

01. Introduzione all'Informatica

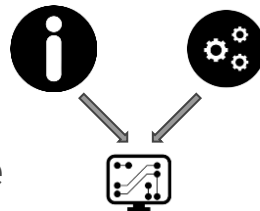
Corso di Algoritmi e Linguaggi di programmazione Python/C

Outline

- Introduzione all'Informatica
- Introduzione agli algoritmi
- Risolvere un problema
- Caratteristiche di un algoritmo risolutivo
- I tipi di dato
- Diagrammi di flusso
- La programmazione strutturata
- Variabili

Introduzione all'Informatica

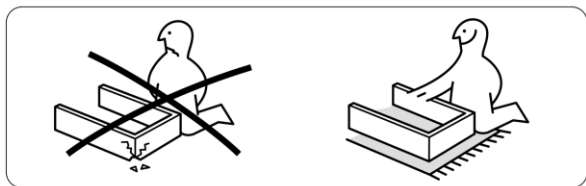
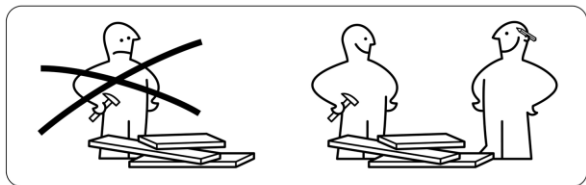
- Deriva dal francese **informatique**, crasi dei termini **informa**(tion) ed (automa)**tique**
- Si occupa di automatizzare **gestione** e **trattamento** dell'informazione.
- E' anche una disciplina scientifica, come rimarcato dalla denominazione inglese di **computer science**.
- La definizione non può prescindere dal ruolo che ha nella società moderna.



L'Informatica è la scienza che si occupa dell'ordinamento, trattamento e trasmissione delle informazioni per mezzo dell'elaborazione elettronica, la quale rende possibile gestire ed organizzare le ingenti masse di dati prodotte dal moderno sviluppo sociale, scientifico e tecnologico.

Introduzione all'informatica

Il concetto di informazione



- L'informazione è associata ai concetti di:
 - **conoscenza** (ad esempio, data dal libretto per il montaggio di un mobile di una nota catena svedese)
 - **esperienza** (ad esempio, data dal consiglio di un esperto nell'interpretazione del libretto di cui sopra)
- Caratterizzata da **vastità** ed **eterogeneità**

Introduzione agli algoritmi

Formulazione di un problema

- La definizione di un **algoritmo** parte da quella di un **problema**.
- Il dizionario De Mauro - Paravia definisce un problema come un:

“...quesito da risolvere mediante la determinazione di uno o più enti, partendo da elementi noti e condizioni fissate in precedenza.”

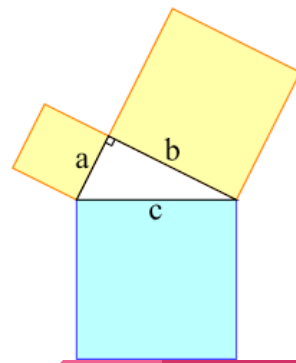
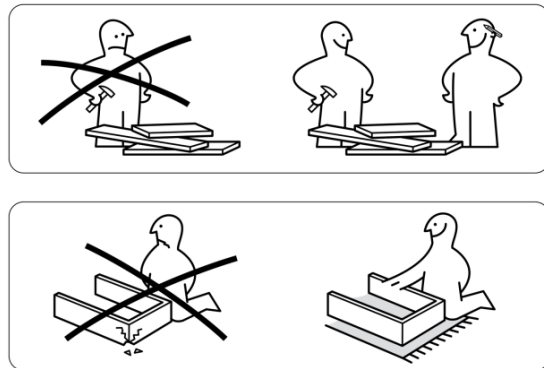
- Abbiamo tre elementi:
 - la **definizione del quesito**;
 - l'**ente risolutore**;
 - **gli elementi noti**.

