Università degli Studi Roma Tre
Dipartimento di Scienze della Formazione
Laboratorio di Matematica per la Formazione Primaria

Algoritmi e risoluzione "automatica" di problemi

Mini corso "Informatica e Matematica nella Scuola Primaria"

Marco Liverani
liverani@mat.uniroma3.it

13 gennaio 2015

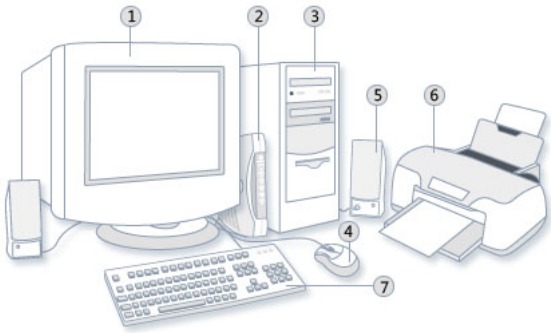
Algoritmi e scuola primaria: motivazione

- Qual è la motivazione di avvicinare i ragazzi della **scuola primaria** agli **algoritmi**...?
- Dopo una breve riflessione la risposta è naturale:
  - il concetto di algoritmo è strettamente legato alla matematica, all'abilità di **eseguire un calcolo**;
  - l'abilità nel costruire algoritmi sollecita una **comprensione piena del problema** e la capacità di **generalizzare il procedimento risolutivo**, sollecita il **ragionamento logico**
  - l'abilità nel costruire algoritmi porta con sé la capacità di **rapportarsi con l'altro da sé**, mettersi "nei panni dell'altro"
  - costruire algoritmi richiede di potenziare la capacità di individuare una **regola**, una **relazione tra i dati**, una **"regolarità" in una sequenza**

Algoritmi e scuola primaria: motivazione

- È necessario giungere fino alla **codifica di un programma** sul computer per ragionare sugli algoritmi?
- **No, non è *necessario*, ma è *utile***:
  - l'uso del computer come strumento da programmare, come esecutore di un algoritmo/programma è **stimolante**, **divertente**
  - getta una nuova luce (forse inaspettata) su uno strumento non estraneo, ma spesso "subìto" nelle modalità di utilizzo imposte da programmi scritti da altri

Computer



1 Monitor
2 Modem
3 Unità di sistema
4 Mouse
5 Altoparlante
6 Stampante
7 Tastiera

CPU:
- Unità di sistema, bus, Processore Intel/AMD, ...

Memoria:
- RAM, hard disk, ...

Unità di input:
- Tastiera, mouse, modem, ...

Unità di output:
- Monitor, modem, stampante, altoparlante, ...

CD, chiavette USB, DVD, sono unità di memoria di massa removibili, che possono essere usate per riversare dati da un computer (output) e immetterli in un altro (input)

Computer

Cos'è un algoritmo?

Un **algoritmo** è una **procedura di calcolo**
suddivisa in un **numero finito** di **passi elementari**
che **termina** dopo aver eseguito un **numero finito di operazioni**

1. Procedura di calcolo:
   - A cosa serve? A risolvere una specifica **istanza** di un determinato **problema**
   - A chi è destinata? Ad un **esecutore** incaricato di eseguire la procedura per risolvere una specifica istanza di un determinato problema
2. Numero finito di passi:
   - La procedura è una successione di operazioni (i *passi* dell'algoritmo): deve essere descritta con "**poche**" operazioni, non infinite
   - Le operazioni **devono essere specificate tutte**, senza lasciare sottintesi...
3. Passi elementari:
   - Le operazioni devono essere "**semplici**"
   - Chi stabilisce il livello di difficoltà delle operazioni? Come si fa a stabilire se i passi sono elementari? Dipende dalle **capacità dell'esecutore**
4. Termina dopo un numero finito di operazioni:
   - La procedura **non può durare un tempo infinito**, prima o poi deve terminare producendo il risultato, altrimenti... non serve a niente!

Serve un esempio!

- Dati due numeri x e y calcolare il *quoziente* e il *resto* della divisione x:y

problema

- Cerchiamo un modo elementare per risolvere il problema... se ben ricordo:
  ... il quoziente q è dato da "quante volte y entra in x"
  ... il resto r è ciò che avanza

strategia risolutiva

- Dunque:

algoritmo

1. acquisisci x e y
2. sia $q = 0$
3. se $x<y$ allora sia $r = x$, scrivi "quoziente = q, resto = r" e fermati
4. altrimenti sia $q = q+1$, $x = x-y$
5. vai al passo n. 3

Serve un esempio!

