Programmazione

Loriano Storchi

loriano@storchi.org

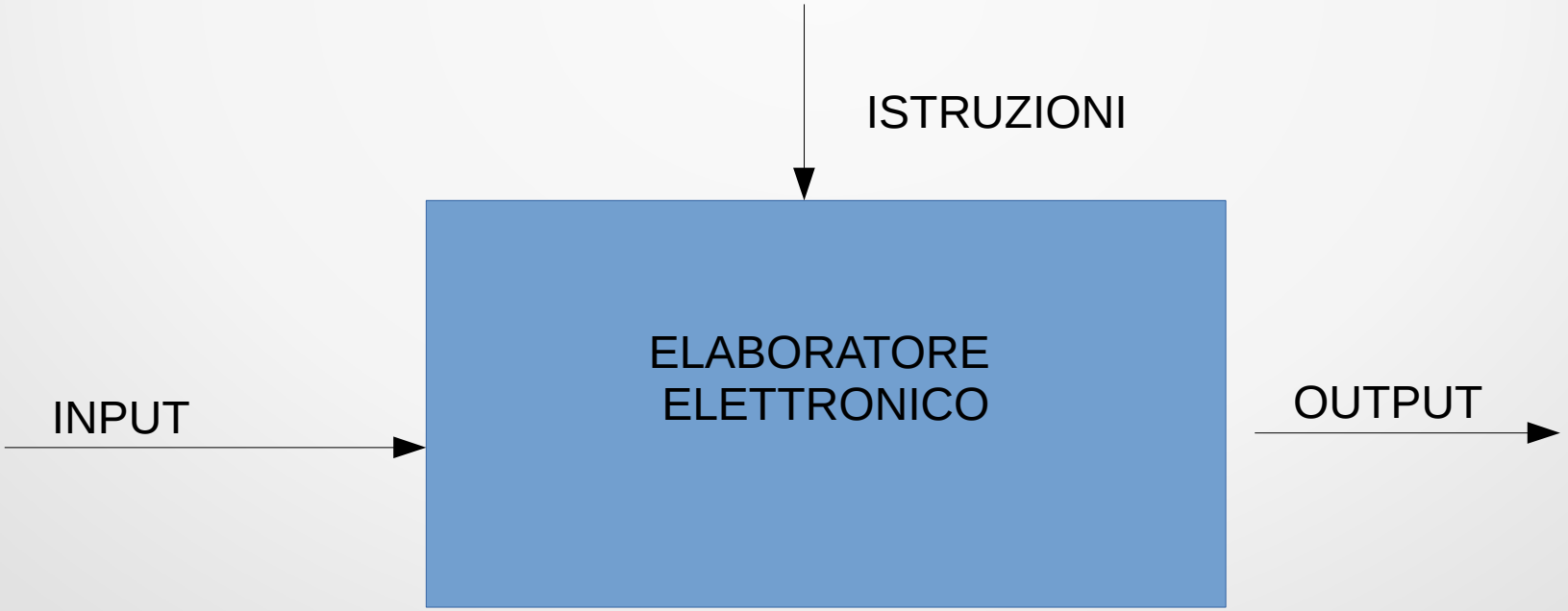
http::://www.storchi.org/

Algoritmi

- Gli algoritmi descrivono il modo in cui trasformare l'informazione. L'informatica si occupa della loro teoria, analisi, progettazione, della loro efficienza realizzazione ed applicazione.
- Un algoritmo è un procedimento formale che risolve un determinato problema attraverso un numero finito di passi. Il termine deriva dalla trascrizione latina del nome del matematico persiano al-Khwarizmi, che è considerato uno dei primi autori ad aver fatto riferimento a questo concetto. L'algoritmo è un concetto fondamentale dell'informatica, anzitutto perché è alla base della nozione teorica di calcolabilità: un problema è calcolabile quando è risolvibile mediante un algoritmo. (Wikipedia)

Programmazione

- Identifica l'attivita' mediante la quale si “istruisce” un calcolatore ed eseguire un particolare insieme di azioni, che agiscono su dati di ingresso (input), allo scopo di risolvere un problema e dunque produrre opportuni dati di uscita (output). Implementazione di un dato algoritmo.



Linguaggi e livelli

Il livello L0 rappresenta il computer reale ed il linguaggio macchina che esso è in grado di eseguire direttamente. Ogni livello superiori rappresenta una macchina astratta. I programmi (istruzioni) di ogni livello superiore devono essere o tradotti in termini di istruzioni di uno dei livelli inferiori, o interpretati da un programma che gira su di una macchina astratta di livello strettamente inferiore.

Linguaggio Macchina e ASSEMBLY

Ogni calcolatore e' in grado di interpretare un linguaggio di basso livello detto linguaggio macchina, le istruzioni (OPCODE) sono semplici sequenze di bit che il processore interpreta e seguendo una serie precisa di operazioni. Ogni istruzione a questo livello e' costituita da operazione estremamente basilari.

Per rendere piu' immediata la programmazioni il primo passo e' stato l'introduzione dell'ASSEMBLY, dove il codice binario e' sostituito da istruzioni mnemoniche umanamente piu' facilmente utilizzabili.

Linguaggio Macchina:

```
00000000000000000000001000000
00000000001000000000001000100
00000010000000010000000000000000
00000001000000000000000000111100
```

PSEUDO-ASSEMBLY:

```
z : INT;
x : INT 8;
y : INT 38;
LOAD R0,x;
LOAD R1,y;
ADD R0,R1;
STORE R0,z;
```

LINGUAGGI

- Oggi sono presenti numerosi linguaggi di programmazione. In generale ogni linguaggio risulta piu' o meno adaguato ad uno scopo specifico.
- **Linguaggi naturali**: sono dato spontaneamente sono estremamnte espressivi ma ambigui: **la vecchia porta la sbarra**
- **Linguaggi artificiali**: sono linguaggi che hanno una data precisa di nascita ed una lista di autori. I linguaggi artificiali possono essere **formali** e non-formali.
- **Linguaggi formali**: non ambigui costituiti da un insieme finito di stringhe costruite a partire da un alfabeto finito. E' un linguaggio per cui la forma delle frasi (**sintassi**) ed il significato dell stesse (**semantica**) sono definite in modo preciso non ambiguo. E' dunque possibile definire una procedura algoritmica in grado di verificare la correttezza **grammaticale** delle frasi.

LINGUAGGI

- Per definire un linguaggio rigorosamente occorrono alcuni strumenti di base:
  - **Alfabeto**: insieme dei simboli di base necessari a costituire le parole
  - **Lessico**: insieme delle regole necessarie a acrivere le parole di un linguaggio (vocabolario)
  - **Sintassi** (regole **grammaticali**): insieme di regole che permettono di stabilire se una frase (insieme di parole) e' corretta
  - **Semantica**: definisce il significato di una “frase” sintatticamente corretta ad esempio: **int a[5];** in linguaggio C permette di riservare spazio in memoria necessario a contenere 5 interi

LINGUAGGI

- Abbiamo gia' accennato a Linguaggi artificiali, Linguaggio macchina e Linguaggio di basso livello (ASSEMBLY)
- Linguaggi di alto livello si allontanano dalla logica del processore e sono costruiti per essere semplici, efficienti e leggibili, oltre che indipendenti dalla macchina
- Sono ad oggi presenti tantissimi linguaggi di programmazione, anche se quelli effettivamente utilizzati sono una decina.
- Di seguito riporteremo una classificazione sommaria di tali linguaggi (C, C++, C#, JAVA, PYTHON, FORTRAN, PASCAL, BASIC, Objectice-C e tantissimi altri). E' chiaro che ogni paradigma di programmazione e' piu' o meno adatto ad uno scopo piu' o meno specifico.

CLASSIFICAZIONE DEI LINGUAGGI

LINGUAGGI IMPERATIVI

- La componente fondamentale del programma e' l'istruzione, ed ogni istruzione indica l'operazione che deve essere eseguita. Le singole istruzioni che operato su i dati del programma.
- Le istruzioni vengono eseguite una dopo l'altra
- Ogni programma e' costituito da due parti fondamentali la dichiarazione dei dati e l'algoritmo inteso come sequenza di operazioni
- Ogni istruzione e' un ordine (programmazione dichiarativa il programma e' una serie di affermazioni)
- Di fatto da un punto di vista sintattico molti linguaggi imperativi utilizzano appunto verbi all'imperativo (i.e. PRINT, READ, ….)

LINGUGGI IMPERATIVI

```
READ *, A, B
C = A + B
PRINT C
```

Quindi una serie di istruzioni leggi A e B, calcola C come somma di A piu’ B ed infine stampa il risultato

PROGRAMMAZIONE PROCEDURALE

- Possiamo considerarlo un sotto-paradigma della programmazione imperativa.
- Viene introdotto il concetto di sotto programma (subroutine) o funzioni.
- Quindi si introduce la possibilita' di creare porzioni di codice sorgente utili ad eseguire funzioni specifiche.
- Questi sottoprogrammi possono ricevere parametri di input e restituire valori di output.

PROGRAMMAZIONE PROCEDURALE

```
SUB EXSUMMA (A, B, C)
    C = A + B
END SUB

FUNCTION SUM (A, B)
    C = A + B
    RETURN C
END FUNCTION
```

```
MAIN
   READ A, B
   PRINT SUM (A,B)
   EXSUM (A,B,C)
   PRINT C
```

Esempio di utilizzo di una subroutine e di una funzione per il calcolo della somma (riusabilita' del codice, librerie di funzioni)

PROGRAMMAZIONE STRUTTURATA

- Possiamo considerarlo un sotto-paradigma della programmazione imperativa.
- In pratica il programmatore e' vincolato ad usare olo strutture di controllo canoniche che non includono le istruzioni di salto incondizionato (GOTO). Dunque la sintassi del linguaggio impedisce l’uso di strutture che non seguono certi vincoli. (non solo)
- L'uso dell'istruzione GOTO porta inevitabilmente ad una scarsa leggibilita' del codice (spaghetti-code)

PROGRAMMAZIONE STRUTTURATA

- Esempio da wikipedia

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

```
function square(i)
    square = i * i
end function
dim i
for i = 1 to 10
    print i & " al quadrato = " & square(i)
next
print "Programma terminato."
```

PROGRAMMAZIONE ORIENTATA AGLI OGGETTI

- Possiamo considerarlo un sotto-paradigma della programmazione imperativa.
- Tale paradigma di programmazione permette di definire **Oggetti** Software in grado di interagire gli uni con gli altri.
- L'organizzazione del software sotto forma di oggetti permette un piu' facile riuso dello stesso. Una migliore organizzazione di progetti di grandi dimensioni.
- I Linguaggi OOP prevedono il raggruppamento di parte del codice sorgente in classi, ogni classe comprende dati e metodi (funzioni) che operano sui dati stessi. Le classi sono dei modelli astratti che al momento dell'esecuzione vengono invocate per creare od istanziare oggetti software.
- Un linguaggio orientato agli oggetti permette di implementare tre meccanismi di base utilizzando la sintassi nativa del linguaggio : **incapsulamento, polimorfismo, ereditarieta'**.

PROGRAMMAZIONE ORIENTATA AGLI OGGETTI

ESEMPIO OGGETTI

CLASSE QUADRATO
  ATTRIBUTI
    LATO
    COLORE
  METODI
    REAL OTTIENI_AREA()
    SET_LATO (VAL)

QUADRATO 1
  LATO = 1.0
  COLORE = VERDE

QUADRATO 2
  LATO = 1.5
  COLORE = GIALLO

PROGRAMMAZIONE ORIENTATA AGLI OGGETTI

- **Incapsulamento:** separazione precisa fra implementazione ed interfaccia della classe. Chi usa la classe (oggetto) non deve conoscere il dettaglio implementativo. Utilizza la classe mediante i metodi ed i dati pubblici interagendo con l'oggetto senza appunto consocere il dettaglio dell'implementazione

MAIN

TRIANGOLO T1

T1.COLORE = GRIGIO

T1.SET_LATO(2.0)

PRINT T1.CALCOLA_AREA ()

PROGRAMMAZIONE ORIENTATA AGLI OGGETTI

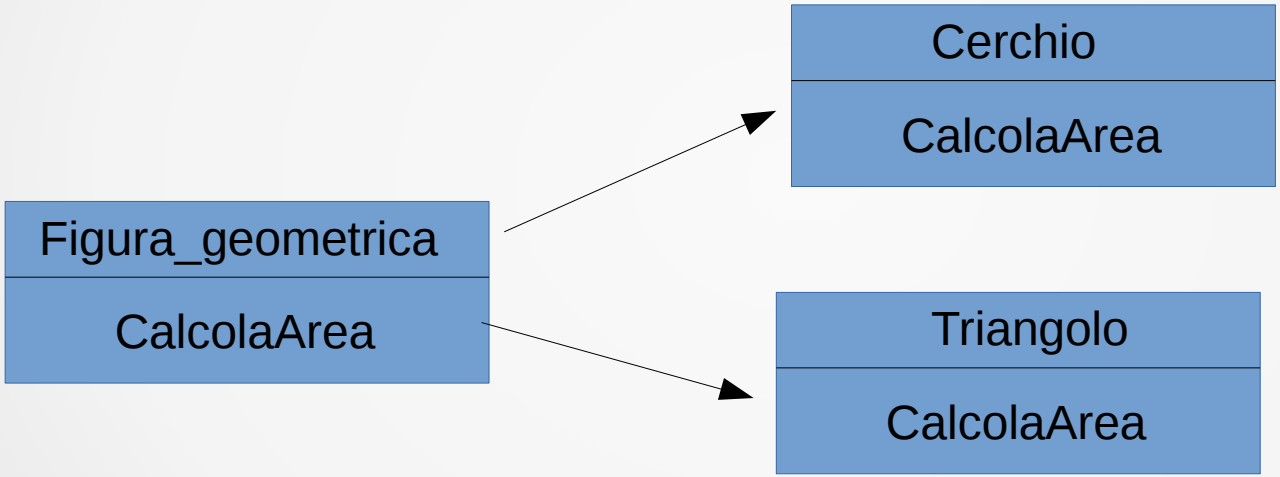
- **Ereditarieta**': una classe puo' ereditare da una classe base ed evolverne o specializzarne le funzionalita'.
- Ad esempio posso immaginare una classe base figura_geometriche da cui classi come trinagolo, cerchio, quadrato … derivano.
- Le classi che derivano da una classe base ereditano tutti i metodi e le proprieta' della classe base, puo' pero' specializzarsi definendo metodi e dati propri.
- Se ad esempio B e' una sottoclasse (oppure piu' in generale un sottotipo) di A, ogni programma/funzione che puo' usare A puo' usare anche B

PROGRAMMAZIONE ORIENTATA AGLI OGGETTI

- **Polimorfismo**: possiamo cercare di esemplificare questo concetto dicendo: molteplici definizione della stessa funzione (overloading), classi e funzioni parametriche rispetto al tipo di dato. Esempio semplice overloading di funzioni.
- Possiamo formalmente distinguere in almeno quattro tipi di polimorfismo: per inclusione, parametrico, overloading, coercion.
- Noi faremo solamente qualche semplice esempio utile a chiarire il concetto generale.

PROGRAMMAZIONE ORIENTATA AGLI OGGETTI

- Immaginiamo la solita classe figura_geometriche da cui avremo fatto derivare due classi cerchio e triangolo



- Quando l'utente richiama il metodo calcola area questo eseguira' una determinata azione (calcolo dell'area nei due casi pur avendo lo stesso nome

PROGRAMMAZIONE FUNZIONALE

- Come dice il nome stesso il flusso di esecuzione assume la forma di valutazione di una serie di valutazioni di funzioni matematiche. Il programma e' quindi un'insieme di funzioni
- Nei linguaggi funzionali puri non esiste il concetto di assegnazione, o di allocazione esplicita della memoria.
- I valori non si trovano cambiando lo stato del programma, non esiste appunto l'assegnazione di valore, ma costruendo il nuovo stato mediante funzioni a partire dallo stato precedente.
- Usanti nell'ambito dell' AI (poco usanti o assenti in ambito industriale)
- Le funzioni possono essere passate come parametri e ritornate come “risultato” da altre funzioni.

LINGUAGGI DICHIARATIVI (O LOGICI)

- Rispetto al paradigma imperativo il programma consiste di una serie di affermazioni e non di ordini.
- Nel programma si specifica il COSA si voglia ottenere non il COME. Il come e' lasciato all'esecutore
- In pratica il programma (o la sua esecuzione) si puo' considerare come la dimostrazione della verita' di un'affermazione.

