Insegnare Informatica a scuola: sfide e strategie.

Violetta Lonati

Università degli studi di Milano
Dipartimento di Informatica

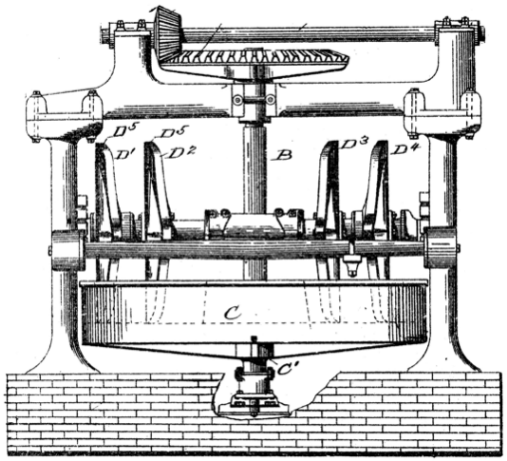
Bologna, 12 novembre 2019

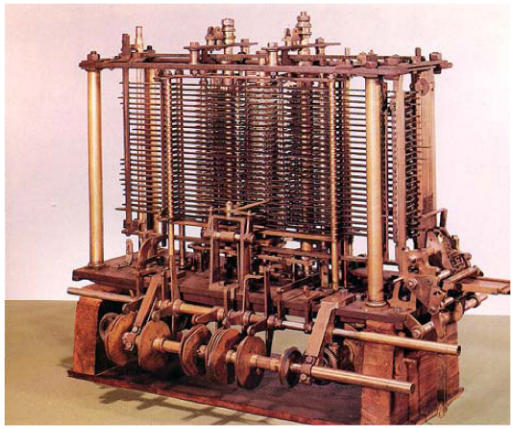
Tre parole chiave per l'informatica

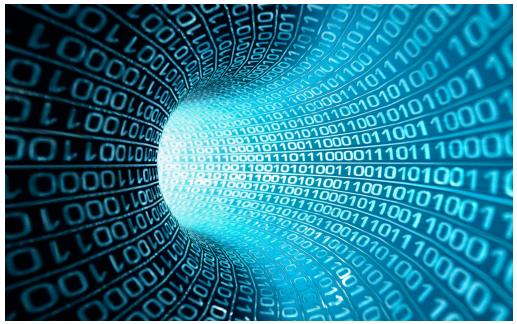
L'informatica è la disciplina scientifica che studia i principi e i metodi per l'elaborazione automatica dell'informazione.

Tre parole chiave per l'informatica

L'informatica è la disciplina scientifica che studia i principi e i metodi per l'elaborazione automatica dell'informazione.







ELABORAZIONE
Come si può trasformare l'informazione al fine di produrre nuova conoscenza?

AUTOMATICA
Quali manipolazioni possono essere eseguite da un interprete meccanico? E come?

INFORMAZIONE
Che cosa è l'informazione? Come si possono usare simboli o numeri per rappresentarla?

Informatica e programmazione

INFORMATICA
disciplina scientifica che studia i principi e i metodi
per l'elaborazione automatica dell'informazione

Informatica e programmazione

INFORMATICA
disciplina scientifica che studia i principi e i metodi
per l'elaborazione automatica dell'informazione

PROGRAMMAZIONE
come metodo per automatizzare la soluzione di problemi
e/o lo svolgimento di compiti

Informatica e programmazione

INFORMATICA
disciplina scientifica che studia i principi e i metodi
per l'elaborazione automatica dell'informazione

