

# Cosa fa un calcolatore

- ▶ Fondamentalmente:
  - ▶ Esegue **calcoli**  
un miliardo al secondo
  - ▶ **ricorda** i risultati  
centinaia di gigabyte di memoria
- ▶ Che tipo di calcoli?
  - ▶ **incorporati** nel linguaggio
  - ▶ quelli **definiti** dal programmatore.
- ▶ i computer fanno solo quello che gli dici di fare

# Tipi di conoscenza

- ▶ **conoscenza dichiarativa** è un'affermazione di fatto

prima che la lezione termini qualcuno vincerà la lotteria

- ▶ **conoscenza imperativa** è una **ricetta** o un "come fare".
  1. Ad ogni studente verrà assegnato un numero da 1 a  $n$
  2. Il docente accende il computer
  3. Il docente avvia un programma che sceglie un numero a caso tra 1 e  $n$
  4. Il docente trova lo studente associato al numero, il vincitore!

# Un esempio numerico

- ▶ la radice quadrata di un numero  $x$  ( $= 16$ ) è un numero  $y$  tale che  $y * y = x$
- ▶ la ricetta per calcolare la radice quadrata di un numero  $x$ 
  - ▶ Inizia con una **ipotesi**,  $g$
  - ▶ Se  $g * g$  è **abbastanza vicino** a  $x$ , interrompi e concludi che  $g$  è la risposta
  - ▶ Altrimenti fai una **nuova ipotesi** facendo la media tra  $g$  e  $x/g$
  - ▶ Usando la nuova ipotesi  $g$ , **ripeti** il processo finché  $g * g$  non sarà abbastanza vicino a  $x$ .

$g$	$g * g$	$x/g$	$(g + x/g)/2$
3	9	$16/3$	4.17
4.17	17.36	3.837	4.0035
4.0035	16.0277	3.997	4.000002

# Cos'è una ricetta

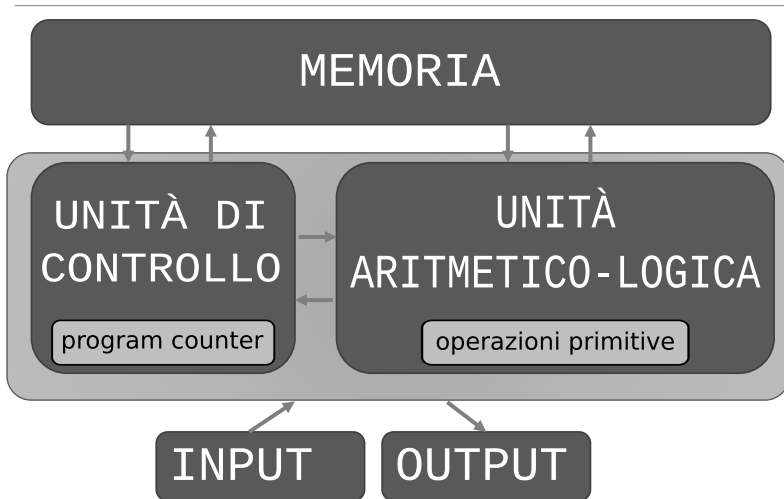
1. una sequenza di semplici **passi**
2. un meccanismo per il **controllo del flusso** che identifica quando ogni passo deve essere eseguito
3. un meccanismo per determinare **quando fermarsi**

$1+2+3 =$  un **algoritmo**

# I computer sono macchine

- ▶ permettono di rendere una ricetta un processo meccanico
- ▶ computer a **programma cablato**
  - ▶ la calcolatrice
- ▶ computer a **programma memorizzato**
  - ▶ la macchina immagazzina ed esegue le istruzioni

# Architettura di base di un calcolatore



# Computer a programma memorizzato

- ▶ sequenza di **istruzioni memorizzate** nel computer
  - ▶ costruito a partire da un insieme di primitive predefinite
    1. aritmetiche e logiche
    2. semplici test
    3. trasferimento di dati
- ▶ uno speciale programma (interprete) che esegue le **istruzioni una alla volta**
  - ▶ attraverso i test stabilisce qual è la prossima istruzione da eseguire tra quelle elencate nel programma
  - ▶ è in grado di fermarsi al termine della computazione

# Computer a programma memorizzato

- ▶ Turing ha dimostrato che bastano 6 istruzioni primitive per **calcolare qualsiasi cosa**
- ▶ i linguaggi di programmazione moderni dispongono di un insieme di primitive più ampio e più comodo
- ▶ e forniscono metodi per **creare nuove primitive**
- ▶ tutto ciò che è calcolabile con un linguaggio lo è in qualsiasi altro linguaggio



# Creare ricette

- ▶ un linguaggio di programmazione fornisce un insieme di **operazioni** primitive
- ▶ le **espressioni** in un linguaggio di programmazione sono combinazioni più complesse di operazioni primitive
- ▶ le espressioni e le computazioni in un linguaggio di programmazione hanno **valori** e significati



# Gli aspetti dei linguaggi

## ► **sintassi**

### ► linguaggio parlato:

- "gatto cane ragazzo" → sintatticamente non valida
- "il gatto abbraccia il ragazzo" → sintatticamente valida

### ► linguaggio di programmazione

- "hi"5 → sintatticamente non valida
- 3.2\*5 → sintatticamente valida

# Gli aspetti dei linguaggi

- ▶ **semantica statica** stabilisce quali stringhe sintatticamente valide hanno un significato
  - ▶ linguaggio parlato:
    - "io abbiamo fame" → sintatticamente valida ma con errore semantico statico
  - ▶ linguaggio di programmazione:
    - $3.2 * 5$  → corretta rispetto la semantica statica
    - $3 + \text{"hi"}$  → errata rispetto la semantica statica