Introduzione agli algoritmi

Il problema come quesito

“...quesito da risolvere mediante la determinazione di uno o più enti, partendo da elementi noti e condizioni fissate in precedenza.”

- Esempi concreti di problema:
 - *Come calcolare l'ipotenusa di un triangolo rettangolo?*
 - *Come montare il mobile IKEA appena acquistato?*

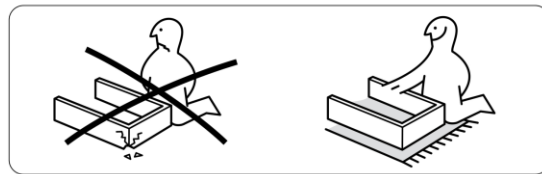
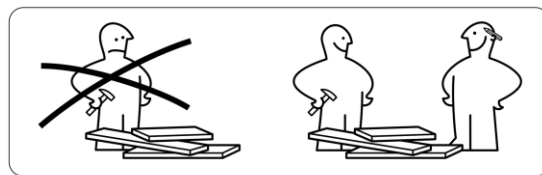
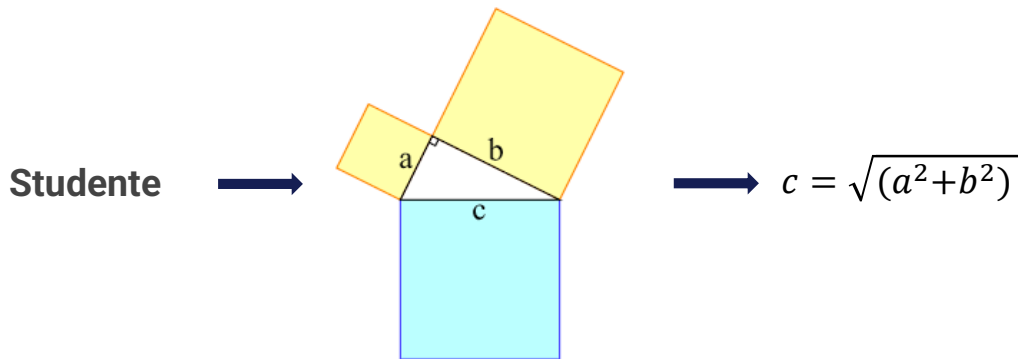


Introduzione agli algoritmi

L'ente risolutore

*"...quesito da risolvere mediante la **determinazione di uno o più enti**, partendo da elementi noti e condizioni fissate in precedenza."*

- Per i problemi precedenti, l'ente risolutore è:
 - lo studente (o la calcolatrice);
 - il montatore.



Montatore

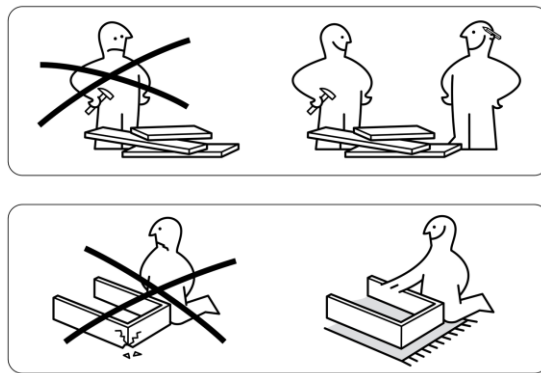


Introduzione agli algoritmi

Gli elementi noti e le condizioni fissate

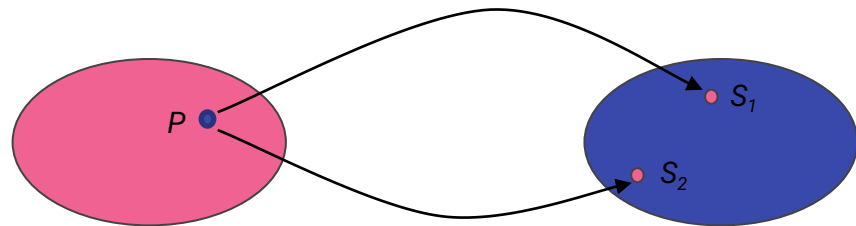
“...quesito da risolvere mediante la determinazione di uno o più enti, partendo da elementi noti e condizioni fissate in precedenza.”