↓

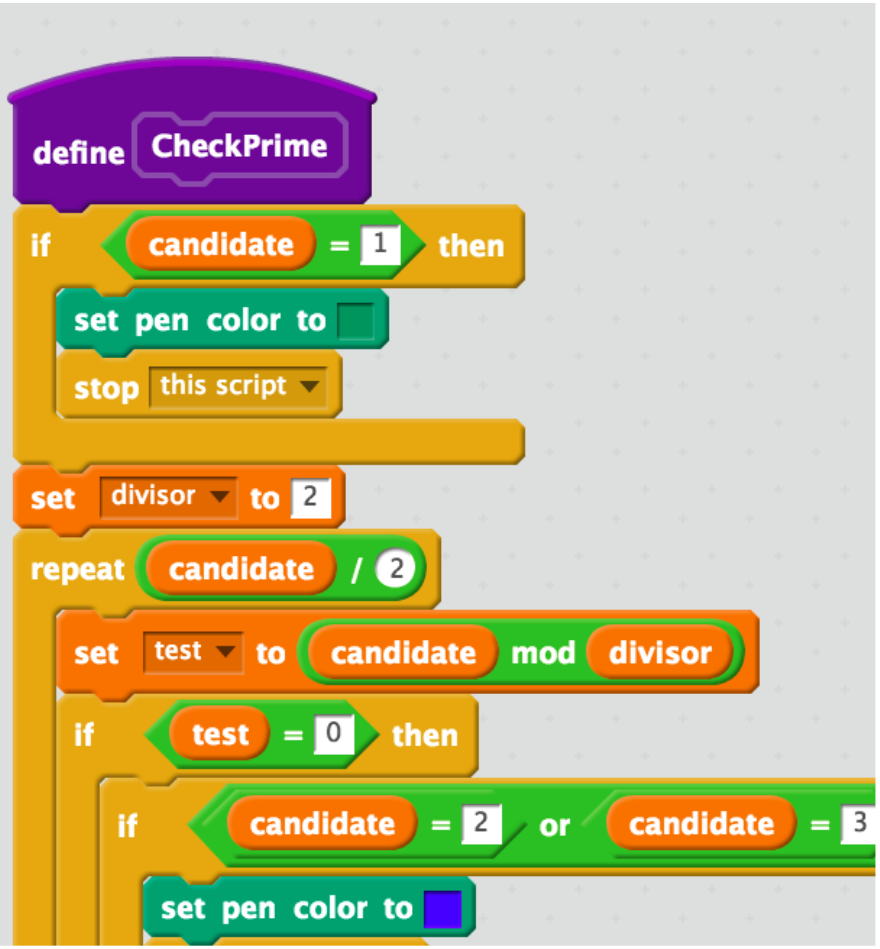
PROGRAMMAZIONE
come metodo per automatizzare la soluzione di problemi
e/o lo svolgimento di compiti

↓

CODING?

Coding?

```
function test_prime(n)
{
  if (n===1)
  {
    return false;
  }
  else if(n === 2)
  {
    return true;
  }else
  {
    for(var x = 2; x < n; x++)
    {
      if(n % x === 0)
      {
        return false;
      }
```



Programmazione. . . in che senso?

- Introduzione alla programmazione come metodo per automatizzare la soluzione di problemi e/o lo svolgimento di compiti
  - concetto di interprete/esecutore
  - differenza tra programmatore e interprete/esecutore
  - scelta e combinazione di primitive
  - importanza di un linguaggio non ambiguo e rigoroso (con regole formali)

Programmazione. . . in che senso?

- Introduzione alla programmazione come metodo per automatizzare la soluzione di problemi e/o lo svolgimento di compiti
  - concetto di interprete/esecutore
  - differenza tra programmatore e interprete/esecutore
  - scelta e combinazione di primitive
  - importanza di un linguaggio non ambiguo e rigoroso (con regole formali)
- Programmazione come tecnica, nel significato etimologico del termine grego $\tau\varepsilon\chi\nu\eta$: “arte” nel senso di “perizia”, “saper fare”, “saper operare”.

Programmazione. . . in che senso?

- Introduzione alla programmazione come metodo per automatizzare la soluzione di problemi e/o lo svolgimento di compiti
  - concetto di interprete/esecutore
  - differenza tra programmatore e interprete/esecutore
  - scelta e combinazione di primitive
  - importanza di un linguaggio non ambiguo e rigoroso (con regole formali)
- Programmazione come tecnica, nel significato etimologico del termine grego $\tau\varepsilon\chi\nu\eta$: “arte” nel senso di “perizia”, “saper fare”, “saper operare”.
- Saper programmare significa aver sviluppato competenze diverse, che richiedono la capacità di ragionare e operare contemporanemente
  - a livello pratico e teorico,
  - in senso astratto e concreto,
  - con una capacità di visione generale e particolare,

L'informatica nelle indicazioni nazionali per i licei

"comprendere la valenza metodologica dell'informatica nella formalizzazione e modellizzazione dei processi complessi e nell'individuazione di procedimenti risolutivi"

...

