programmare”?

- Programmare è un’attività “**creativa**”!
 - Ogni problema è **diverso** da ogni altro problema
 - Non ci sono soluzioni sistematiche/analitiche o soluzioni “**universali**”!
- Programmare è un’operazione complessa
 - Un approccio “**diretto**” (dal problema direttamente al codice sorgente) è **impraticabile**
 - Solitamente, lo sviluppo è organizzato in diverse fasi (**raffinamenti**) successivi

Sviluppo di un programma

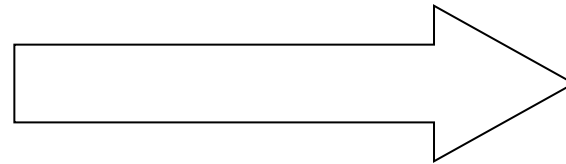


Cosa impareremo in questo corso?

- Dalla specifica di un problema, fino alla realizzazione di una soluzione a tale problema, sotto forma di un programma



Costruire
un programma

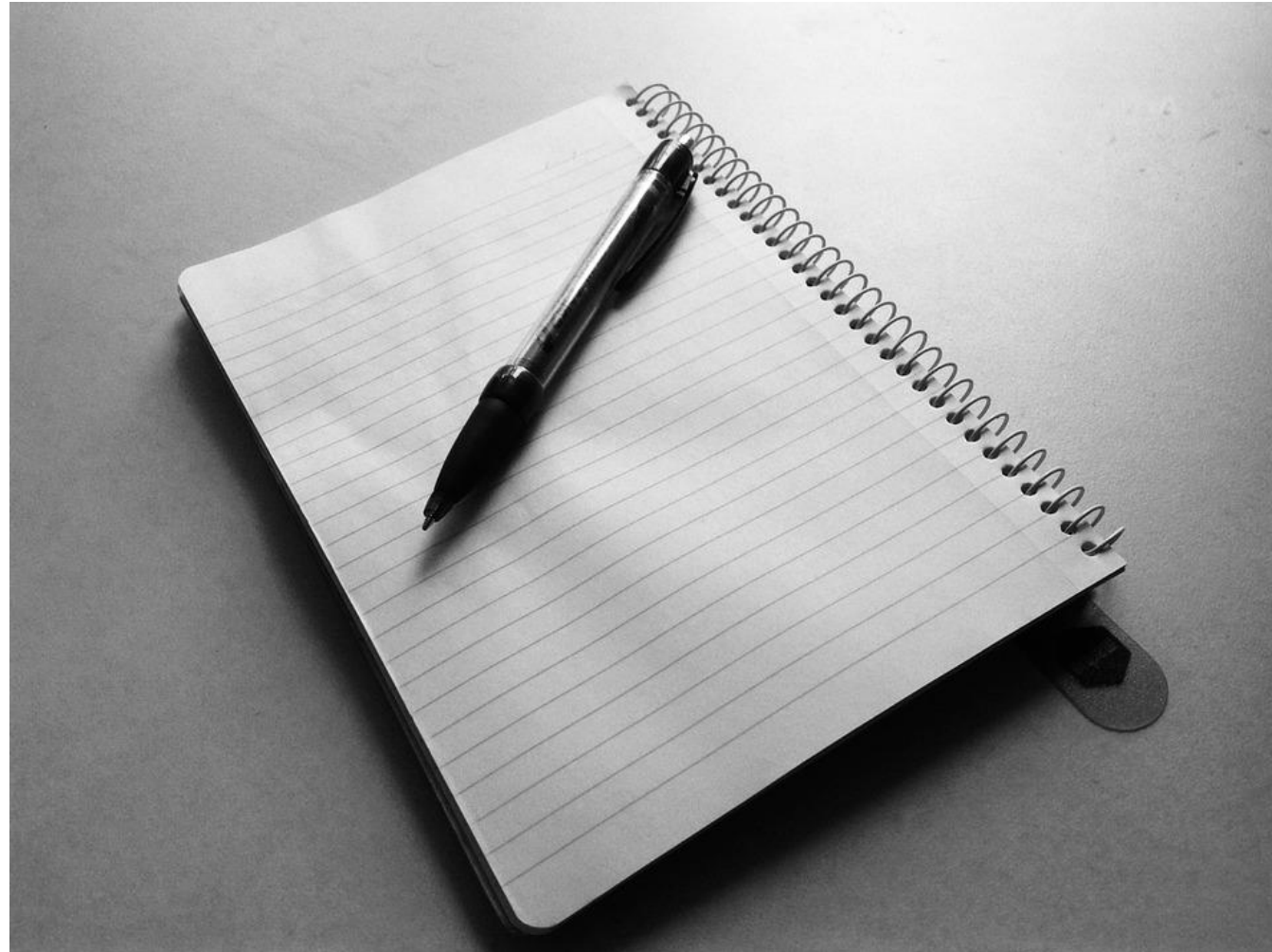


Il primo corso di INGEGNERIA al Politecnico

Cosa impareremo in questo corso?

- Ad acquisire la **predisposizione mentale** (*forma mentis*) necessaria ad affrontare compiti di «**problem posing**» e «**problem solving**»
 - **Analizzare** un problema e **decomporlo** in problemi più piccoli
 - **Descrivere** la soluzione di un problema, in modo **chiaro** e non ambiguo
 - **Analizzare** ed esplicitare i passi del nostro **ragionamento**
- Queste capacità sono utili in tutte le discipline scientifiche (e non scientifiche)
- In particolare...
 - ...a **PENSARE** come ‘pensa’ un computer
 - ... **PARLARE** ad un computer in modo che ci possa ‘comprendere’

Strumenti per il Problem Solving



Le Regole

- Devo risolvere il problema ed immaginare di avere solamente un foglio di carta (memoria) ed una penna
- Ogni informazione che ho bisogno di ricordare deve essere scritta sul foglio di carta

Esercizio

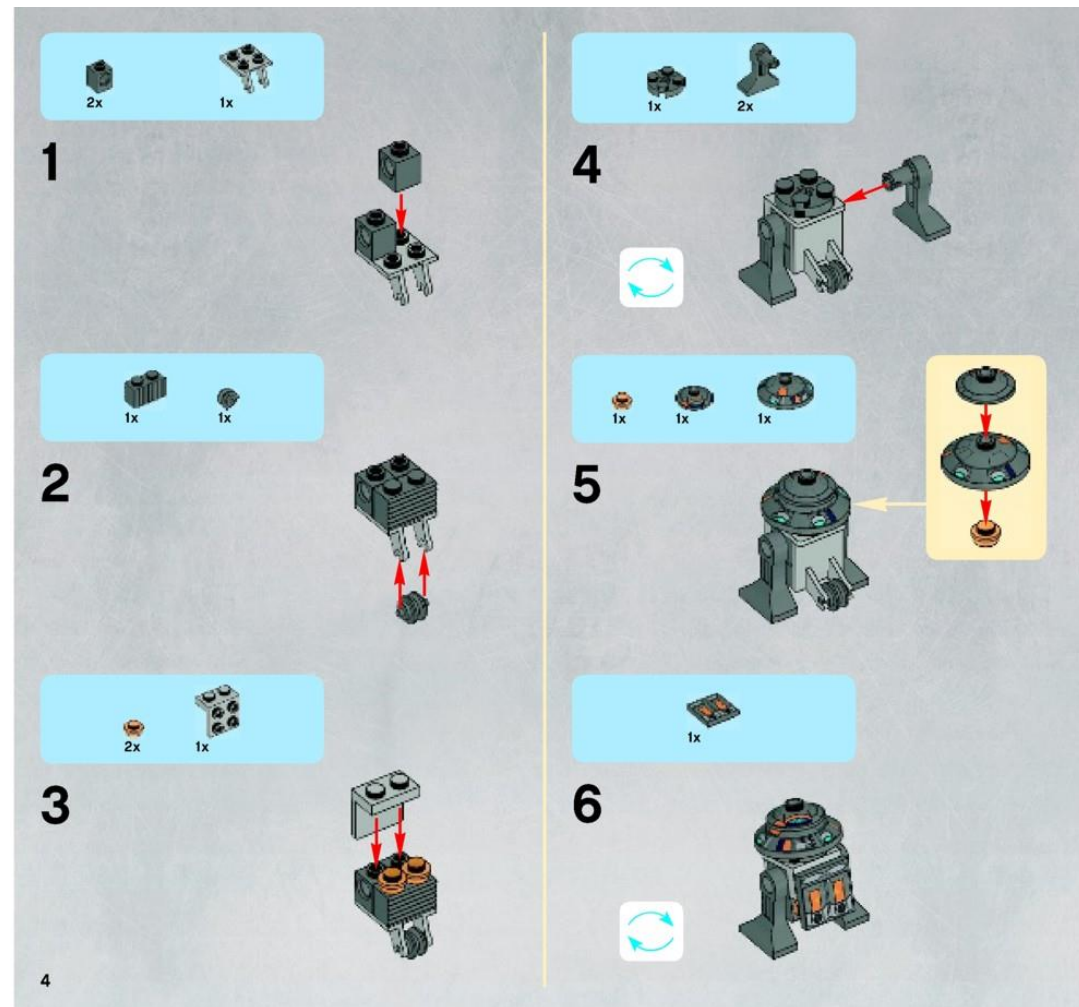
- Trovare chi sia lo studente più anziano presente nell'aula

Algoritmi



1.7

Algoritmo



Linguaggio visuale
strutturato

Operazioni sequenziali

Sotto-operazioni

Ripetizioni

Pensare come un programmatore ☹️

**Wife : Honey, please go to the super market
and get 1 bottle of milk. If they have bananas,
bring 6.**

Pensare come un programmatore ☹️

Wife : Honey, please go to the super market and get 1 bottle of milk. If they have bananas, bring 6.

He came back with 6 bottles of milk.

Wife: Why the hell did you buy 6 bottles of milk?!?!

Husband (confused): BECAUSE THEY HAD BANANAS.

He still doesn't understand why his wife yelled at him since he did exactly as she told him.

Una prima definizione

- **Algoritmo**
- Un algoritmo è una descrizione **passo-passo** di come **risolvere un problema**

Introduzione agli algoritmi

- Per fare in modo che un computer esegua un compito, il primo passo è scrivere un algoritmo
- Un **Algoritmo** è:
 - Una sequenza (l'ordine è importante!) di azioni da compiere (istruzioni) per svolgere il compito dato, e raggiungere un obiettivo specifico
 - Come una 'ricetta'
- Per i problemi più complessi, gli sviluppatori software studiano un algoritmo, poi lo formalizzano come pseudo-codice o diagrammi di flusso, prima di iniziare la scrittura di un programma vero e proprio
- Sviluppare algoritmi richiede fondamentalmente capacità di **problem solving**
 - Tali capacità sono utili in molti campi, anche al di fuori dell'informatica

Algoritmo: Definizione formale

- Un **algoritmo** descrive una sequenza di passi con le seguenti caratteristiche:
 - **Non ambigua**
 - Non vi possono essere delle «assunzioni» sulla conoscenza necessaria per eseguire l'algoritmo
 - L'algoritmo usa istruzioni precise
 - **Eseguibile**
 - L'algoritmo può essere svolto, in pratica
 - **Termina**
 - L'algoritmo, prima o poi, dovrà necessariamente terminare e fermarsi

Problem Solving: Progettazione di algoritmi

- Gli algoritmi sono semplicemente dei piani operativi
 - Piani dettagliati che descrivono i passi per risolvere un problema specifico
- Alcuni li conosciamo già
 - Calcolare l'area di un cerchio
 - Trovare la lunghezza dell'ipotenusa di un triangolo
- Alcuni sono più complessi e richiederanno più passi
 - Risolvere un'equazione di 2° grado
 - Trovare il MCD tra due numeri
 - Calcolare π con 100 cifre decimali
 - Calcolare la traiettoria di un razzo

Algoritmi nella vita quotidiana

1. metti l'acqua
2. accendi il fuoco
3. aspetta
4. se l'acqua non bolle
torna a 3
5. butta la pasta
6. aspetta un po'
7. assaggia
8. se è cruda
torna a 6
9. scola la pasta



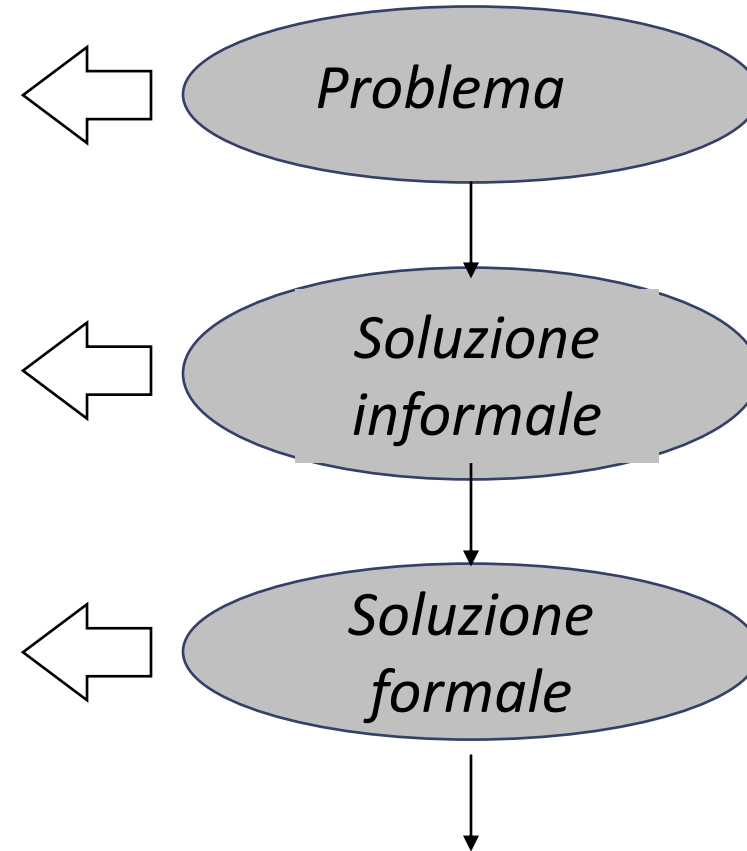
Chi si è accorto che
manca il sale?

