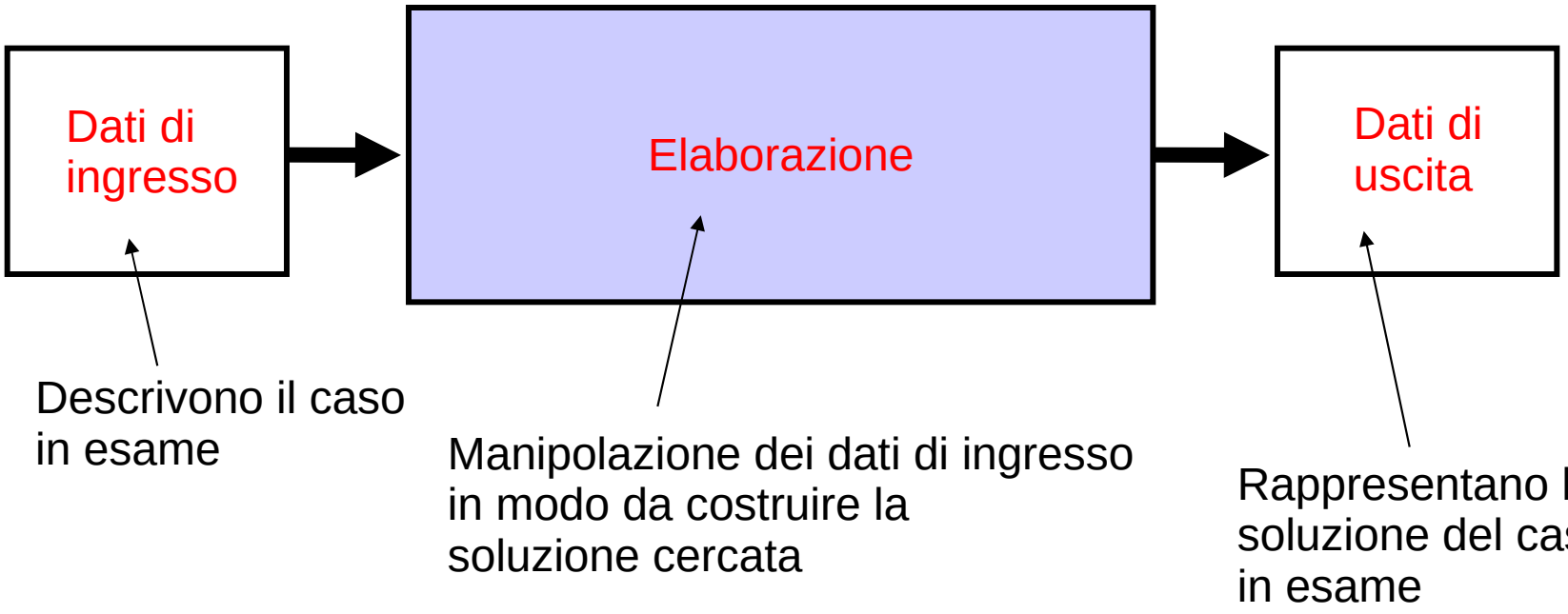
12 - Gli algoritmi e la risoluzione di problemi

Sommario

- Che significa risolvere un problema?
- Algoritmi e programmi
- Rappresentazione di algoritmi
- Lo pseudocodice
- Primitive dello pseudocodice