- Ad esempio:
 - la lunghezza dei cateti, e le regole fissate dal teorema di Pitagora;
 - la posizione del mobile e gli attrezzi necessari.



Risolvere un problema

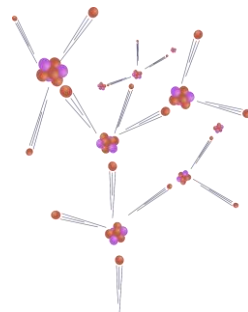
- Abbiamo determinato il **cosa**, il **chi** ed il **da dove partire**.
 - Per quello che riguarda l'ultimo aspetto, abbiamo visto come siano importanti *condizioni iniziali e vincoli*, passati sotto forma di *dati iniziali del problema*.
- Occorre determinare il **come**, trovando un *metodo di risoluzione*.
 - Individuiamo la relazione tra il problema P ed una o più istanze dell'insieme delle soluzioni S .



Risolvere un problema

Costruire la soluzione

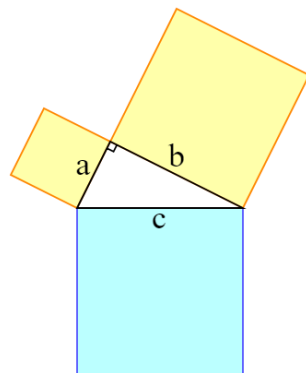
- Costruire la soluzione significa individuare il modo in cui possono combinarsi una serie di **operazioni atomiche**.
 - Un'operazione atomica non è ulteriormente divisibile: ad esempio, la somma di due numeri è un'operazione atomica, mentre la risoluzione di un'equazione di primo grado non lo è.
- Le operazioni atomiche possono combinarsi:
 - in modo **sequenziale**, ovvero concatenando un'azione all'altra;
 - in modo **parallelo**, quando due o più operazioni sono svolte contemporaneamente.
- La soluzione è un **operatore composto** da diverse azioni atomiche.
- Un algoritmo è la **serie di operazioni atomiche da seguire**.



Risolvere un problema

Un esempio

- **Formulazione**
 - *Dati due numeri interi **a** e **b**, rappresentativi della lunghezza dei due cateti di un triangolo rettangolo, calcolare l'ipotenusa **c**.*
- **Dati**
 - *Valori dei cateti **a** e **b**.*
- **Algoritmo risolutivo (in operazioni “quasi” atomiche)**
 - *Calcolare il quadrato di **a**.*
 - *Calcolare il quadrato di **b**.*
 - *Sommare i quadrati calcolati ai punti precedenti.*
 - *Calcolare la radice quadrata della somma ottenuta al punto precedente.*



Caratteristiche degli algoritmi risolutivi

- Caratteristiche principali:
 - **finitezza**, sia spaziale, sia temporale
 - **generalità**
 - **non ambiguità**
 - **eseguibilità**
- **Determinismo**
 - L'algoritmo è **deterministico** se ad ogni step si conosce in maniera univoca l'istruzione da eseguire successivamente.
- **Input, Output e Variabili**
 - **Input**: i dati rappresentativi della situazione iniziale.
 - **Output**: il valore restituito dall'algoritmo.
 - **Variabili**: dati di supporto usati per la risoluzione dell'algoritmo.

I tipi di dato

- In informatica, esistono differenti **tipi** di dato trattabili da problemi ed algoritmi.
- In linea generale, abbiamo **numeri**, dati **booleani** e **caratteri**.
- I **numeri** possono essere **interi** o **decimali**.
 - Esistono diverse rappresentazioni per entrambe le tipologie, le quali si differenziano in termini di **peso** in memoria e, conseguentemente, numero di valori rappresentabili.
- I **dati booleani** permettono di modellare una condizione logica, e possono assumere due valori: **vero** o **falso**.
- I **caratteri** rappresentano, per l'appunto, l'insieme di **tutti** i possibili caratteri.
 - È importante sottolineare come **anche un numero possa essere rappresentato come un carattere**. Ciò però non fa del carattere un numero (*chiariremo questo aspetto nel seguito*).