Per capirci meglio:

12 : 4 = 3 (resto 0)

17 : 5 = 3 (resto 2)

6 : 9 = 0 (resto 6)

Serve un esempio!

- Un'**istanza** del problema:

$$x = 17, y = 5$$

- Soluzione dell'istanza del problema:

ALGORITMO

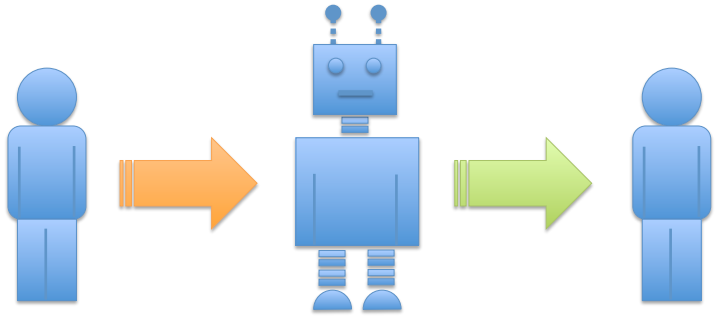
1. acquisisci x e y
2. sia $q = 0$
3. se $x<y$ allora sia $r = x$, scrivi "quoziente = q, resto = r" e fermati
4. altrimenti sia $q = q+1$, $x = x-y$
5. vai al passo n. 3

ESECUZIONE DELL'ALGORITMO

- $x = 17, y = 5, q = 0$
- $17 > 5 \rightarrow q = 1, x = 17 - 5 = 12$
- $12 > 5 \rightarrow q = 2, x = 12 - 5 = 7$
- $7 > 5 \rightarrow q = 3, x = 7 - 5 = 2$
- $2 < 5 \rightarrow$ **RISOLTO!**

quoziente: $q = 3$
resto: $x = 2$

Gli attori del contesto: i personaggi



Progettista

Definisce la procedura (l'algoritmo) per risolvere il problema e la fornisce all'esecutore

Esecutore

Esegue fedelmente la procedura (l'algoritmo) e calcola la soluzione dell'istanza del problema

Utente

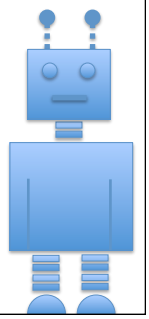
Fornisce i dati dell'istanza del problema all'esecutore e **riceve la soluzione** dell'istanza del problema

Gli attori del contesto: i ruoli

- Per distribuire i ruoli agli attori dobbiamo conoscerne le **capacità**, definirne le **competenze**
- **Utente**: fornisce i dati dell'istanza del problema all'esecutore e riceve la soluzione del problema...
  - come deve comunicare i dati all'esecutore?
  - come riceverà la soluzione del problema dall'esecutore?
  - deve fidarsi del risultato prodotto dall'esecutore?
- **Progettista**: definisce gli algoritmi per il calcolo delle soluzioni dei problemi...
  - come devono essere scritti gli algoritmi affinché l'esecutore possa eseguirli?
  - quali sono le abilità dell'esecutore affinché il progettista possa sfruttarle usandole per definire gli algoritmi?
- **Esecutore**: è essenziale scoprirne le caratteristiche per poter definire il ruolo degli altri attori!

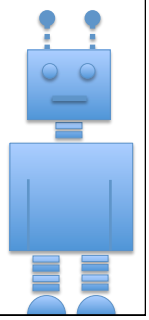
Caratteristiche dell'esecutore

- È una **macchina**, costituita da circuiti elettronici digitali e da componenti elettromeccaniche, ottiche e magnetiche
- È **velocissimo**, essendo una macchina elettronica è molto rapido nel compiere le operazioni per cui è stato progettato
- È **puntuale** nell'applicare le regole che conosce (è *preciso*, ma non nel senso matematico del termine)
- È **duttile** e si adatta bene ad eseguire nuove procedure, purché questo gli venga spiegato in modo ***dettagliato*** e ***privo di ambiguità***
- Ha una **buona memoria**, estremamente ampia ed organizzata in modo razionale, ma parcellizzato
- Essendo una macchina **non si stanca**, **non si annoia**, non crea problemi se il procedimento è lungo e un po' monotono... ne terremo conto!