LINGUAGGI

- Si possono poi classificare i linguaggi anche secondo il tipo di dato **tipizzazione statica** e **tipizzazione dinamica**.
- **Tipizzazione statica :** il programmatore e' costretto a specificare esplicitamente il tipo di ogni elemento sintattico Ad esempio deve specificare il tipo di una variabile ed il linguaggio poi garantira' che quella variabile verra' usata coerentemente con la dichiarazione.
- **Tipizzazione dinamica** : ad esempio in questo caso i dati assumeranno un tipo che varia a runtime in funzione di assegnazioni fatte (vedi Python)
- Possiamo poi distinguere anche fra tipizzazione **debole e forte**

LINGUAGGI

- **Linguaggi o paradigmi di programmazione parallela** (per le moderne architetture) se siete cuirosi potete vedere qui ad esempio http://www.storchi.org/lecturenotes/acr/index.html
- **Linguaggi esoterici** : sono linguaggi voltamente complessi e poco chiari. Popolari solo fra gli utenti piu' abili ed usati al solo scopo di mettere alla prova le capacita' di programmazione (scopo essenzialmente ludico)
- **Scripting**: nati inizialmente per essere usati nelle shell Unix. Sono linguaggi usati per automatizzare compiti ripetitivi e lunghi.

Programmazione altri concetti

- Il sorgente viene scritto in file di testo ASCII.Il sorgente esprime l'algoritmo implementato nel linguaggio scelto. Per scrivere il sorgente si possono usare semplici editor di testo (VI, Emacs). Oppure IDE ambineti di sviluppo integrato con altri strumenti, come compilatori , linker e debugger.
- **Compilazione**: il sorgente viene tradotto (dal compilatore) da linguaggio ad alto livello a codice eseguibile. Il vantaggio e' che l'esecuzione e' “veloce” e che il codice viene ottimizzato per la piattaforma specifica. Lo svantaggio e' che si dovra' ri-compilare per ogni diverso sistema operativo o hardware.
- **Linking**: ogni programma generalmente fa uso di una o piu' librerie ed il linker collega assieme librerie e programma di partenza. I linking puo' essere sia statico che dinamico (ad esempio librerie .so in Linux/Unix-like o .dll in windows)

Programmazione altri concetti

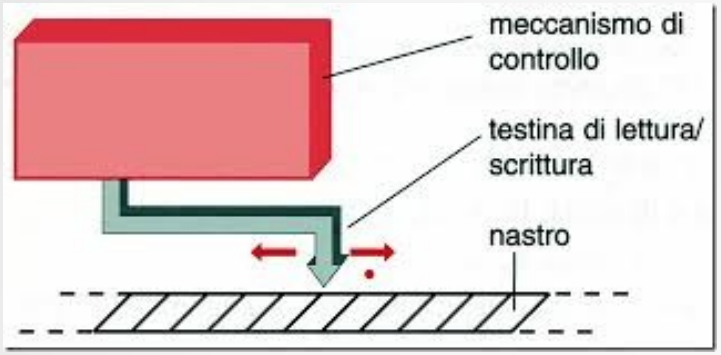
- **Interpretazione**: onde evitare il problema della portabilita' dei programmi si e' ricorso al concetto di interpretate In questo caso il codice sorgente non viene compilato “tradotto” ma eseguito appunto da un'interprete. Questo introduce altri problemi come quello delle prestazioni.
- **Bytecode, P-code**: possiamo definirlo come un approccio intermedio in cui il programma sorgente viene “tradotto” in un codice intermedio che viene interpretato da una macchina virtuale. Questo permette di unire due vantaggi una buona velocita' di esecuzione assieme ad una estrema portabilita' del programma. JIT (Just in Time) al momento dell'esecuzione compilano il codice intermedio in codice macchina.

Programmazione altri concetti

- **Calcolabilita' :** Data una funzione essa e’ detta calcolabile se possiamo trovare un algoritmo (quindi una procedura che esegue meccanicamente un numero finito di passi) che la calcola.
- Tesi di Church-Turing, la classe delle funzioni calcolabili coindicide con la classe delle funzioni calcolabili da una macchina di Turing.
- Tutte le macchine di calcolo (computer) possono essere ricondotte ad una macchina di Turing.

Programmazione altri concetti

- **MdT: macchina di Turing**



Modello deterministico con nastro e istruzioni a 5 campi:

1 - Un nastro lungo a piacere che puo' contenere caratteri oppure spazi vuoti
2 - testina/dispositivo di lettura e e scrittura con cui appunto leggere e scrivere sul nastro ed ovviamente la testina puo' muovere il nastro a desta o sinistra
3 - La macchina ha uno stato interno

La MdT ad ogni passo legge un simbolo ed ed in funzione del suo stato interno puo' cambiare stato e poi scrivere un simbolo nel nastro e poi muovere il nastro a destra o sinitra.
Il comportamento della MdT e' programmato definendo delle regole o quadruple del tipo:

(stato interno, simbolo-letto, nuovo-stato, simbolo-scritto/direzione)

Programmazione altri concetti

- **MdT e problema della terminazione:** dato un certo programma e dato un certo input e' impossibile stabilire se tale programma terminera' o meno. (questo problema e' fortemente legato al teorema della incompletezza di Gödel)

Programmazione altri concetti

- **Complessita' algoritmica :** e' la misura' della difficolta' di un calcolo (algoritmo + input)
- La bonta' di un algoritmo si valuta in funzione del tempo e dello spazio necessario alla sua esecuzione, in genrale quindi in funzione delle risorse richieste.
- Chiaramente il tempo di esecuzione e' funzione del tipo di input oltre che del tipo di hardware utilizzato, non ha quindi senso classificare gli algoritmi in funzione del numero di secondo richiesti alla sua esecuzione.
- Il tempo di calcolo si esprime dunque come il numero di operazioni elementari in funzione della dimensione N dei dati di input

Programmazione altri concetti

- Esempio calcolo dell'efficienza di un algoritmi in cui cerchiamo il minimo **m** all'interno di un insieme di **N** numeri {x1, x2, …, xN}
- Immaginiamo di affrontare il problema come segue:
  - Scelgo x1 come possibile minimo
  - lo confronto con x2, poi x3 e cosi' via
  - Se trovo un xi piu' piccolo continuo i confronti con quello cosi' come fatto con x1
  - Al termine avro' trovato il minimo
- Per fare tutto avro' fatto N confronti quindi l'efficienza dell'algoritmo e' direttamente proporzionale alle dimensioni N dell'input

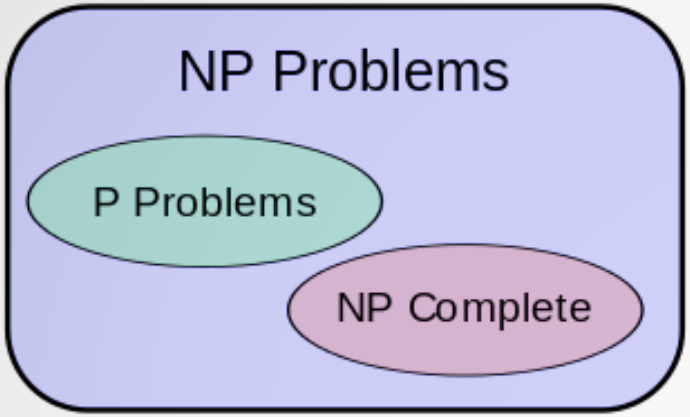
Programmazione altri concetti

- L’efficienza di un dato algoritmo e’ quindi esprimbile come una funzione f(N), quindi funzione della variabile N che rappresenta la dimensione dei dati di input.
  - Tale funzione esprime dunque il numero di operazioni elementari necessarie per risolvere il problema mediante l’algoritmo dato in funzione della dimensione dell’input
  - Rappresenta quindi la complessita’ computazione
  - Dato N un algoritmo A e’ piu’ efficiente di un altro B se al crescere di N $f_A(N)$ s’ minore o uguale ad $f_B(N)$
- Il tempo di esecuzione di un programma quindi dipende dalla complessita’ dell’algoritmo, dalla dimensione di N ed ovviamente dalla “ velocita’ ” della macchina sul quale e’ eseguito

Programmazione altri concetti

- Per suddividere gli algoritmi in classi di complessita' si usa il segunte criterio:
- Una funzione f(N) si dice che e' di ordine g(N) e si indica con **f(N) = O(g(N))** se esiste una costante K tale che , a meno che per un numero finito di valori di N sia sempre vera la seguente disuguaglianza: **f(N) ≤ K*g(N)**. Si puo' anche scrivere **f(n) / g(N) ≤ K**
- Ad esempio 2*N + 5 = O(N) infatti 2*N + 5 ≤ 7*N per ogni N maggiore di zero

Programmazione altri concetti



Classi di complessita’ P e NP , se P sia uguale ad NP o meno ad oggi e’ un problema ancora aperto (problema da un milione di dollari Clay Math Institute)

Classe P problemi risolvibili in una macchina di Turing deterministica in tempi polinomiali.

Classe NP problemi per i quali non e’ noto un algoritmo con complessita’ polinomiale. Ma sono verificabili invece velocemente

Problemi NP-completi il modo piu’ semplice di descriverla se uno dei problemi NP-completi e’ facile allora lo sono tutti visto che posso “convertire” la risoluzione di uno nell’altro. Allo stesso modo se uno e’ difficile sono tutti difficili.

ESEMPIO: la fattorizzazione in numeri primi di un numero intero problema NP (molto probabilmente non e’ NP-completo ancora non possiamo dirlo) dato c trovare i fattori primo a e b tali che a * b = n

Programming, control structures

Loriano Storchi

loriano@storchi.org

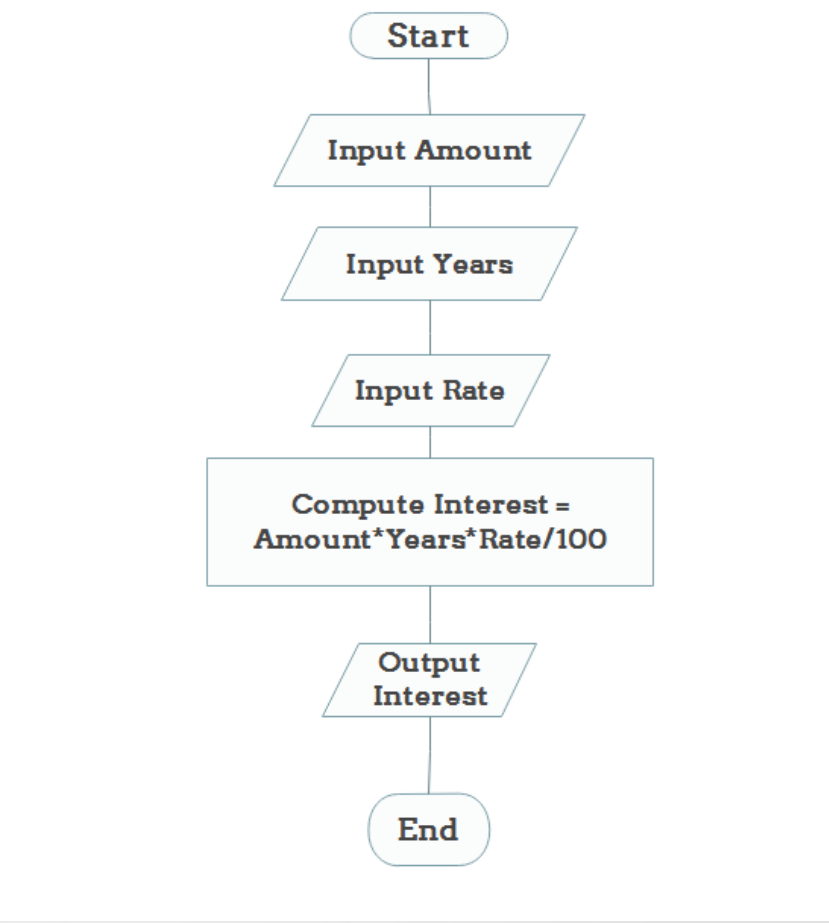
http::://www.storchi.org/

FLOWCHART AND PSEUDOCODE

FLOWCHART AND PSEUDOCODE

- Esamineremo solo alcuni esempi di base: calcolare l'interesse di un deposito bancario

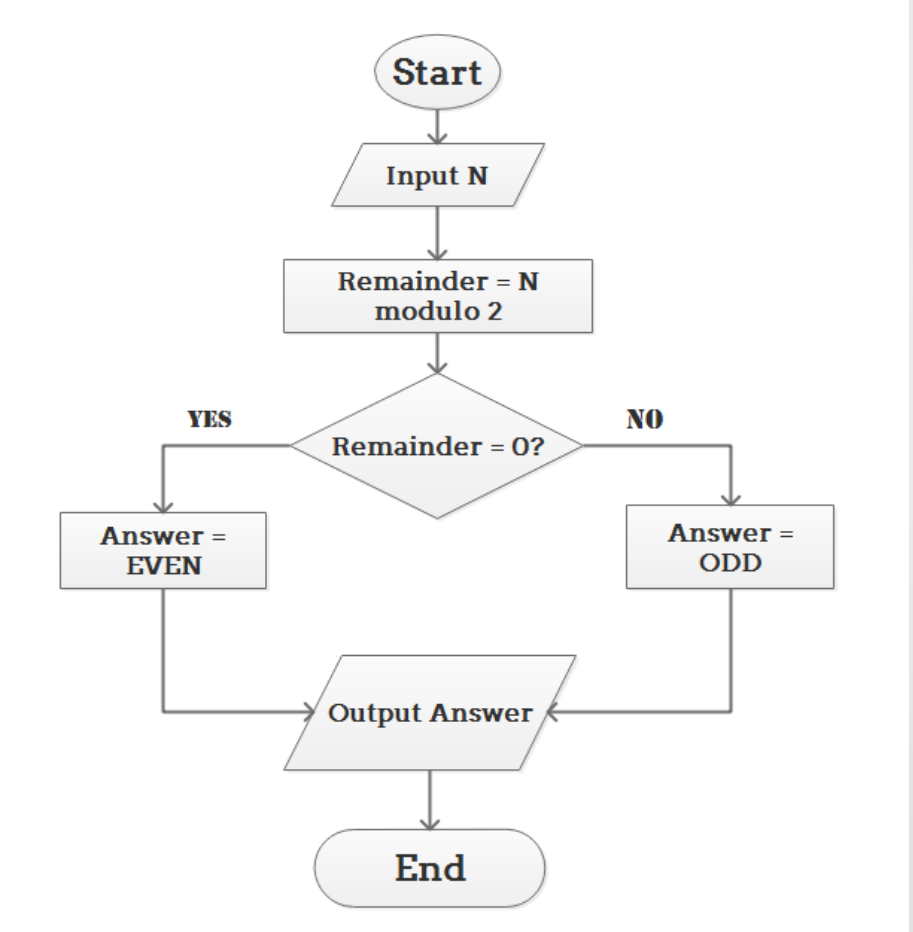
Step 1: Read amount,
Step 2: Read years,
Step 3: Read rate,
Step 4: Calculate the interest with formula
"Interest=Amount*Years*Rate/100
Step 5: Print interest,



FLOWCHART AND PSEUDOCODE

- Determina e mostra se il numero N è pari o dispari

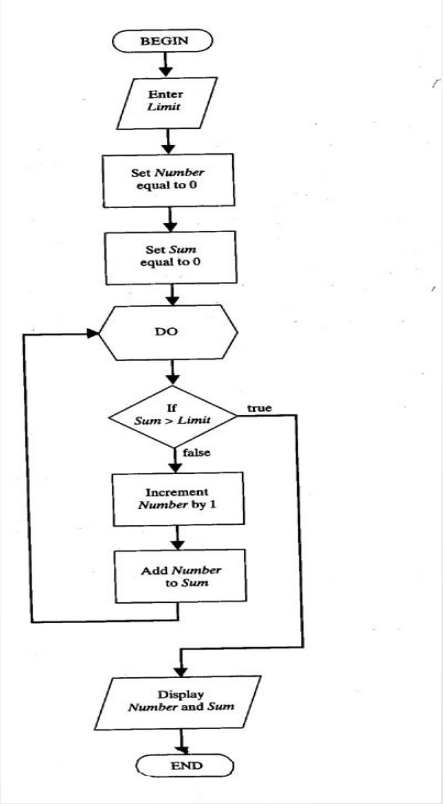
Step 1: Read number N,
Step 2: Set remainder as N modulo 2,
Step 3: If remainder is equal to 0 then number N is even, else number N is odd,
Step 4: Print output.