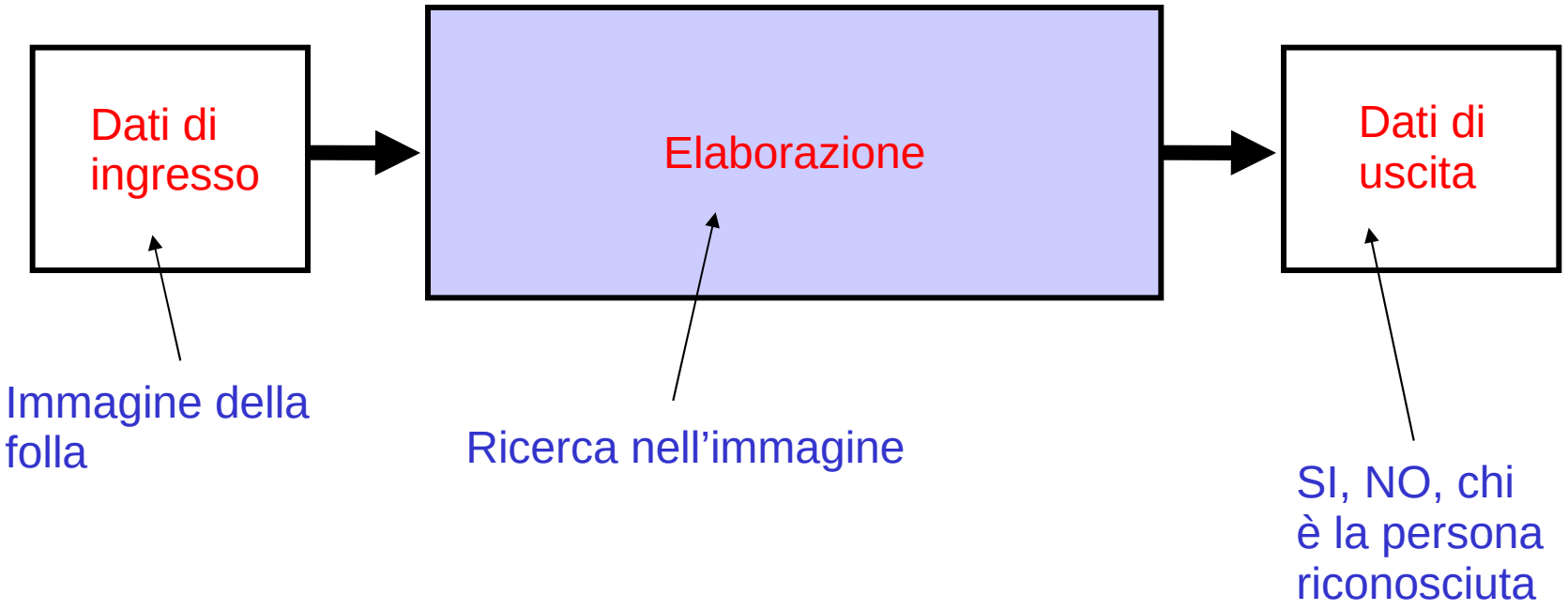
Risolvere un problema

- Come viene risolto un problema :



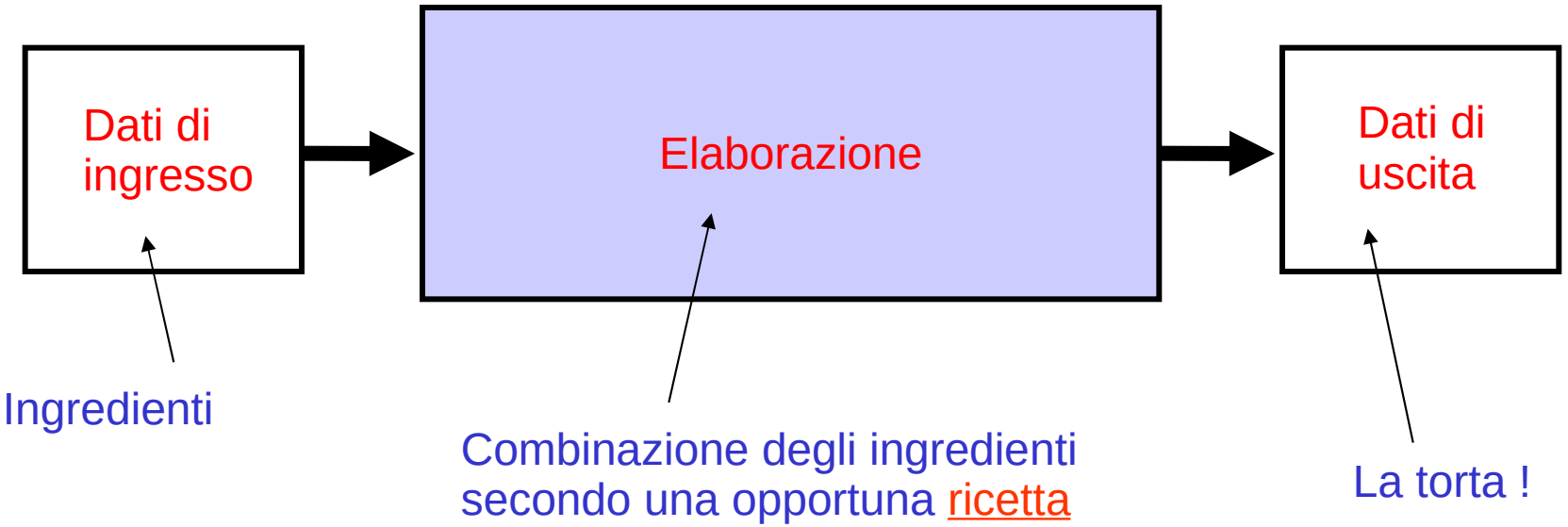
Risolvere un problema (2)

