

Informatica

Corso di Laurea in Scienze Geologiche

Prima Parte

Ugo Dal Lago



ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA



Anno Accademico 2016-2017

Sezione 1

Introduzione al Corso

Questo Corso

- ▶ Questo è Un corso pensato per gli studenti del primo anno della **Laurea Triennale in Scienze Geologiche**.
- ▶ Il corso è diviso in due moduli:
 - ▶ Un modulo di **informatica**, (4 ECTS).
 - ▶ Un modulo di **cartografia numerica** (3 ECTS).
- ▶ Il sottoscritto si occuperà *solamente* del primo modulo, mentre il secondo sarà tenuto dal Dottor Luigi Cantelli, a partire dalla fine di Novembre.
- ▶ Le **esercitazioni** non sono formalmente previste, ma:
 - ▶ Alcune lezioni frontali saranno dedicate alla risoluzione di esercizi, della stessa difficoltà di quelli previsti per la prova scritta.
 - ▶ Durante le lezioni e le esercitazioni, è conveniente l'uso del calcolatore.
 - ▶ L'ideale sarebbe che ogni studente avesse a disposizione un notebook, soprattutto nella seconda e nella terza parte del corso.

- ▶ **Cinque ore** di lezione a settimana.
 - ▶ Il lunedì, dalle 16.00 alle 18.00.
 - ▶ Il martedì, dalle 14.00 alle 17.00.
- ▶ La lezione **inizia** all'orario prestabilito, per poi finire un po' prima.
- ▶ Il **ricevimento studenti** non ha un orario fissato.
 - ▶ Previo appuntamento via email (ugo.dallago@unibo.it).
- ▶ Se avete **dubbi** sui contenuti del corso, potete anche inviarmi una mail.
- ▶ Vi invito a fare tutte le **domande** che volete durante la lezione, ma anche offline.

Modalità d'Esame — I

- ▶ L'esame consiste in una **prova scritta**.
 - ▶ **Sei** appelli d'esame all'anno.
- ▶ Alla fine del modulo (verso la metà di Novembre) il docente proporrà un piccolo **progetto**.
 - ▶ Avrete due settimane di tempo per scrivere un programma **Python** per la risoluzione di un semplice problema.
 - ▶ Il progetto va svolto personalmente, e verrà discusso con il docente nel mese di dicembre.
 - ▶ Chi passa il progetto con una valutazione sufficiente, potrà limitarsi a svolgere *solo una parte* della prova scritta.
 - ▶ Il voto del progetto contribuisce poi per la metà al voto di questo modulo.
 - ▶ Chi non può o non riesce a svolgere il progetto, può comunque partecipare alla prova scritta, svolgendola *tutta*.
- ▶ Il voto che viene registrato è la media pesata del voto ottenuto in questo modulo e del voto ottenuto nel modulo di Cartografia Numerica.

Modalità d'Esame — II

- ▶ La **prova scritta** consiste in un certo numero di semplici esercizi e domande.
- ▶ Porrò la massima attenzione nel rendere le prove **diverse** le une dalle altre.
 - ▶ Si studia il **corso**, non si studia **come superare l'esame**.
- ▶ Gli esercizi riguardano l'informatica e la programmazione Python.

Contenuti del Corso

- ▶ Questo è un corso introduttivo sull'**informatica** e sulla **programmazione**.
- ▶ Più nello specifico, il modulo è suddiviso in tre parti:
 1. Una prima parte introduttiva e generale sull'**informatica**.
 2. Una seconda parte in cui verranno studiati i fondamenti del **linguaggio Python**.
 3. Una terza parte, in cui le conoscenze acquisite nella prima parte verranno messe a frutto nel **problem solving**.
- ▶ Non c'è nessun prerequisito.
 - ▶ Non è necessario aver precedentemente studiato l'informatica.
 - ▶ Si presuppone però che lo studente abbia una conoscenza di base dello *strumento* informatico.

Libri di Testo e Materiale Didattico

- ▶ La pagina web del corso è

<http://www.cs.unibo.it/~dallago/INFGE01617/>

A partire da essa trovate:

- ▶ Il syllabus del corso;
- ▶ Alcuni riferimenti bibliografici;

- ▶ **Libro di testo:**

John V. Guttag. Introduction to Computation and Programming Using Python (Revised and Expanded Edition). MIT Press, 2013

- ▶ Sono disponibili delle **trasparenze** , non necessariamente per tutte le parti del corso.
 - ▶ Sicuramente non per le esercitazioni.
 - ▶ Le trasparenze verranno messe a disposizione durante il corso.