Diagrammi di flusso

Cosa sono, e perché usarli?

- Un algoritmo può presentare sequenze arbitrariamente complesse di operazioni atomiche.
- I Diagrammi di Flusso (comunemente chiamati anche *Flow Charts*, dalla loro denominazione inglese) ci permettono di gestire il flusso di tali operazioni.
- Sono degli **strumenti visivi**, atti a schematizzare un algoritmo, che permettono al progettista di valutare rapidamente come cambiano i dati (input, output e variabili) all'interno dello stesso.

Diagrammi di flusso

Componenti fondamentali

Forma	Descrizione
	Indica l'inizio o la fine dell'algoritmo.
	Indica un'istruzione da eseguire nel programma.
	Indica un <i>input</i> o un <i>output</i> .
	Indica una ramificazione del percorso dell'algoritmo.
	Usata per collegare tra loro più parti dell'algoritmo.

Nota: non confondere un input/output con un'istruzione di assegnazione!

Diagrammi di flusso

Un esempio

1. Leggi $c_1 = 3; c_2 = 4;$

Input

2. $v_1 = c_1 * c_1 = 9$

Assegnazione

3. $v_2 = c_2 * c_2 = 16$

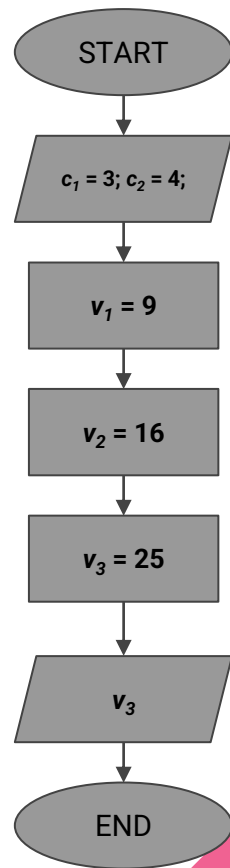
Assegnazione

4. $v_3 = v_1 + v_2 = 25$

Assegnazione

5. Scrivi v_3

Output



La programmazione strutturata

Lo spaghetti code

- Anni '60: **Spaghetti Code!**
 - *Gli algoritmi allora si basavano prevalentemente sul costrutto **go to**, che indicava al programma l'istruzione verso la quale 'saltare'.*
 - *Vedete un esempio di queste meraviglie a lato.*
- Approccio fortemente criticato
 - *Ad esempio, Dijkstra ne discusse gli effetti deleteri in **Go To Statement Considered Harmful***
- Codice strutturato più semplice da strutturare e mantenere!

```
10 int i = 0
20 i = i + 1
30 i = i + 2
40 if i <= 10 then goto 70
50 print "Programma terminato.«
60 end
70 print i & " al quadrato = " & i * i
80 goto 20
```



```
for (int i = 0; i <= 10; i++)
{
    print(i & " al quadrato = " & i * i);
}
print("Programma terminato");
```