FLOWCHART AND PSEUDOCODE

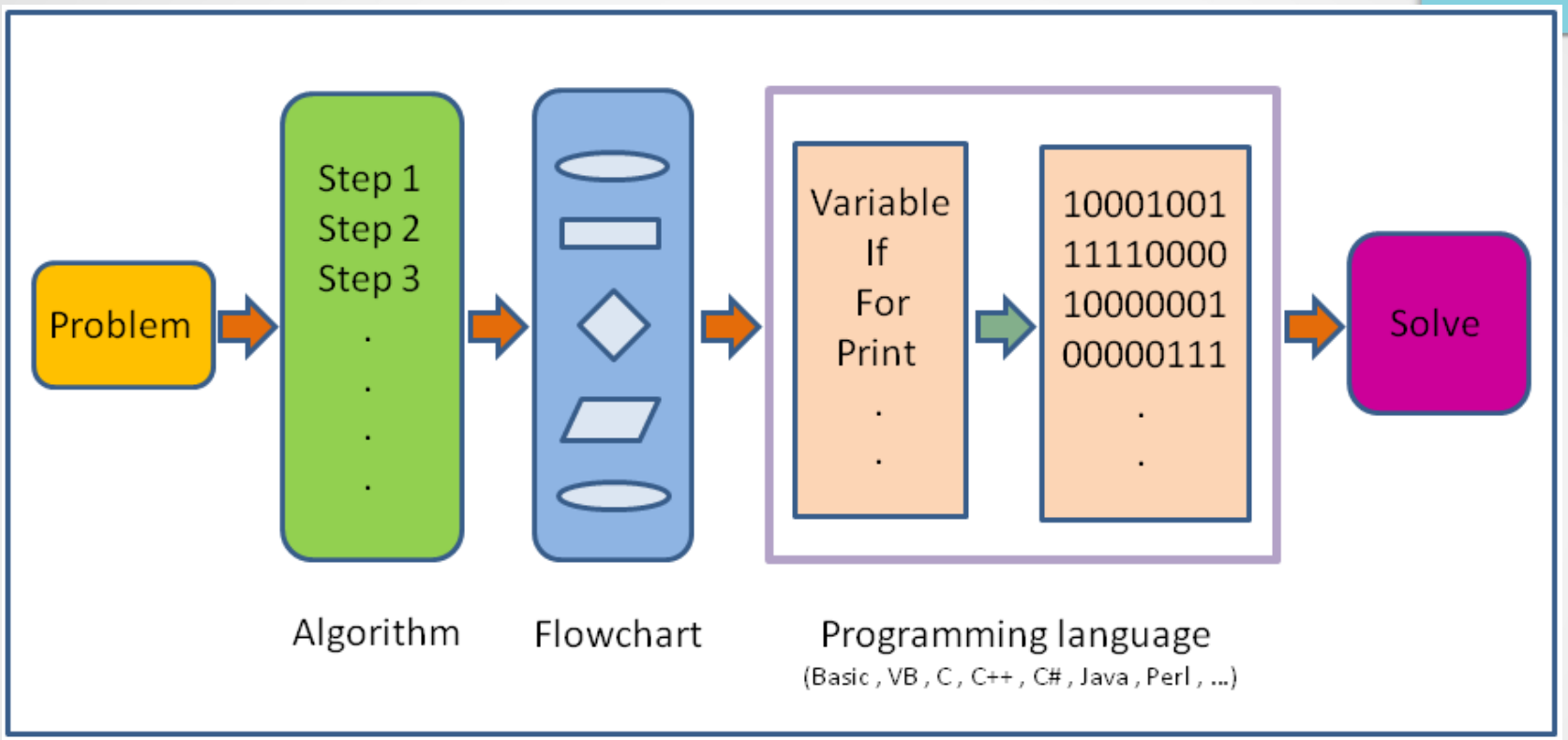
- Per un dato valore, **Limite**, qual è il numero intero positivo più piccolo per cui la somma **Somma = 1 + 2 + ... + Numero** è maggiore di Limite. Qual è il valore di questa somma?

1. Enter Limit
2. Set Number = 0.
3. Set Sum = 0.
4. Repeat the following:
   a. If Sum > Limit, terminate the repitition, otherwise.
   b. Increment Number by one.
   c. Add Number to Sum and set equal to Sum.
5. Print Number and Sum.



STRUTTURE DI CONTROLLO

CONTROL STRUCTURES



Sono tre le strutture fondamentali che vengono utilizzate per la risoluzione algoritmica dei problemi: selezione, iterazioni e sequenza (sequenza di istruzioni) (il GOTO presente nei linguaggi macchina dagli anni '70 è stato progressivamente scoraggiato / eliminato)

Examples: https://bitbucket.org/lstorchi/teaching
https://github.com/lstorchi/teaching

SEQUENZE

Sequences

- **La struttura di controllo della sequenza** si riferisce all'esecuzione riga per riga mediante la quale le istruzioni vengono eseguite in sequenza, nello stesso ordine in cui appaiono nel programma. Potrebbero, ad esempio, eseguire una serie di operazioni di lettura o scrittura, operazioni aritmetiche o assegnazioni a variabili.

Step 1: Read amount,
Step 2: Read years,
Step 3: Read rate,
Step 4: Calculate the interest with formula
"Interest=Amount*Years*Rate/100
Step 5: Print interest,

SCELTE

Selections

- La struttura generale di un'istruzione di **selezione** è la seguente:

if (condition 1)

statements 1

else if (condition 2)

statements 2

...

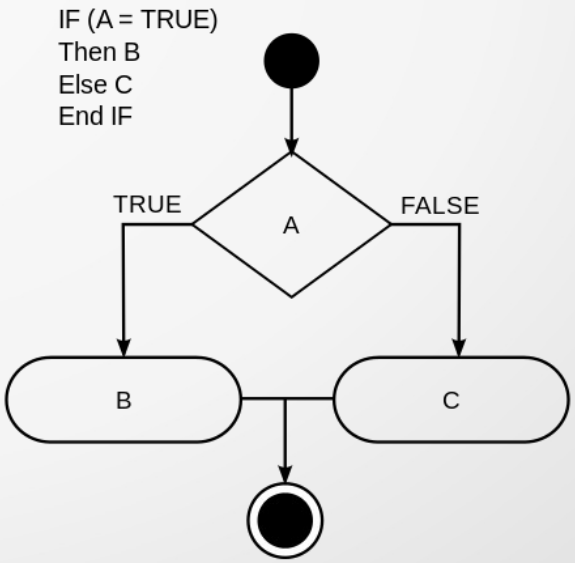
else

statements N

endif

Ci POSSONO ESSERE MOLTI else if

Non devono necessariamente mostrare tutti e tre gli elementi, IF, ELSE IF e ELSE, posso anche avere un solo IF



Selections

- **Annidamento**,posso ovviamente avere if..then.else annidati:

```
if (condition 1)
  statement 1
  if (condituion 2)
    statements 2
  end if
else
  statements 3
end if
```

Operators

- In tutti i linguaggi di programmazione, posso utilizzare operatori di relazione per confrontare numeri e variabili, ad esempio:

Description	Java, C, C++	Fortran
Greater than	>	.GT.
Greater than or equal	>=	.GE.
Less than	<	.LT.
Less than or equal	<=	.LE.
Equal	==	.EQ.
Not Equal	!=	.NE.

```
Int a;

a = 4 // operatori di  assegnazione

If  (a == 5)
{
   cout << “Hello” << std::endl;
}
else if (a > 5)
{
    cout <<  a << “ > 5 “ << std::endl;
}
```

Logical Operators

- Dò per scontati gli operatori logici AND, OR e NOT

Logical Operators		
Operator	Description	Example
&&	AND	x=6 y=3 x<10 && y>1 Return True
||	OR	x=6 y=3 x==5 || y==5 Return False
!	NOT	x=6 y=3 !(x==y) Return True

Esempio la seguente disuguaglianza

5 < a < 7

Nei linguaggi di programmazione è suddiviso in due espressioni elementari collegate dall'operatore AND:

(a > 5) AND (a < 7)

Bitwise operations

- Non confondere il precedente con le operazioni bit per bit
- Queste sono le operazioni che servono per manipolare dati bit per bit, da non confondere con quanto visto nella diapositiva precedente
  - & (AND)
  - | (OR)
  - ^ (XOR I.e. 1 XOR 1 is zero)
  - ~ (ones' complement i.e. 0 to 1 and 1 to 0)
  - >> (right shift 11100101 >> 1 is 01110010)
  - << (left shift)

Bitwise operations

- Un semplice esempio:

```
└─ $ python3
Python 3.6.9 (default, A
[GCC 8.4.0] on linux
Type "help", "copyright"
>>> a = 60
>>> b = 13
>>> print(a&b)
12
>>> exit()
```

```
a = 60
b = 13

print(a&b)
```

```
12
```

```
a = 0011 1100

b = 0000 1101

---------------

a&b = 0000 1100
```

Case structure

- Fondamentalmente una serie di if-then-else con qualche vincolo. In pratica, la scelta tra i blocchi di istruzione è guidata dal valore di una certa variabile:

```
switch variable:
   case val1:
      statements 1
   case val2:
      statements 2
   …
   default:
     default statements
```

```
If (variable == val1)
{
    Statements 1
}
else if (variables == val2)
{
   Statements 2
}
…
else
{
  Default statements
}
```

Example

```
int i;
i = 4;
switch (i)
{
  case 1:
    printf("vale uno \n");
    break;
  case 2:
    printf("vale due \n");
    break;
  default:
    printf("non uno non due\n");
    break;
}
```

LOOPS

Loops

- Ci sono tre diversi tipi: while..do, do.. while e for
- Non tutti I linguaggi hanno necessariamente tutte e tre queste strutture
- Queste strutture consentono di ripetere un blocco di istruzioni finché non si verifica una condizione
- Anche in questo caso è possibile annidare più loop uno dentro l'altro

While..do e Do..while

- Il blocco di istruzioni inoltre non può mai essere eseguito, poiché la condizione è verificata all'inizio, finché la condizione è vera, il blocco di istruzioni viene eseguito

  WHILE (condition)

  statements

  END DO

- do..while esegue invece il blocco di istruzioni finché la condizione non è falsa

  DO

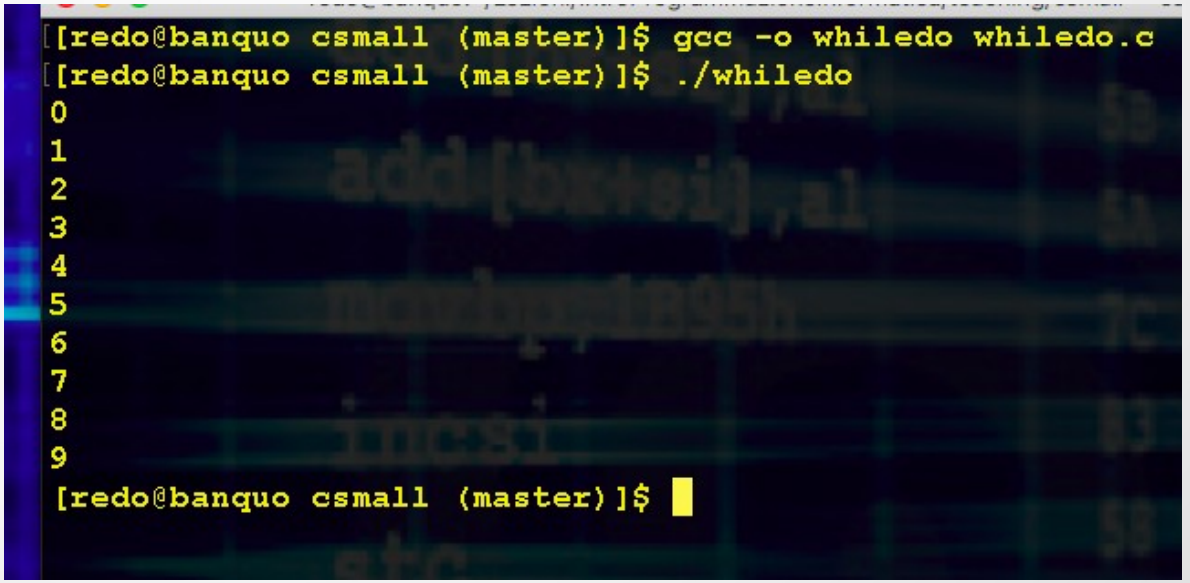
  statements

  WHILE (condition)

Example

- A simple C example:

```
int i = 0, N = 10;
while (i != N)
{
    printf("%d \n", i);
    i++;
}
```



Example

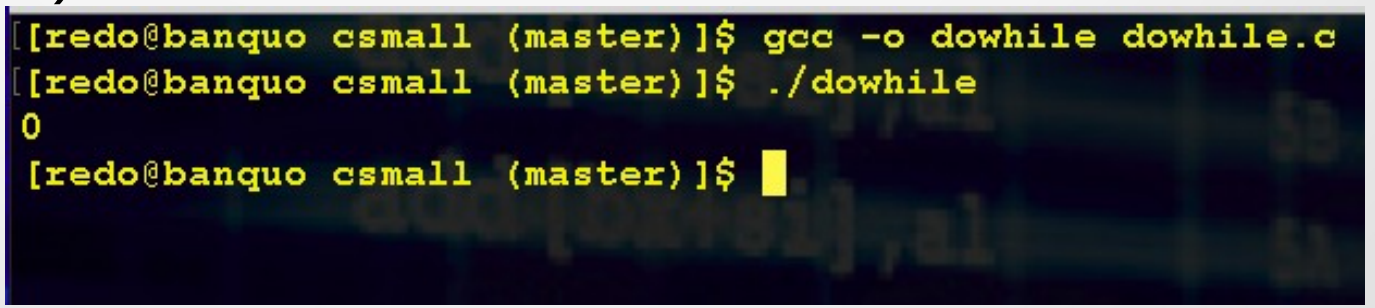
```
Type "help", "copyright"
>>> while True:
...     print("here")
...
```

```
while True:
    print("here")
```

```
here
here
here
here
here
here
here
here
here
here
here
```

Example

```
Int i = 0, N = 10;


do
{
    printf (“%d \n”, i);
    i++; // i = i + 1
} while (i >= N);
```



For loop

- Esegue un blocco di istruzioni un numero di volte noto dall'inizio. Molti linguaggi di programmazione costringono il programmatore a utilizzare un contatore.
- Generalmente esiste un contatore che è una variabile intera il cui valore viene "modificato" passo dopo passo
- La condizione finale viene generalmente determinata confrontando la variabile contatore con un valore

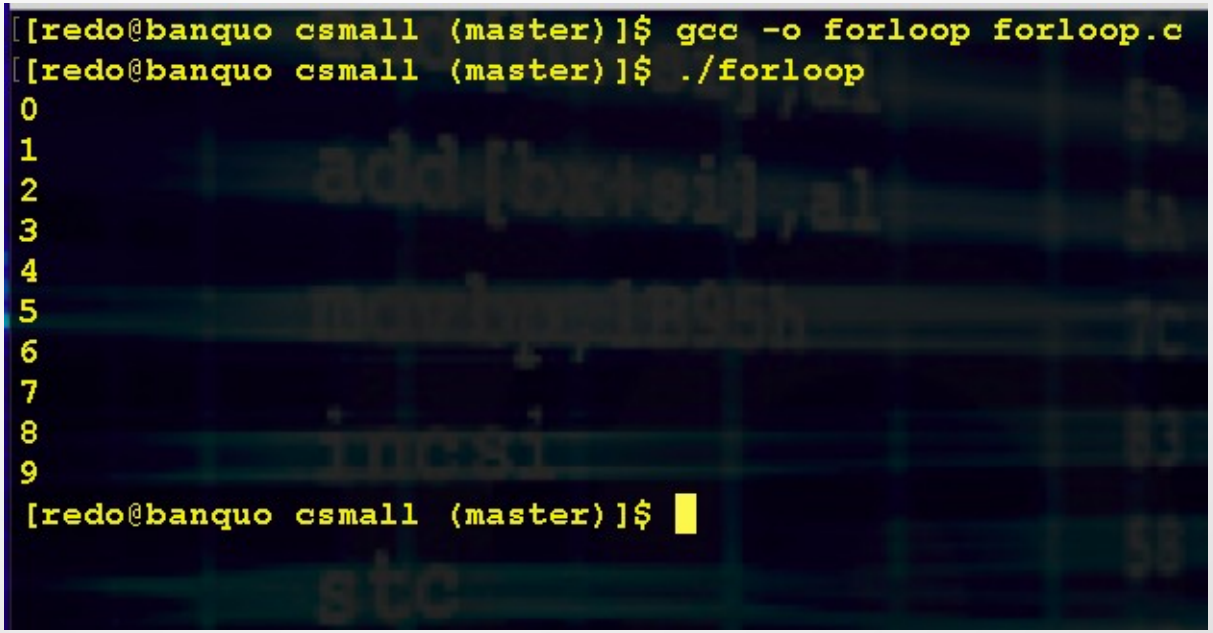
for (counter = startingvalue A endvalue STEP = stepvalue)