Perché questo corso?

- ▶ Imparare ad utilizzare **software generico**?
 - ▶ *No*, questo non è un corso di alfabetizzazione informatica.
 - ▶ Presupponiamo che lo studente sia in grado di utilizzare gli strumenti di base (posta elettronica, web).

Perché questo corso?

- ▶ Imparare ad utilizzare **software generico**?
 - ▶ *No*, questo non è un corso di alfabetizzazione informatica.
 - ▶ Presupponiamo che lo studente sia in grado di utilizzare gli strumenti di base (posta elettronica, web).
- ▶ Imparare ad utilizzare i **GIS**?
 - ▶ *No*, i geographical information systems verranno introdotti nel secondo modulo e in altri corsi di natura “geologica”.

Perché questo corso?

- ▶ Imparare ad utilizzare **software generico**?
 - ▶ *No*, questo non è un corso di alfabetizzazione informatica.
 - ▶ Presupponiamo che lo studente sia in grado di utilizzare gli strumenti di base (posta elettronica, web).
- ▶ Imparare ad utilizzare i **GIS**?
 - ▶ *No*, i geographical information systems verranno introdotti nel secondo modulo e in altri corsi di natura “geologica”.
- ▶ Apprendere i rudimenti del **pensiero computazionale**?
 - ▶ *Sì*, questo è esattamente ciò che faremo in questo corso.
 - ▶ Non basta saper utilizzare software esistente.
 - ▶ Occorre essere in grado affrontare problemi *sempre nuovi*.
 - ▶ E per fare ciò bisogna conoscere *i principi*, anche se per sommi capi.

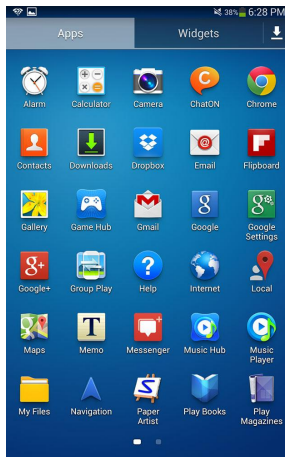
Sezione 2

Introduzione all'Informatica

Cos'è un Computer?



Cos'è un Computer?



Macchine Astratte — Esempio

Guidatore
Meccanico
Ingegnere

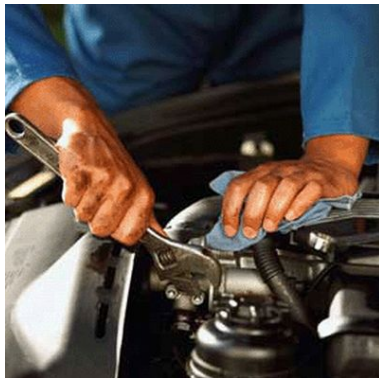
Macchine Astratte — Esempio

Guidatore
Meccanico
Ingegnere



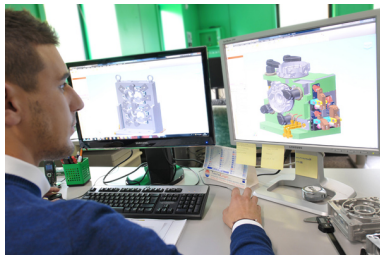
Macchine Astratte — Esempio

Guidatore
Meccanico
Ingegnere



Macchine Astratte — Esempio

Guidatore
Meccanico
Ingegnere



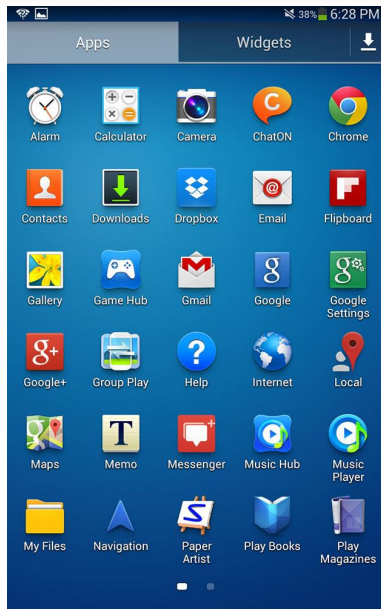
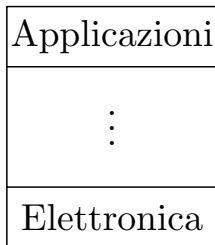
Macchine Astratte

- ▶ Ogni livello è **costruito** sul livello sottostante, e ne usa le funzionalità.
- ▶ Per comprendere il funzionamento di una macchina **ad un certo livello**, non è necessario comprendere i dettagli dei livelli sottostanti.
- ▶ Certamente, i livelli più bassi impongono dei **limiti** a ciò che si può fare nei livelli più alti.