Un semplice esempio

- Un semplice algoritmo per prepararti un succo d'arancia
 - Per semplicità, assumiamo che le seguenti condizioni siano verificate:
 - Hai un bicchiere pulito nell'armadio
 - Hai del succo d'arancia nel frigorifero
- Un algoritmo valido potrebbe essere:
 - Prendi un bicchiere dall'armadio
 - Vai al frigorifero e prendi la bottiglia del succo d'arancia
 - Apri la bottiglia del succo d'arancia
 - Versa il succo d'arancia dalla bottiglia al bicchiere
 - Rimetti la bottiglia del succo d'arancia nel frigorifero
 - Bevi il succo

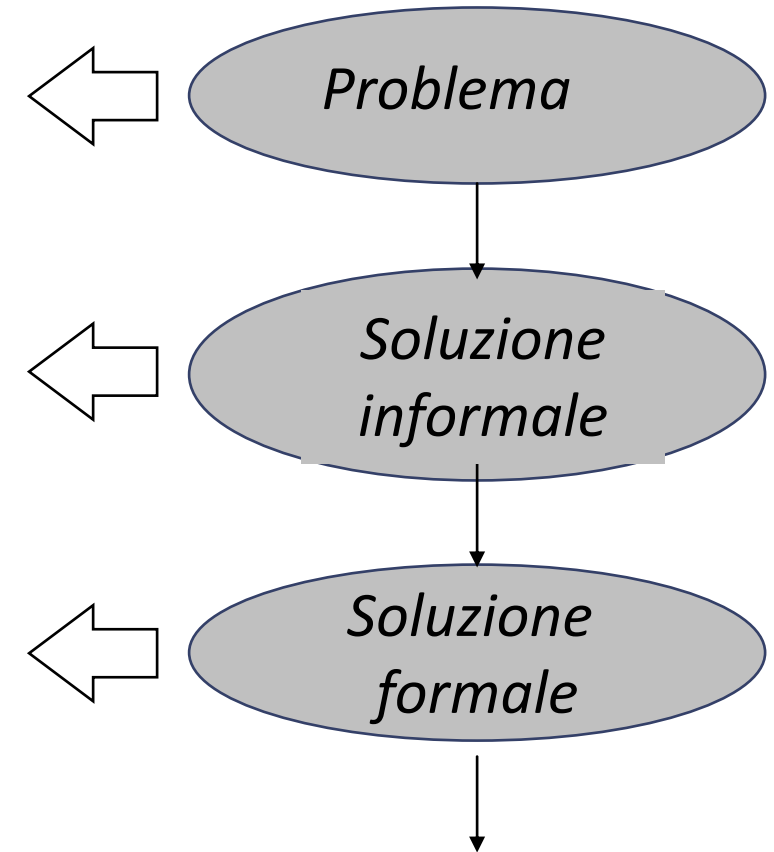
Esempio di Problem Solving

- Problema: Calcola il massimo tra A e B
- Soluzione: Il massimo è in numero maggiore tra A e B...
- Soluzione formale:
 1. inizialmente: $max = 0$
 2. se $A > B$ allora $max = A$; stop
 3. altrimenti $max = B$; stop



Esempio di Problem Solving

- Problema: Calcola il Massimo Comun Divisore (MCD) tra A e B
- Soluzione: Per definizione il MCD è il numero intero più grande che divide esattamente (sia A che B). Possiamo allora provare con tutti i numeri interi tra 1 ed A oppure B
- Soluzione formale: ???



Esempio: scegliere un'automobile

- Definizione del problema:
 - Avete la scelta di scegliere l'acquisto di un'auto tra due possibili
 - Una ha una maggiore efficienza (minor consumo di benzina), ma è anche più costosa
 - Conoscete il prezzo e l'efficienza (in km per litro) di entrambe le auto
 - Prevedete di tenere l'auto per 10 anni
 - Quale auto è più conveniente?

Sviluppare l'algoritmo

- Determinare gli input e l'output
- Dalla definizione del problema, sappiamo:
 - Car 1: Prezzo d'acquisto, Efficienza
 - Car 2: Prezzo d'acquisto, Efficienza
 - Prezzo al litro = \$1.40
 - Km annuali percorsi = 15,000
 - Durata = 10 anni
- Per ciascuna auto dovremo calcolare:
 - Effettivo carburante consumato in un anno
 - Costo annuale del carburante
 - Costi operativi di ciascuna auto
 - Costo totale dell'auto
- Sceglieremo l'auto con il costo minore

Formalizzazione della soluzione

■ Pseudo-codice

○ Pro

- Immediato

○ Contro

- Descrizione dell'algoritmo non molto astratta
- L'interpretazione è più complessa

■ Flow Chart

○ Pro

- Più intuitivo: formalismo grafico
- Descrizione dell'algoritmo più astratta

○ Contro

- Richiede l'apprendimento del significato dei blocchi
- Difficile rappresentare operazioni più complesse o troppo astratte

Tradurre l'algoritmo in pseudo-codice

- Dividere il problema in sotto-problemi più semplici
 - 'Calcola il costo totale' di ciascuna auto
 - Per calcolare il costo totale annuale dobbiamo calcolare i costi operativi
 - I costi operativi dipendono dal costo annuale del carburante
 - Il costo annuale è il prezzo al litro * il consumo annuale
 - Il consumo annuale è la percorrenza totale / l'efficienza
- Descrivere ciascuno dei sotto-problemi come pseudo-codice
 - $\text{costo totale} = \text{prezzo di acquisto} + \text{costi operativi}$

Pseudo-codice

- Per ciascuna auto, calcola il costo totale
 - Carburante consumato all'anno = chilometri percorsi all'anno / efficienza
 - Costo annuo carburante = prezzo al litro * carburante consumato all'anno
 - Costo operativo = durata * costo annuo carburante
 - Costo totale = prezzo di acquisto + costo operativo
- Se costo totale 1 < costo totale 2
 - Scegli auto 1
- Altrimenti
 - Scegli auto 2

Esempio: conto corrente bancario

- Definizione del problema:
 - Depositare \$10,000 in un conto corrente che garantisce un interesse del 5 per cento all'anno. Quanti anni sono necessari affinché il saldo del conto corrente arrivi al doppio della cifra originaria?
- Come lo risolviamo?
 - Metodo manuale
 - Fare una tabella
 - Aggiungere nuove righe fino a trovare il risultato
 - Usare un foglio di calcolo!
 - Costruiamo una formula
 - Calcoliamo ogni linea a partire da quella precedente

year	balance
0	10000
1	$10000.00 \times 1.05 = 10500.00$
2	$10500.00 \times 1.05 = 11025.00$
3	$11025.00 \times 1.05 = 11576.25$
4	$11576.25 \times 1.05 = 12155.06$

Sviluppo dei passi dell'algoritmo

- Depositare \$10,000 in un conto corrente che garantisce un interesse del 5 per cento all'anno. Quanti anni sono necessari affinché il saldo del conto corrente arrivi al doppio della cifra originaria?

■ Dividiamolo in passi

- Iniziamo con un valore 0 per l'anno ed un saldo di \$10,000
- Ripetiamo le seguenti operazioni finché il saldo rimane inferiore a \$20,000
 - Aggiungi 1 al valore dell'anno
 - Moltiplica il saldo per 1.05 (corrisponde all'aumento del 5%)
- La risposta sarà il valore finale dell'anno

year	balance
0	10000

year	balance
0	10000
1	10500

14	19799.32
15	20789.28

Tradurre in pseudo-codice

- Pseudo-codice
 - Via di mezzo tra il linguaggio naturale ed un linguaggio di programmazione
- Passi da eseguire
 - Poni il valore dell'anno a 0
 - Poni il valore del saldo a \$10,000
 - Finché il saldo è minore di \$20,000
 - Aggiungi 1 al valore dell'anno
 - Moltiplica il saldo per 1.05
 - Restituisci il valore dell'anno come risposta finale
- Lo pseudo-codice si traduce facilmente in Python

Dalla Soluzione al Programma

- Scrivere un programma è un'operazione quasi immediata, se si parte da una soluzione formale (pseudo-codice o flow chart)
- I linguaggi di programmazione offrono diversi costrutti ed istruzioni, di complessità variabile
 - Dipende dal linguaggio utilizzato

Quali linguaggi?

- Diversi livelli di astrazione
 - Linguaggi ad alto livello
 - Elementi del linguaggio hanno complessità equivalente ai blocchi dei diagrammi di flusso strutturati (condizionali, cicli,...)
 - Esempi: C, C++, Java, JavaScript, Python, ecc.
 - Indipendenti dall'hardware
 - Linguaggi 'assembler'
 - Elementi del linguaggio sono istruzioni microarchitetturali
 - Fortemente dipendenti dall'hardware
 - Esempio: Linguaggio Assembler del microprocessore Intel Core

Quali linguaggi? – Esempi

- Linguaggi ad alto livello

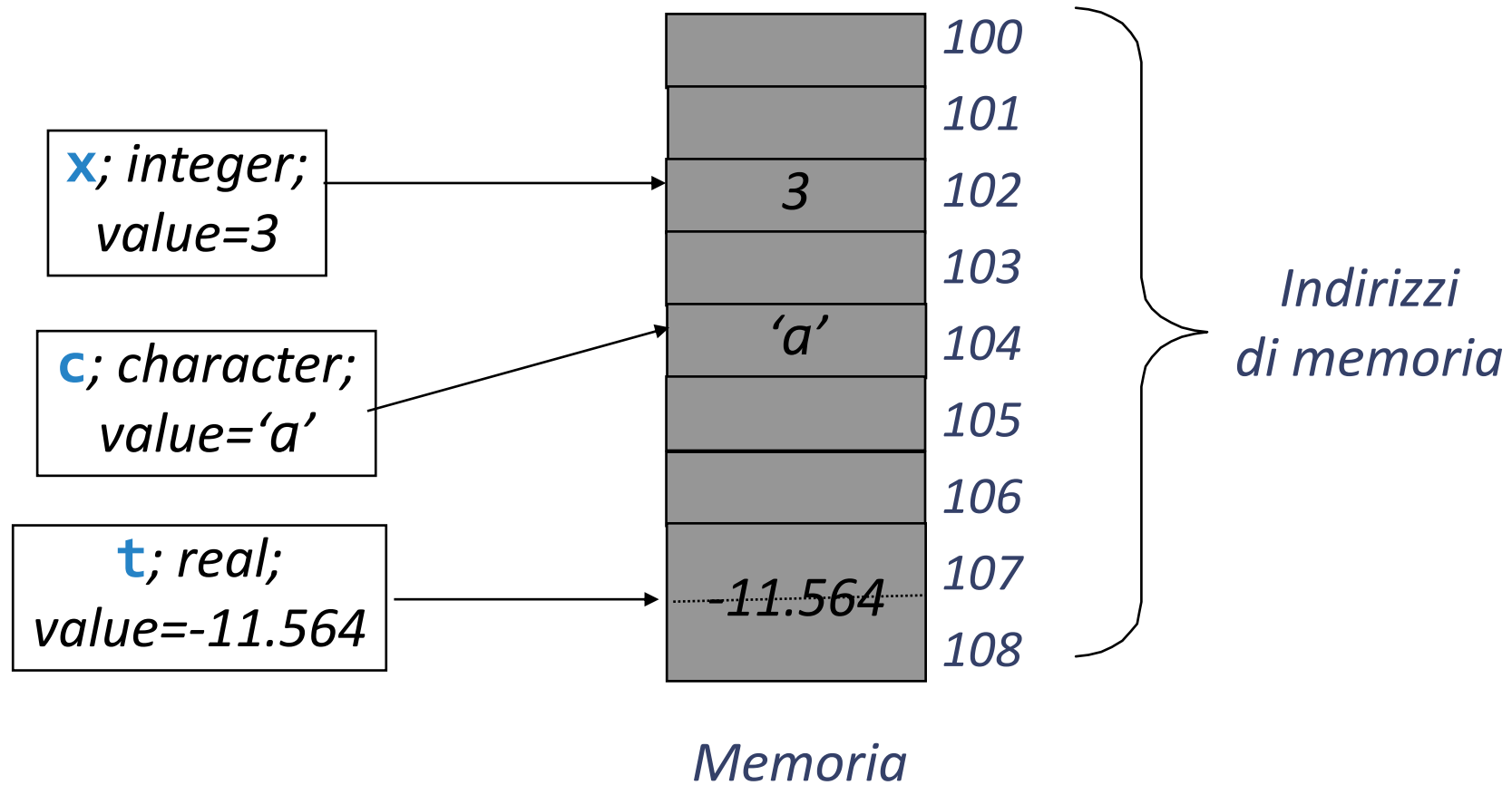
```
...  
if x > 3:  
    x = x + 1  
...
```