"algoritmi e linguaggi di programmazione"

L'informatica nelle indicazioni nazionali per i licei

"comprendere la valenza metodologica dell'informatica nella formalizzazione e modellizzazione dei processi complessi e nell'individuazione di procedimenti risolutivi"

...

"algoritmi e linguaggi di programmazione"

LA SFIDA

Insegnare la programmazione
non tanto per far acquisire un insieme di tecniche
ma per la sua valenza metodologica e culturale.

Quali strategie?

Programmare implica:

- formalizzare/modellizzare il problema e la sua soluzione (o il compito e cosa vuol dire portarlo a termine)
- individuare di procedimenti risolutivi al problema (procedure automatiche per lo svolgimento del compito)
- implementare - realizzare programmi (ovvero traduzioni concrete di tali procedimenti in un linguaggio di programmazione) affiché siano eseguibili da un agente autonomo

Quali strategie?

Programmare implica:

- formalizzare/modellizzare il problema e la sua soluzione (o il compito e cosa vuol dire portarlo a termine)
- individuare di procedimenti risolutivi al problema (procedure automatiche per lo svolgimento del compito)
- implementare - realizzare programmi (ovvero traduzioni concrete di tali procedimenti in un linguaggio di programmazione) affiché siano eseguibili da un agente autonomo

Quali strategie?

Alcuni spunti dalla letteratura sulla didattica dell'informatica

- Syntactic, conceptual, strategic knowledge
- La *notional machine*
- Scrittura Vs lettura di programmi

Syntactic, conceptual, strategic knowledge

- Syntactic knowledge: conoscenza della sintassi del linguaggio di programmazione.
- Conceptual knowledge: riguarda la dinamica del programma in esecuzione ("the program as a working mechanism"), cioè è la conoscenza di come funzionano i costrutti della programmazione e di come questi determinano l'esecuzione del codice (vedi sotto: Notional machine).
- Strategic knowledge: riguarda la capacità di applicare la conoscenza sintattica e concettuale al fine di risolvere nuovi problemi e raggiungere obiettivi specifici.

Esempio

```c
int main( void ) {
  int n, x = 0;

  do {
    scanf( "%d", &n );
    x = x + n;
  } while ( n != 0 );

  printf( "%d\n", x );
  return 0;

}
```

Syntactic knowledge

sintassi del ciclo `do-while` o dell'operatore += in C

Conceptual knowledge

quante volte viene eseguito il ciclo for, quando si interrompe

Strategic knowledge

uso della variabile x come *accumulatore* per calcolare la somma dei valori letti

La *notional machine*

> *The notional machine is an idealized, conceptual computer whose properties are implied by the constructs in the programming language employed.*

[du Boulay, B. (1989). *Some difficulties of learning to program.*]

- Il senso della notional machine è fornire delle basi per capire il comportamento dei programmi in esecuzione.
- Questo modello astratto di macchina nei principianti è incompleto o scorretto e questo è fonte di molte misconceptions.

[Sorva, J. 2013. *Notional machines and introductory programming education*. ACM Transactions on Computing Education]

Scrittura o lettura di programmi

- Di solito l'obiettivo dei corsi di programmazione introduttivi è imparare a programmare, cioè scrivere programmi
- La capacità di leggere e comprendere il codice, invece, di solito non è un obiettivo esplicito, non viene insegnata, ma ci si aspetta che venga sviluppata come "effetto collaterale".
- Anche solo la capacità di tracciare passo-passo l'esecuzione di codice risulta essere assai poco sviluppata (è una cosa che ci vedono sicuramente fare ma che raramente insegniamo o chiediamo di fare esplicitamente)
- Le capacità di scrivere e di comprendere codice risultano essere molto correlate.