- es : riconoscere qualcuno fra la folla



Risolvere un problema (3)

- es : torta di carote



Risolvere un problema (4)

- Una prima considerazione :

  essere capaci di risolvere un problema non significa essere capaci di spiegare esattamente *come* questo avviene

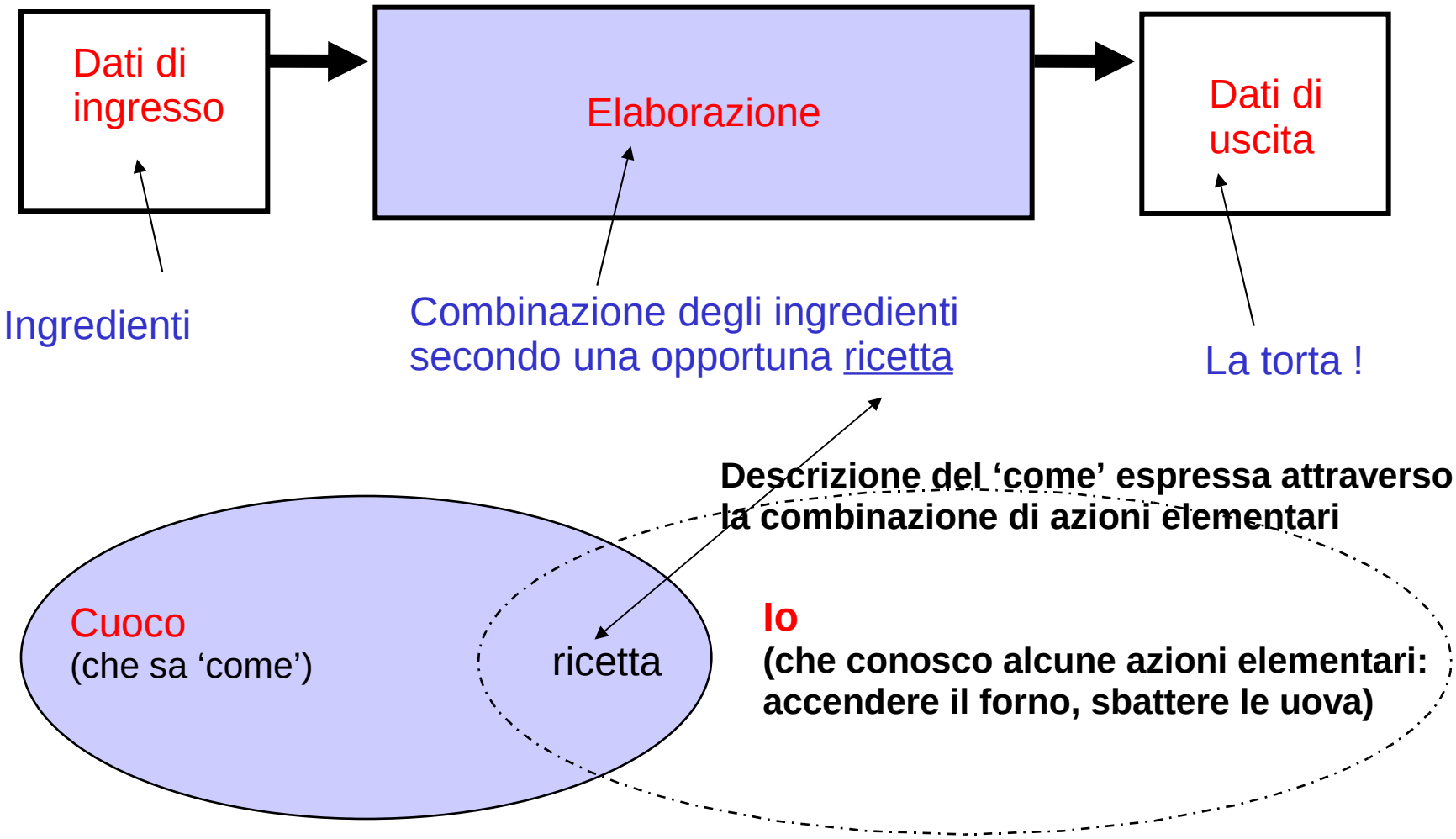
Risolvere un problema (5)