statements

end for

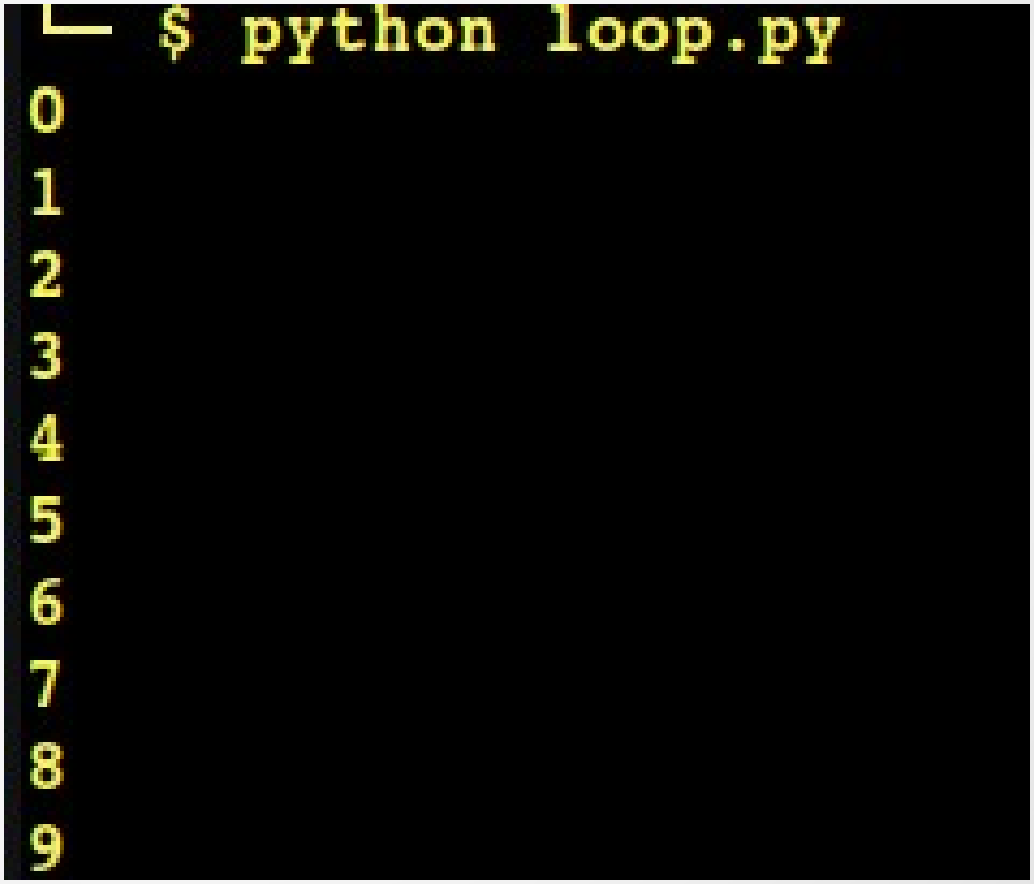
Example

```
int i;
for (i=0; i<10; ++i)
{
   printf (“%d \n”, i);
}
```



Example

```
for i in range(10):
    print(i)
```

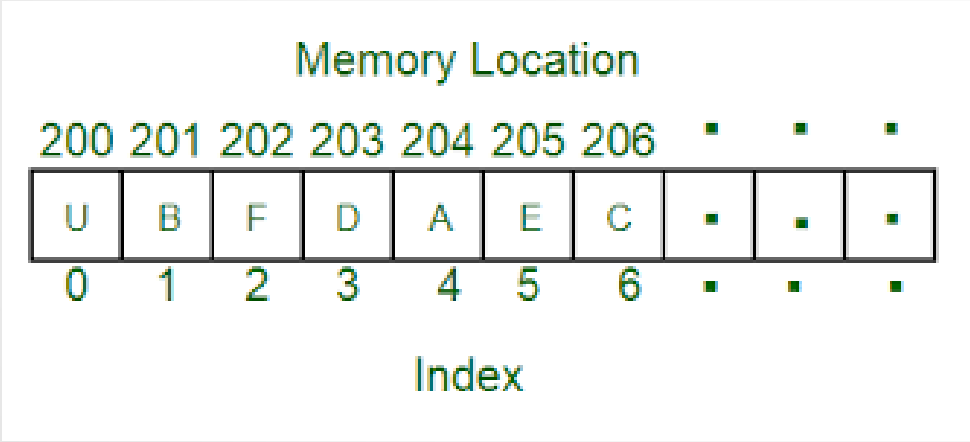


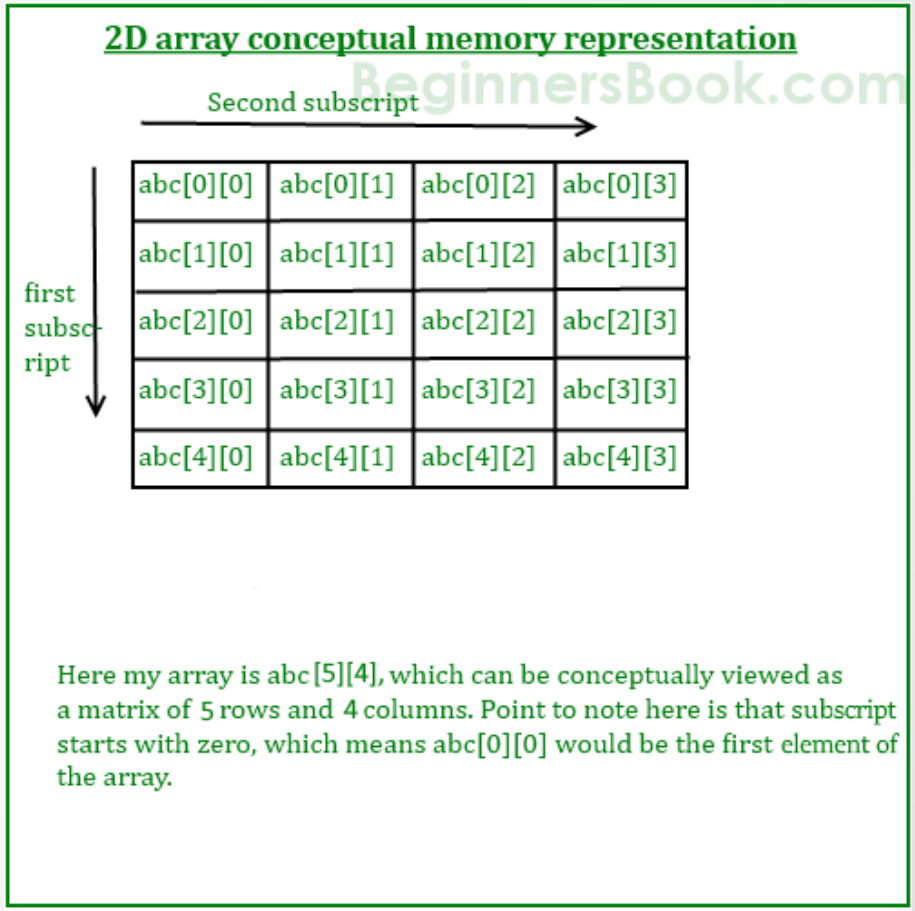
ARRAY

Array

- Come posso gestire facilmente le informazioni strutturate per natura? Ad esempio un numero complesso, o un vettore o una matrice?
- Le variabili strutturate tipiche sono gli array
- Ad esempio, un vettore di numeri in virgola mobile in C può essere dichiarato come: float v [10];
- Possiamo quindi accedere all'elemento i-esimo del vettore: v [i-1] = 3.5;
- Allo stesso modo posso definire una matrice (array bidimensionale) come: float m [10] [10];

Array

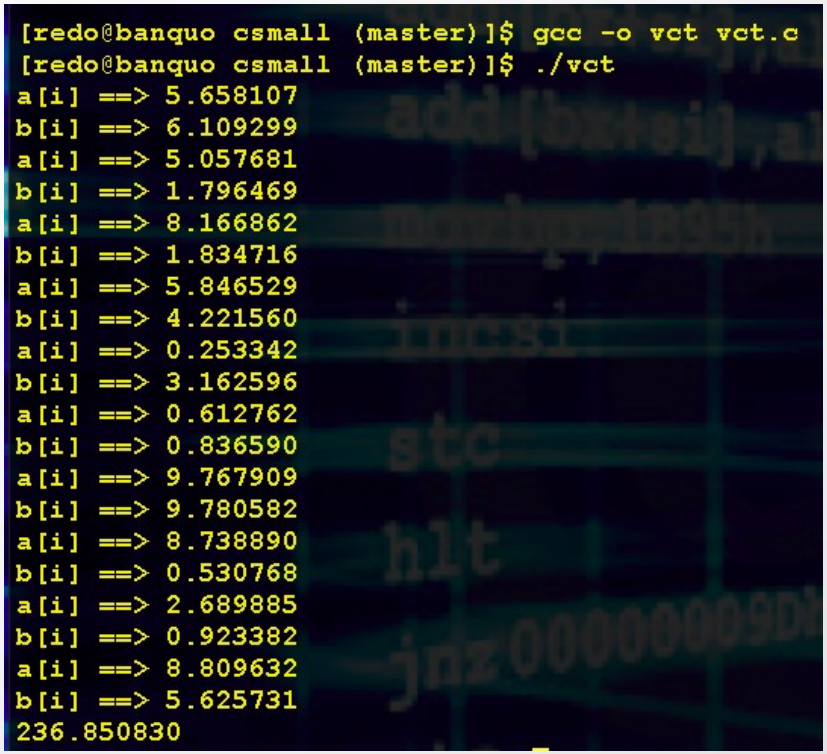

Memory Location
200 201 202 203 204 205 206
U B F D A E C
0 1 2 3 4 5 6
Index


2D array conceptual memory representation
BeginnersBook.com
Second subscript
first subscript
abc[0][0] abc[0][1] abc[0][2] abc[0][3]
abc[1][0] abc[1][1] abc[1][2] abc[1][3]
abc[2][0] abc[2][1] abc[2][2] abc[2][3]
abc[3][0] abc[3][1] abc[3][2] abc[3][3]
abc[4][0] abc[4][1] abc[4][2] abc[4][3]
Here my array is abc[5][4], which can be conceptually viewed as a matrix of 5 rows and 4 columns. Point to note here is that subscript starts with zero, which means abc[0][0] would be the first element of the array.

Example

- Loop di esempio per eseguire una moltiplicazione scalare tra i vettori

```
float a[N], b[N];
for (i=0; i<N; ++i)
{
  a[i] = (float)rand()/(float)(RAND_MAX/N);
  b[i] = (float)rand()/(float)(RAND_MAX/N);
}
s = 0.0;
for (i=0; i<N; ++i)
{
    s = s + a[i]*b[i];
}
```



Example

```
import random

N = 20

a = []
b = []

for i in range(N):
    a.append(random.randrange(1, 30))
    b.append(random.randrange(1, 30))

for i in range(N):
    print("%3d %3d"%(a[i], b[i]))

s = 0.0
for i in range(10):
    s = s + float(a[i]*b[i])

print("")
print("S = %.5e"%s)
```

Programming, control structures

Loriano Storchi

loriano@storchi.org

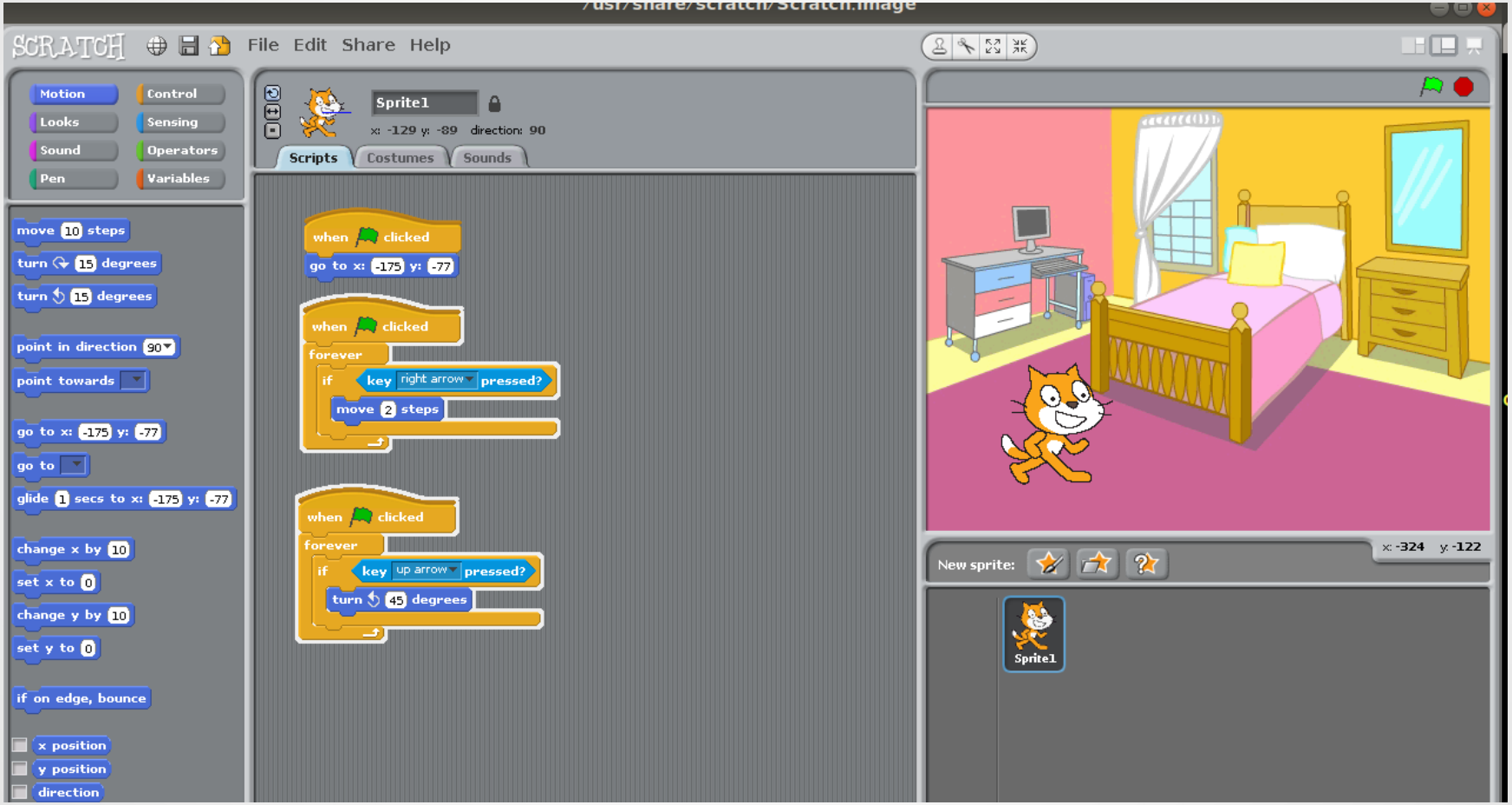
http::/www.storchi.org/

YOU CAN TRY USING SCRATCH

Scratch

- **Scratch is a block-based visual programming language and online community targeted primarily at children.**

Scratch


/usr/share/scratch/Scratch.image
SCRATCH
File Edit Share Help
Motion
Control
Looks
Sensing
Sound
Operators
Pen
Variables
move 10 steps
turn 15 degrees
turn 15 degrees
point in direction 90
point towards
go to x: -175 y: -77
go to
glide 1 secs to x: -175 y: -77
change x by 10
set x to 0
change y by 10
set y to 0
if on edge, bounce
x position
y position
direction
Sprite1
x: -129 y: -89 direction: 90
Scripts
Costumes
Sounds
when clicked
go to x: -175 y: -77
when clicked
forever
if key right arrow pressed?
move 2 steps
when clicked
forever
if key up arrow pressed?
turn 45 degrees
New sprite:
x: -324 y: -122
Sprite1

Python

Loriano Storchi

loriano@storchi.org

http::/www.storchi.org/

Programming Laguanges

- Abbiamo visto che i linguaggi di programmazione possono essere classificati come (non solo ma ...):
  - Dichiarativo
    - Logica
    - Funzionale
  - Imperativo
    - Procedurale
    - Orientato agli oggetti
- Python è imperativo e sia procedurale che orientato agli oggetti

Programming languages

- **Possiamo classificare i linguaggi di programmazione anche come tipizzazione dinamica o statica e tipizzazione forte e debole**
- Tipizzazione debole e dinamica, esempio utilizzando Perl:

```
[redo@banquo tipiz (master)]$ cat test.pl
$i = 2 + "3";
print $i, "\n";
[redo@banquo tipiz (master)]$ perl test.pl
5
[redo@banquo tipiz (master)]$
```