- Linguaggio assembler

```
...  
LOAD Reg1, Mem[1000]  
ADD Reg1, 10  
...
```

Specifico per una specifica
architettura (microprocessore)

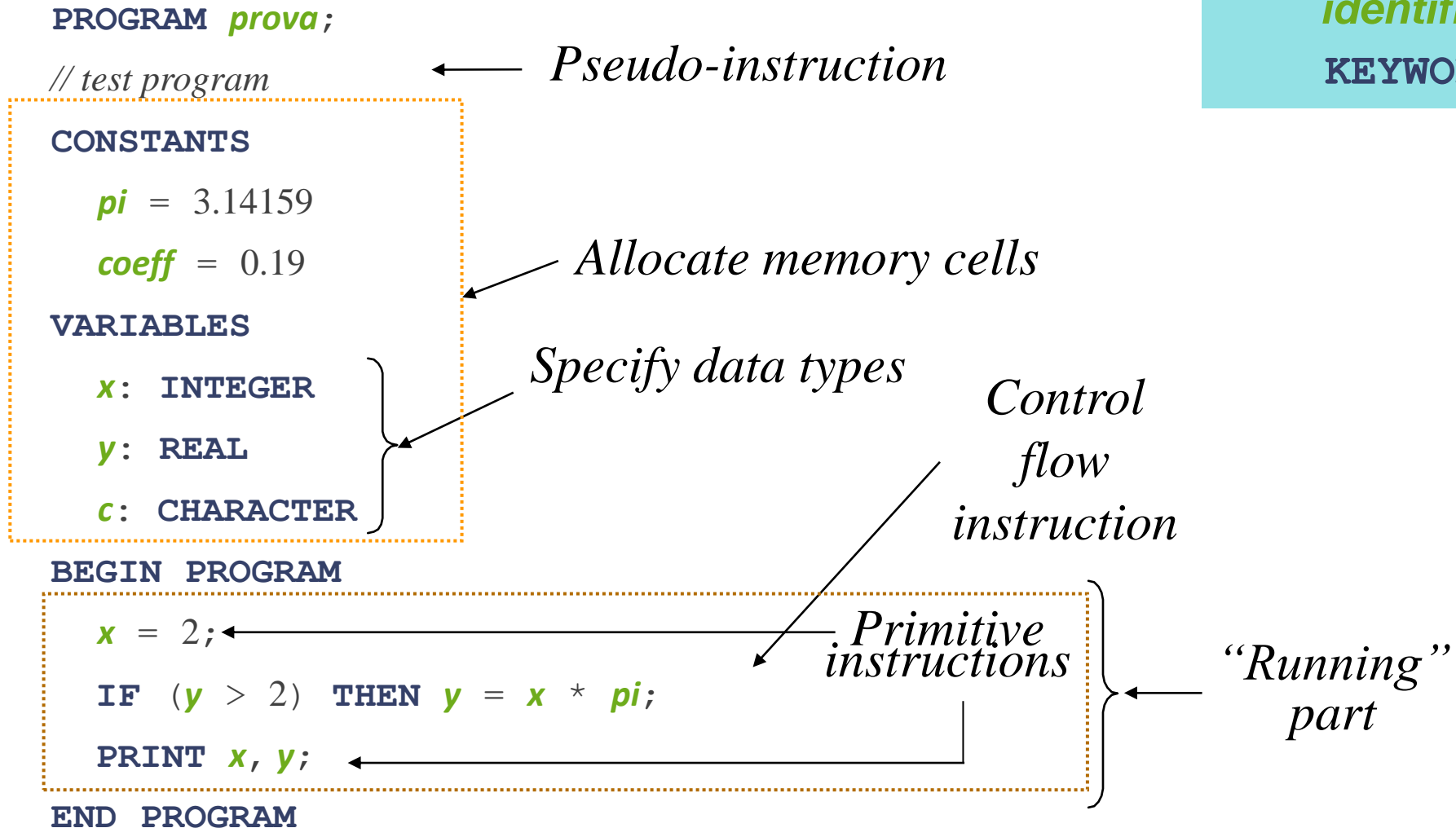
Astrazione dei dati



Istruzioni

- Operazioni supportate dal linguaggio di programmazione, che saranno eseguite traducendole in codice macchina
 - Pseudo-istruzioni
 - Direttive all'interprete o compilatore del linguaggio, non corrispondono ad effettivo codice eseguibile
 - Istruzioni elementari (o primitive)
 - Operazioni che corrispondono direttamente ad operazioni hardware
 - Esempio: interazioni con dispositivi di I/O, accesso ai dati, modifica dei dati
 - Istruzioni di controllo del flusso
 - Permettono l'esecuzione di operazioni complesse, controllando l'esecuzione di sequenze di istruzioni elementari

Esempio di programma



identifiers
KEYWORDS

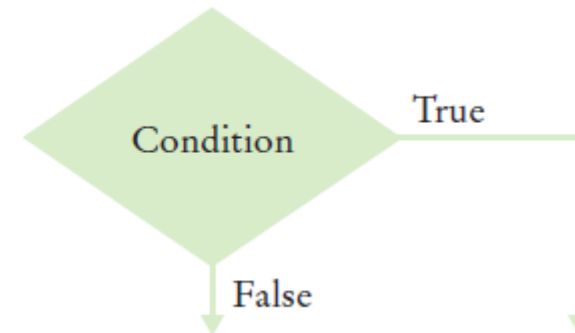
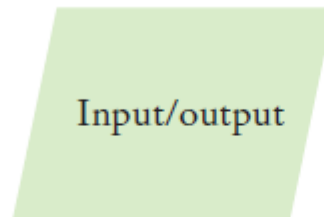
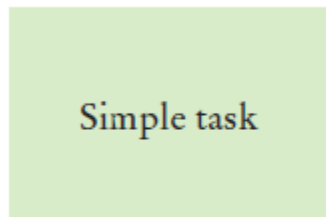
Usare Flow Chart per Sviluppare e Raffinare Algoritmi



3.5

Problem Solving: Flow Chart

- Un diagramma di flusso (flow chart) mostra la struttura delle decisioni e delle attività necessari a risolvere un problema
- Elementi principali dei flow chart:



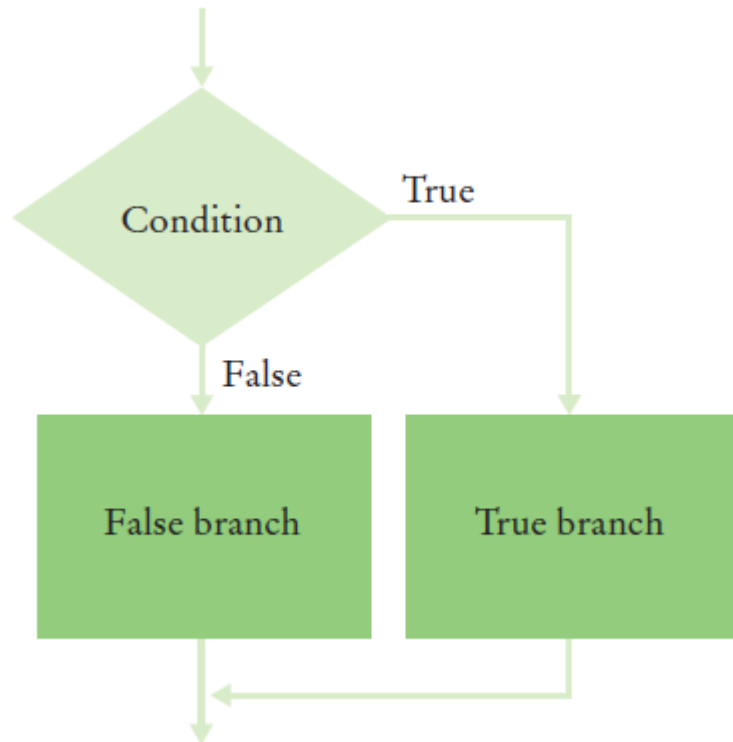
- Connetterli con frecce
 - Le frecce devono andare verso i blocchi, non verso altre frecce
- Ciascun ramo di una decisione può contenere altre attività ed altre decisioni

Usare i Flow Chart

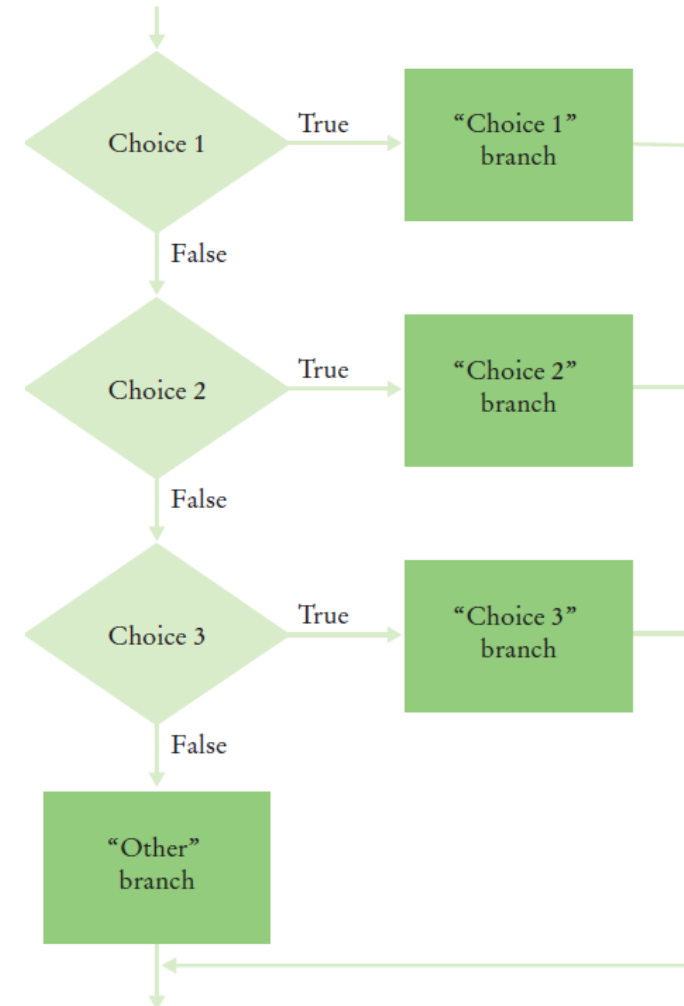
- I Flow Chart sono uno strumento eccellente
- Aiutano a visualizzare il flusso, lo svolgimento, dell'algoritmo
- Costruire un flow chart
 - Collegare le attività ed i blocchi di input / output nella sequenza in cui dovranno essere eseguiti
 - Quando occorre prendere una decisione, usare il rombo (istruzione condizionale) che possiede due uscite
 - Mai indirizzare una freccia verso un ramo «interno»