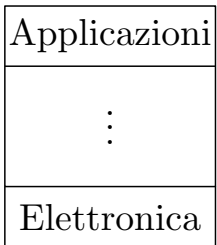
Macchine Astratte — il Computer

Applicazioni
⋮
Elettronica

Macchine Astratte — il Computer



Macchine Astratte — il Computer



Livello delle Applicazioni

- ▶ Gli oggetti con cui abbiamo a che fare sono le **applicazioni**.
- ▶ Le eseguiamo in un ambiente detto **sistema operativo**.
- ▶ Ciascuna applicazione ha un particolare, specifico, compito.

Ad esempio:

- ▶ *Editor di testi*;
- ▶ *Browser*;
- ▶ *Client email*;
- ▶ *Foglio di calcolo*;
- ▶

Livello delle Applicazioni

- ▶ Gli oggetti con cui abbiamo a che fare sono le **applicazioni**.
- ▶ Le eseguiamo in un ambiente detto **sistema operativo**.
- ▶ Ciascuna applicazione ha un particolare, specifico, compito.
Ad esempio:
 - ▶ *Editor di testi*;
 - ▶ *Browser*;
 - ▶ *Client email*;
 - ▶ *Foglio di calcolo*;
 - ▶
- ▶ Nessuna delle applicazioni ci permette di usare **direttamente** le risorse del calcolatore.

Livello delle Applicazioni

- ▶ Gli oggetti con cui abbiamo a che fare sono le **applicazioni**.
- ▶ Le eseguiamo in un ambiente detto **sistema operativo**.
- ▶ Ciascuna applicazione ha un particolare, specifico, compito.
Ad esempio:
 - ▶ *Editor di testi*;
 - ▶ *Browser*;
 - ▶ *Client email*;
 - ▶ *Foglio di calcolo*;
 - ▶
- ▶ Nessuna delle applicazioni ci permette di usare **direttamente** le risorse del calcolatore.
- ▶ E se avessimo a che fare con un problema che **nessuna applicazione** permette di risolvere?

Sotto il Livello delle Applicazioni

- ▶ Talvolta occorre scendere e mettersi nel **livello dei programmi**.

Sotto il Livello delle Applicazioni

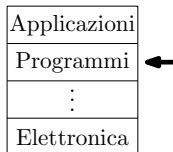
- ▶ Talvolta occorre scendere e mettersi nel **livello dei programmi**.
- ▶ In questo modo avremo più **controllo** sulle risorse di calcolo del calcolatore, e potremo disporre liberamente di esse.

Sotto il Livello delle Applicazioni

- ▶ Talvolta occorre scendere e mettersi nel **livello dei programmi**.
- ▶ In questo modo avremo più **controllo** sulle risorse di calcolo del calcolatore, e potremo disporre liberamente di esse.
- ▶ Questo corso (almeno in questo modulo) si occupa proprio di familiarizzarvi con il livello dei programmi.
- ▶ Occorre, almeno all'inizio, mettere in campo la propria **capacità di astrazione**.

Sotto il Livello delle Applicazioni

- ▶ Talvolta occorre scendere e mettersi nel **livello dei programmi**.
- ▶ In questo modo avremo più **controllo** sulle risorse di calcolo del calcolatore, e potremo disporre liberamente di esse.
- ▶ Questo corso (almeno in questo modulo) si occupa proprio di familiarizzarvi con il livello dei programmi.
- ▶ Occorre, almeno all'inizio, mettere in campo la propria **capacità di astrazione**.



Cosa Sa Fare il Calcolatore?

1. Fare Conti.

- ▶ I calcolatori moderni sono in grado di eseguire una *quantità enorme* di calcoli in poco tempo (circa 10^9 ogni secondo!)
- ▶ Ciascuno dei calcoli elementari che il calcolatore esegue, però, deve essere *molto semplice*.
- ▶ L'*accuratezza* dei calcoli è molto alta: la probabilità che i calcoli risultino in un errore è veramente minima.

2. Memorizzare Dati.

- ▶ Un'altra cosa che i calcolatori sanno fare molto bene è *tener traccia* dei dati.
- ▶ Anche qui, la *quantità* di dati che un calcolatore può immagazinare al suo interno, e l'*accuratezza* con cui lo può fare, sono molto alte. Un *gigabyte* corrisponde a 2^{30} informazioni elementari.