- *Python dynamic and strong typing*

the Python interpreter

- Molto materiale / documentazione disponibile (infatti, questa introduzione è fortemente basata su:
  - http://tdc-www.harvard.edu/Python.pdf
- Python può essere trovato pronto per l'uso sia su Linux che su Mac OS X. Per Windows potete trovare facilmente i binari al seguente URL: http://python.org/
- Molti moduli (librerie) disponibili, nel nostro caso alcuni utili / interessanti: numpy, matplotlib e pystat
- python 2.x vs python 3.x, useremo python 3.x la versione 2.x è stata ora ignorata.

the Python interpreter

```
[redo@virtualinux ~]$ python
Python 2.7.10 (default, Jun 20 2016, 14:45:40)
[GCC 5.3.1 20160406 (Red Hat 5.3.1-6)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> (4*5)/3
6
>>> exit()
[redo@virtualinux ~]$
```

To exit press CTRL-D or type exit()

the Python interpreter

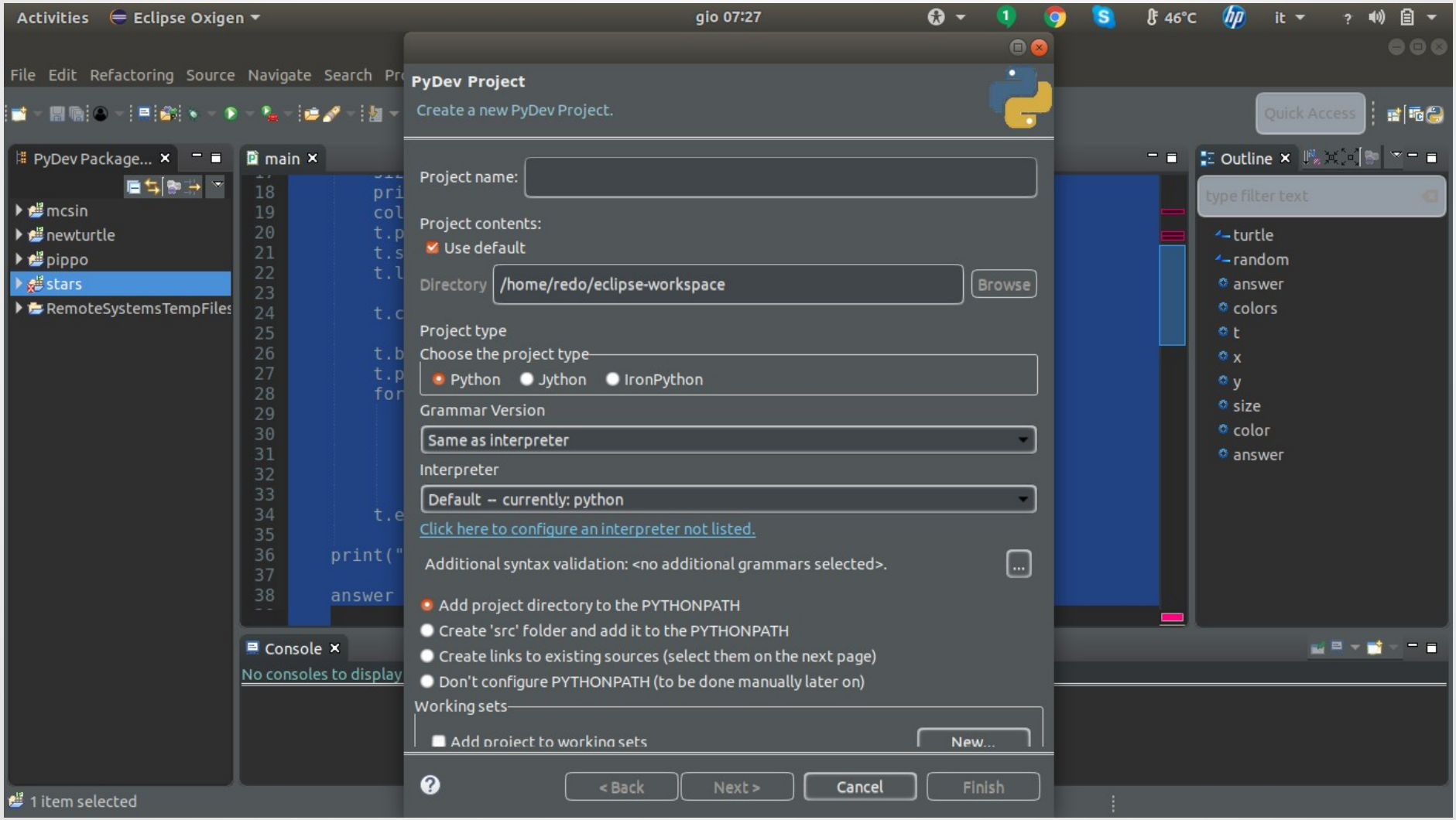
Possiamo semplicemente scrivere il sorgente usando un banale editor ASCII

$ python file.py

```
└─ $ cat file.py
a = 5
b = 5

c = a/b

print(c)

┌redo@buchner /home/redo
└─ $ python3 file.py
1.0
```

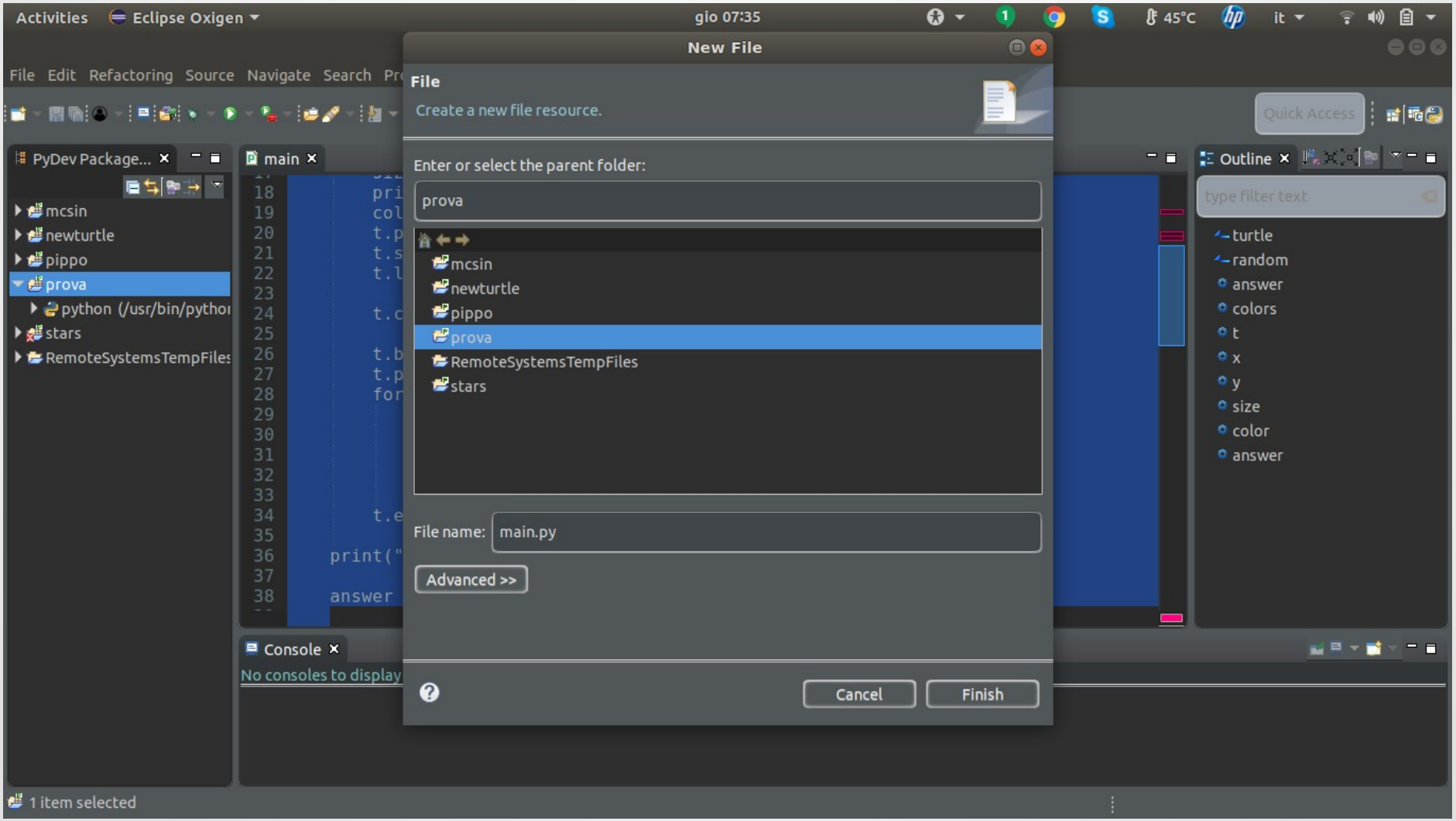
the Python interpreter

Usare un IDE: File → PyDev Project



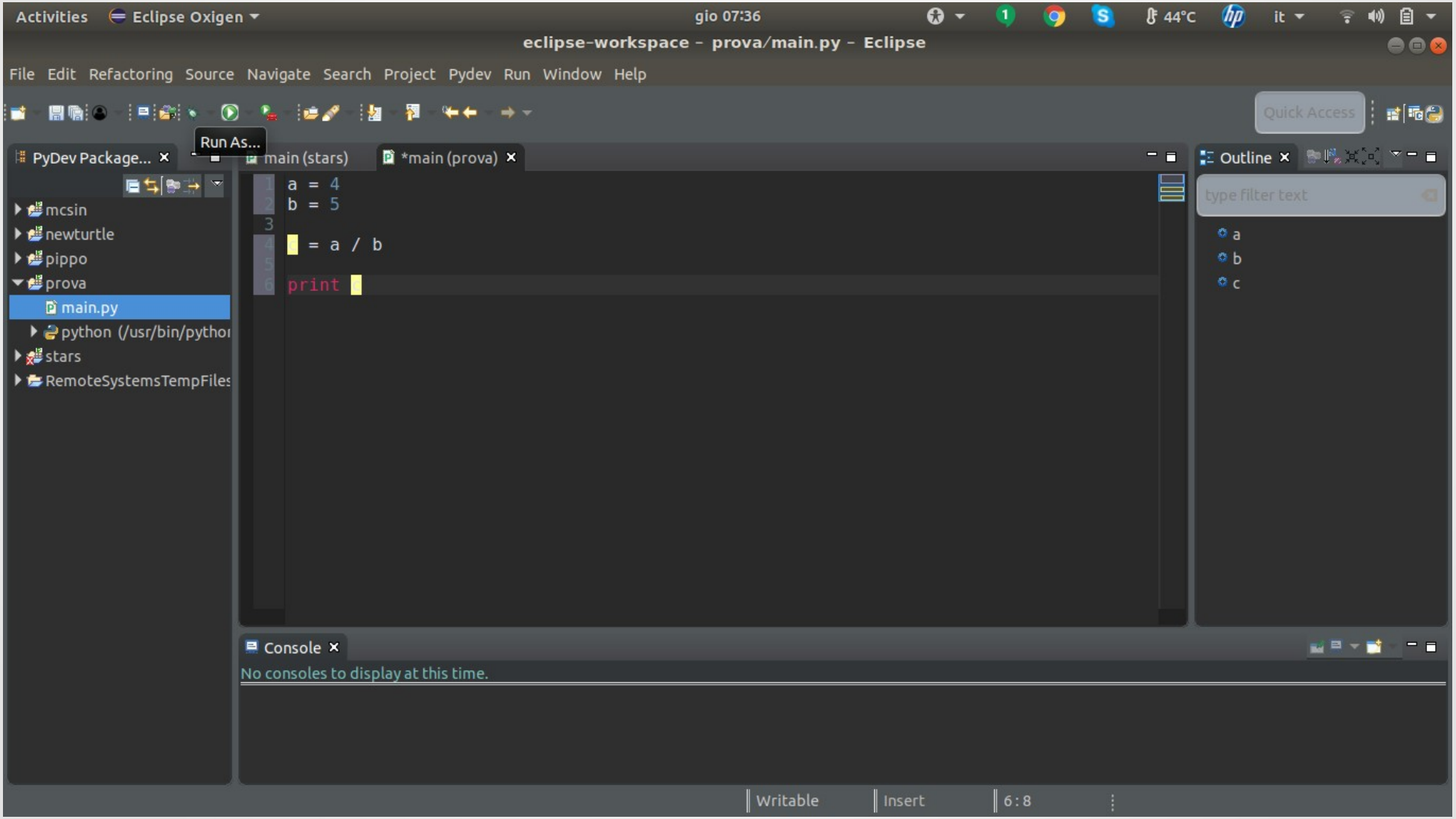
the Python interpreter

Click su Project e poi con il pulsante destro New → File



the Python interpreter

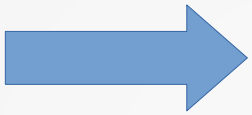
Click Run As

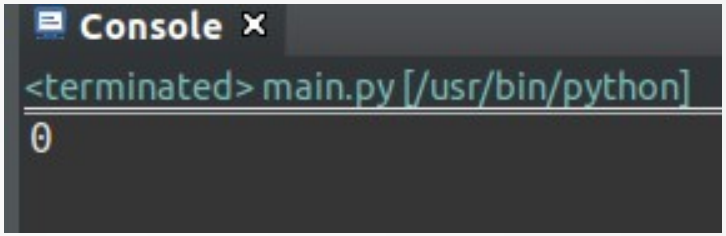


the Python interpreter

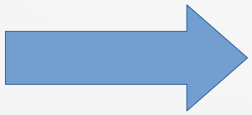
Click Run As

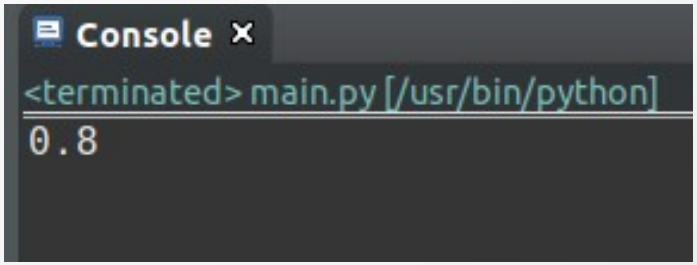
```
1 a = 4
2 b = 5
3
4 c = a / b
5
6 print c
```






```
1 a = 4.0
2 b = 5.0
3
4 c = a / b
5
6 print c
```



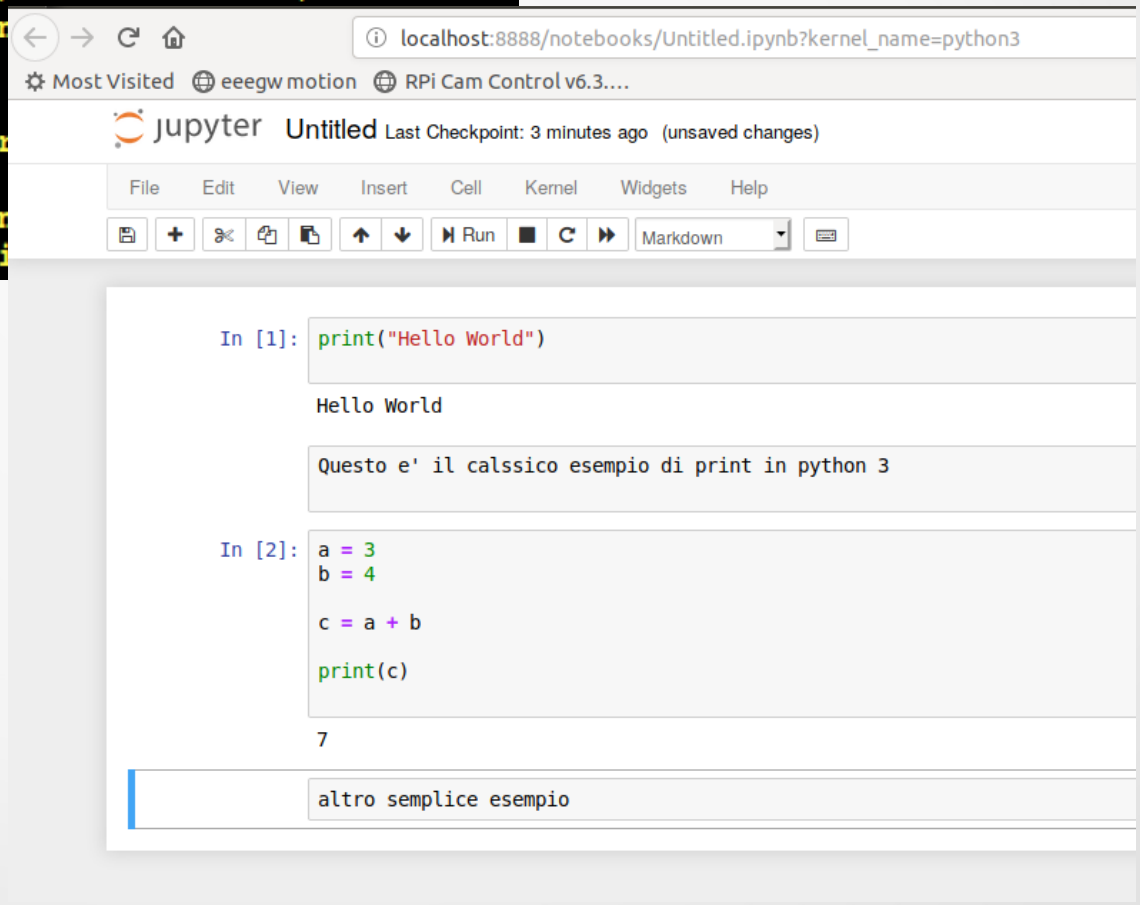


Or Jupyter

Applicazione web in cui puoi creare e condividere documenti che contengono codice live, equazioni, visualizzazioni e testo, Jupyter Notebook è uno degli strumenti ideali

Or Jupyter

```
[redo@buchner FondamentiDiProgrammazione]$ jupyter notebook
[I 18:32:09.367 NotebookApp] Serving notebooks from local dir
mmazione
[I 18:32:09.367 NotebookApp] The Jupyter Notebook is running
[I 18:32:09.367 NotebookApp] http://localhost:8888/?token=510
[I 18:32:09.367 NotebookApp] Use Cor
[C 18:32:09.368 NotebookApp]

    Copy/paste this URL into your br
    to login with a token:
        http://localhost:8888/?toker
[I 18:32:09.617 NotebookApp] Accepti
```



Or Colab

What is Colaboratory?