- Seconda considerazione :

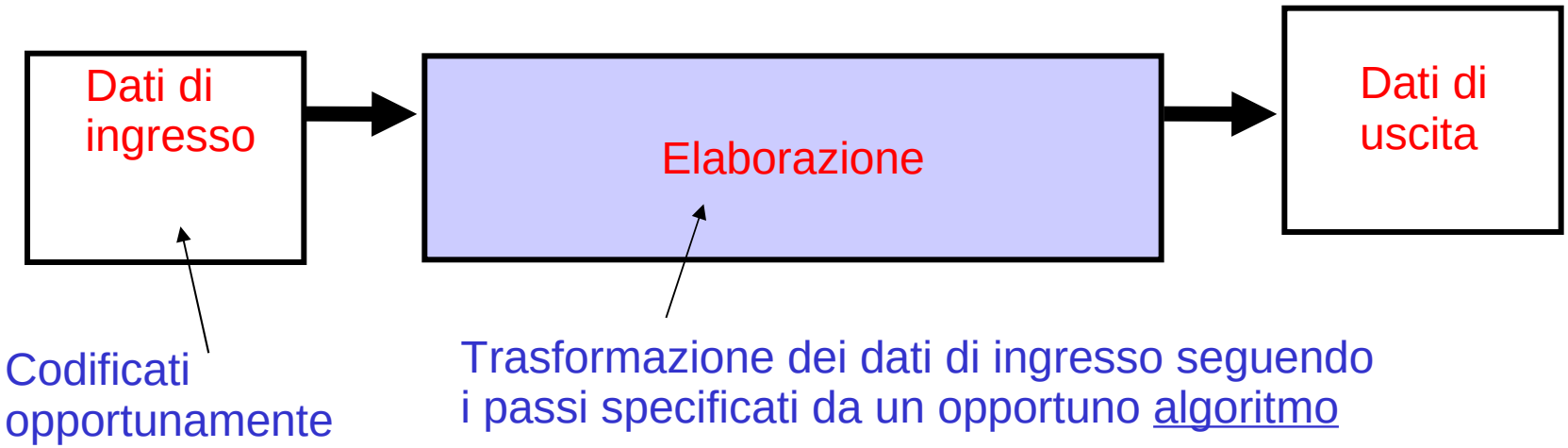
  vogliamo essere capaci di specificare la strategia seguita dal passo di elaborazione in modo da farla eseguire 'automaticamente' dal computer quindi dobbiamo :

  - *riuscire a descrivere accuratamente i vari passi della soluzione attraverso azioni che il calcolatore è in grado di effettuare e con un linguaggio che è in grado di comprendere*

Risolvere un problema (6)



Problemi, algoritmi e programmi



Ovvero la descrizione dell'algoritmo secondo un linguaggio comprensibile al calcolatore

Umano
(che conosce l'algoritmo)

programma

Calcolatore
(che conosce alcune azioni elementari: es confrontare due numeri, eseguire semplici operazioni aritmetiche

Algoritmi e programmi (2)

- **Algoritmo** (def) :
  - una sequenza di azioni **non ambigue** che trasformi i dati iniziali nel risultato finale utilizzando un insieme di **azioni elementari** che possono essere eseguite da un opportuno **esecutore** (e definiscono un processo **terminante**).
- **Programma** (def)
  - **rappresentazione** di un algoritmo utilizzando un linguaggio non ambiguo e direttamente comprensibile dal computer

Algoritmi e Ricette

- Ma insomma, una ricetta è proprio un algoritmo?