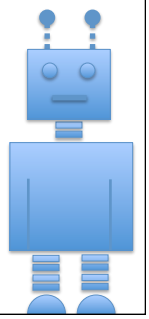
Caratteristiche dell'esecutore

- **Non è intelligente**: qualunque sia l'accezione di questo termine, non è adatta a descrivere le caratteristiche di un computer
- **Non è in grado di compiere deduzioni** o ragionamenti di altro tipo in modo autonomo
- **Non è in grado di capire un problema**
- **Non è in grado di capire la soluzione di un problema**, né è in grado di capire in modo autonomo se il risultato raggiunto è la soluzione del problema



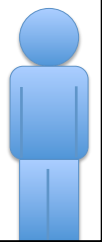
Capacità dell'esecutore

- Sa **memorizzare** le informazioni
- Sa **eseguire alcune operazioni elementari**: addizione, sottrazione, prodotto e rapporto fra numeri, concatenazione di parole
- Sa **eseguire il confronto fra informazioni** dello stesso tipo: confronto fra numeri (es.: $a>b$, $a=b$, $a\geq b$) e sa verificare l'uguaglianza fra due parole
- Sa **leggere** le informazioni dall'esterno (*input*)
- Sa **scrivere** le informazioni all'esterno (*output*)
- Sa **memorizzare sequenze di istruzioni elementari** (programma) e le sa **eseguire** secondo un ordine stabilito dal programma stesso



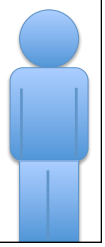
Compiti del progettista

- L'esecutore quindi si limita ad **eseguire in modo efficiente e disciplinato una procedura di calcolo** (un algoritmo) che lo condurrà a produrre la soluzione di una determinata istanza di un problema
- L'onere di **definire la procedura risolutiva** per un determinato problema è a carico del **progettista/programmatore**
- Egli deve:
  - **Analizzare il problema** riducendolo in termini astratti, eliminando ogni componente non indispensabile e formulando un modello del problema
  - Individuare una **strategia risolutiva** e ricondurla ad un **algoritmo**
  - **Codificare l'algoritmo** in modo tale da renderlo comprensibile all'esecutore



Competenze del progettista

- Deve essere in grado di **capire i problemi** e **schematizzarli**, distinguendone le diverse componenti (dati in *input*, parametri del problema, dati in *output*)
- Deve essere in grado di risolvere problemi mediante un **approccio algoritmico**, individuando gli aspetti del problema che possano essere risolti reiterando più volte operazioni simili
- Deve conoscere alcuni **metodi fondamentali** di risoluzione dei problemi, gli approcci più comuni, le strade notoriamente meno convenienti
- Deve conoscere a fondo le **caratteristiche e le capacità dell'esecutore**
- Deve essere in grado di comunicare con l'esecutore: ne deve **conoscere il linguaggio**



Serve un linguaggio!

- È emersa la necessità di definire un **linguaggio** con cui descrivere la procedura di calcolo, l'algoritmo, definita dal progettista ed eseguita dall'esecutore
- Il linguaggio deve essere
  - **semplice**, commisurato alle scarse abilità dell'esecutore, senza aspetti grammaticali o semantici complicati
  - **chiaro** e **non ambiguo**, in modo da esprimere esattamente le operazioni da compiere, senza lunghi giri di parole
  - **compatto**, dotato del minimo indispensabile insieme di termini

Un linguaggio imperativo

- Il linguaggio che vogliamo scegliere deve descrivere delle operazioni che l'esecutore deve svolgere
- Scegliamo un **linguaggio imperativo**, in cui ogni frase/istruzione è un comando dato all'esecutore
- Nei linguaggi imperativi è centrale il concetto di **variabile** (come nelle espressioni matematiche):
  - la variabile è un **simbolo** che rappresenta **un'area di memoria** dell'esecutore, in cui questo può memorizzare un **dato**
  - Esempio: $a=5$, memorizza nella **variabile** a il **valore** 5

Un linguaggio imperativo

I linguaggi imperativi sono basati su **sei istruzioni fondamentali**:

1. **Assegna**: assegna ad una *variabile* un *valore* o il risultato del calcolo di un'operazione
2. **Leggi**: *acquisisce in input* dall'esterno un valore e lo memorizza in una variabile
3. **Scrivi**: scrive in output il valore di una espressione o di una variabile (locazione di memoria)
4. **Se … allora … altrimenti …**: modifica il "flusso" dell'algoritmo sulla base del valore di una espressione logica
5. **Vai al passo …**: modifica il "flusso" dell'algoritmo incondizionatamente
6. **Fermati**: termina l'esecuzione dell'algoritmo

Un linguaggio imperativo

- **Assegna**: memorizza un valore o il risultato di un calcolo in una variabile
- Esempi:
  - **assegna a = 57** — Assegna alla variabile **a** il valore **57**
  - **assegna b = a** — Assegna alla variabile **b** lo stesso valore della variabile **a**
  - **assegna c = (a+b) : 2** — Assegna alla variabile **c** il risultato dell'operazione **(a+b):2**

Un linguaggio imperativo

- **Leggi**: acquisisce in input un dato e lo memorizza in una variabile
- Esempi:
  - **leggi a** — Acquisisce in input un dato e lo memorizza nella variabile **a**

    Il dato viene fornito in input dall'utente: l'utente comunica il dato all'esecutore (ad esempio mediante la tastiera l'utente fornisce un dato in input al computer)
  - **leggi x_1, ..., x_n** — Acquisisce in input *n* dati (una sequenza di *n* parole o di *n* numeri) e li memorizza nelle variabili **x_1, ..., x_n** (un dato per ciascuna delle *n* variabili)

Un linguaggio imperativo

- **Scrivi**: scrive in output una frase, un dato, il contenuto di una variabile di memoria, il risultato di un'espressione aritmetica, ecc.
- Esempi:
  - **scrivi "ciao!"** — Scrive la frase "**ciao!**" (tipicamente un computer scrive visualizzando l'output su uno schermo)
  - **scrivi a** — Scrive il contenuto della variabile **a**
  - **scrivi a+b** — Scrive il risultato dell'espressione aritmetica **a+b**, sommando il valore contenuto nella variabile **a** al contenuto della variabile **b**

Un linguaggio imperativo

- **Se ... allora ... altrimenti ...**: verifica una certa **condizione logica** e ne valuta il valore: ***se*** la condizione è vera, ***allora*** esegue determinate istruzioni, ***altrimenti*** ne esegue altre
- È l'operazione attraverso cui l'esecutore mette in pratica la capacità di eseguire confronti

- Esempi:
  - **se a ≥ 18 allora scrivi "Promosso!" altrimenti scrivi "Bocciato"**
  - **se a = b+c allora d = 45**
  - **se x > y e y > z allora scrivi "il più grande è" x**

Un linguaggio imperativo

- **Vai al passo ...**: salta all'istruzione ... dell'algoritmo

- Esempio:
  - **vai al passo 7** Invece di eseguire l'istruzione successiva nella sequenza delle istruzioni dell'algoritmo, salta all'istruzione n. 7

- **Stop**: interrompe l'esecuzione dell'algoritmo

- Esempio:
  - **stop** Termina l'esecuzione dell'algoritmo **:-)**

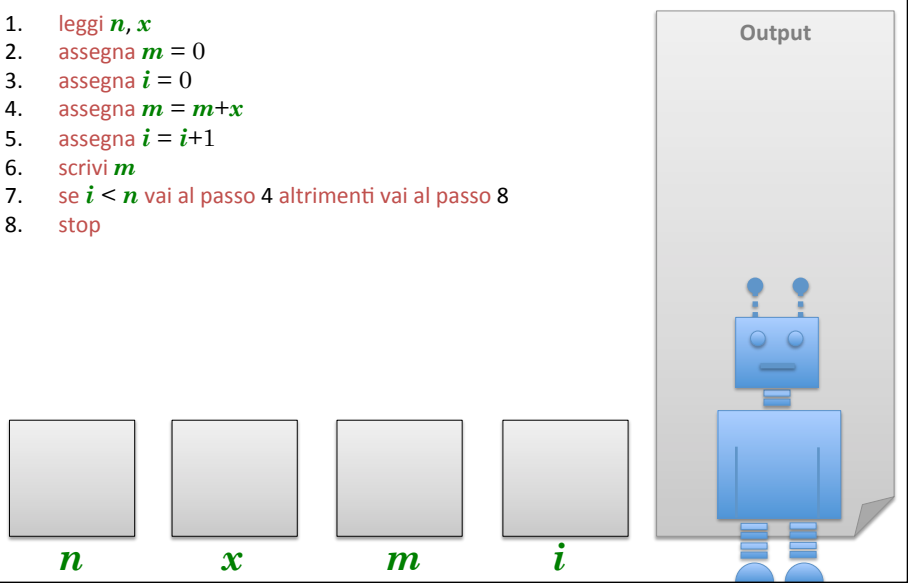
Serve un altro esempio!