3. Comunicare con Dispositivi Esterni.

- ▶ I calcolatori moderni possono interagire con periferiche sempre più *complessi* (schermi, tastiere, sensori, apparecchiature biomediche, etc.).
- ▶ Per la maggior parte, le periferiche sono entità *passive*.

Cosa Non Sa Fare il Calcolatore?

1. Ragionare per Analogia;
2. Riconoscere l'Ironia;
3. Essere Creativo;
4. Risolvere l'Ambiguità;
5. ...

Conoscenza Dichiarativa

- ▶ È composta da **enunciati** che descrivono in modo preciso un certo oggetto o un certo fenomeno.
- ▶ Possiamo ampliare la nostra conoscenza dichiarativa semplicemente applicando le regole della *logica*.
- ▶ Tipico esempio sono le definizioni matematiche tradizionalmente concepite.
- ▶ **Esempio:**
 - ▶ La *radice quadrata* di un numero reale positivo x è un numero y tale che $y \cdot y = x$.
 - ▶ Da questa definizione possiamo dedurre che il modulo di qualunque radice quadrata di un numero x , è unico. Infatti, se $y \cdot y = x$ e $z \cdot z = x$

$$\begin{aligned}y \cdot y &= z \cdot z \Rightarrow y \cdot y - z \cdot z = 0 \\&\Rightarrow (y + z)(y - z) = 0 \\&\Rightarrow (y = -z) \text{ oppure } (y = z).\end{aligned}$$

Conoscenza Imperativa — I

- ▶ La conoscenza dichiarativa relativa ad un certo oggetto o concetto, in molte contesti è *sufficiente* a descrivere completamente l'entità in gioco.
- ▶ C'è però una cosa che la conoscenza dichiarativa non riesce, per sua natura, a catturare, ossia l'aspetto **operativo** e **di calcolo**.
 - ▶ *Come* calcolo un certo valore?
 - ▶ *In che modo* devo procedere, operativamente, se voglio fare uso dell'oggetto?

Conoscenza Imperativa — I

- ▶ La conoscenza dichiarativa relativa ad un certo oggetto o concetto, in molte contesti è *sufficiente* a descrivere completamente l'entità in gioco.
- ▶ C'è però una cosa che la conoscenza dichiarativa non riesce, per sua natura, a catturare, ossia l'aspetto **operativo** e **di calcolo**.
 - ▶ *Come* calcolo un certo valore?
 - ▶ *In che modo* devo procedere, operativamente, se voglio fare uso dell'oggetto?
- ▶ Abbiamo quindi bisogno della **conoscenza imperativa**!

Conoscenza Imperativa — I

- ▶ La conoscenza dichiarativa relativa ad un certo oggetto o concetto, in molte contesti è *sufficiente* a descrivere completamente l'entità in gioco.
- ▶ C'è però una cosa che la conoscenza dichiarativa non riesce, per sua natura, a catturare, ossia l'aspetto **operativo** e **di calcolo**.
 - ▶ *Come* calcolo un certo valore?
 - ▶ *In che modo* devo procedere, operativamente, se voglio fare uso dell'oggetto?
- ▶ Abbiamo quindi bisogno della **conoscenza imperativa**!
- ▶ Ad **esempio**, dato x posso calcolare la sua radice quadrata come segue:
 1. Faccio un tentativo g ;
 2. Se $g \cdot g$ è abbastanza vicino a x , allora g è una buona approssimazione della radice di x ;
 3. Altrimenti, calcoliamo un nuovo g come media tra g e $\frac{x}{g}$, ovvero $(g + x/g)/2$.
 4. Torniamo al punto 2.

Conoscenza Imperativa — II

- Vediamo se la la procedura di calcolo descritta funziona quando $x = 25$:

$$x = 25 \qquad g = 3$$

Conoscenza Imperativa — II

- Vediamo se la procedura di calcolo descritta funziona quando $x = 25$:

$$x = 25$$

$$g = 3$$

$$x = 25$$

$$g = 5.67$$

Conoscenza Imperativa — II

- Vediamo se la procedura di calcolo descritta funziona quando $x = 25$:

$x = 25$	$g = 3$
$x = 25$	$g = 5.67$
$x = 25$	$g = 5.04$

- Osserviamo come la procedura che abbiamo appena descritto consti di alcuni semplici passi *elementari*, ciascuno dei quali prevede solo semplici operazioni aritmetiche.
- Abbiamo dato, in altre parole, un esempio di *algoritmo*.