[Whalley J., Lister R, et al. 2006. An Australasian study of reading and comprehension skills in novice programmers, using the bloom and SOLO taxonomies] e lavori seguenti

Scrittura o lettura di programmi

Le capacità di scrivere e di comprendere codice risultano essere molto correlate.

> *We believe that the more likely explanation for a correlation between writing code and explaining code is that both writing and explaining depend upon a common set of skills concerned with reasoning about programs. If reasoning about code is the underlying skill that is common to both code writing and code explaining, then the crucial pedagogical question is how to most efficiently develop that underlying skill.*

[Lister et al. 2014. 'Explain in plain english' questions revisited: data structures problems. SIGCSE '14. ACM]

Esercizi di lettura e comprensione del codice

- Traccia l'esecuzione di programmi in maniera sistematica
- Parsons puzzles
- Spiega con parole tue cosa fa una data porzione di codice
- Dai un nome più significativo a una variabile o a una funzione/metodo

Esempio

Questa funzione realizza in modo assai discutibile un calcolo che può essere descritto brevemente:

```
func f(x int) int {
  var a, b, c int

  for x > 0 {
    a = x % 10
    b = 1 - a%2
    c += b
    x /= 10
  }
  return c
}
```

- Senza eseguire il programma al computer, tracciatene l'esecuzione quando `x` è uguale a 1344.
- Date un nome più significativo alle variabili `a`, `b` e `c`.
- Riassumete con una frase cosa restituisce la funzione `f`.

Esempio

Questa funzione realizza in modo assai discutibile un calcolo che può essere descritto brevemente:

```
func f(x int) int {
  var a, b, c int

  for x > 0 {
    a = x % 10
    b = 1 - a%2
    c += b
    x /= 10
  }
  return c
}
```

- Senza eseguire il programma al computer, tracciatene l'esecuzione quando `x` è uguale a 1344.
- Date un nome più significativo alle variabili `a`, `b` e `c`.
- Riassumete con una frase cosa restituisce la funzione `f`.

Esempio

Questa funzione realizza in modo assai discutibile un calcolo che può essere descritto brevemente:

```
func f(x int) int {
  var a, b, c int

  for x > 0 {
    a = x % 10
    b = 1 - a%2
    c += b
    x /= 10
  }
  return c
}
```

- Senza eseguire il programma al computer, tracciatene l'esecuzione quando `x` è uguale a 1344.
  La funzione restituisce il valore 2.
- Date un nome più significativo alle variabili `a`, `b` e `c`.
- Riassumete con una frase cosa restituisce la funzione `f`.

Esempio

Questa funzione realizza in modo assai discutibile un calcolo che può essere descritto brevemente:

```
func f(x int) int {
  var a, b, c int

  for x > 0 {
    a = x % 10
    b = 1 - a%2
    c += b
    x /= 10
  }
  return c
}
```

- Senza eseguire il programma al computer, tracciatene l'esecuzione quando `x` è uguale a 1344.
  La funzione restituisce il valore 2.
- Date un nome più significativo alle variabili `a`, `b` e `c`.
  `cifra, pari, contaPari`
- Riassumete con una frase cosa restituisce la funzione `f`.

Esempio

Questa funzione realizza in modo assai discutibile un calcolo che può essere descritto brevemente:

```
func f(x int) int {
  var a, b, c int

  for x > 0 {
    a = x % 10
    b = 1 - a%2
    c += b
    x /= 10
  }
  return c
}
```

- Senza eseguire il programma al computer, tracciatene l'esecuzione quando `x` è uguale a 1344.
  La funzione restituisce il valore 2.
- Date un nome più significativo alle variabili `a`, `b` e `c`.
  `cifra, pari, contaPari`
- Riassumete con una frase cosa restituisce la funzione `f`.
  Calcola e restituisce il numero di cifre pari del parametro `x`

Esempio

Questa funzione realizza in modo assai discutibile un calcolo che può essere descritto brevemente:

```
func f(x int) int {
  var a, b, c int

  for x > 0 {
    a = x % 10
    b = 1 - a%2
    c += b
    x /= 10
  }
  return c
}
```