- Dati due numeri x e n stampa i primi n multipli di x
- Esempio:
  - istanza del problema: n=4, x=7
  - soluzione dell'istanza del problema: 7, 14, 21, 28
- Algoritmo espresso con il linguaggio imperativo:
  1. leggi n, x
  2. assegna $m = 0$
  3. assegna $i = 0$
  4. assegna $m = m+x$
  5. assegna $i = i+1$
  6. scrivi m
  7. se $i < n$ vai al passo 4 altrimenti vai al passo 8
  8. stop

Eseguiamo l'algoritmo (...mettiamoci nei panni dell'esecutore)

ALGORITMO

1. leggi n, x
2. assegna $m = 0$
3. assegna $i = 0$
4. assegna $m = m+x$
5. assegna $i = i+1$
6. scrivi m
7. se $i < n$ vai al passo 4 altrimenti vai al passo 8
8. stop



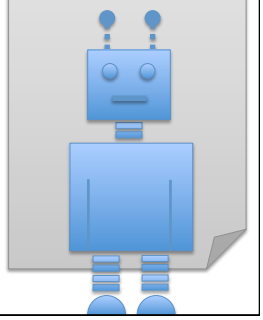
Eseguiamo l'algoritmo (...mettiamoci nei panni dell'esecutore)

ALGORITMO

1. leggi n, x
2. assegna $m = 0$
3. assegna $i = 0$
4. assegna $m = m+x$
5. assegna $i = i+1$
6. scrivi m
7. se $i < n$ vai al passo 4 altrimenti vai al passo 8
8. stop

SOLUZIONE!

Output: 7, 14, 21, 28



4	7	28	4
n	x	m	i

Diagrammi di flusso

- Per **rappresentare in modo efficace un algoritmo** sono stati sviluppati dei modelli grafici (i **diagrammi di flusso**) che associano alle istruzioni del programma dei simboli grafici:

Assegnazioni	$a = 17$
Input/Output	Leggi b
Condizioni	$x<3$?
Salti (*vai al passo...*)	→
Start/Stop	Fermati

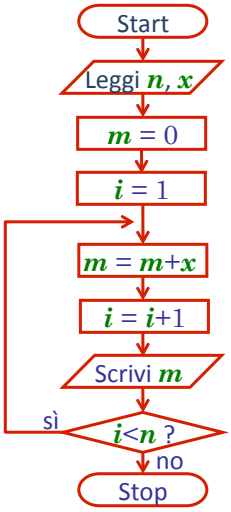
Diagrammi di flusso

PSEUDO-CODICE DELL'ALGORITMO

1. leggi n, x
2. assegna $m = 0$
3. assegna $i = 0$
4. assegna $m = m+x$
5. assegna $i = i+1$
6. scrivi m
7. se $i < n$ vai al passo 4 altrimenti vai al passo 8
8. stop

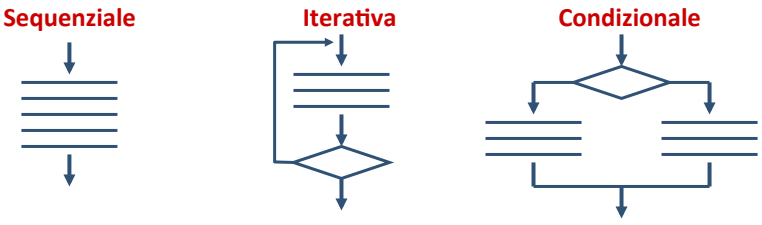
Due rappresentazioni equivalenti dello stesso algoritmo per il calcolo dei primi n multipli di x