Algoritmi

- ▶ Un **algoritmo** è:
 - ▶ Una sequenza di semplici *passi*;
 - ▶ Assieme ad un insieme di regole che prescrivano *quando* ciascun passo debba essere eseguito;
 - ▶ Che permettono in un tempo *finito*;
 - ▶ Di risolvere una *classe* di problemi
- ▶ Le *proprietà* più importanti che un algoritmo deve soddisfare sono:
 - ▶ **Finitezza**: il numero dei passi che l'algoritmo richiede deve essere finito;
 - ▶ **Non ambiguità**: ogni passo di cui si compone l'algoritmo deve essere univocamente interpretabile;
 - ▶ **Effettività**: ogni operazione presente nell'algoritmo deve essere sufficientemente semplice.
- ▶ L'algoritmo è forse il concetto più importante in informatica:
 - ▶ I calcolatori eseguono *algoritmi*;
 - ▶ Si rende quindi necessaria una conoscenza *imperativa* del problema se si vuole essere in grado di far risolvere un problema da un calcolatore.

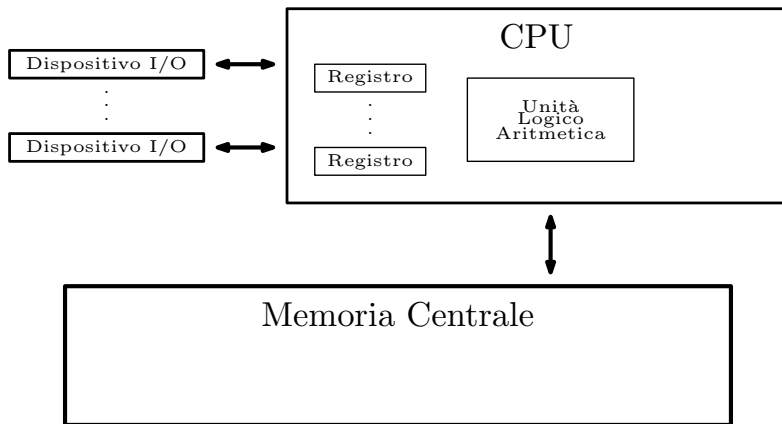
L'Architettura di von Neumann — I

- ▶ Ma *in che senso* i calcolatori eseguono algoritmi?

L'Architettura di von Neumann — I

- ▶ Ma *in che senso* i calcolatori eseguono algoritmi?
- ▶ Un buon modello dell'architettura interna di un calcolatore è dovuto al matematico von Neumann.
- ▶ Un calcolatore si compone di:
 1. **Unità Centrale di Elaborazione** (o CPU).
 - ▶ E' la componente del calcolatore che si occupa di svolgere i calcoli elementari.
 - ▶ Dispone di una sottocomponente che esegue materialmente i calcoli (l'*unità logico-aritmetica*) e di un limitato numero di *registri* in cui memorizzare i risultati intermedi.
 2. **Memoria Centrale.**
 - ▶ E' un grande deposito in cui tutti i dati necessari alla risoluzione del problema possono essere *memorizzati*.
 - ▶ Tali dati possono essere letti o scritti con facilità dalla CPU, la quale può trasferirli nei registri.
 - ▶ Possiamo sovrascrivere lo stesso spazio di memorizzazione più volte, ossia possiamo *riutilizzarlo*.
 3. **Dispositivi di Input/Output**
 - ▶ Ad esempio mouse, tastiera, monitor, stampante, etc.
 - ▶ La CPU può inviare i dati ai dispositivi di output o ricevere i dati dai dispositivi di input.