La programmazione strutturata

Il Teorema di Böhm - Jacopini

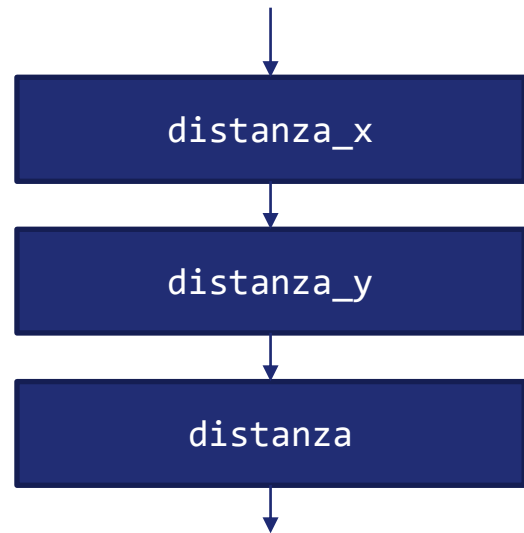
- Il Teorema di Böhm – Jacopini stabilisce che:
Ogni algoritmo può essere costruito a partire da tre strutture di controllo fondamentali, ovvero sequenza, selezione ed iterazione.
- Il teorema ha avuto un forte impatto nel passaggio dalla programmazione non strutturata a quella strutturata.
- Le sue implicazioni sono, chiaramente, estremamente importanti.

La programmazione strutturata

Le Strutture di Controllo - Sequenza

- Istruzioni realizzate **sequenzialmente**, ovvero l'una in cascata all'altra
- Esempio*: calcolo distanza euclidea

```
distanza_x = (x_a - x_b)^2  
distanza_y = (y_a - y_b)^2  
distanza = (distanza_x + distanza_y)^(1/2)
```

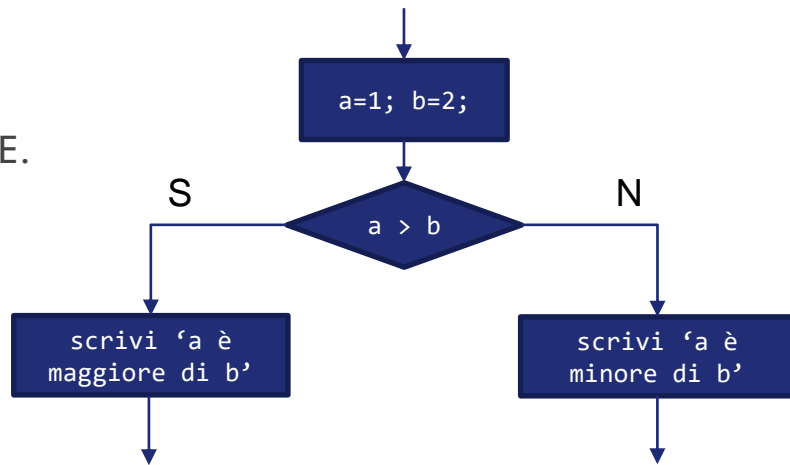


La programmazione strutturata

Le Strutture di Controllo - Selezione

- La struttura di **selezione** ci permette di scegliere tra due diverse opzioni in base ad una condizione.
- Per farlo, si usa il costrutto IF - THEN - ELSE.
- I due rami del programma sono divergenti e mutualmente esclusivi.*

```
a = 1;  
b = 2;  
if (a > b):  
    then scrivi 'a è maggiore di b'  
else:  
    scrivi 'b è maggiore di a'
```