DIAGRAMMA DI FLUSSO DELL'ALGORITMO



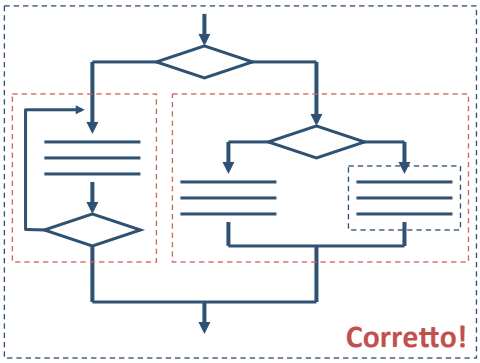
Programmazione strutturata

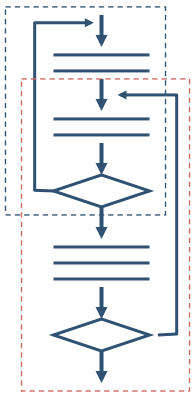
- I programmatori inesperti tendono ad "aggrovigliare" il programma introducendo numerosi salti privi di regole (*spaghetti programming*)
- È stato dimostrato (**Teorema fondamentale della programmazione strutturata di Jacopini e Böhm**) che ogni programma può essere codificato attenendosi esclusivamente a tre strutture fondamentali:



Programmazione strutturata

- Le tre strutture possono essere **concatenate** una di seguito all'altra oppure **nidificate** una dentro l'altra
- Non possono essere "intrecciate" o "accavallate"



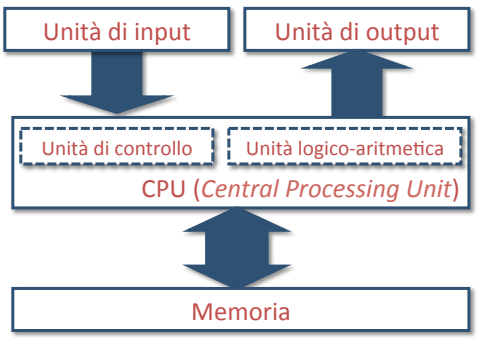


Sbagliato!

È un tipico esempio di *spaghetti programming*

Un esecutore reale: il computer

- Il **computer** è un esempio concreto di quell'esecutore un po' astratto di cui abbiamo parlato fino ad ora
- Le caratteristiche del "nostro" esecutore si sposano bene con quelle del calcolatore, come viene schematizzato nel **modello di Von Neumann** (1945)



John Von Neumann
(1903-1957)

Un esecutore reale: il computer

- Nel **modello di Von Neumann** è facile riconoscere gli elementi costitutivi di ogni moderno calcolatore
- Le componenti sono le seguenti:
  - Le **unità di input** tramite cui la macchina acquisisce informazioni dall'esterno
  - Le **unità di output** tramite cui la macchina produce (stampa) informazioni all'esterno
  - L'**unità centrale di elaborazione** (CPU) che elabora le istruzioni del programma (i passi dell'algoritmo), composta da due elementi:
    - L'**unità di controllo**: stabilisce l'ordine con cui devono essere eseguite le operazioni
    - L'**unità logico-aritmetica**: esegue operazioni aritmetiche e risolve espressioni logiche
  - La **memoria** in cui l'unità centrale deposita ed estrae le informazioni per poterle elaborare

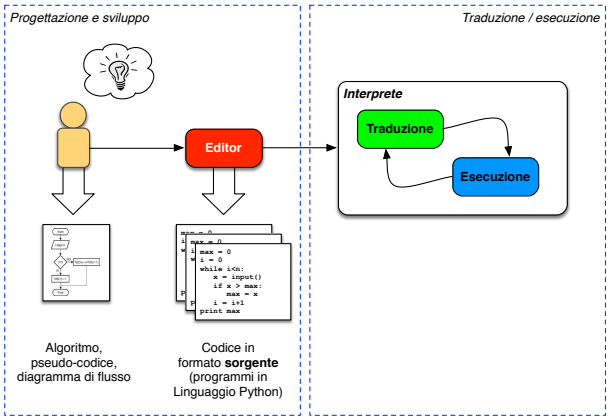
Linguaggi di programmazione

- Un **linguaggio di programmazione** è lo strumento con cui codificare gli algoritmi per renderli "comprensibili" all'esecutore/ computer
- Per poter tradurre ed eseguire le istruzioni di un programma scritto in un linguaggio di programmazione c'è bisogno di un **programma interprete** che traduca dal linguaggio di programmazione al linguaggio macchina
- Il linguaggio di programmazione è quindi un compromesso tra il nostro *linguaggio naturale* (troppo complesso e ambiguo) ed il *linguaggio macchina* (troppo povero e difficile da utilizzare)