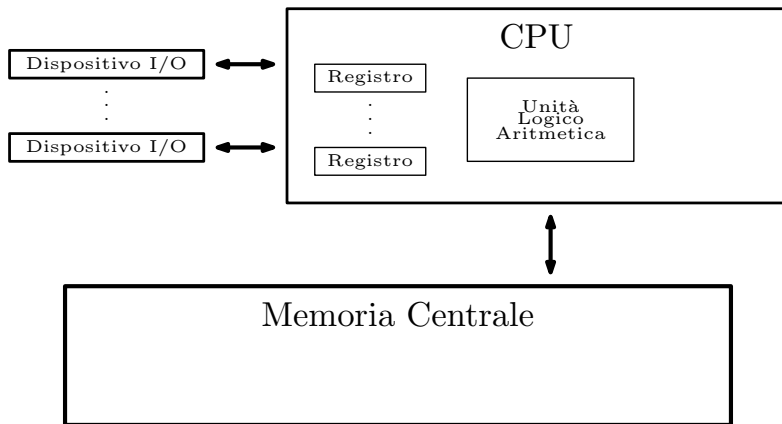
L'Architettura di von Neumann — II



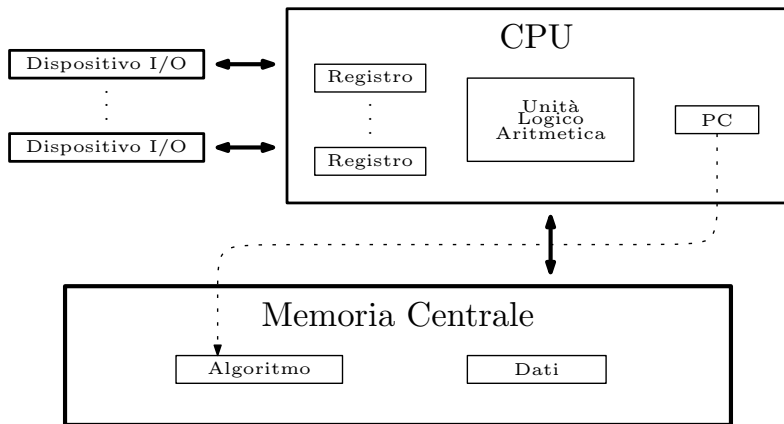
Dove sono i Dati? E l'Algoritmo?

- ▶ I *dati* di cui l'algoritmo ha bisogno possono risiedere in memoria centrale, ed alcuni di essi possono essere memorizzati nei registri.
- ▶ Per quanto riguarda l'*algoritmo*, abbiamo essenzialmente due possibilità:
 1. L'algoritmo risiede nella *CPU*.
 - ▶ Di conseguenza, il calcolatore può eseguire *uno e un solo* algoritmo.
 - ▶ Parliamo in tal caso di **fixed-program computer**.
 - ▶ E' chiaro che in questo caso ciò che possiamo fare con il nostro calcolatore è limitato.
 2. L'algoritmo risiede in *memoria centrale*.
 - ▶ L'algoritmo è trattato alla stregua dei dati.
 - ▶ Può essere eseguito *qualsiasi* algoritmo.
 - ▶ Un particolare registro, detto *program counter*, identificherà la prossima istruzione da eseguire.
 - ▶ La CPU non farà altro che leggere la prossima istruzione, eseguirla, e passare alla successiva.
 - ▶ Parliamo in questo caso di **stored-program computer**.

Lo Stored-Program Computer



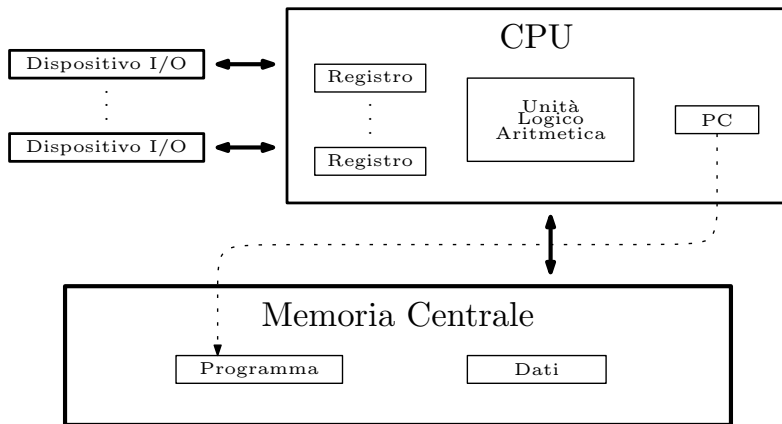
Lo Stored-Program Computer



Linguaggi di Programmazione — I

- ▶ Gli algoritmi vengono spesso scritti usando il **linguaggio naturale**.
- ▶ In questo modo, però, risulta difficile per il calcolatore (per la CPU) *eseguire* l'algoritmo.
 - ▶ Il linguaggio naturale è per sua stessa natura fortemente *ambiguo*.
 - ▶ E il calcolatore non è fatto, lo sappiamo, per risolvere l'ambiguità.
- ▶ Occorre quindi passare ad un **linguaggio di programmazione**.
 - ▶ Un linguaggio *artificiale*, pensato per essere comprensibile al calcolatore.
 - ▶ Le frasi di questo linguaggio vengono dette *programmi*, e ciascuna di esse corrisponde ad un algoritmo o ad un suo frammento.
 - ▶ Le regole che prescrivono quali siano i programmi e come essi vengono eseguiti dal calcolatore *devono essere rigide*.