  … NO, ovvero è molto simile ma con due importanti differenze:

  - La sequenza di azioni contiene spesso degli elementi di *ambiguità* risolti da un esecutore intelligente
    - es: spesso non si specificano gli strumenti da utilizzare, confidando che l'esecutore umano *sbatta le uova* nel posto giusto
    - es: sale q.b.
  - Non tutti i possibili casi vengono specificati

    es: è chiaro che se c'e' puzza di bruciato conviene spegnere il forno, anche se la ricetta non lo specifica

    (si confida nelle capacità deduttive dell'esecutore)

Ancora una definizione di algoritmo

- Un'altra definizione, per ribadire il concetto: ce ne sono tante...

La descrizione di un procedimento risolutivo di un problema può considerarsi un algoritmo se rispetta alcuni requisiti essenziali:

- ***Finitezza**: un algoritmo deve essere composto da una sequenza finita di passi elementari*
- ***Eseguibilità**: il potenziale esecutore deve essere in grado di eseguire ogni singola azione in tempo finito con le risorse a disposizione*
- ***Non-ambiguità**: l'esecutore deve poter interpretare in modo univoco ogni singola azione*

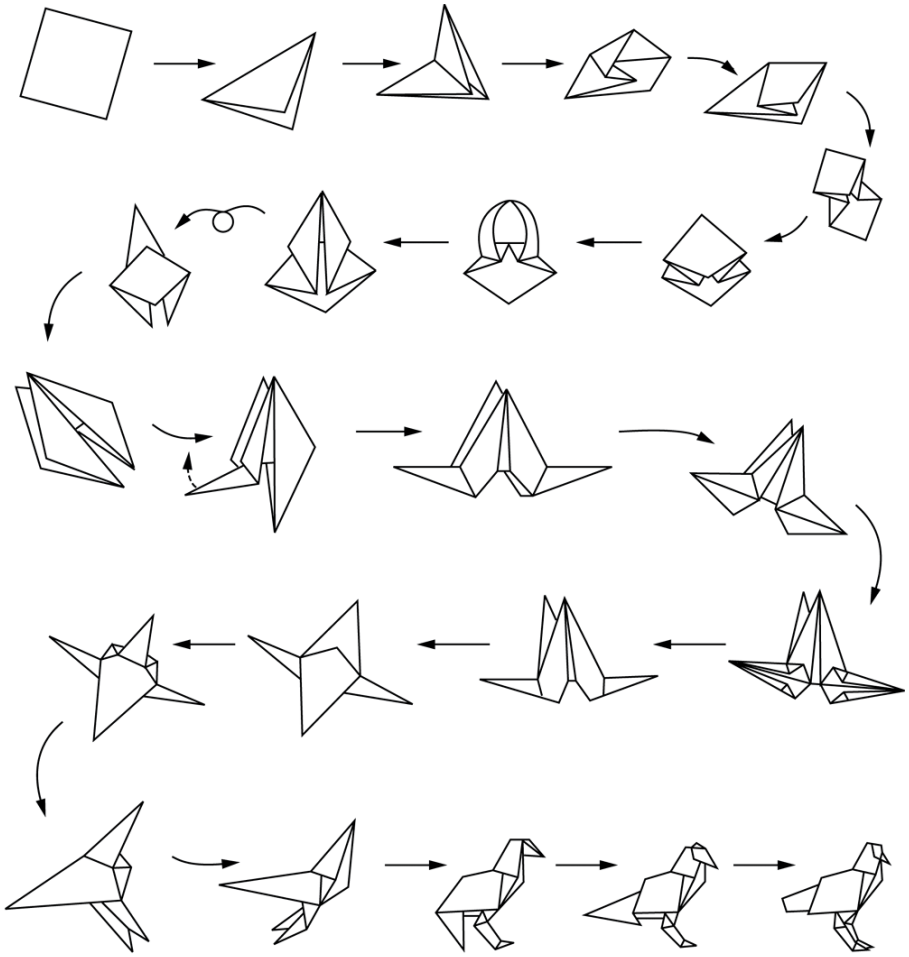
Problemi e algoritmi

- Un algoritmo, in generale, risolve un problema
- Esempi di algoritmi:
  - addizione o moltiplicazione di più numeri a più cifre
  - estrazione della radice quadrata
  - complemento a due di un numero binario con NOT e +1
- Un problema può essere risolto da più algoritmi
  - quanti algoritmi conoscete per il complemento a due?
  - altri problemi con più algoritmi di risoluzione?