- Un **programma** per il computer è un *algoritmo codificato con un linguaggio di programmazione*

L'interprete

- **Interprete**: itera più volte questo processo
  - Legge un'istruzione del programma "sorgente"
  - Traduce l'istruzione in linguaggio macchina
  - Esegue l'istruzione
  - Passa all'interpretazione dell'istruzione successiva
- È necessario disporre dell'interprete per poter eseguire il programma



Un linguaggio di programmazione reale: Python

- **Python** è un linguaggio di programmazione moderno e molto diffuso, si presta bene per scopi didattici perché è meno complicato di altri
- Il nome del linguaggio deriva dalla passione dell'autore del linguaggio per i comici britannici **Monty Python**
- Per eseguire un programma in Python serve:
  - un **programma editor** di testo per scrivere il programma e salvarlo sul computer (Notepad, TextEdit, Brackets, Kwrite, ...)
  - un **interprete del linguaggio Python**, disponibile gratuitamente per tutti i sistemi operativi (Microsoft Windows, Apple OS X, Linux, ...)
  - un buon punto di partenza: **http://www.python.it**

Un linguaggio di programmazione reale: Python

Pseudo-codice	Python	Esempio
Assegna	*variabile* = *valore*	a = 17 b = (c+d)/3
Leggi	*variabile* = **input**(“messaggio”)	nome = input(“Come ti chiami?”) n = int(input(“Inserisci numero:”))
Scrivi	**print**(“messaggio”, *variabile*)	print(“Ciao!”) print(“Risultato:”, x)

Se...allora...altrimenti...	**if** *condizione*:
	  istruzioni
	else:
	  altre istruzioni
		if a>b:
		  print(a, “è più grande di ”, b)
		else:
		 print(b, “è più grande di ”, a)

- Per le iterazioni:

  while *condizione*:
  istruzioni

  for *variabile* **in** *lista*:
  istruzioni

- Esempi:

  while x<5:
  print x
  x = x+1

  for x **in** 1, 2, 3, 4, 5:
  print x

Un linguaggio di programmazione reale: Python

- L’interprete del linguaggio è “interattivo”:

```
C:\> python

Python 3.4.2
Type "help", "copyright", "credits" or "license" for more information.
>>> a=3;
>>> b=a*2;
>>> print(b);
6
>>>
```

Python: qualche esempio

- Letti in input due numeri a e b stampa la somma in output

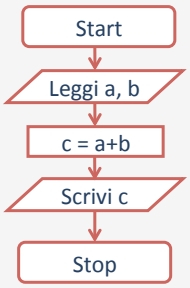
Algoritmo in pseudo-codice

leggi a, b
$c = a+b$
scrivi c

Programma in Python

```
a = int(input("Inserisci un numero:"))
b = int(input("Inserisci un numero:"))
c = a+b
print("La somma di ", a, " e ", b, " è ", c)
```

Diagramma di flusso



Python: qualche esempio

- Letti in input due numeri x e y, scrivi in output il massimo

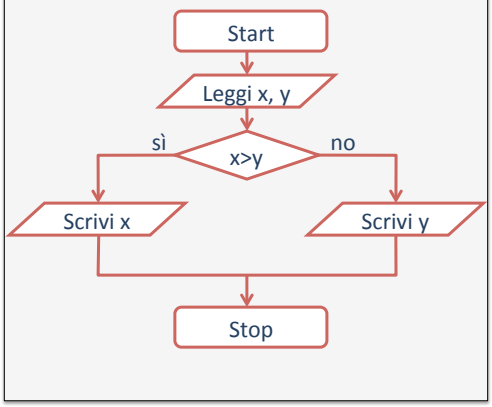
Algoritmo in pseudo-codice

leggi x, y
se $x>y$
allora scrivi x
altrimenti scrivi y

Programma in Python

```
x = int(input("Inserisci x: "))
y = int(input("Inserisci y: "))
if x>y:
    print(x)
else:
    print(y)
```