Una variante migliore:

```
for x > 0 {
      a = x % 10
      if ( a % 2 == 0 ) {
          c++
      }
      x /= 10
}
```

Esempio

Questa funzione realizza in modo assai discutibile un calcolo che può essere descritto brevemente:

```
func f(x int) int {
  var a, b, c int

  for x > 0 {
    a = x % 10
    b = 1 - a%2
    c += b
    x /= 10
  }
  return c
}
```

Una variante migliore:

```
for x > 0 {
      a = x % 10
      if ( a % 2 == 0 ) {
          c++
      }
      x /= 10
}
```

Una variante ancora migliore:

```
for x > 0 {
      cifra = x % 10
      if ( cifra % 2 == 0 ) {
          contaPari++
      }
      x /= 10
}
```

Strategic knowledge: goals and plans

- Molti studi confermano che la conoscenza concettuale (notional machine) e soprattutto la conoscenza strategica sono le più difficili da costruire per chi sta imparando a programmare.
- Una delle differenze che si osserva tra gli esperti e i novizi quando leggono il codice è che i novizi tendono ad analizzarlo riga per riga sequenzialmente mentre gli esperti cercano dei "beacons" (segnali, letteralmente "fari") e li usano per identificare dei "chunks" (porzioni di codice significative).
- La didattica tuttavia (es: libri di testo) tende a dare maggior spazio alla conoscenza sintattica o al più concettuale e poco a quella strategica. Ci si aspetta che questa venga costruita da sé.

Strategic knowledge: goals and plans

[Soloway, E. 1989 *Learning to Program = Learning to Construct Mechanisms and Explanations*]

- Analizzare i problemi in termini di goals che devono essere affrontati e di plans, cioè soluzioni pronte per l'uso ("stereotypical canned solution").
  - Esempio: per sommare un insieme di valori faccio un ciclo che esamina i valori uno alla volta e li accumulo in una variabile `sum` opportunamente inizializzata.
- Combinare i plans - una delle difficoltà maggiori
  - Esempio: per fare la media di un insieme di valori è necessario combinare un ciclo di conteggio con un ciclo che calcola la somma, poi dividere controllando prima che il denominatore non sia 0...

Pattern e ruoli delle variabili

- Il concetto di *Plan* è tuttavia abbastanza fumoso.
- In letteratura il tema è stato ripreso con molti nomi diversi.
- Alcuni approcci molto operativi:
  - Pattern (elementari)
    `https://www.cs.uni.edu/~wallingf/patterns/elementary/`
  - Ruoli delle variabili (es: stepper, gatherer, counter,...)
    [Jorma Sajaniemi and Marja Kuittinen. 2005. An Experiment on Using Roles of Variables in Teaching Introductory Programming. Computer Science Education.]

Esempio

Scrivi un programma che calcola la media.

Goals

1. Sommare i valori
2. Contare i valori
3. Dividere somma per numeri di valori (controllando che tale numero sia positivo!)

Plans

1. Sum loop (con variabile *accumulatore*)
2. Count loop (con variabile *contatore*)
3. Guarded division (controllata da `if`)

Conclusioni

- Programmazione come metodo per automatizzare la soluzione di problemi e/o lo svolgimento di compiti

Conclusioni

- Programmazione come metodo per automatizzare la soluzione di problemi e/o lo svolgimento di compiti
- Saper programmare significa aver sviluppato competenze diverse, che richiedono la capacità di ragionare e operare contemporanemente
  - a livello pratico e teorico,
  - in senso astratto e concreto,
  - con una capacità di visione generale e particolare,

Conclusioni

- Programmazione come metodo per automatizzare la soluzione di problemi e/o lo svolgimento di compiti
- Saper programmare significa aver sviluppato competenze diverse, che richiedono la capacità di ragionare e operare contemporanemente
  - a livello pratico e teorico,
  - in senso astratto e concreto,
  - con una capacità di visione generale e particolare,
- Alcuni spunti dalla letteratura sulla didattica dell'informatica