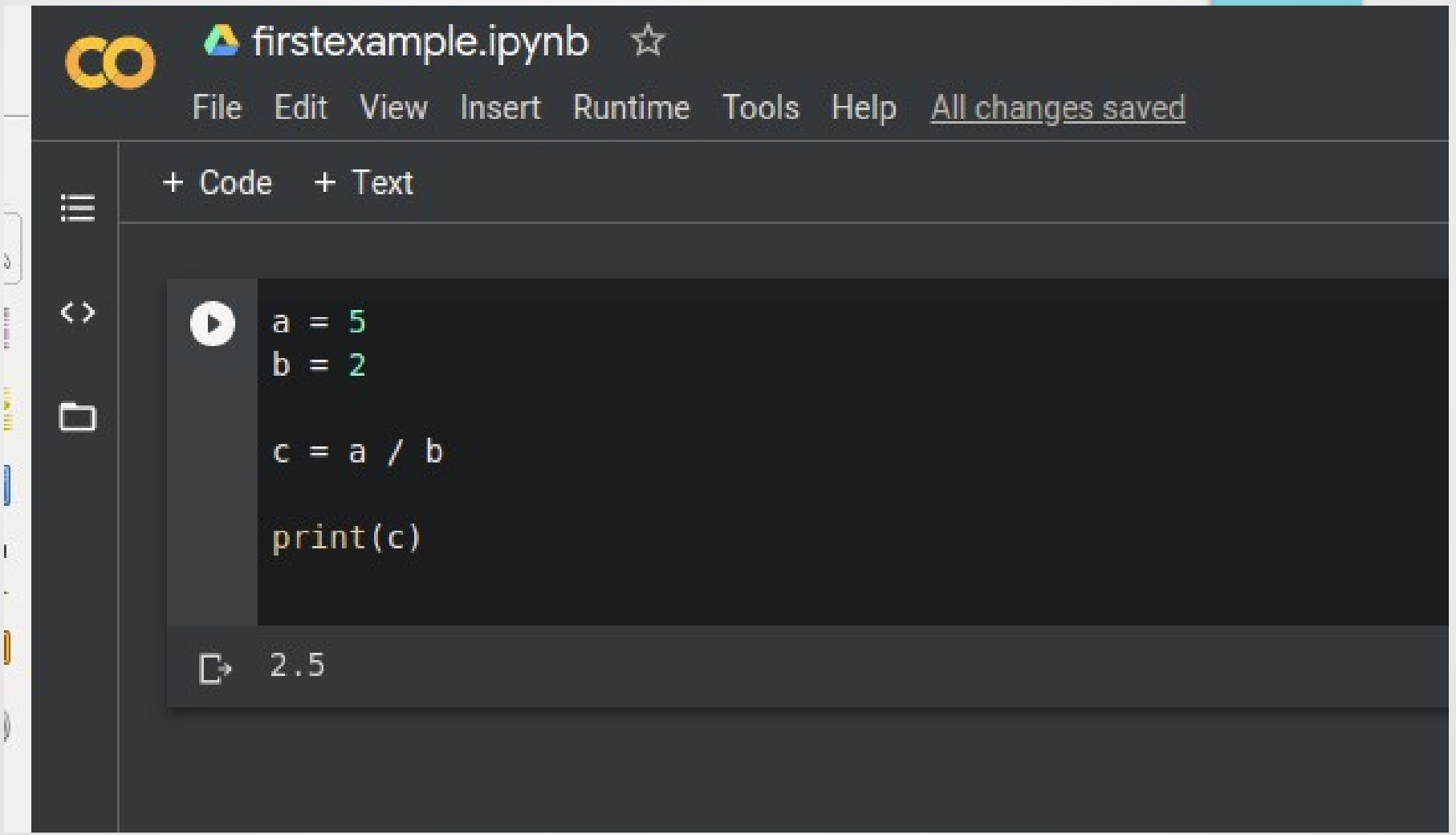
Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab to learn more, or just get started below!

Let’s start →

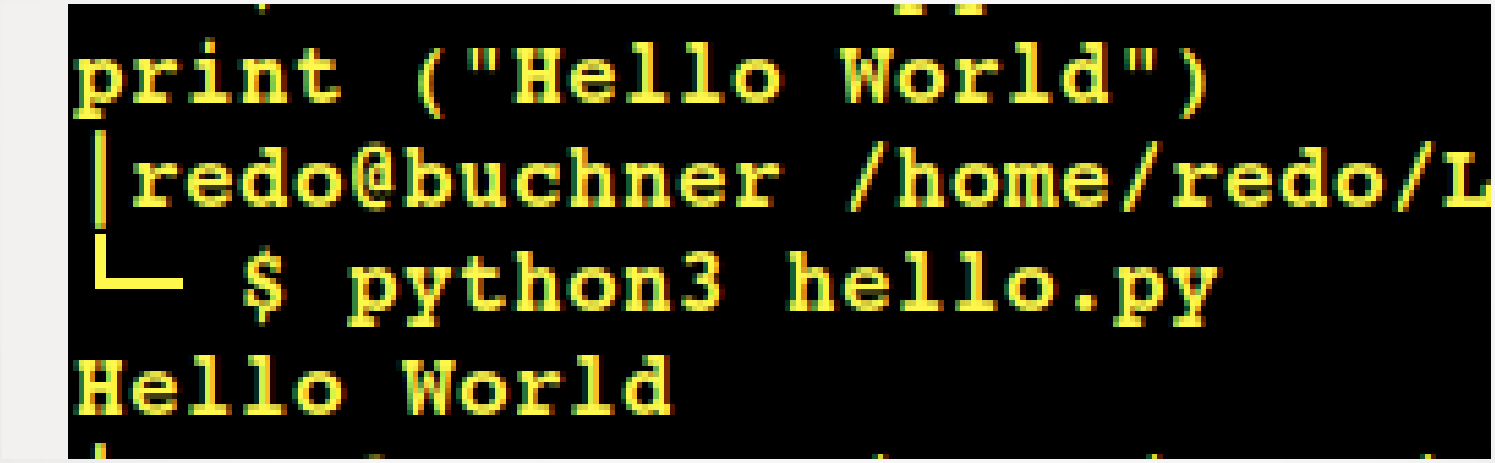
Or Colab


firstexample.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
a = 5
b = 2
c = a / b
print(c)
2.5

HELLO WORLD!

Hello world

- Il classico programma utilizzato per illustrare le basi sintattiche di qualsiasi linguaggio di programmazione



HELLO WORLD 2

Hello world

```
└─ $ cat name.py
name = input("Insert your name: ")

print("Ciao ", name)
```

```
└─ $ python3 name.py
Insert your name: Loriano
Ciao  Loriano
```

Hello world

The variable has no value until the user enters a name.

PYTHON BASICS

Python Basics

- Vediamo un semplice codice che ci permette di illustrare alcune delle caratteristiche di base della sintassi di Python:
  - Posso aggiungere commenti usando il carattere #
  - = viene utilizzato per assegnare valori alle variabili
  - Per fare operazioni tra numeri e variabili posso usare i soliti operatori +, -, *, /

```
x = 34 - 23 # commentare il codice
y = "Hello"
z = 3.45
```

Python Basics

- == è l'operatore che serve a confrontare i valori
- Gli operatori logici sono invece: and, or, not
- L'operatore + può essere utilizzato anche per concatenare le stringhe

```
if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + " World"
```

Python Basics

- print è il comando di base utilizzato per stampare a schermo
- **Le variabili non devono essere dichiarate esplicitamente la prima volta che assegno un valore alla variabile viene creata e gli viene assegnato un tipo**

```
print("Valore di x: ", x)
print("Valore di y: ", y)

print(z)
```

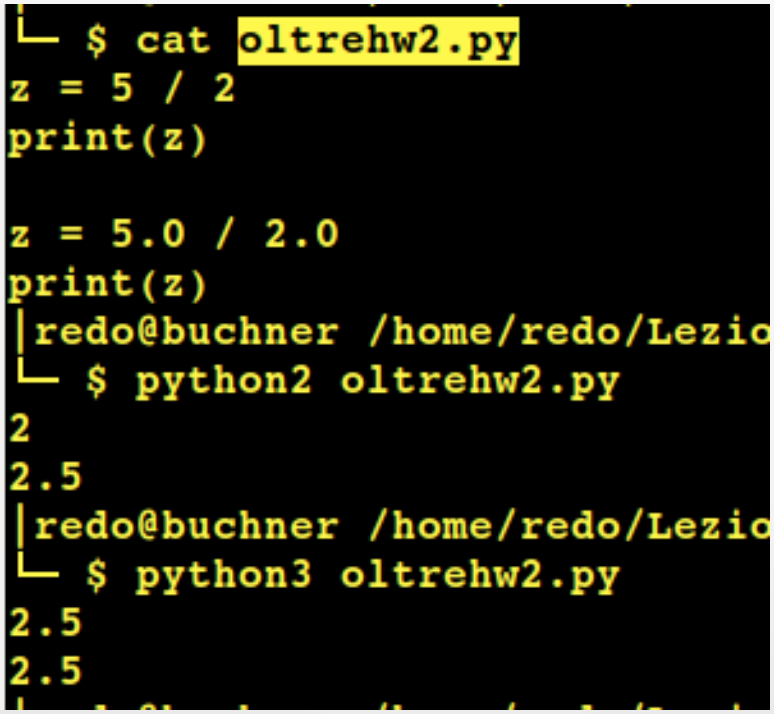
Python Basics

```
x = 34 - 23 # commentare il codice
y = "Hello"
z = 3.45

if z == 3.45 or y == "Hello":
    x = x + 1
    y = y + " World"

print("Valore di x: ", x)
print("Valore di y: ", y)

print(z)
```

```
Valore di x:  12
Valore di y:  Hello World
3.45
```

Python Basics

- Non ci sono caratteri di fine riga, se una riga di codice deve essere interrotta su più righe, usa \
- Python 2 Per impostazione predefinita i numeri sono interi, quindi z = 5/2 darà come risultato 2



IF … THEN … ELSE

If … then … else

```
INSERT NUMBER I
IF I LOWER THEN 0
    PRINT “il numero è minore di zero”
ELSE IF I EQUAL TO 0
    PRINT “Il numero è uguale a zero”
ELSE
    PRINT “Il numero è maggiore di zero”
ENDIF
```

If … then … else

- Per identificare i blocchi di codice **in Python, vengono utilizzati spazi vuoti**, non ad esempio {} come in C / C ++

```
└─ $ cat oltrehw4.py
si = input ("inserisci un numero: " )
i = float(si)

if i < 0 :
    print("numero inferiore a zero")
elif i == 0:
    print("inseiro uguale a zero")
else:
    print("numero maggiore di zero")
```

```
inserisci un numero: -31
numero inferiore a zero
inserisci un numero: 3.0
numero maggiore di zero
```

If … then … else

- Oppure puoi usare eval()

```
si = eval(input ("inserisci un numero: " )) # The expression argument
                                            # is parsed and evaluated as a
                                            # Python expression (technically
                                            # speaking, a condition list)
                                            # using the globals and locals
                                            # dictionaries as global and local
                                            # namespace.

if si < 0 :
    print("numero inferiore a zero")
elif si == 0:
    print("inseiro uguale a zero")
else:
    print("numero maggiore di zero")
```

```
└─ $ python3 oltrehw4.py
inserisci un numero: 10
numero maggiore di zero
inserisci un numero: print("si ", si)
si  10
Traceback (most recent call last):
  File "oltrehw4.py", line 20, in <module>
    if si < 0 :
TypeError: '<' not supported between instances of 'NoneType' and 'int'
```

LOOPS

Loops

SET N TO 0

SET n TO 0

Repeat the following:

a. If n >= 10, terminate the repetition, otherwise.
b. Increment N by n
c. PRINT n

PRINT N

Loops

- Per identificare i blocchi di codice in Python, vengono utilizzati spazi vuoti, non ad esempio {} come in C / C ++
- **Python e' case sensitive**

```
└─ $ less oltrehw3.py
N = 0

for n in range(0,10):
  N = N + n
  print(n)

print("Valore finale: ", N)
```

Loops

```
N = 0

for n in range(0,10):
  N = N + n
  print(n)

print("Valore finale: ", N)
```

```
0
1
2
3
4
5
6
7
8
9
Valore finale:  45
```

EXAMPLE

Example of a numerical procedure

- È possibile trovare un algoritmo per risolvere quasi tutti i problemi, ma non tutti. Ad esempio, calcola le soluzioni di un'equazione del secondo ordine:

```
INPUT  A, B, C
COMPUTE D = (B*B)-(4 * A * C)
IF D >= 0.0
  SOL1 = (-1,0 * B + SQRT(D)) / (2,0 * A)
  SOL2 = (-1,0 * B - SQRT(D)) / (2,0 * A)
  PRINT SOL1 AND SOL2
ELSE
  PRINT “non ci sono soluzioni reali”
```

Example of a numerical procedure

```
└─ $ cat solv.py
import math

a = float(input("insert a:"))
b = float(input("insert b:"))
c = float(input("insert c:"))

print("a = ", a, " b = ", b, " c = ", c)

delta = math.pow(b, 2.0) - (4.0 * a * c)

if (delta >= 0):
  tn = math.sqrt(delta)
  sol1 = ((-1.0 * b) + tn) / (2.0 * a)
  sol2 = ((-1.0 * b) - tn) / (2.0 * a)
  print(sol1, sol2)
else:
  print("No real solutions")
```

Example of a numerical procedure

```
    c = float(c1)
```

Console

```
<terminated> main.py [/usr/bin/python]
insert a:1
insert b:15
insert c:2.5
a =  1.0  b =  15.0  c =  2.5
-0.168560850692 -14.8314391493
```

ESERCIZIO 1

EXERCISE 1

- Scrivi un programma in Python che legga 10 numeri, dopo aver calcolato il valore medio e stampato il risultato

```
[redo@banquo esercizipython (master)]$ python ex1.py
inserisci il numero 1 45
inserisci il numero 2 67
inserisci il numero 3 84
inserisci il numero 4 2
inserisci il numero 5 4
inserisci il numero 6 6
inserisci il numero 7 7
inserisci il numero 8 8
inserisci il numero 9 9.0
inserisci il numero 10 13.0
la somma:  245.0
valore medio:  24.5
[redo@banquo esercizipython (master)]$
```

BREAK

Break

- L'interruzione solitamente nidificata sintatticamente in un ciclo for o while, termina il ciclo più vicino, saltando la clausola else opzionale se il ciclo ne ha una.