Flow chart condizionali

- Due alternative

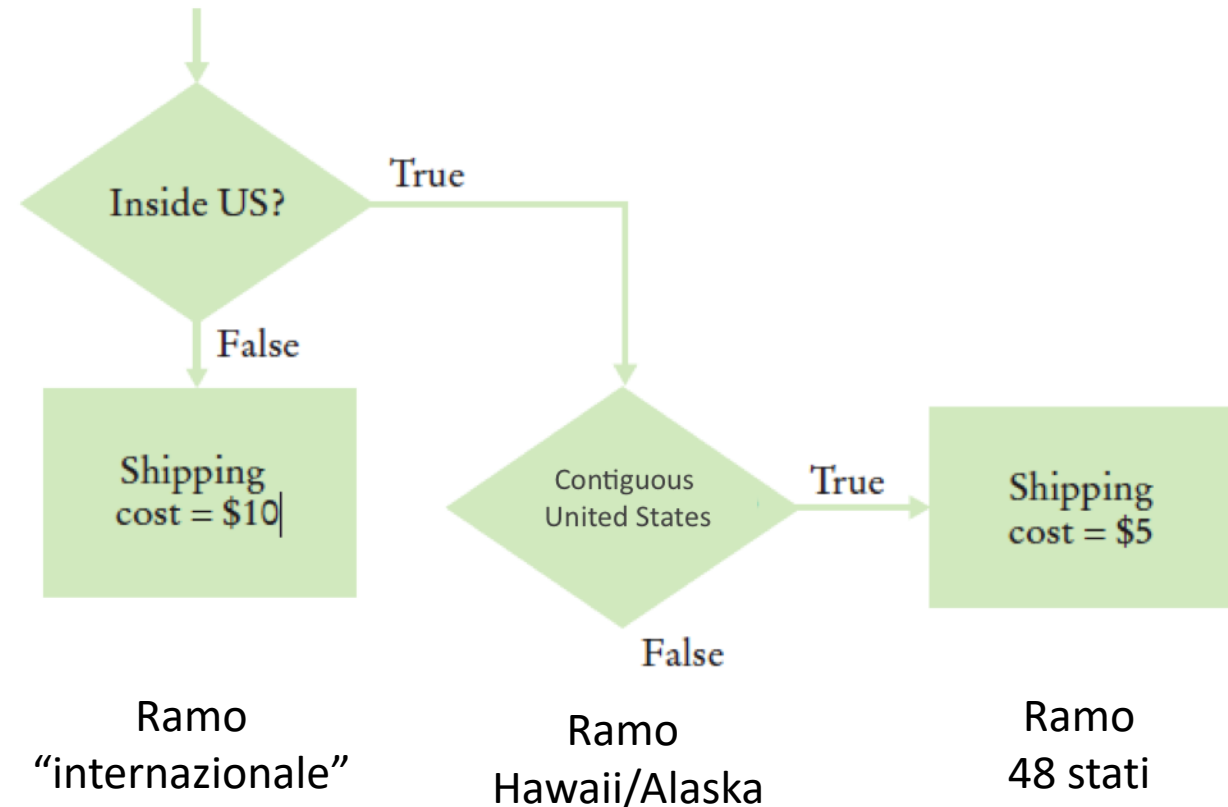


- Più di due alternative



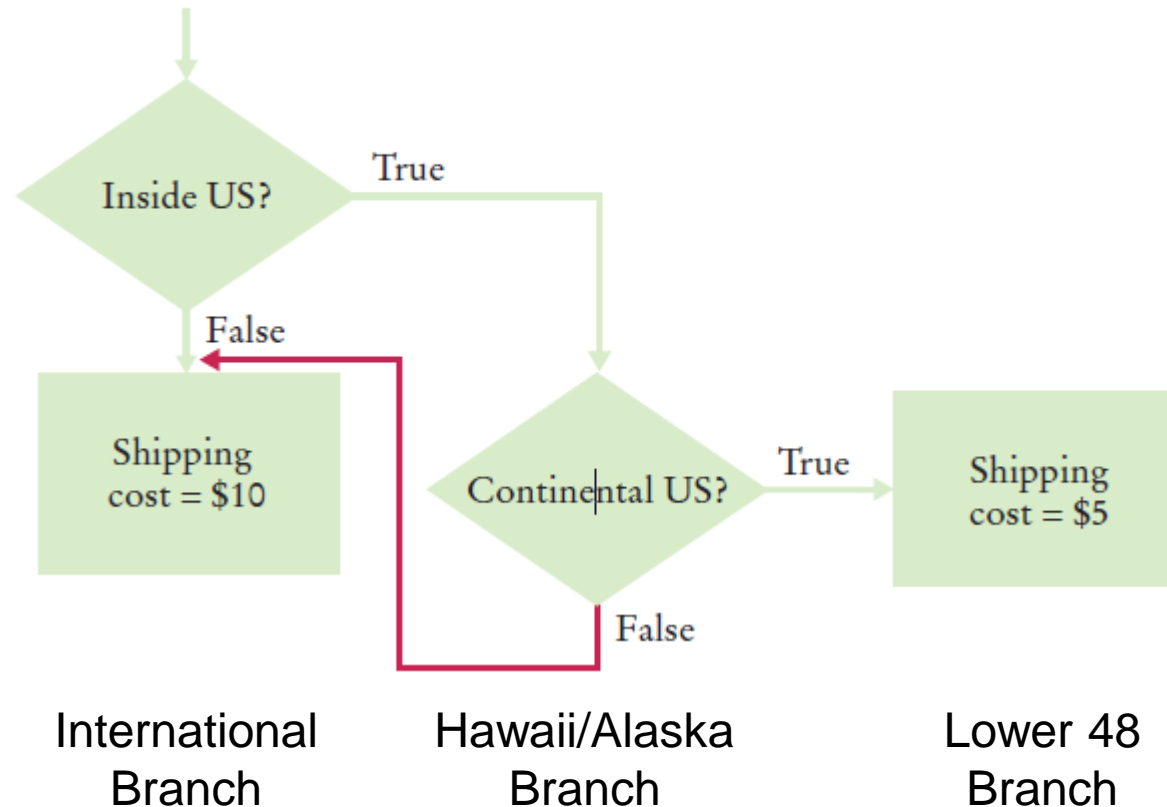
Flow Chart: Spese di spedizione

- Le spese di spedizione sono \$5 negli Stati Uniti continentali (48 stati) e \$10 verso Hawaii ed Alaska. Le spedizioni estere (extra-USA) costano anch'esse \$10
- Tre rami alternativi:



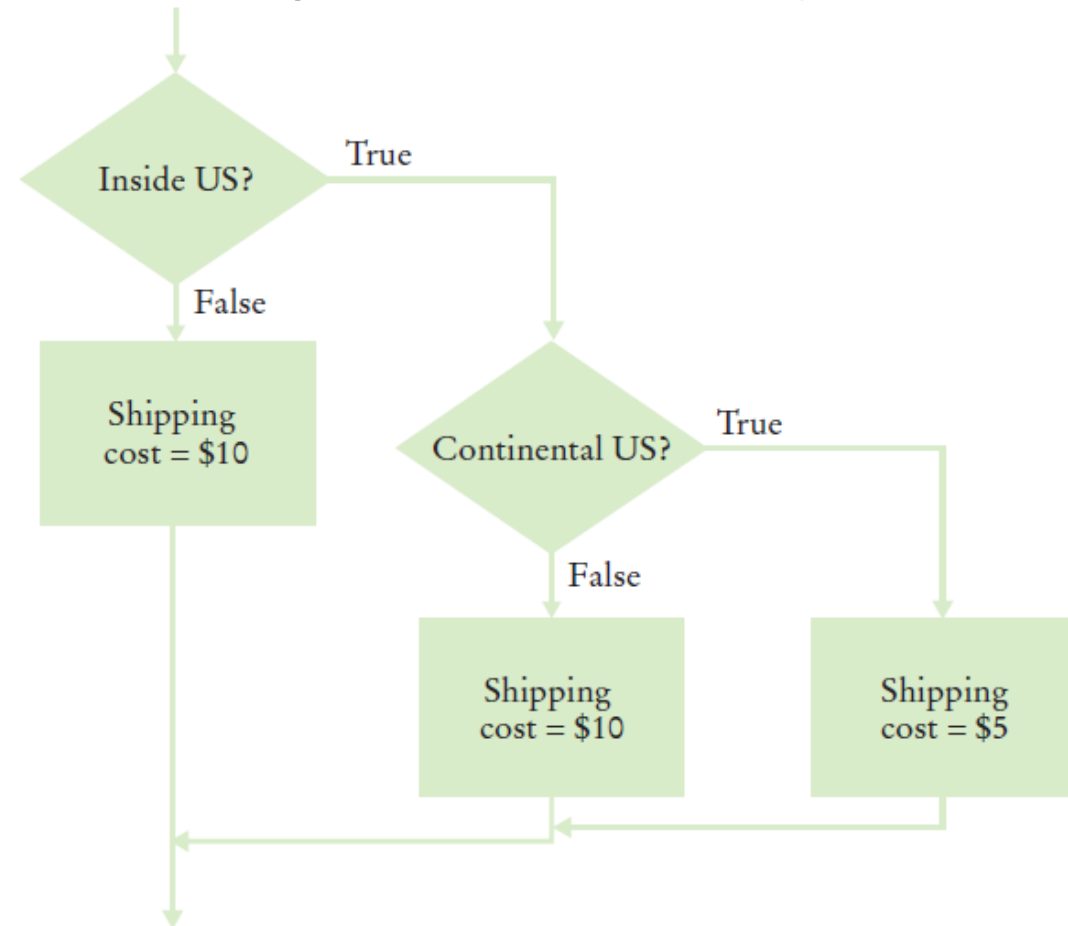
Non incrociare i rami

- Le spese di spedizione sono \$5 negli Stati Uniti continentali (48 stati) e \$10 verso Hawaii ed Alaska. Le spedizioni estere (extra-USA) costano anch'esse \$10
- **Non fate così!**



Flow Chart finale

- Le spese di spedizione sono \$5 negli Stati Uniti continentali (48 stati) e \$10 verso Hawaii ed Alaska. Le spedizioni estere (extra-USA) costano anch'esse \$10



Scelte e decisioni complesse sono difficili!

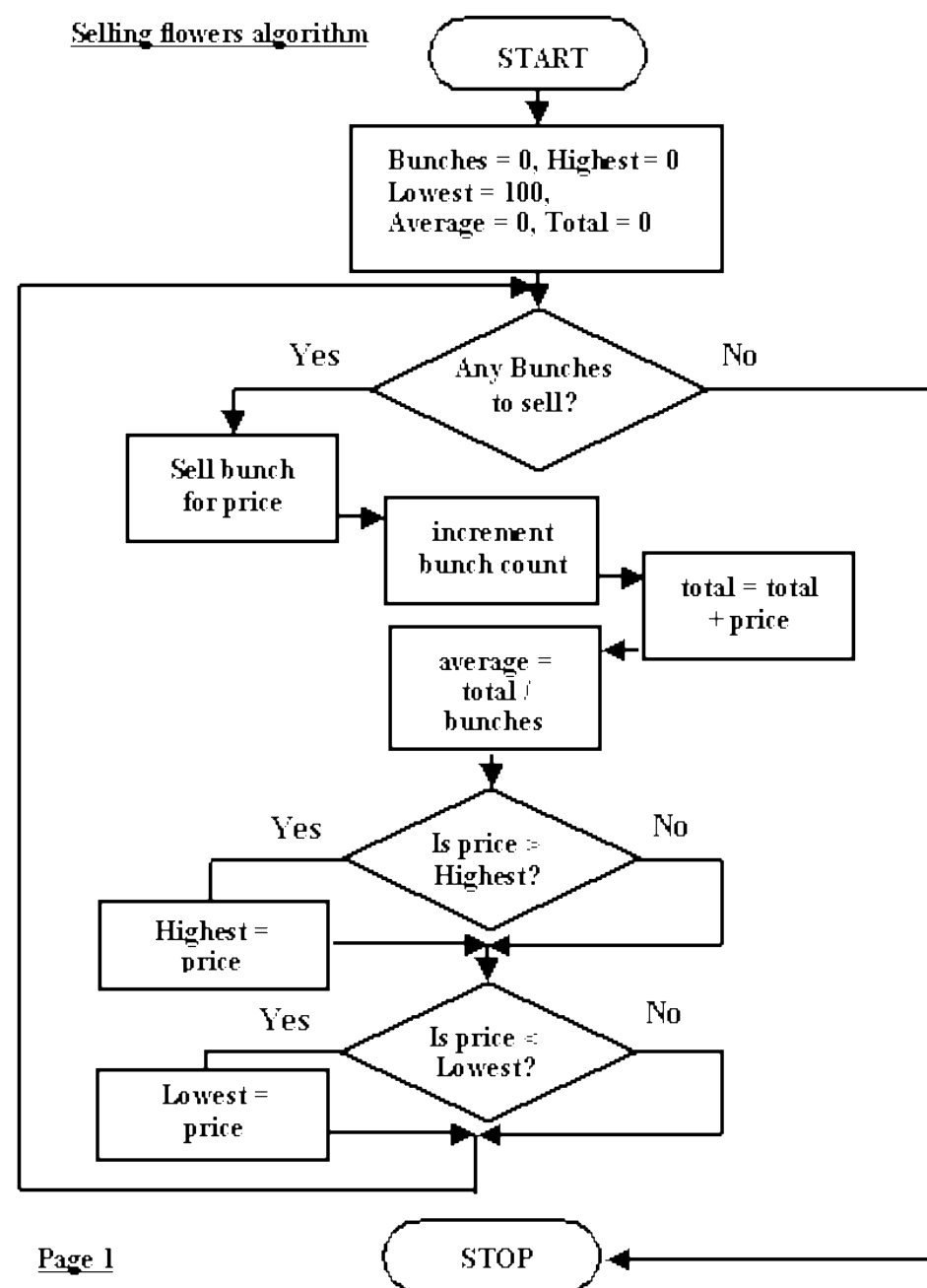
- I computer sono usati anche per ordinare e dirigere i bagagli negli aeroporti
- Questi sistemi:
 - Leggono le etichette sulle valigie
 - Ordinano gli elementi
 - Instradano gli elementi presso i nastri trasportatori
 - Operatori umani poi metteranno le valigie sui bus
- Nel 1993, Denver costruì un nuovo aeroporto con un sistema di gestione bagagli “allo stato dell’arte” in cui sostituiva gli operatori umani con carrelli robotizzati
 - Il sistema fallì
 - L’aeroporto non poteva aprire senza la gestione bagagli...
 - Il sistema fu sostituito (ci volle oltre 1 anno)
 - Il costo stimato: circa 1 miliardo di dollari (del 1994)
 - L’azienda che aveva progettato il sistema (ovviamente?) fallì