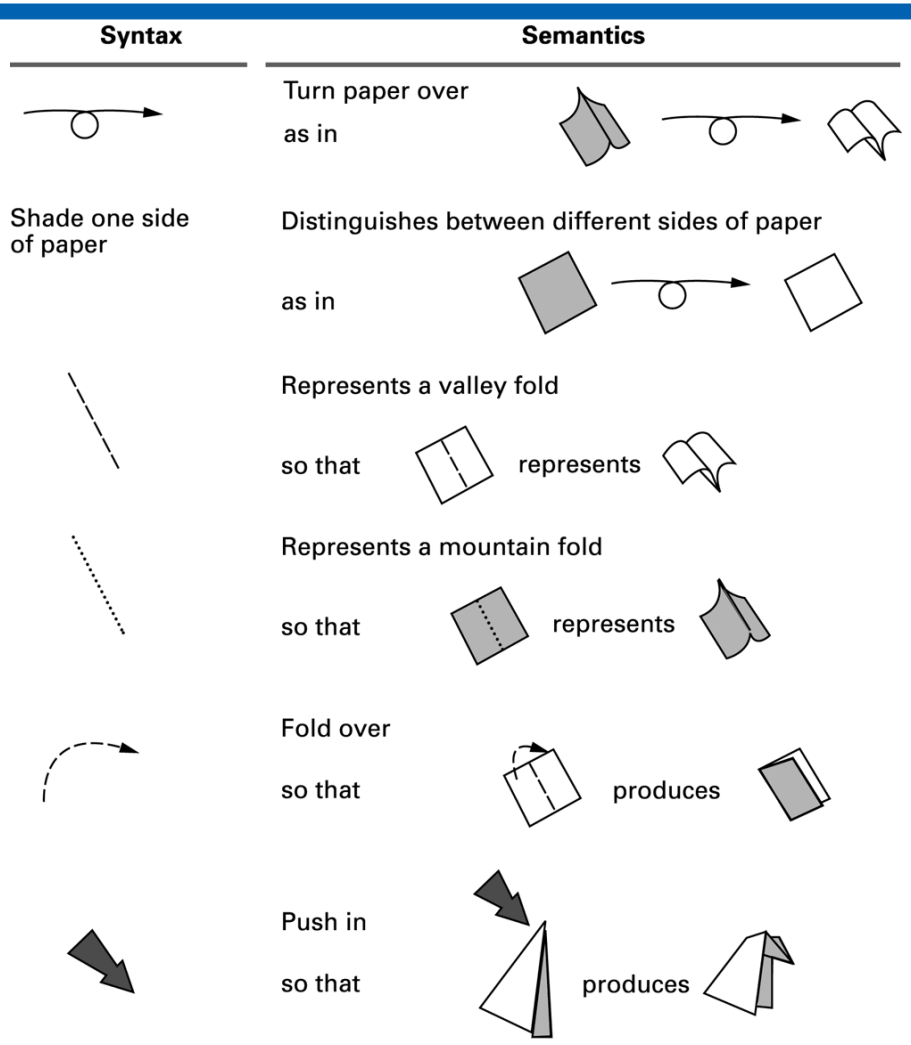
Rappresentazione di algoritmi

- Differenza tra algoritmo (**semantica**) e rappresentazione (**sintassi**) [come per i numeri...]
- Lo stesso algoritmo può essere rappresentato in vari modi
  - formula, sequenza di istruzioni, disegno, a parole...
  - a diversi **livelli di astrazione**
- Ogni rappresentazione si basa su un insieme di **primitive** ben definite, comprensibili all'**esecutore**
- **Un primitiva ad un livello di astrazione può essere definita da un algoritmo che usa primitive di livello di astrazione inferiore.**