# Gli aspetti dei linguaggi

- ▶ **semantica** è il significato associato ad una stringa di simboli sintatticamente corretta e senza errori di semantica statica
  - ▶ linguaggio parlato: possibili più significati come "Ho visto un uomo col binocolo"
  - ▶ linguaggi di programmazione: è possibile un unico significato ma potrebbe non essere quello voluto

# Quando le cose vanno storte

- ▶ **errori sintattici**

- ▶ comuni e di facile individuazione

- ▶ **errori di semantica statica**

- ▶ alcuni linguaggi li verificano prima di eseguire il programma
- ▶ possono provocare comportamenti indesiderati

- ▶ nessun errore semantico ma **un significato diverso da quello voluto dal programmatore**

- ▶ il programma si arresta in modo anomalo
- ▶ il programma va avanti all'infinito
- ▶ il programma dà risposte diverse da quelle volute

# Programmi Python

- ▶ un **programma** è una sequenza di definizioni e comandi
  - ▶ le definizioni sono **valutate**
  - ▶ i comandi sono **eseguiti** da un interprete Python in un terminale (o *console* o *shell*)
- ▶ i **comandi** (*dichiarazioni*) istruiscono l'interprete a fare qualche cosa
- ▶ possono essere digitati direttamente sulla **shell** oppure memorizzati in un **file** che è letto e valutato nella shell

# Oggetti

- ▶ i programmi manipolano **oggetti**
- ▶ gli oggetti hanno un **tipo** che definisce cosa un programma può fare con loro
  - ▶ Luca è un umano quindi può camminare, parlare, etc
  - ▶ Chewbacca è un wookiee perciò può camminare, "mwaaarhrhh", etc.
- ▶ gli oggetti possono essere
  - ▶ scalari (non possono essere scomposti)
  - ▶ non-scalari (hanno una struttura interna accessibile)



# Oggetti scalari

- ▶ int – rappresentano gli **interi**, esempio 5
- ▶ float – rappresentano i numeri **reali**, esempio 3.27
- ▶ bool – rappresentano i valori **booleani** True e False
- ▶ NoneType – un tipo **speciale** con unico valore, None
- ▶ la funzione type() mostra il tipo di un oggetto

```
>>> type(5)
```

```
int
```

```
>>> type(3.0)  
float
```

*quello che scrivi nella shell  
Python*

*cosa viene mostrato dopo  
aver premuto INVIO*

# Conversioni tra tipi (casting)

- ▶ si può **convertire un oggetto da un tipo ad un altro**
- ▶ `float(3)` converte l'intero 3 nel float 3.0
- ▶ `int(3.9)` tronca il float 3.9 all'intero 3

# Stampare sulla console

- ▶ per mostrare l'output dal codice all'utente si utilizza la funzione `print`

```
In [11]: 3+2
```

```
In [11]: 5
```

```
In [12]: print(3+2)
```

```
5
```

*“Out” ci dice che stiamo  
interagendo soltanto  
con la shell*





*Nessun “Out”, significa  
che quando il pro-  
gramma viene lanciato,  
il valore viene mostrato  
all'utente*

# Espressioni

- ▶ **combinano oggetti ed operatori**
- ▶ una espressione ha un **valore** di un determinato tipo
- ▶ la sintassi di una semplice espressione:

`<oggetto> operatore <oggetto>`

# Operatori su int e float

- ▶  $i+i \rightarrow$  **somma**  se gli operandi sono entrambi int
- ▶  $i-i \rightarrow$  **differenza**  il risultato è int; se uno dei due
- ▶  $i*i \rightarrow$  **prodotto**  è float il risultato è float
- ▶  $i/i \rightarrow$  **divisione**  il risultato è float
  
- ▶  $i\%j \rightarrow$  **resto** della divisione di  $i$  per  $j$
- ▶  $i**j \rightarrow$   $i$  con **esponente**  $j$ .

# Semplici operazioni

- ▶ le parentesi (...) vengono usate per dire a Python quali operazioni vanno eseguite per prime
- ▶ **precedenze tra operatori** senza parentesi
  - ▶ \*\*
  - ▶ \*
  - ▶ /
  - ▶ + e - eseguiti da sinistra a destra rispetto a come appaiono nell'espressione

# Legare variabili e valori

- ▶ il simbolo di uguale è un **assegnamento** di un valore ad un nome di variabile

*variabile*

*valore*

`pi` = `3.14159`

`pi_approx` = `22/7`

- ▶ i valori sono salvati in memoria
- ▶ un assegnamento lega nomi a valori
- ▶ si può recuperare il valore associato ad un nome o variabile invocando il nome, ad esempio digitando `pi`

# Astrarre le espressioni

- ▶ perché **assegnare nomi** ai valori delle espressioni?
- ▶ per **riutilizzare i nomi** anziché i valori
- ▶ semplifica le successive modifiche al codice

```
pi = 3.14159  
radius = 2.2  
area = pi*(radius**2)
```



# Programmazione vs matematica

- nella programmazione non "risolvi per x"<sup>6</sup>

```
pi = 3.14159
```

```
radius = 2.2
```

```
# area del cerchio
```

```
area = pi*(radius**2)
```

```
radius = radius+1
```

*un assegnamento  
\* sulla destra una espressione valutata ad un valore  
\* sulla sinistra un nome di variabile  
\* espressione equivalente radius += 1*

# Modificare i legami

- ▶ si possono **riassegnare** nomi di variabili utilizzando nuovi operatori di assegnamento
- ▶ il valore precedente potrebbe essere ancora in memoria ma senza modo per accedervi
- ▶ il valore per area non viene modificato finché non lo facciamo ricalcolare dal computer

```
pi = 3.14159  
radius = 2.2  
area = pi*(radius**2)  
radius = radius+1
```