Esercizio

- Fred vende mazzi di fiori al centro commerciale
- Un giorno Joe, il capo di Fred, gli dice che in qualunque momento della giornata, egli (Joe) ha bisogno di sapere:
 - Quanti mazzi di fiori sono stati venduti
 - Qual era il valore del mazzo più costoso venduto
 - Qual era il valore del mazzo meno costoso venduto
 - Qual era il valore medio dei mazzi venduti

Esercizio

Selling flowers algorithm



Costruire Casi di Prova (Test Case)



3.6

Problem Solving: Test

- Per verificare la correttezza di un programma, occorre testarlo (collaudarlo)
 - Un Test (o Caso di Test) è un insieme di input che viene usato per verificare se il programma genera l'output corretto, in quel caso specifico
- Nessun test sarà mai completo al 100%, ma dovrebbe coprire tutti i possibili comportamenti del programma
- Cercare di ottenere una copertura completa di tutti i punti di decisione (alternative)
 - Creare test separati per ogni alternativa del programma (es., nazionale/internazionale)
 - Creare test le “condizioni limite”: il valore minimo (massimo), un valore appena sopra il minimo (max), appena sotto il min (max), appena sopra/sotto una soglia interna al programma, ...
 - Creare test per valori speciali (0, 1, numeri molto grandi, ...)
 - Creare test per valori non validi (valori negativi, stringhe anziché numeri, valori vuoti, ...)

Pianificare...

- Cercare di fare una stima ragionevole del tempo necessario a:
 - Progettare l'algoritmo
 - Sviluppare i casi di test
 - Tradurre l'algoritmo in codice, e scrivere tale codice
 - Testare e correggere (debug) il programma
- Lasciare sempre del tempo extra per problemi imprevisti
- Con il crescere dell'esperienza le stime diventeranno via via più precise. È comunque meglio avere del tempo in più rispetto a consegnare in ritardo

Introduzione a Python



1.3, 1.4,
1.5, 1.6

Il linguaggio Python

- All'inizio degli anni '90 Guido van Rossum progettò ciò che sarebbe diventato il linguaggio di programmazione Python
- Van Rossum non era soddisfatto dei linguaggi esistenti
 - Erano ottimizzati per scrivere grandi programmi, eseguibili in modo efficiente
- Voleva un linguaggio che permettesse di creare rapidamente i programmi, ma anche modificarli in modo semplice
 - Progettato per avere una sintassi più semplice e pulita degli altri linguaggi come Java, C and C++ (più facile da apprendere)
 - L'ambiente Python aveva un approccio "batterie comprese", offrendo subito la disponibilità di molte funzioni utili in modo standard
 - Python è interpretato, rendendo più facile lo sviluppo ed il test di brevi programmi
- I programmi Python sono eseguiti dall'**interprete Python**
 - L'interprete legge il programma e lo esegue



<https://www.python.org/>

The screenshot shows the Python.org homepage with a dark blue header and a white footer. The main content area is dark blue and features a code editor with a Fibonacci function, a 'Functions Defined' section, and a navigation menu. The footer contains four columns of information: 'Get Started', 'Download', 'Docs', and 'Jobs'.

Python

PSF

Docs

PyPI

Jobs

Community

python™

Donate

Search

GO

Socialize

About

Downloads

Documentation

Community

Success Stories

News

Events

```
# Python 3: Fibonacci series up to n
>>> def fib(n):
>>>     a, b = 0, 1
>>>     while a < n:
>>>         print(a, end=' ')
>>>         a, b = b, a+b
>>>     print()
>>> fib(1000)
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

Functions Defined

The core of extensible programming is defining functions. Python allows mandatory and optional arguments, keyword arguments, and even arbitrary argument lists. [More about defining functions in Python 3](#)

1 2 3 4 5

Python is a programming language that lets you work quickly and integrate systems more effectively. >>> [Learn More](#)

Get Started

Whether you're new to programming or an experienced developer, it's easy to learn and use Python.

Start with our [Beginner's Guide](#)

Download

Python source code and installers are available for download for all versions!

Latest: [Python 3.8.5](#)

Docs

Documentation for Python's standard library, along with tutorials and guides, are available online.

docs.python.org

Jobs

Looking for work or have a Python related position that you're trying to hire for? Our **relaunched community-run job board** is the place to go.

jobs.python.org

<https://docs.python.org/>

Python » English » 3.8.5 » Documentation » | [modules](#) | [index](#)

Python 3.8.5 documentation

Welcome! This is the documentation for Python 3.8.5.

Parts of the documentation:

- [What's new in Python 3.8?](#)
or all "What's new" documents since 2.0
- [Tutorial](#)
start here
- [Library Reference](#)
keep this under your pillow
- [Language Reference](#)
describes syntax and language elements
- [Python Setup and Usage](#)
how to use Python on different platforms
- [Python HOWTOs](#)
in-depth documents on specific topics
- [Installing Python Modules](#)
installing from the Python Package Index & other sources
- [Distributing Python Modules](#)
publishing modules for installation by others
- [Extending and Embedding](#)
tutorial for C/C++ programmers
- [Python/C API](#)
reference for C/C++ programmers
- [FAQs](#)
frequently asked questions (with answers!)

Indices and tables:

- [Global Module Index](#)
quick access to all modules
- [General Index](#)
all functions, classes, terms
- [Glossary](#)
the most important terms explained

Meta information:

- [Search page](#)
search this documentation
- [Complete Table of Contents](#)
lists all sections and subsections

Download
Download these documents

Docs by version

- [Python 3.10 \(in development\)](#)
- [Python 3.9 \(pre-release\)](#)
- [Python 3.8 \(stable\)](#)
- [Python 3.7 \(security-fixes\)](#)
- [Python 3.6 \(security-fixes\)](#)
- [Python 3.5 \(security-fixes\)](#)
- [Python 2.7 \(EOL\)](#)
- [All versions](#)

Other resources

- [PEP Index](#)
- [Beginner's Guide](#)
- [Book List](#)
- [Audio/Visual Talks](#)
- [Python Developer's Guide](#)

Altri siti utili

- <https://realpython.com/>
 - Molti tutorial a diversi livelli di approfondimento
- <https://devdocs.io/python/>
 - Guida alle funzioni della libreria standard ed elenco dei moduli disponibili

Ambienti di programmazione

- Ci sono vari modi per creare un programma
 - Usando un sistema integrato di sviluppo (IDE, Integrated Development Environment)
 - IDLE, PyCharm, Visual Studio Code, Wing 101, ...
 - Usando un semplice editor di testi
 - Blocco note, Notepad++, Atom, vi, gedit, ...
- Usate il metodo ed il tool con cui vi trovate più a vostro agio
 - Nel corso useremo l'IDE **PyCharm** (versione Edu) o l'ide on-line **replit.com**
 - Gli esempi del libro usano l'IDE Wing o Spyder

Componenti di un IDE

- L'editor del codice sorgente aiuta il programmatore con:
 - Visualizzazione dei numeri di linea del codice
 - Evidenziazione e colorazione della sintassi (commenti, testi, ...)
 - Indentazione automatica del codice
 - Evidenziazione degli errori di sintassi
 - Completamento automatico dei nomi
- Finestra di output
 - L'output (testuale) generato dal programma
- Debugger
 - Strumenti di ausilio alla ricerca degli errori logici nel programma

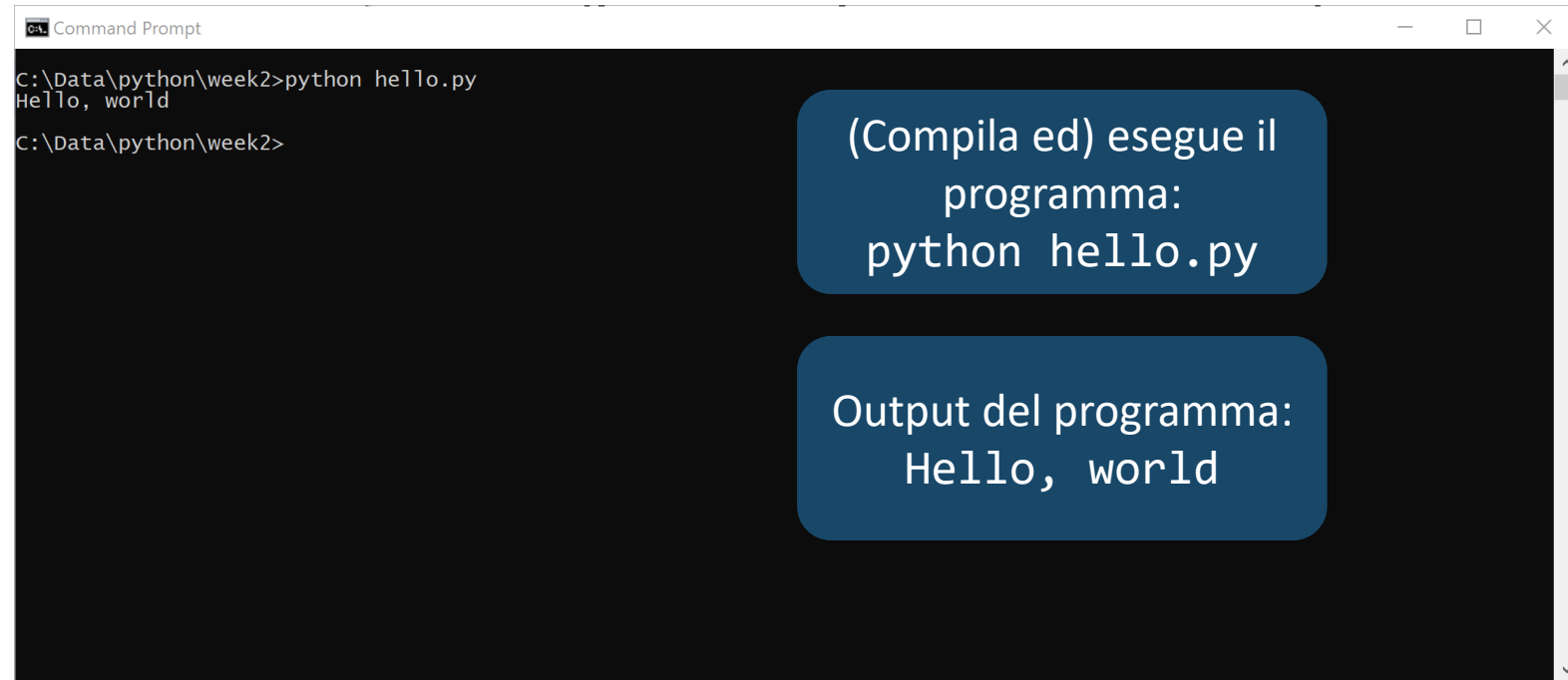
Programmazione con un editor di testi

- Si può anche usare un semplice editor di testi per scrivere il codice
- Salvando il file come `hello.py`, usare una finestra di comando per:
 - Compilare & eseguire il programma