Linguaggi di Programmazione — II



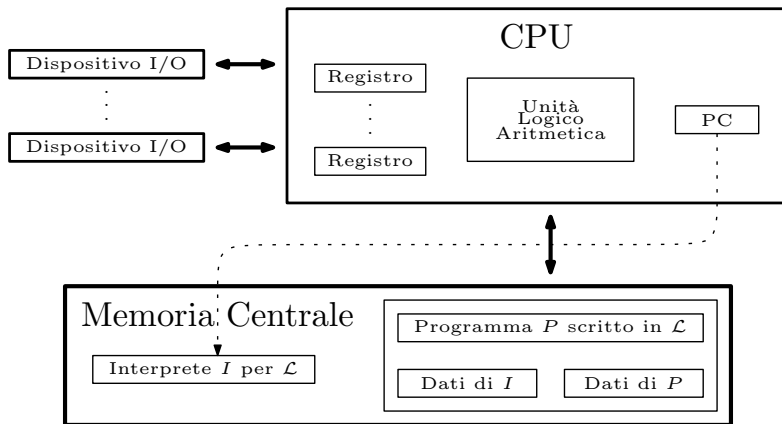
Linguaggi di Programmazione — III

- ▶ Esistono migliaia di linguaggi di programmazione.
 - ▶ In questo corso ci concentreremo su uno di essi, ovvero `Python`.
- ▶ Ogni calcolatore lavora con uno e un solo linguaggio?

Linguaggi di Programmazione — III

- ▶ Esistono migliaia di linguaggi di programmazione.
 - ▶ In questo corso ci concentreremo su uno di essi, ovvero **Python**.
- ▶ Ogni calcolatore lavora con uno e un solo linguaggio?
- ▶ In un certo senso sì!
 - ▶ Il linguaggio in cui sono scritti i programmi che un certo calcolatore riesce ad eseguire *direttamente* si chiama **linguaggio macchina**.
- ▶ E' possibile però fare in modo che un calcolatore esegua programmi scritti in un qualunque linguaggio di programmazione:
 1. **Compilazione**
 - ▶ Programmi scritti in un qualunque linguaggio \mathcal{L} vengono tradotti in programmi equivalenti nel linguaggio macchina.
 - ▶ Il programma che esegue la traduzione si chiama *compilatore* per \mathcal{L} .
 2. **Interpretazione**
 - ▶ Programmi scritti in un linguaggio \mathcal{L} vengono visti come dati ed eseguiti da un programma apposito, che si chiama *interprete* per \mathcal{L} .

Linguaggi di Programmazione — IV



Sintassi — I

- ▶ I programmi in un qualunque linguaggio di programmazione \mathcal{L} sono nient'altro che *frasi* (o *stringhe*) in un certo alfabeto Σ .
 - ▶ Tipicamente, Σ include:
 - ▶ I caratteri a, A, b, B, \dots
 - ▶ Le cifre $1, 2, 3, \dots$
 - ▶ Alcuni simboli di punteggiatura (ad esempio $;$, $:$, $<$, $>$, \dots).
- ▶ Non tutte le frasi in Σ *hanno senso* nel linguaggio \mathcal{L} .
- ▶ La sintassi di un linguaggio di programmazione è un insieme di regole che definiscono quali frasi in Σ possano dirsi almeno **ben formate**.
- ▶ Ciò richiede la definizione di certe **categorie** sintattiche, e di un insieme di regole che descrivano precisamente come generare le parole ben formate.
- ▶ **Esempio:** in Python la sequenza di caratteri `2.3 * 5.3` è ben formata, mentre la sequenza `2.3 5.3` non lo è.