Esempio di algoritmo: origami



Primitive per origami



Algoritmi e linguaggi di programmazione

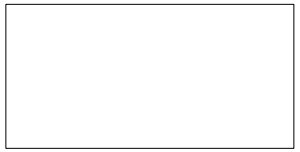
- Se l'esecutore è un computer, una collezione di primitive per decrivere un algoritmo, insieme alle regole per comporle, costituisce un **linguaggio di programmazione**.
- Es: il **linguaggio macchina** che abbiamo visto
  - **Primitive**: istruzioni macchina (LOAD, STORE, ADD...)
  - **Regole** per comporle: si scrivono in sequenza
  - **Livello di astrazione**: adeguato per la CPU, non per umani
- Vedremo **linguaggi di programmazione ad alto livello**, con primitive adeguate per umani
- Per il momento usiamo uno **pseudocodice** che permette una rappresentazione informale degli algoritmi.

Lo pseudocodice

- Costituito da **istruzioni elementari** e **strutture di controllo** per comporle
  - **assegnamento**
  - altre istruzioni descritte informalmente (stampa, ...)
  - sequenzializzazione
  - **selezione condizionale** (**if ... then ... else**)
  - **iterazione** (**while ... do**, **repeat ... until**)
  - possibilità di definire **procedure**, utilizzabili come nuove istruzioni elementari
- Poiché è un linguaggio informale, lo modificheremo quando conviene. Non possibile per linguaggi formali

Primitive dello pseudocodice: assegnamento

- .**Assegnamento:** ***nome ← espressione***
  - (anche ***nome := espressione***)
- Effetto: "associa" a ***nome*** il valore ottenuto valutando ***espressione***. Successivamente ***nome*** può essere usato per richiamare quel valore.
- Possiamo pensare a ***nome*** come a una scatola (cella di memoria) in cui mettiamo il valore



nome

50
2.50
125.0

```
quantità ← 50
prezzoUnitario ← 2.50
prezzoTotale ← quantità * prezzoUnitario
```

Primitive dello pseudocodice: selezione condizionale

- Serve per scegliere cosa fare in base al valore di una espressione booleana (quindi **true** o **false**)
- **Sintassi:**

  if *condizione* then *azione* else *azione*
- **Significato:** Si valuta la ***condizione***
  - Se il risultato è **true**, si esegue l'azione dopo **then**
  - Se il risultato è **false**, si esegue l'azione dopo **else**

```
if (l'anno è bisestile)
then (totaleGiornaliero ← totale/365)
else (totaleGiornaliero ← totale/366)
```

Primitive dello pseudocodice: iterazione

- Serve per ripetere un insieme di azioni per un certo numero di volte
- **Sintassi:**

  while *condizione* do *attività*
- **Significato:** Si valuta la ***condizione***
  - Se il risultato è **true**, si esegue l'attività, quindi si riesegue tutto il **while**
  - Se il risultato è **false**, si termina l'esecuzione (continuando con la prossima istruzione

```
while (ci sono esercizi da svolgere)
do (svolgi il prossimo esercizio;
    controlla con il vicino;
     riposati un po')
```

Primitive dello pseudocodice: procedure

- Una **procedura** associa delle azioni da eseguire a un nome. E' come definire una nuova primitiva.

```
procedure Saluti
Contatore ← 3;
while (Contatore > 0) do
     (stampa il messaggio "Saluti";
      Contatore ← Contatore -1)
```

Effetto della procedura: stampa tre volte la stringa "**Saluti**"; provare a eseguirla!
Esempio di uso della procedura:

```
if (l'ospite è simpatico)
then (esegui la procedura Saluti)
else (...)
```

Primitive dello pseudocodice: procedure con parametri

- Per renderle più flessibili, le procedure possono avere **parametri** (o nomi generici). Un parametro viene usato come se fosse un valore nella procedura. Quando si esegue la procedura va fornito un valore per il parametro.

```
procedure SalutiBis(Contatore)
while (Contatore > 0) do
     (stampa il messaggio "Saluti";
      Contatore ← Contatore -1)

if (l'ospite è molto simpatico)
then (esegui la procedura SalutiBis(5))
else (esegui la procedura SalutiBis(1))
```