```
1 # My First Python program
2 print("Hello, world")
```

Sorgente del programma in `hello.py`



```
C:\Data\python\week2>python hello.py
Hello, world
C:\Data\python\week2>
```

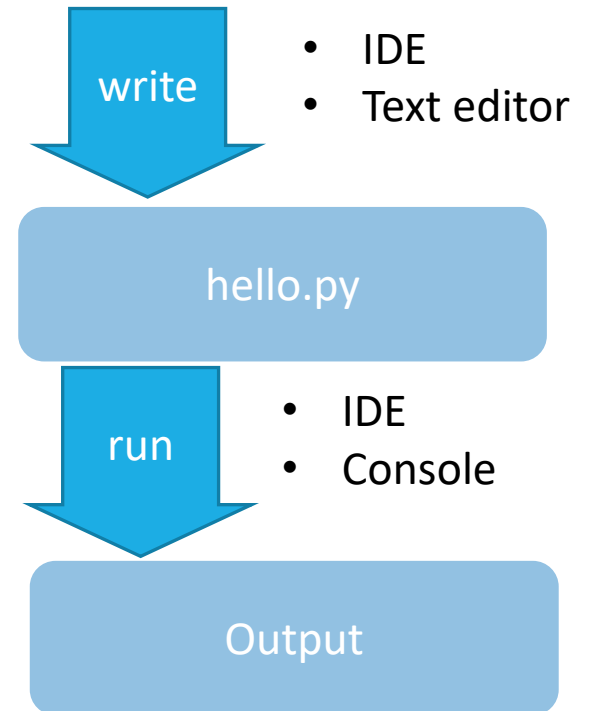
(Compila ed) esegue il programma:
`python hello.py`

Output del programma:
Hello, world

Il nostro primo programma

- Il classico programma 'Hello World' in Python
 - `print` è un esempio di una **istruzione (statement)** Python