Diagramma di flusso



Python: qualche esempio

- Letti in input tre numeri x, y e z scrivi in output il massimo

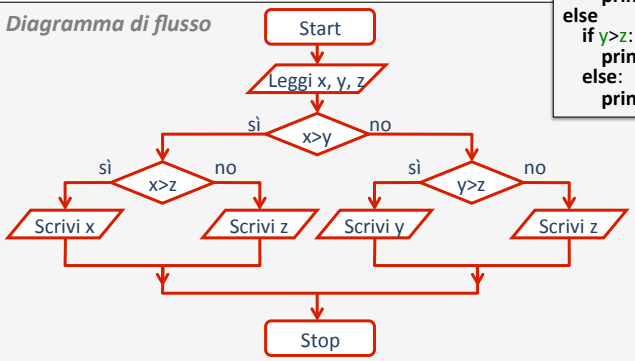
Algoritmo in pseudo-codice

leggi x, y, z
se $x>y$ allora
  se $x>z$
    allora scrivi x altrimenti scrivi z
  altrimenti
    se $y>z$
    allora scrivi y altrimenti scrivi z

Programma in Python

```
x = int(input("Inserisci x: "))
y = int(input("Inserisci y: "))
z = int(input("Inserisci z: "))
if x>y:
   if x>z:
      print(x)
   else:
      print(z)
else
   if y>z:
      print(y)
   else:
      print(z)
```

Diagramma di flusso



Python: qualche esempio

- Letti in input n numeri, scrive in output il massimo

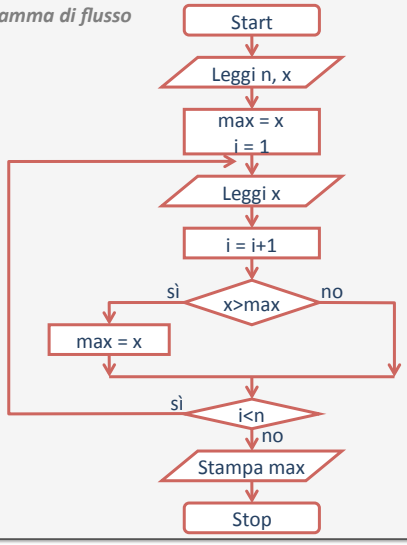
Algoritmo in pseudo-codice

1. leggi n
2. $max = 0$
3. $i = 0$
4. se $i \geq n$ allora vai al passo 9
5. leggi x
6. se $x > max$ allora $max = x$
7. $i = i+1$
8. vai al passo 4
9. scrivi max
10. stop

Programma in Python

```
n = int(input("Numero di elementi: "))
print("Inserisci ", n, " numeri: ")
max = 0
i = 0
while i<n:
  x = int(input());
  if x > max:
    max = x
  i = i+1
print("Massimo: ", max)
```

Diagramma di flusso



Python: qualche esempio

- Letti in input due numeri x e y, calcola il **quoziente** e il **resto** della divisione $x:y$.

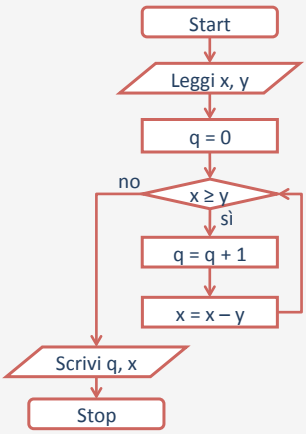
Algoritmo in pseudo-codice

1. acquisisci x e y
2. sia $q = 0$
3. se $x<y$ allora
   scrivi "quoziente = q, resto = x" e fermati
4. altrimenti
   sia $q = q+1$, $x = x-y$
5. vai al passo n. 3

Programma in Python

```
x = int(input("inserisci un numero:"))
y = int(input("inserisci un numero:"))
q = 0
while x≥y:
    q = q + 1
    x = x – y
print("quoziente ", q, "resto ", x)
```

Diagramma di flusso



Algoritmi e risoluzione "automatica" di problemi

... e non finisce qui, ma per il momento ...

Grazie per l'attenzione!