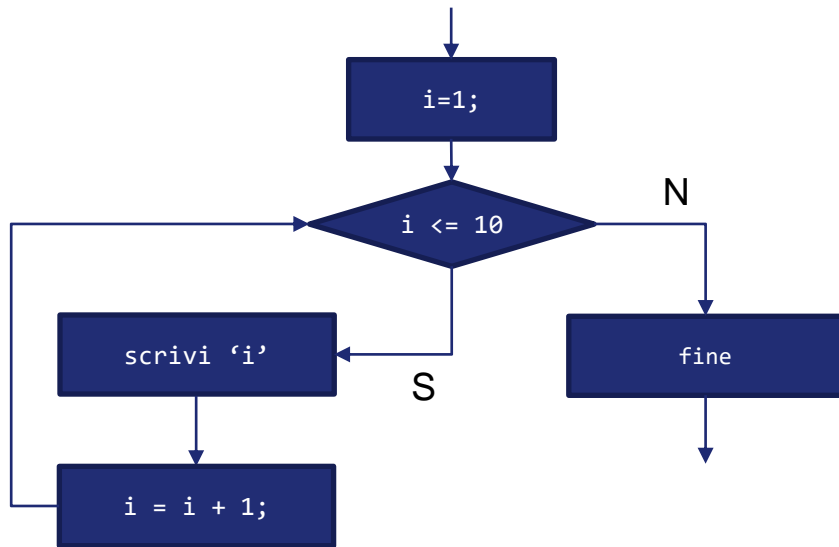


La programmazione strutturata

Le Strutture di Controllo - Iterazione

- E' una struttura di controllo che **reitera** (ovvero, **ripete**) un'istruzione fino al verificarsi di una condizione.
- Quando la condizione non è più verificata, il programma prosegue.

```
i=1;  
for (i che va da 1 a 10):  
    scrivi 'i';  
endfor  
scrivi 'fine';
```



La programmazione strutturata

I costrutti FOR e WHILE

- Esistono due modi per implementare l'iterazione.
- Il primo è il costrutto **FOR**, utilizzato per ripetere un'istruzione un certo numero di volte.
- Il secondo è il costrutto **WHILE**, usato per ripetere un'istruzione fino a che è verificata una certa condizione.

```
i = 1;
for (i che va da 1 a 10):
    scrivi 'i';
endfor
scrivi 'fine';
```

```
condizione = vero;
while (condizione diverso da falso):
    do [...] // istruzioni
    aggiorna condizione;
endwhile
```

La programmazione strutturata

Esercizi

- *Scrivere un diagramma di flusso che confronti due numeri letti da un input esterno. Scrivere a schermo se i numeri sono uguali o meno.*
- *Scrivere un algoritmo che aumenti il valore di un numero letto in ingresso di dieci unità in maniera iterativa, e poi verifichi che il valore del numero stesso sia superiore a quindici.*
- *Scrivere un algoritmo che generi numeri interi casuali fino a che l'ultimo numero generato non sia superiore a 10. Si supponga che i numeri casuali siano generati mediante un'istruzione chiamata "genera magia". In particolare, scrivendo: `a = genera magia`; supponiamo che ad `a` sia assegnato un certo valore intero casuale.*

Variabili e funzioni

Cosa è una variabile?

- Usata come **dato di supporto** negli algoritmi.
 - *Non è un input o un output!*
- Rappresenta un particolare **stato interno** dell'algoritmo.
 - *Esempi: contatori, valori intermedi in calcoli complessi.*
 - *Può (ed anzi spesso deve) variare durante l'esecuzione dell'algoritmo.*
- Esistono anche le **costanti**.
 - *Rispetto alle variabili, non possono modificare il loro valore durante l'esecuzione dell'algoritmo.*
 - *Esempio: una costante che definisce il valore del π (anche se molto spesso sono integrate nel linguaggio di programmazione stesso)*

Variabili e funzioni

Dichiarazione ed inizializzazione

- La **dirchiarazione** associa un nome ed un tipo alla variabile

`int numero;`

- L'**inizializzazione** associa un valore mediante l'operatore =

`numero = 1;`

- Le operazioni possono essere combinate in un unico step

`int altro_numero = 2;`

Variabili e funzioni

Tipo della variabile (1)

- Determina il **formato** del dato.
- Importante per due ragioni:
 - *Flusso logico*
 - *Memoria allocata*
- Flusso logico: non è sempre possibile passare da un tipo ad un altro!

```
float x = 1.1;  
float y = 2.2;  
int z;  
z = x + y;  
// Risultato: 3!
```

```
int a = 1;  
char b = '1';  
int c = a + b;  
// Risultato: 50!
```

Variabili e funzioni

Tipo della variabile (2)

- Memoria allocata: possibili errori di troncamento!