Sintassi — II

- ▶ La sintassi di un linguaggio si può definire in maniera precisa attraverso una **grammatica**.
- ▶ Un **esempio** di regole grammaticali Python potrebbero essere le seguenti:

$$\langle \textit{espressione} \rangle \rightarrow \langle \textit{letterale} \rangle \langle \textit{operatore} \rangle \langle \textit{letterale} \rangle$$
$$\langle \textit{letterale} \rangle \rightarrow \langle \textit{numero} \rangle$$
$$\langle \textit{numero} \rangle \rightarrow 2.3$$
$$\langle \textit{numero} \rangle \rightarrow 5.3$$
$$\langle \textit{numero} \rangle \rightarrow \dots$$
$$\langle \textit{operatore} \rangle \rightarrow *$$
$$\langle \textit{operatore} \rangle \rightarrow \dots$$

Sintassi — II

- ▶ La sintassi di un linguaggio si può definire in maniera precisa attraverso una **grammatica**.
- ▶ Un **esempio** di regole grammaticali Python potrebbero essere le seguenti:

$$\langle \textit{espressione} \rangle \rightarrow \langle \textit{letterale} \rangle \langle \textit{operatore} \rangle \langle \textit{letterale} \rangle$$
$$\langle \textit{letterale} \rangle \rightarrow \langle \textit{numero} \rangle$$
$$\langle \textit{numero} \rangle \rightarrow 2.3$$
$$\langle \textit{numero} \rangle \rightarrow 5.3$$
$$\langle \textit{numero} \rangle \rightarrow \dots$$
$$\langle \textit{operatore} \rangle \rightarrow *$$
$$\langle \textit{operatore} \rangle \rightarrow \dots$$

- ▶ Un **esempio** di regola grammaticale della lingua italiana è invece la seguente:

$$\langle \textit{periodo} \rangle \rightarrow \langle \textit{soggetto} \rangle \langle \textit{verbo} \rangle \langle \textit{oggetto} \rangle$$

- ▶ Solo le espressioni che possiamo dimostrare essere in una categoria sintattica tramite una derivazione finita sono ben formate.
- ▶ In tutti gli altri casi, parliamo di **errore sintattico**
- ▶ L'interprete e/o il compilatore sono sempre in grado di segnalare la presenza di un errore sintattico in un programma.
 - ▶ In tal caso il programma non viene neanche eseguito!

Semantica Statica — I

- ▶ Non tutte le sequenze di stringhe ben formate **hanno effettivamente un senso** in un linguaggio di programmazione.

- ▶ **Esempio:**

- ▶ Sempre in Python, abbiamo anche le regole

$$\langle \textit{letterale} \rangle \rightarrow \langle \textit{stringa} \rangle$$
$$\langle \textit{numero} \rangle \rightarrow \text{'hello'}$$
$$\langle \textit{numero} \rangle \rightarrow \dots$$

- ▶ La sequenza di caratteri ben formata, quindi, è anche `2.3 * 'hello'`. Infatti:
 - ▶ Ovviamente, però, a tale sequenza non è possibile *dare un senso*.
- ▶ **Esempio:**
 - ▶ Nella lingua italiana, la frase **Io mangiamo una pizza** è ben formata, ma certamente non ha senso!

Semantica Statica — II

- ▶ La **semantica statica** è lo strumento che abbiamo a disposizione per identificare tra le frasi ben formate, quelle che *hanno un senso*.
- ▶ Spesso, e in **Python** in particolare, la semantica statica è formulata tramite il concetto di **tipo**.
 - ▶ Ad ogni frase ben formata è attribuito un tipo.
 - ▶ I tipi verranno poi usati per evitare di costruire frasi senza significato.
 - ▶ Ad esempio, in

$$\langle \text{espressione} \rangle \rightarrow \langle \text{letterale} \rangle \langle \text{operatore} \rangle \langle \text{letterale} \rangle$$

l'espressione ha tipo intero se l'operatore è `*` e se entrambi i letterali hanno tipo intero.

- ▶ A differenza degli errori sintattici, gli errori di semantica statica sono in genere catturati dall'interprete **Python**, ma quando il programma è già in esecuzione
 - ▶ Tecnicamente, si dice che **Python** *non ha* tipi statici.
 - ▶ In altri linguaggi di programmazione vale il viceversa.

Semantica Dinamica

- ▶ La semantica dinamica definisce **il senso**, **il significato** di una qualunque frase in un linguaggio di programmazione.
- ▶ La semantica dinamica può essere specificata in modo formale, attraverso il linguaggio matematico.
- ▶ In questo corso, per ovvie ragioni, non ci sarà modo di formalizzare la semantica dinamica di **Python** in modo preciso.
- ▶ Useremo però una rappresentazione diagrammatica, informale ma certamente sufficiente ai nostri scopi.
- ▶ In *tutti* i linguaggi di programmazione, gli errori di semantica statica non sono rilevabili.
 - ▶ Come è possibile, per l'elaboratore, accorgersi che il programma che avete scritto non fa quello che dovrebbe fare?