```
1 # My first Python program.  
2 print("Hello, World!")  
3
```

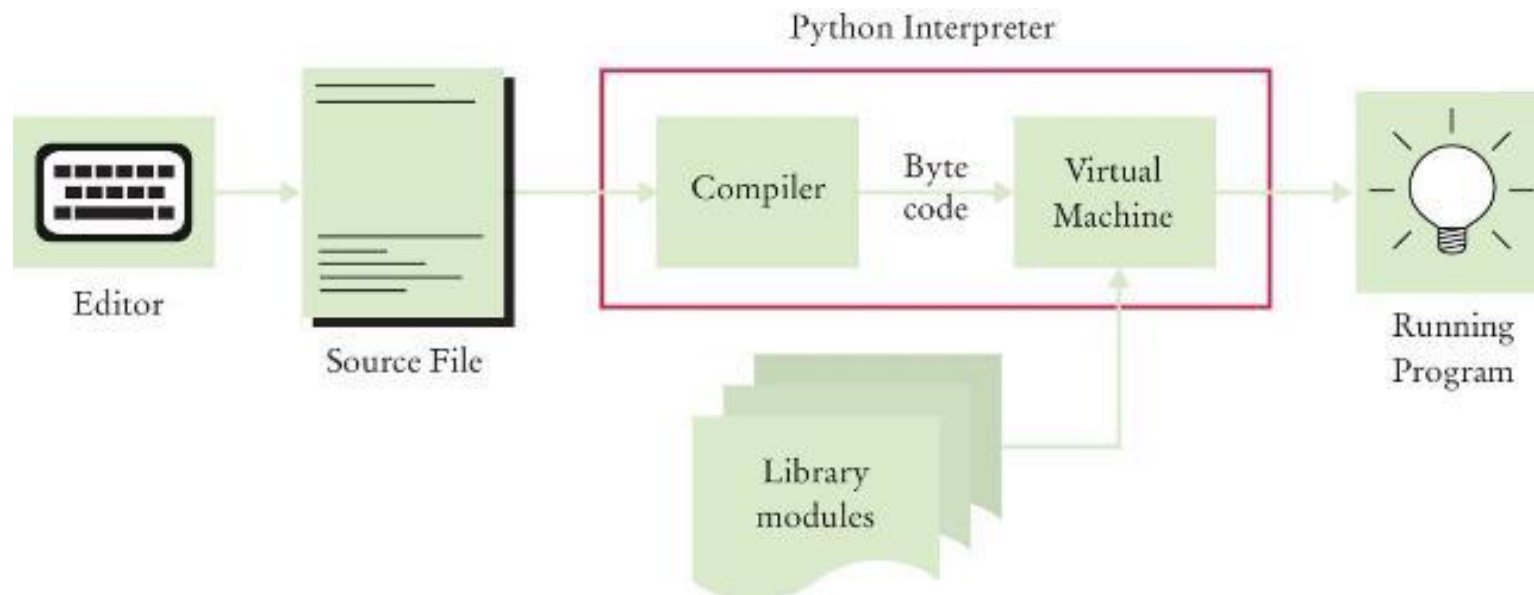


Scrivere un programma in Python

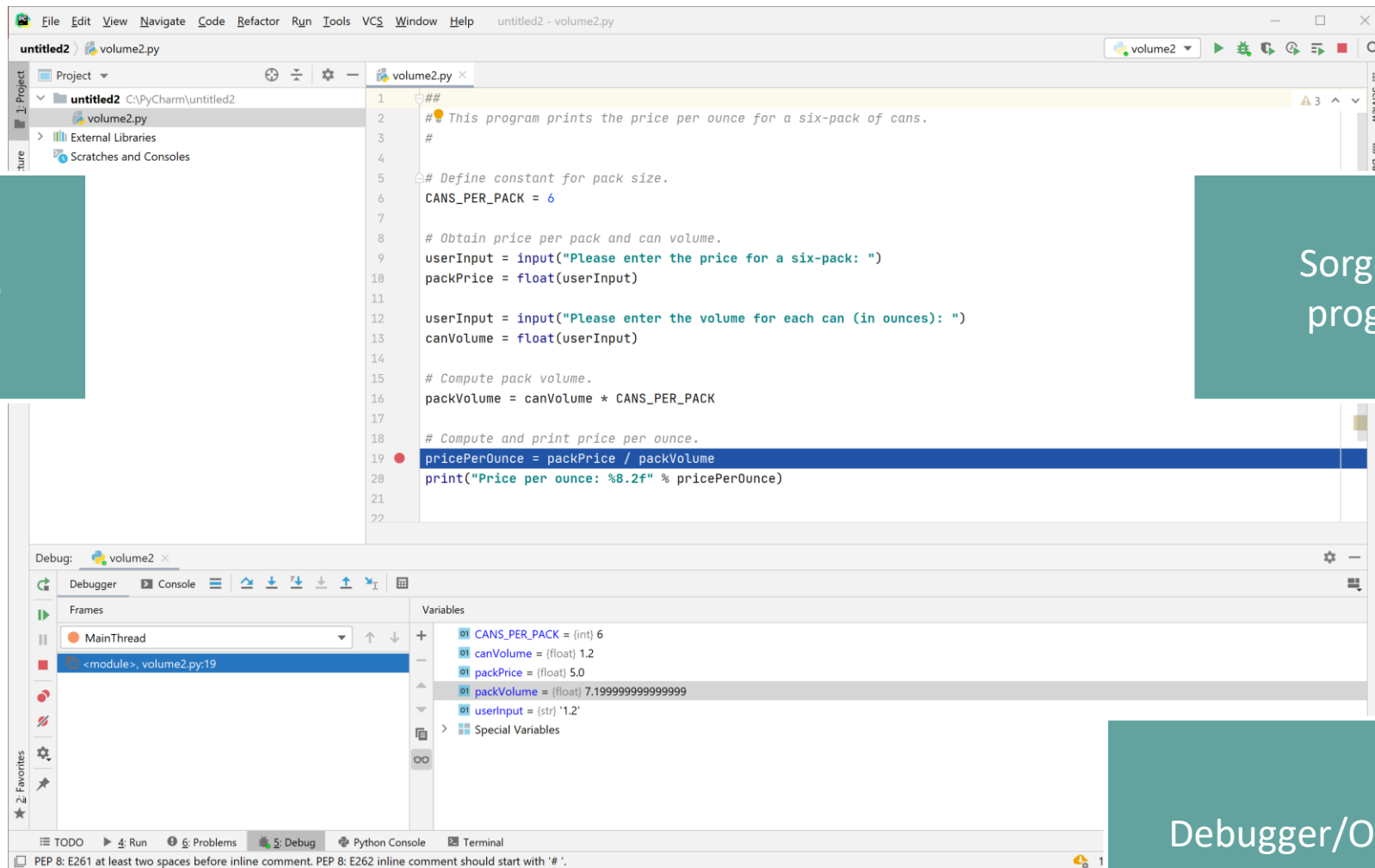
- Attenzione agli errori di battitura – es.: `'print'` vs. `'primt'`
- PyTHon **fA diFFereNza** tra MAIUscole e minuSCOLE
- Gli **spazi** sono importanti, soprattutto all'inizio della linea (indentazione o rientro)
- Le linee che iniziano con **#** sono commenti (vengono ignorati da Python)

Dal sorgente all'esecuzione del programma

- Il compilatore legge il programma e genera le istruzioni binarie (byte code, semplici istruzioni per la Macchina Virtuale Python)
 - La Macchina Virtuale Python è un programma che si comporta come la CPU del computer (esegue istruzioni, in software)
 - Ogni libreria necessaria (es. per la grafica) viene automaticamente trovata ed inclusa dalla macchina virtuale



L'IDE di PyCharm



File di progetto

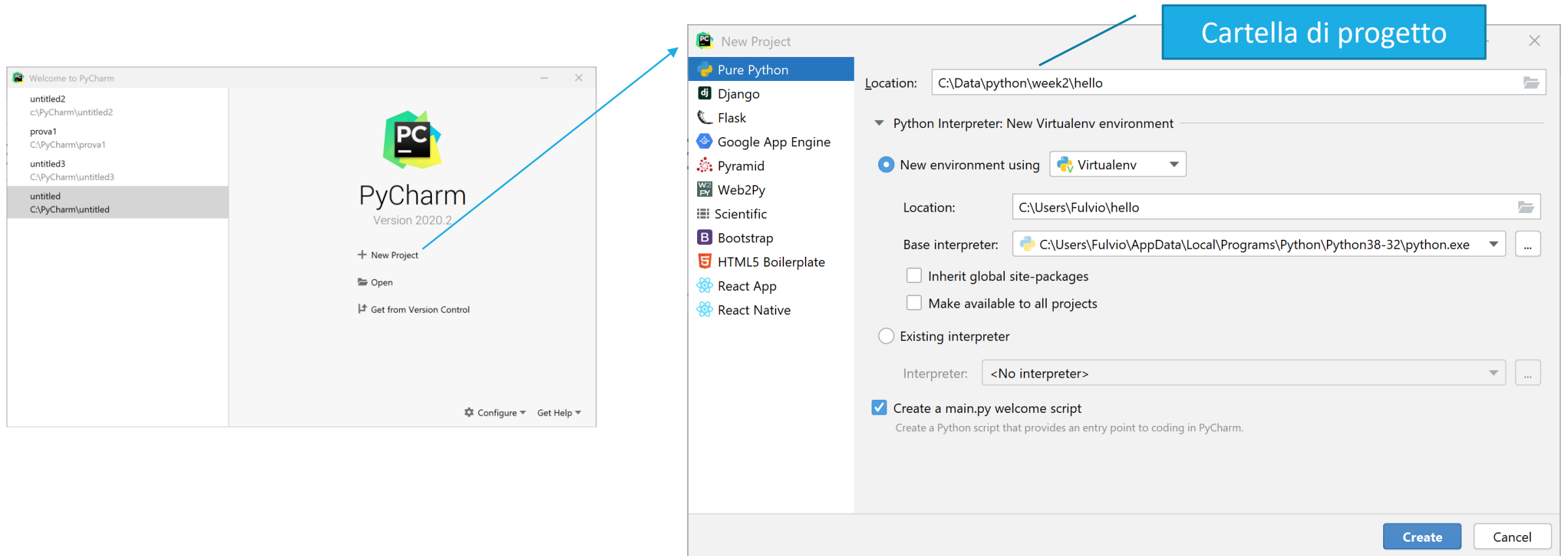
Sorgente del programma

Debugger/Output

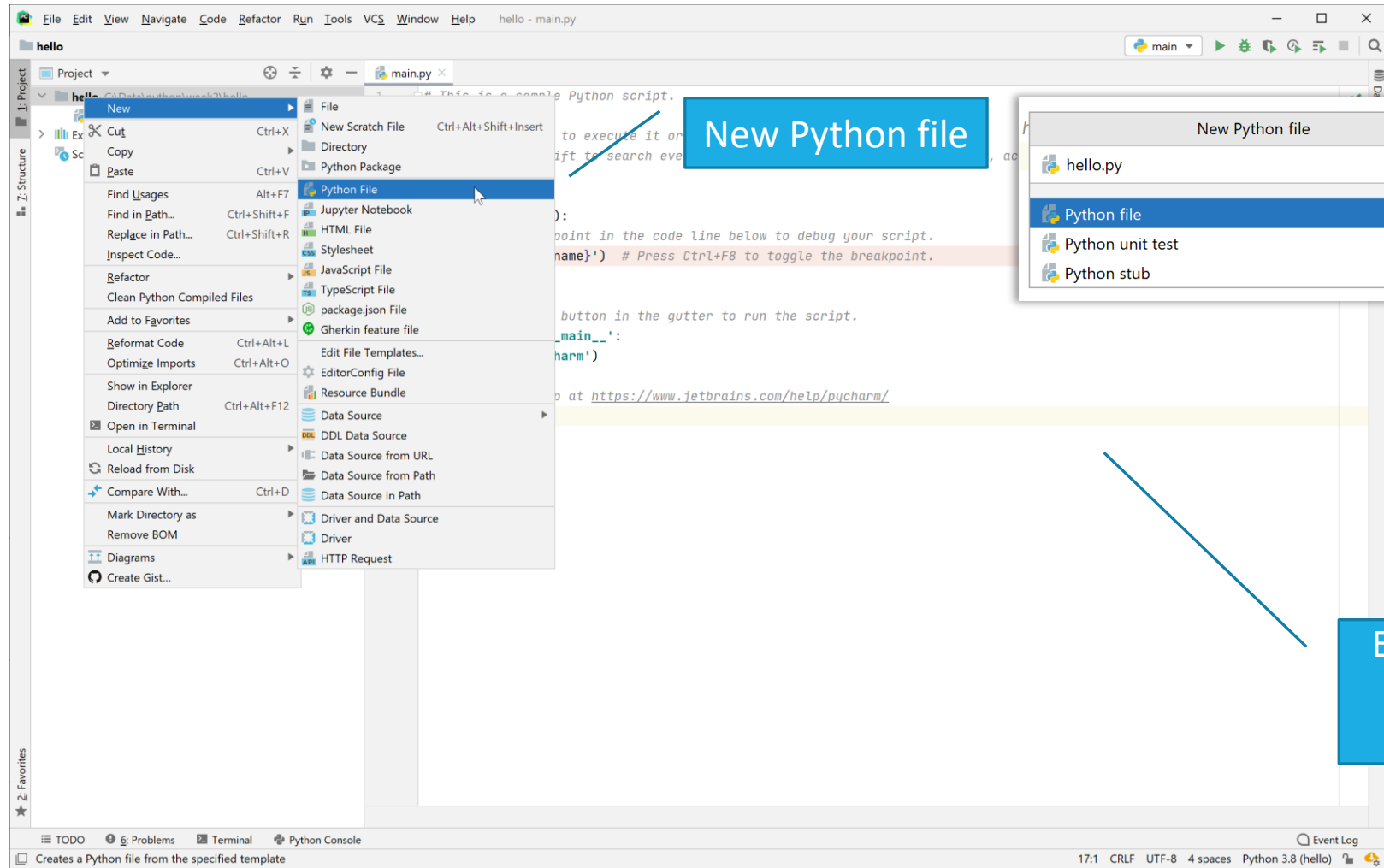
Progetto vs. Programma vs. File

- Un singolo **Programma** potrebbe essere molto grande, in tal caso sarà composto da molti **File** diversi
- Gli IDE permettono di raggruppare un insieme di **File** correlati in un “**Progetto**”
- Ogni volta che vogliamo creare un nuovo **Programma**, dovremo
 - Creare un nuovo **Progetto**
 - Creare uno (o più) **File** Python all’interno del **Progetto**

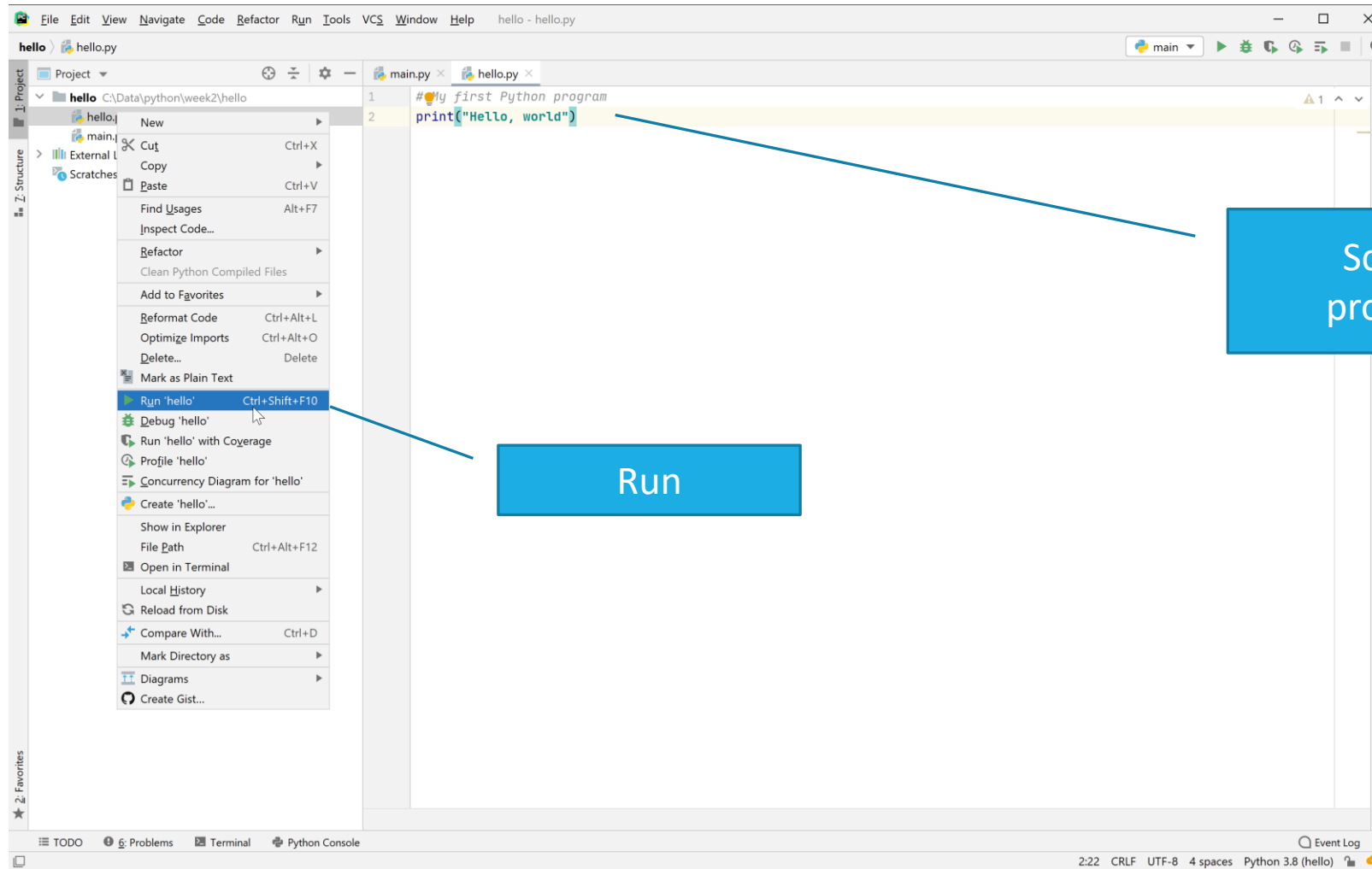
Programmare con l'IDE PyCharm



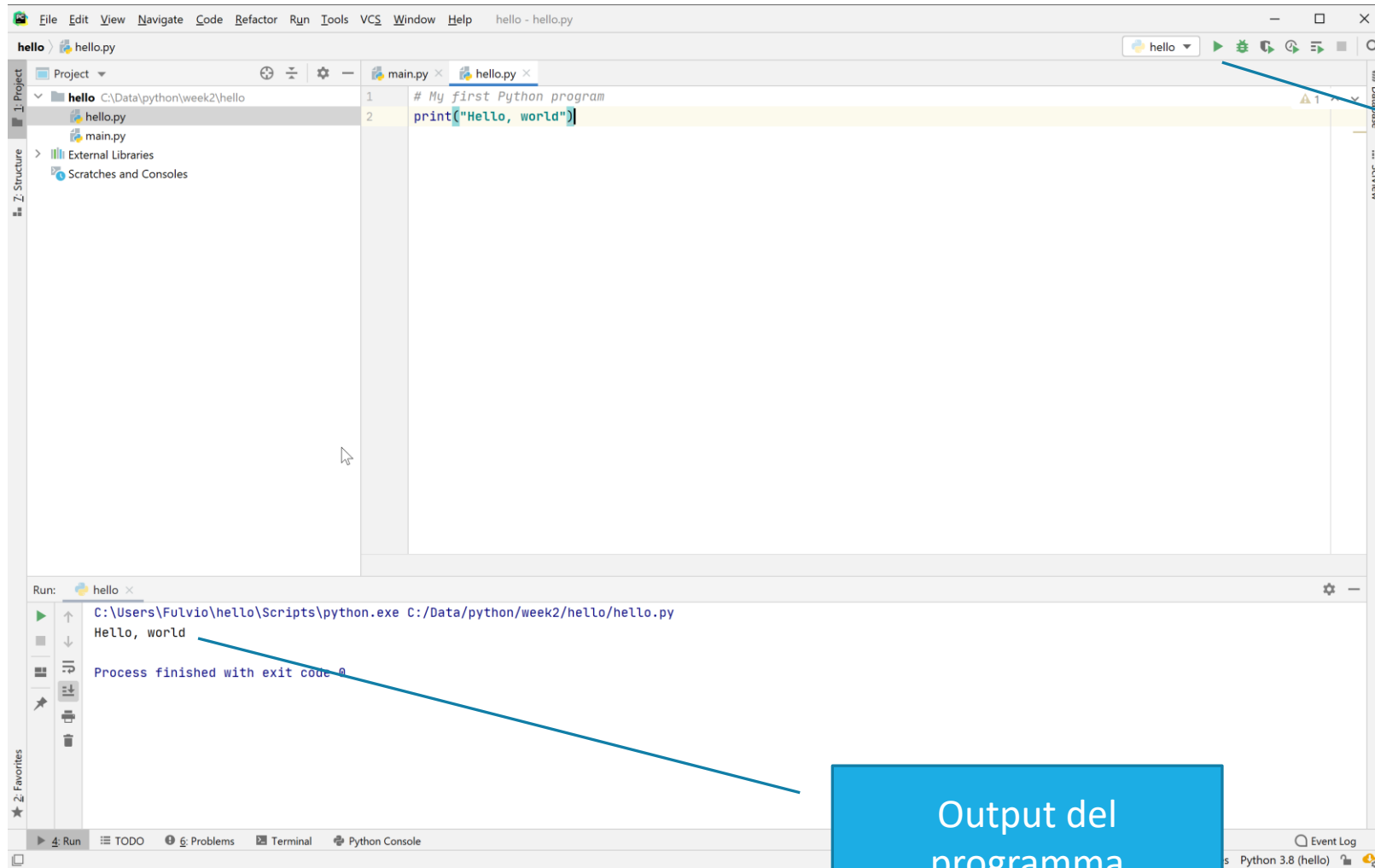
Programmare con l'IDE PyCharm



Programmare con l'IDE PyCharm



Programmare con l'IDE PyCharm



Scorciatoia
"Run"

Output del
programma

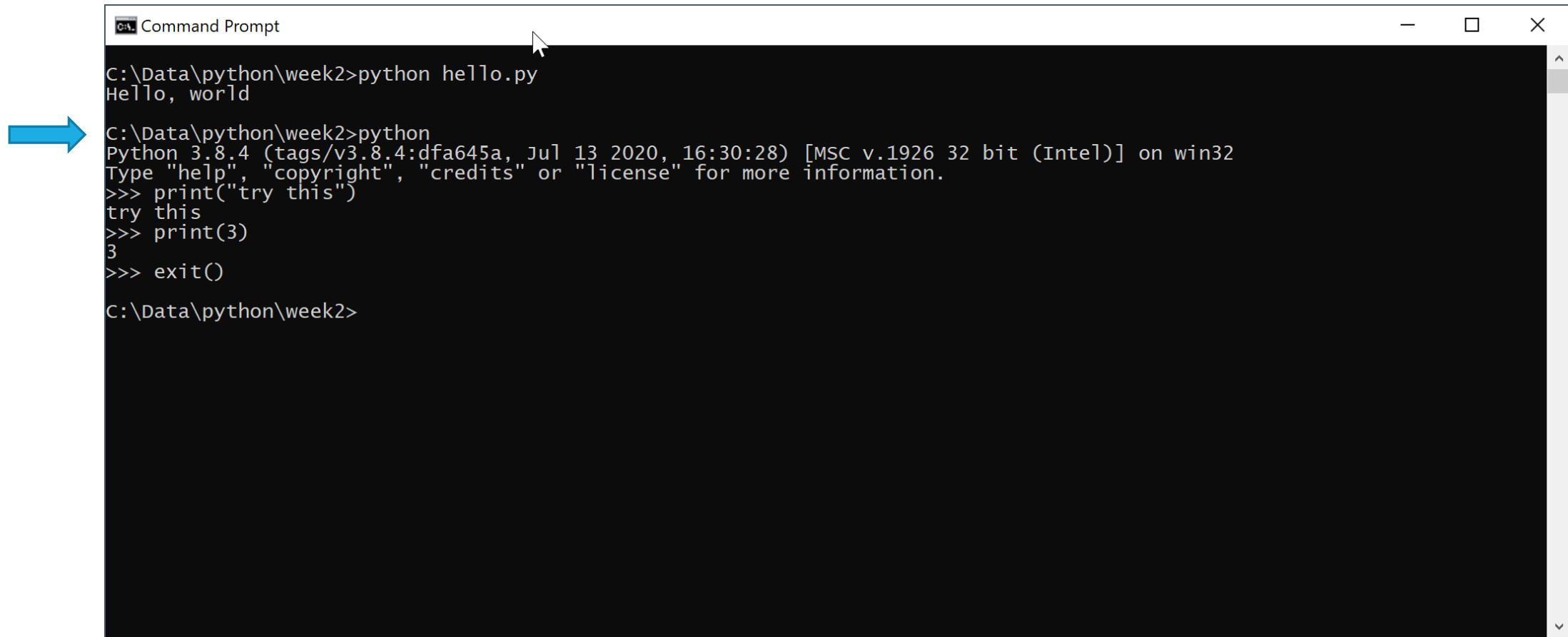
Organizzare il lavoro

- Il 'codice sorgente' è salvato il file `.py`
- Creiamo una cartella per il corso di Informatica
- Creiamo una **cartella di progetto** per ciascun programma, all'interno della cartella di Informatica
 - Un programma sarà composto da diversi file `.py`
- Fare backup regolari e frequenti dei propri dati
 - Su chiavetta USB (o più di una)
 - Su un disco di rete (servizio cloud) or hard disk esterno
 - Fatelo. Davvero. Subito.

Modalità interattiva di Python

- L'interprete di Python normalmente carica un intero programma ed esegue le istruzioni in esso contenute
 - Procedimento simile ad altri linguaggi (compilati)
- In alternativa: in **modo interattivo**, Python può eseguire un'istruzione per volta
 - Permette di scrivere velocemente dei 'programmini di test'
 - Permette di provare e sperimentare con le varie istruzioni
 - Permette di scrivere istruzioni Python direttamente nella finestra di console

Modalità interattiva di Python

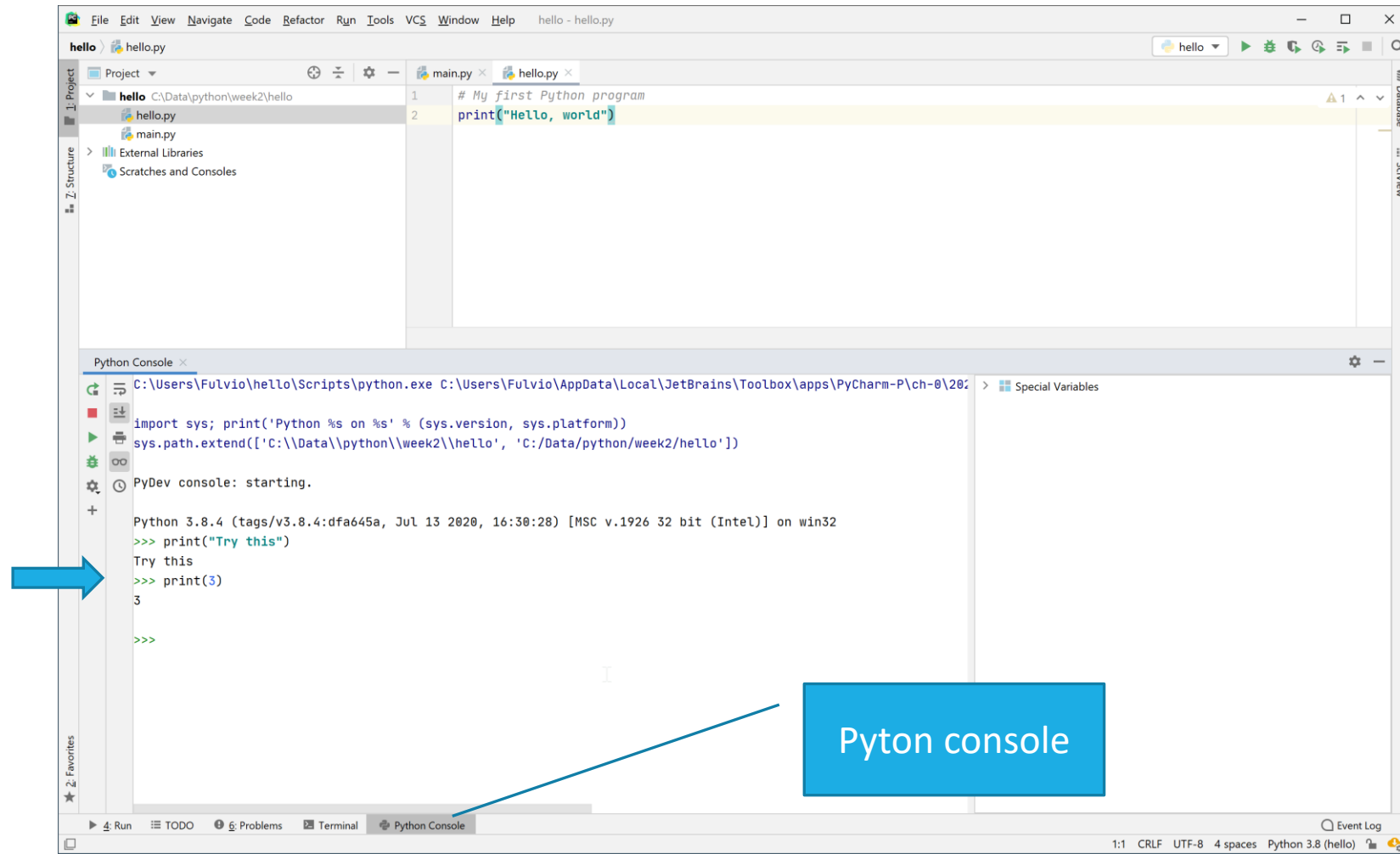