Denominazione	Spazio occupato	Descrizione
short	16 bit	Rappresenta un tipo di dato intero con precisione a 16 bit.
int	32 bit	Rappresenta un dato di tipo intero con segno.
uint	32 bit	Rappresenta un tipo di dato intero senza segno (<i>unsigned</i>)
long	64 bit	Rappresenta un tipo di dato intero con precisione a 64 bit.
float	32 bit	Rappresenta un tipo di dato reale con precisione a 32 bit.
double	64 bit	Rappresenta un tipo di dato reale con precisione a 64 bit.
bool	1 bit	Rappresenta un valore nell'algebra booleana (ovvero uno 0 o un 1).
char	1 byte	Rappresenta un singolo carattere.

Variabili e funzioni

Definizione di funzione

- Costrutto che permette di raggruppare una o più istruzioni eseguite più volte all'interno del nostro programma
- Due parti principali: **firma** e **corpo**

```
// Questa è la firma!
tipo_ritorno nome_funzione(tipo_par_1 par_1, tipo_par_2 par_2)
{
    // Questo è il corpo
    istr_1;
    istr_2;
    tipo_ritorno valore_ritorno = istr_3;
    return valore_ritorno;
}
```

Variabili e funzioni

Riutilizzo del codice

- Le funzioni servono a minimizzare il codice scritto
- Risultati immediati:
 - *Minor numero di errori*
 - *Coerenza*

```
def ipotenusa(c_1, c_2):  
    c_1_quad = c_1**2  
    c_2_quad = c_2**2  
    return (c_1_quad + c_2_quad)**1/2  
}  
  
if __name__ == "__main__":  
    a = 3  
    b = 4  
    i_1 = ipotenusa(a, b)  
    c = 6  
    d = 8  
    i_2 = ipotenusa(c, d)
```

Variabili e funzioni

Modularità

- Ogni funzione deve essere intesa come *atomica*.
 - *In pratica, le elaborazioni compiute all'interno della funzione devono, per quanto possibile, essere indipendenti dal resto del programma.*
- Questo rende il programma *modulare*.
 - *Potremo usare la funzione ipotenusa in altri programmi, o magari cambiare esclusivamente questa se dovesse insorgere un errore.*

```
def ipotenusa(c_1, c_2):  
    return (c_1 + c_2)**1/2  
}  
  
if __name__ == "__main__":  
    a = 3  
    b = 4  
    i_1 = ipotenusa(a, b)  
    c = 6  
    d = 8  
    i_2 = ipotenusa(c, d)
```

Variabili e funzioni

Ambito di una variabile

- Ad ogni variabile è associato un *ambito* di validità.
- Ciò significa che una data variabile può operare esclusivamente all'interno del suo ambito.
- Una variabile definita all'interno di una funzione o di un ciclo ha ambito *locale*, mentre una definita all'esterno di ogni funzione ha ambito solitamente *globale*.

```
numero_esami = 20;
miei_voti = lista_miei_voti;

float calcolo_voto_accesso_laurea(int[] voti_esami):
    somma_voti = 0;
    for i che va da 1 a numero_esami:
        somma_voti = somma_voti + voto_esame_i;
    endfor
    voto_medio = somma_voti / numero_esami;
    voto_accesso = voto_medio / 3 * 11;
    return voto_accesso;
```

Variabili e funzioni

Ambito di una variabile – Analisi dell'esempio

- Le variabili **numero_esami** e **miei_voti** sono variabili *globali*.
 - Ciò implica che possono essere accedute all'interno della funzione `calcola_voto_accesso_laurea`.
- Le variabili **somma_voti**, **voto_medio** e **voto_accesso** sono *variabili locali* relativamente all'ambito definito dalla funzione `calcola_voto_accesso_laurea`.
 - Non sono accessibili dall'esterno della funzione, ma rimangono accessibili nell'ambito individuato dal ciclo `for`.

```
numero_esami = 20;
miei_voti = lista_miei_voti;

float calcola_voto_accesso_laurea(int[] voti_esami):
    somma_voti = 0;
    for i che va da 1 a numero_esami:
        somma_voti = somma_voti + voto_esame_i;
    endfor
    voto_medio = somma_voti / numero_esami;
    voto_accesso = voto_medio / 3 * 11;
    return voto_accesso;
```


Domande?

42