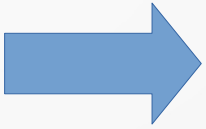
```
└─ $ cat testbreaks.py
for i in range(0,10):
    print("i: ", i)
    if i > 5:
        break;
```



```
i:  0
i:  1
i:  2
i:  3
i:  4
i:  5
i:  6
```

Break and nested loops

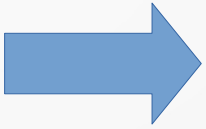
```
└─ $ cat testbreaks1.py
for i in range(10):
    print("loop 1: ", i)
    for j in range(10):
        print("    loop 2: ", j)
        if (j > 5):
            break;
```



```
loop 1:  0
    loop 2:  0
    loop 2:  1
    loop 2:  2
    loop 2:  3
    loop 2:  4
    loop 2:  5
    loop 2:  6
loop 1:  1
    loop 2:  0
    loop 2:  1
    loop 2:  2
    loop 2:  3
    loop 2:  4
    loop 2:  5
    loop 2:  6
loop 1:  2
    loop 2:  0
    loop 2:  1
    loop 2:  2
```

Break and nested loops

```
└─ $ cat testbreaks2.py
for i in range(10):
    print("1) loop 1: ", i)
    for j in range(10):
        print("    loop 2: ", j)
        if (j > 5):
            break;
    print("2) loop 1: ", i)
```



```
1) loop 1:  0
    loop 2:  0
    loop 2:  1
    loop 2:  2
    loop 2:  3
    loop 2:  4
    loop 2:  5
    loop 2:  6
2) loop 1:  0
1) loop 1:  1
    loop 2:  0
    loop 2:  1
    loop 2:  2
    loop 2:  3
    loop 2:  4
    loop 2:  5
    loop 2:  6
2) loop 1:  1
1) loop 1:  2
```

RANDOM NUMBER

Random Number

- https://docs.python.org/2/library/random.html
  - Pseudo Random vs Random
  - cat /dev/urandom
  - cat /dev/urandom | od -vAn -N2 -tu
  - cat /dev/random
  - cat /dev/random | od -vAn -N2 -tu

```
[redo@buchner ~]$ cat /dev/random | od -vAn -N2 -tu2
 38004

^C
[redo@buchner ~]$ cat /dev/random
```

Random Number

- https://docs.python.org/2/library/random.html
  - random.randint(a, b) genera un numero intero random N nell'intevallo a <= N <= b

```
└─ $ cat rnd.py
import random

for i in range(100):
  print(random.randint(0,10))
```

Random Number

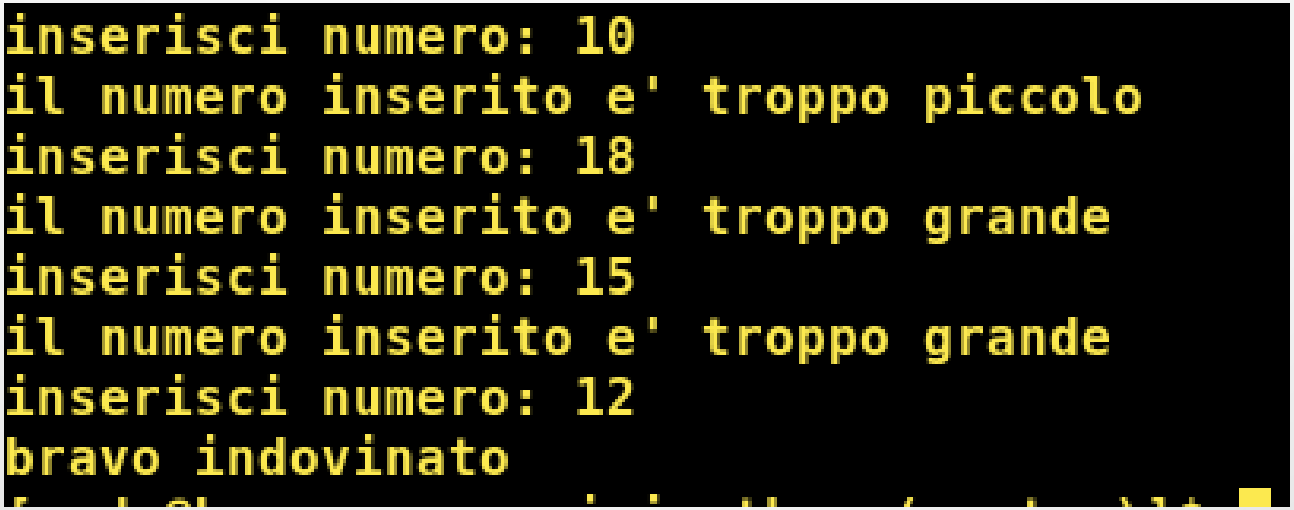
```python
import random

for i in range(100):
  print(random.randint(0,10))
```

```
9
3
3
4
2
4
0
5
4
10
10
2
3
10
```

ESERCIZIO 2

Exercise 2

- Scrivi un programma che generi un numero casuale R compreso tra 0 e 20 e chieda all'utente di indovinare il numero con un massimo di 10 tentativi. Ogni volta che il programma scriverà semplicemente se il numero inserito è maggiore o minore di R. Chiaramente se il numero inserito è uguale al numero casuale R generato il programma termina



Pseudocode

```
GENERATE A RANDOM NUMBER rnd

Repeat the following:
    INPUT b
    IF b IS EQUAL TO rnd
          PRINT “well done”
          BREAK
     ELSE IF b < rnd
          PRINT “inserted number is too small”
     ELSE
          PRINT “inserted number is too big”
    ENDIF
```

Python (strutture dati)

Loriano Storchi

loriano@storchi.org

http::/www.storchi.org/

I numeri complessi

- I numeri complessi sono intrinsecamente definiti ed usabili in python

```
a = 2 + 3j
b = 4 - 1j

print a
print a.real, " ", a.imag
print b
print b.real, " ", b.imag

c = a * b
print type(c), " valore ", c

if type(c) == complex:
    print "c e\' un numero complesso"

[redo@banquo datastr (master)]$ python complex.py
(2+3j)
2.0   3.0
(4-1j)
4.0   -1.0
<type 'complex'>  valore  (11+10j)
c e' un numero complesso
[redo@banquo datastr (master)]$
```

Strutture dati

- Le strutture dati permettono di organizzare i dati in modo da rendere il loro uso e la loro manipolazione piu' efficiente
- Vedremo in particolare le stringhe, le liste, le tuple, i dizionari ed i set.
- Noi vedremo solo le basi minime di uso utili allo svolgimento di esercizi base

Le stringhe

- Una stringa e' una sequenza di caratteri, in python ci sono diversi metodi/operazioni utili alla manipolazione delle stringhe, una stringa in generale puo' essere vista come un array di caratteri

```
str1 = "hello"
str2 = "world"

print str1 + " " + str2
print 3*(str1+" ")
print str1[0]
print str1[0:3]
print str1[-2:]
[redo@banquo datastr (master)]$ python strin1.py
hello world
hello hello hello
h
hel
lo
[redo@banquo datastr (master)]$
```

Le stringhe

- Come detto la classe in questione ha diversi metodi vediamo solo alcuni

```
[redo@banquo datastr (master)]$ cat strin2.py
str = "Hello, World!"
index = str.find("ll")
if index >= 0:
  print "a ", index, " trovato ", str[index]

res = str.split(" ")
idx = 0
for r in res:
    idx += 1
    print idx , " - ", r

for i in range(0,len(str)):
    print str[i]
[redo@banquo datastr (master)]$ python strin2.py
a  2  trovato  l
1  -  Hello,
2  -  World!
H
e
l
```

Liste

- Le liste sono sequenze ordinate di oggetti, molte operazioni di base sono in comune con le stringhe

```
a = [1, "pippo", 4.5, "pluto"]
print a[0]
for i in range(0,len(a)):
    print a[i]

for l in a:
    print l
[redo@banquo datastr (master)]$ python liste1.py
1
1
pippo
4.5
pluto
1
pippo
4.5
pluto
[redo@banquo datastr (master)]$
```

Liste

- La lista ha molti metodi utili vediamone alcuni

```
[redo@banquo datastr (master)]$ cat liste2.py
a = [1, 3.5, -6.0, 5]
a.append(46)
print a
print "rimuove l\'ultimo elemento: ", a.pop()
a.append(46)
i = len(a) - 1
print "rimuove l\'elemento ", i, " ", a.pop(i)
a.sort()
print a
[redo@banquo datastr (master)]$ python liste2.py
[1, 3.5, -6.0, 5, 46]
rimuove l'ultimo elemento:  46
rimuove l'elemento  4   46
[-6.0, 1, 3.5, 5]
[redo@banquo datastr (master)]$
```

Tuples

- Le tuples in python sono molti simili alle liste solo che la loro manipolazione e' piu' rapida visto che sono “immutabili”

```
t = (1,3.5,8,10.0)
for i in range(len(t)):
    print t[i]

for val in t:
    print val

# ma posso modificare un valore ?
t[1] = 0
[redo@banquo datastr (master)]$ python tuples.py
1
3.5
8
10.0
1
3.5
8
10.0
Traceback (most recent call last):
  File "tuples.py", line 9, in <module>
    t[1] = 0
TypeError: 'tuple' object does not support item assignment
[redo@banquo datastr (master)]$
```

Piccola precisazione

```
[redo@banquo datastr (master)]$ cat tuples_pres.py
t = (1,3.5,8,10.0,[10,5])

for val in t:
    print val

# ma posso modificare un valore se mutabile
t[4][1] = 0
print t
[redo@banquo datastr (master)]$ python tuples_pres.py
1
3.5
8
10.0
[10, 5]
(1, 3.5, 8, 10.0, [10, 0])
[redo@banquo datastr (master)]$
```

Dizionari

- Un dizionario e' una sequenza di elementi , ogni elemento a' una coppia chiave, valore. Le chiavi sono uniche ed i dizionari si creano usando le parentesi graffe

```
d = {"k1":1, "k2":"valore", 3:"val3"}
print d[3], d["k1"]
d["quattro"] = 4
print d
if d.has_key("quattro"):
    print "chiave presente"
if not d.has_key(4):
    print "chiave non presente"
[redo@banquo datastr (master)]$ python diz1.py
val3 1
{'k2': 'valore', 'k1': 1, 3: 'val3', 'quattro': 4}
chiave presente
chiave non presente
[redo@banquo datastr (master)]$
```

Dizionari

```
d = {"k1":1, "k2":"valore", 3:"val3", "quattro":4}
for x in d.itervalues():
    print x
print " "
for x in d.iterkeys():
    print x
print " "
for x in d.iteritems():
    print x
    print x[0], x[1]
[redo@banquo datastr (master)]$ python diz2.py
valore
1
val3
4

k2
k1
3
quattro

('k2', 'valore')
k2 valore
('k1', 1)
k1 1
(3, 'val3')
```

Dizionari

```
d = {"k1":1, "k2":"valore", 3:"val3", "quattro":4}
del d["k1"]
for x in d.iteritems():
    print x
d.clear()
print len(d)
for x in d.iteritems():
    print x
[redo@banquo datastr (master)]$ python diz3.py
('k2', 'valore')
(3, 'val3')
('quattro', 4)
0
```

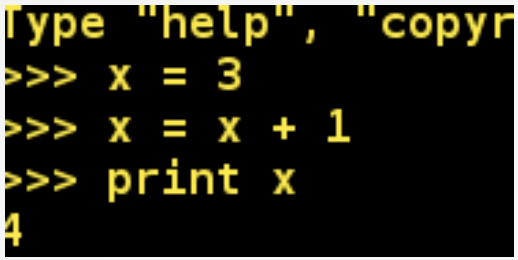
Reference

- Quando dichiaro una variabile sto chiedendo una certa quantita' di spazio in memoria
- In python l'operazione di assegnamento manipola i riferimenti , quindi x = y non crea una copia dei dati contenuti in y in x, ma crea un riferimento a y, x punta ad y

```
a = [1,2,3,4]
b = a
a.append(5)
print b
x = 3
y = x
x = 4
print y
[redo@banquo datastr (master)]$ python refer.py
[1, 2, 3, 4, 5]
3
```

Quando scrivo x = 4 viene allocato lo spazio in memoria necessario a contenere l'intero 4 e poi viene “memorizzato l'indirizzo della” (creato il riferimento alla) locazione di memoria in x

Reference

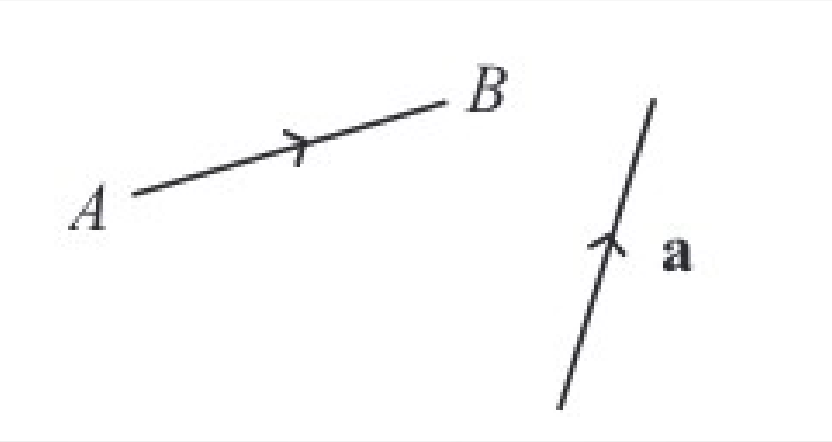


- Cosa succede veramente quando incremento x ?
  - L'interprete recupera il valore contenuto nell'indirizzo di memoria a cui x fa riferimento
  - Viene calcolato il risultato dell'operazione 3 + 1 ed il risultato viene memorizzato in una nuova locazione di memoria
  - Si cambia il riferimento in x, x adesso fara' riferimento al nuovo indirizzo in memoria dove e' memorizzato il valore 4
  - Python ha un garbage collector che elimina libera tutta la memoria allocata quando non ci sono piu' nomi che fanno riferimento alle zone di memoria in questione

VETTORI

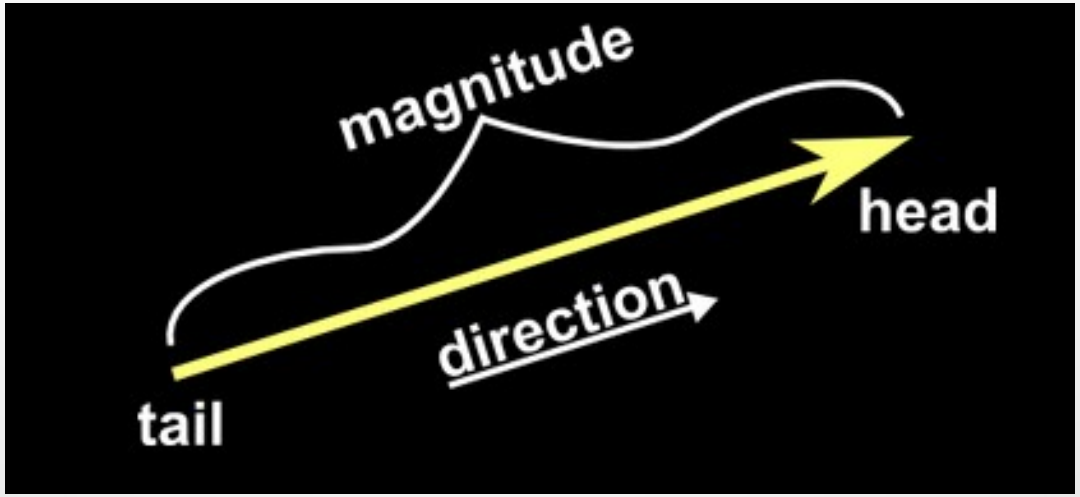
Vettori

- I vettori sono estremamente utili in fisica. La caratteristica principale di un vettore è che ha sia una grandezza (o una dimensione) che una direzione.
- Un esempio di una grandezza vettoriale è velocità. Questa è la velocità, in una particolare direzione.



Vettori

- I vettori sono estremamente utili in fisica. La caratteristica principale di un vettore è che ha sia una grandezza (o una dimensione) che una direzione.
- Un esempio di una grandezza vettoriale è velocità. Questa è la velocità, in una particolare direzione.



Vettori

- I vettori sono estremamente utili in fisica. La caratteristica principale di un vettore è che ha sia una grandezza (o una dimensione) che una direzione.
- Un esempio di una grandezza vettoriale è velocità. Questa è la velocità, in una particolare direzione.