```
C:\Data\python\week2>python hello.py
Hello, world

C:\Data\python\week2>python
Python 3.8.4 (tags/v3.8.4:dfa645a, Jul 13 2020, 16:30:28) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("try this")
try this
>>> print(3)
3
>>> exit()

C:\Data\python\week2>
```

Modalità interattiva di Python



IDE On-line : <https://replit.com>

The screenshot displays the Replit IDE interface. At the top, the user profile is '@anonymous / GentleGiftedGeneric' with a Python logo, a dropdown menu, a refresh icon, and a play button. On the right, there are 'talk' and 'Sign up' buttons. The main workspace is divided into three sections: a file explorer on the left, a code editor in the center, and a terminal on the right. The file explorer shows a 'Files' panel with a 'main.py' file highlighted. The code editor shows the following Python code:

```
main.py  
1 # My first Python program  
2 print("Hello, world")
```

The terminal on the right shows the output of the program: 'Hello, world'. Three blue callout boxes with arrows point to specific elements: 'File di progetto' points to the 'main.py' file in the file explorer; 'Sorgente del programma' points to the code in the editor; and 'Output del programma e console interattiva' points to the terminal output. A green rounded rectangle at the bottom contains the text: 'Opzione fantastica per esempi rapidi, per testare frammenti di programma, per evitare di creare un intero Progetto per una piccola parte di codice,, ...'. A help icon (?) is visible in the bottom left corner of the IDE interface.

Sintassi Python: Print

- Usare la funzione `print()` in Python
 - Una funzione è un insieme di istruzioni (con un **nome**) che svolge un compito (task) particolare (in questo caso, stampare un valore su schermo)
 - È codice che qualcun altro ha scritto per noi!

Syntax `print()`
`print(value1, value2, ..., valuen)`

All arguments are optional. If no arguments are given, a blank line is printed.

```
print("The answer is", 6 + 7, "!")
```

The values to be printed, one after the other, separated by a blank space.

Sintassi per le funzioni Python

- Per usare (o 'chiamare') una funzione in Python, occorre specificare:
 - Il nome della funzione che vogliamo usare
 - Nell'esempio precedente, il nome era `print`
 - Tutti i valori (argomenti, parametri) di cui la funzione ha bisogno per svolgere il proprio compito
 - In questo caso, `"Hello World!"`
 - Gli argomenti sono racchiusi tra parentesi tonde
 - Se vi sono più argomenti, sono separati da virgole.

Stringhe

- Una *sequenza di caratteri* racchiusa tra apici o virgolette è chiamata *Stringa*
 - Può essere racchiusa tra 'apici singoli'
 - Può essere racchiusa tra "apici doppi" o "virgolette"

Altri esempi della funzione `print`

- Stampare valori numerici
 - `print(3 + 4)`
 - Valuta l'espressione `3 + 4` e visualizza `7`
- Passare più valori alla funzione
 - `print("The answer is", 6 * 7)`
 - Visualizza `The answer is 42`
 - Tutti i valori passati alla funzione vengono visualizzati, uno dopo l'altro, separati da uno spazio
- Per default, la funzione `print` crea una nuova linea (va «a capo») ogni volta che stampa i suoi argomenti
 - `print("Hello")`
 - `print("World!")`
 - Stampa due linee di testo:
 - Hello
 - World!

Il nostro secondo programma (printtest.py)



```
##  
# Sample Program that demonstrates the print function  
#  
# Prints 7  
  
print(3 + 4)  
  
# Print Hello World! on two lines  
print("Hello")  
print("World!")  
  
# Print multiple values with a single print function call  
print("My favorite numbers are", 3 + 4, "and", 3 + 10)  
  
# Print Hello World! on two lines  
print("Goodbye")  
print()  
print("Hope to see you again")
```

Errori

ERRORI A TEMPO DI **COMPILAZIONE**

o ERRORI DI **SINTASSI**

- Scrittura, maiuscole, punteggiatura
- Ordine delle istruzioni, corrispondenza delle parentesi, virgolette, indentazione, ...
- Il compilatore non crea alcun programma eseguibile
- Correggere il primo errore evidenziato, poi ri-compilare
 - Ripetere finché tutti gli errori non sono corretti
- Solitamente rivelati ed evidenziati direttamente dall'IDE

ERRORI A TEMPO DI **ESECUZIONE (RUN-TIME)**

o ERRORI **LOGICI**

- Il programma viene eseguito, ma non produce il risultato corretto
- Il programma può andare in 'crash'
- Sono i più difficili da trovare e correggere
 - Anche per programmatori più esperti

Errori di sintassi

- Gli errori di sintassi vengono catturati dal compilatore
- Verificare cosa succede se...
 - Sbagliamo una maiuscola `Print("Hello World!")`
 - Dimentichiamo le virgolette `print(Hello World!)`
 - Virgolette non corrispondenti `print("Hello World!')`
 - Parentesi non corrispondenti `print('Hello'`
- Proviamo ciascun esempio nell'IDE
 - Nel sorgente del programma
 - Nella console Python interattiva
 - Quali messaggi di errore vengono generati?

Errori Logici

- Verificare cosa succede se...
 - Dividiamo per zero `print(1/0)`
 - Sbagliamo il testo `print("Hello, Word!")`
 - Dimentichiamo q.cosa `(cancellare linea 2)`
- Il programma compila «normalmente» e viene eseguito
 - L'output però non è quello che ci aspettiamo
- Proviamo ciascun esempio nell'IDE
 - Quali errori vengono generati?