Vettori

- TODO come rappresento un vettore ? Vedi una lista e poi magari rappresenta anche con matplotlib poi somman sottrazione prodotto scalare e vettoriale
-

MATRICI

Una matrice

- Posso usare una lista di liste per memorizzare una matrice in modo semplice

```
import random
import math

A =  [[0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0],
      [0.0, 0.0, 0.0]]

for i in range(len(A)):
  for j in range(len(A[0])):
    A[i][j] = random.uniform(0.0, 1.0)

print "Matrix A"
print A
[redo@banquo mtxmtx (master)]$ python storemtx.py
Matrix A
[[0.1975653801959704, 0.4483683937014895, 0.278
 0.9080131010804726, 0.3679889526650569], [0.471
.5738920118128268]]
```

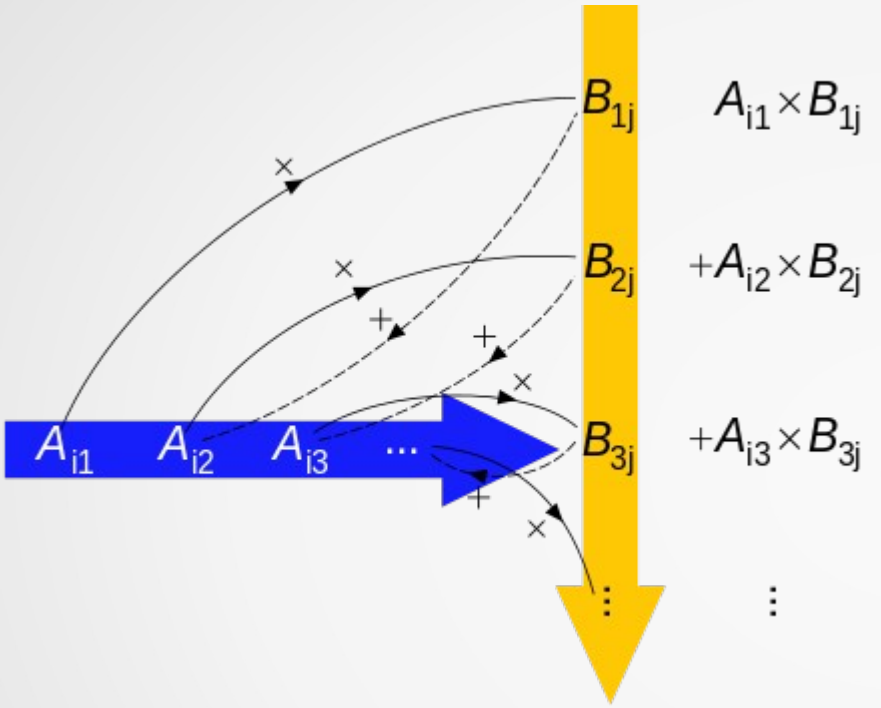
Moltiplicazione matrice matrice

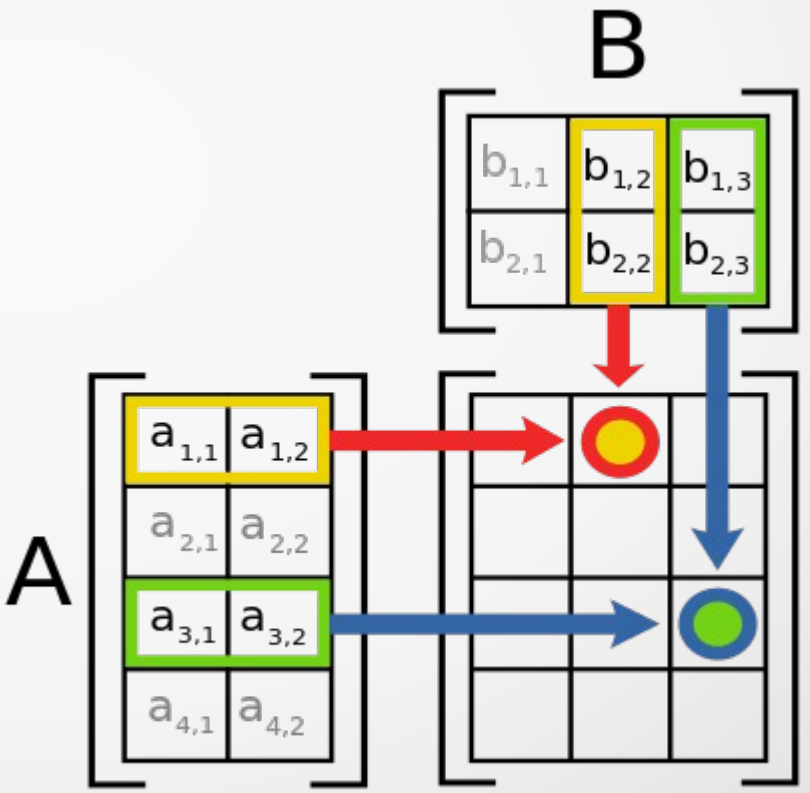
$$\mathbf{A} = \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} B_{11} & B_{12} & \cdots & B_{1p} \\ B_{21} & B_{22} & \cdots & B_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mp} \end{pmatrix}$$

$$(\mathbf{AB})_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj} \ .$$

Il prodotto e' definito solo per matrici con dimensioni compatibili

Moltiplicazione matrice matrice





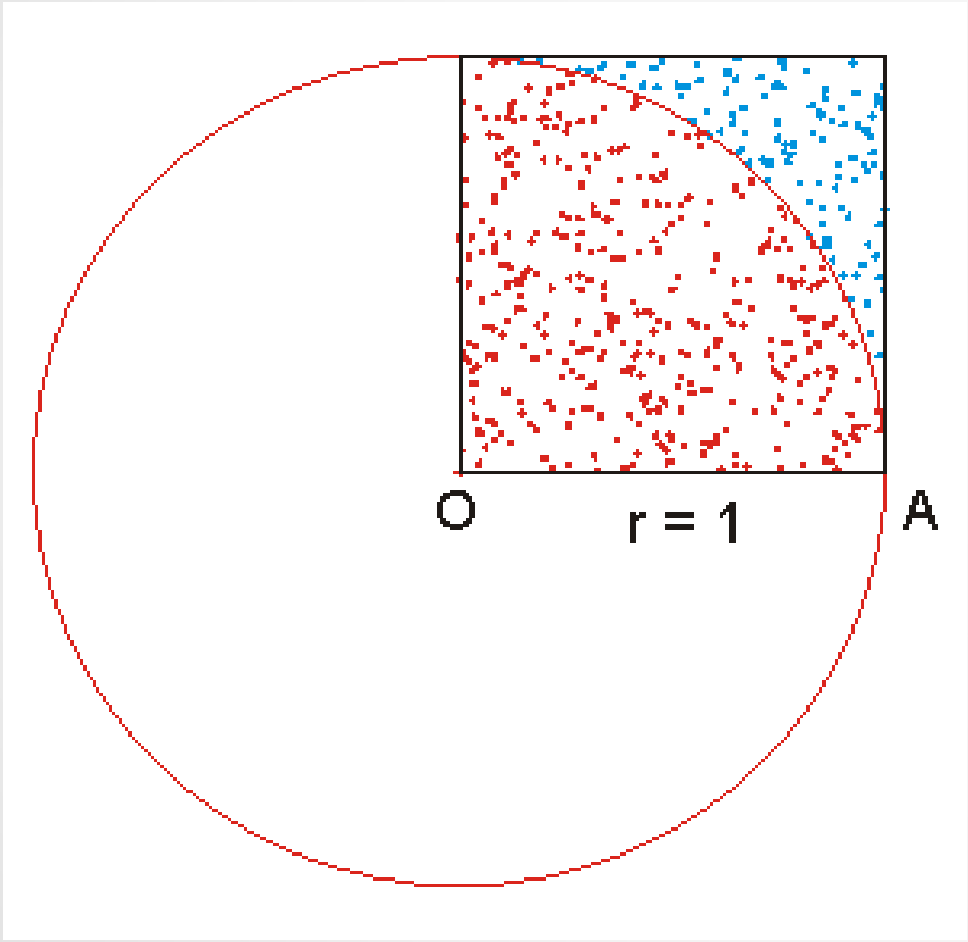
Esercizio

- Scrivere un programma che date due matrici 3x3 riempite con numeri random calcoli e stampi il risultato del prodotto matrice matrice

```
[redo@banquo mtxmtx (master)]$ python mtx.py
Matrix A
[0.1924696838424818, 0.969694776511462, 0.8237325914685499]
[0.177986003994455, 0.975535311913037, 0.8217085413339825]
[0.4127948602822054, 0.05779395844699264, 0.4020898243848140]
Matrix B
[0.7196544280415835, 0.8235257700392403, 0.9349263737223458]
[0.2739352426172888, 0.7093719818717363, 0.4407551720290670]
[0.1260881966959214, 0.939121875851728, 0.4639718068159948]
Matrix C
[0.5080081911273185, 1.619963346520346, 0.9895316704002202]
[0.4989296664411049, 1.610277945334311, 0.9776256401093906]
[0.3636002319703472, 0.7585559701630582, 0.5979641302345579]
```

ESERCIZI ED ESEMPI

Calcolo PI con metodo MC



$$A_S = (2r)^2 = 4r^2$$

$$A_C = \pi r^2$$

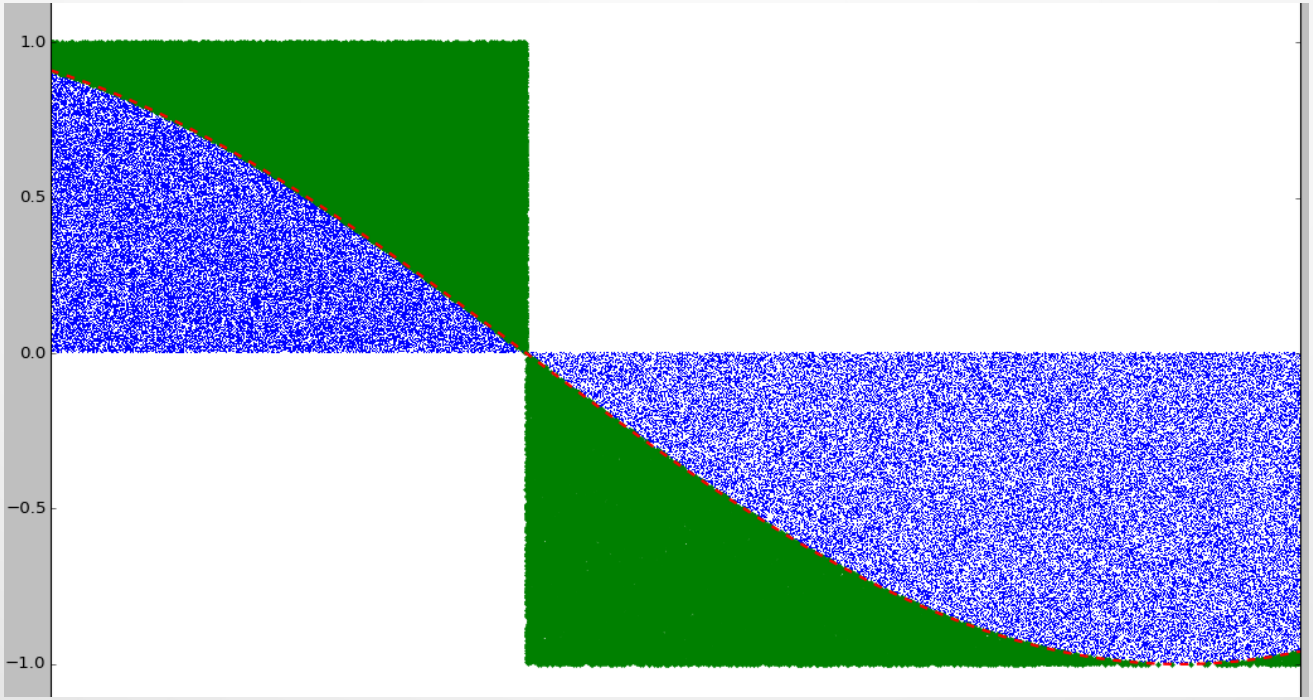
$$\pi = 4 \times \frac{A_C}{A_S}$$

```
import random
import math
import sys

DIM = 100000

"""
if len(sys.argv) != 2:
    print("usage: ", sys.argv[0] , " NUM ")
    exit(1)
else:
    DIM = int(sys.argv[1])
"""

circle_count = 0

for i in range(0,DIM):

   x = random.uniform(0.0, 1.0)
   y = random.uniform(0.0, 1.0)

   if (math.sqrt((math.pow(x, 2.0) + math.pow(y, 2.0))) < 1.0):
     circle_count = circle_count + 1

pi = float(circle_count) / float(DIM)

print(4.0 * pi)
```

Esercizio

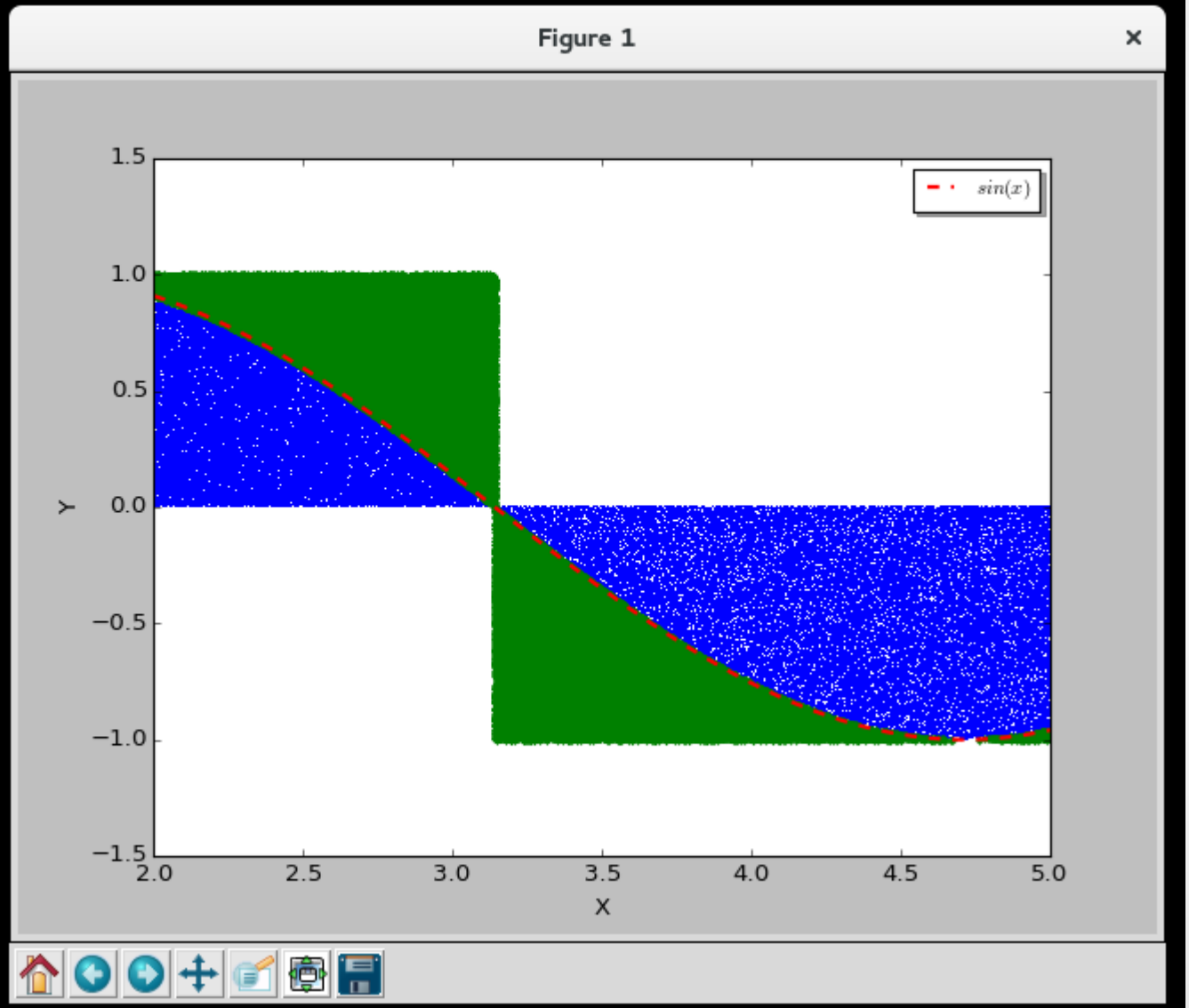
- Calcolo di integrale con metodo MC:

$$\int_2^5 \sin(x)\,dx = \cos(2) - \cos(5) = -0{,}69981$$



```
[redo@banquo mcsin (master)]$ time python mcsin.py 100
-0.18

real    0m0.013s
user    0m0.010s
sys     0m0.003s
[redo@banquo mcsin (master)]$ time python mcsin.py 1000
-0.726

real    0m0.017s
user    0m0.014s
sys     0m0.003s
[redo@banquo mcsin (master)]$ time python mcsin.py 10000
-0.6834

real    0m0.049s
user    0m0.040s
sys     0m0.008s
[redo@banquo mcsin (master)]$ time python mcsin.py 100000
-0.70938

real    0m0.115s
user    0m0.108s
sys     0m0.007s
[redo@banquo mcsin (master)]$ time python mcsin.py 1000000
-0.703062

real    0m1.049s
user    0m1.021s
sys     0m0.027s
[redo@banquo mcsin (master)]$ time python mcsin.py 10000000
-0.7001124

real    0m10.226s
user    0m10.105s
sys     0m0.116s
[redo@banquo mcsin (master)]$
```

```
[redo@banquo mcsin (master)]$ python mcsin_wplt.py 100000
-0.696206603137
```



Vediamo un primo esempio di uso di numpy e matplotlib