Informatica per la didattica delle scienze

Release 0.01

Zambelli Daniele

15 June 2014

Indice

CAPITOLO 1

Presentazione

1.1 Introduzione

1.1.1 Chi sono

- mi sono lareato in matematica nel 1980;
- ho insegnato nella scuola secondaria di primo grado per una ventina di anni;
- per 3 anni sono sato distaccato dall'insegnamento per coordinare un progetto di introduzione dell'informatica nella scuola di base.
- insegno nella secondaria di secondo grado dal 2001;
- ho sempre utilizzato l'informatica nell'insegnamento della matematica:
  - con Logo nelle medie;
  - con Python nelle superiori;
- ho scritto un interprete Logo per DOS;
- ho scritto delle librerie per la geometria con Python:
  - pycart: piano cartesiano,
  - pyplot: grafico di funzioni,
  - pyturtle: geometria della tartaruga,
  - pyig: geometria interattiva;
- da quando lo conosco uso software libero;
- il software e i documenti che produco sono tutti rilasciati con licenze libere.

1.1.2 Riferimenti bibliografici

1. Seymour Papert, "Mindstorms", 1986
2. Abelson, Disessa, "La geometria della tartaruga", 1986
3. Manfred Spitzer, "Demenza digitale", 2012

1.1.3 Licenza

1.2 Informatica e didattica

1.2.1 Informatica

Cos'è l'informatica?

"L'informatica è la scienza che usa il linguaggio per risolvere problemi", Lucio Varagnolo.

1.2.2 Matematica

Cos'è la matematica?

È un linguaggio che permette di descrivere modelli.

1.2.3 Informatica e computer

L'informatica non riguarda i computer più di quanto l'astronomia riguardi i telescopi. (Edsger Wybe Dijkstra)

1.2.4 Informatica per la didattica

Cosa **non** è informatica nella didattica:

- uso il computer;
- office;
- ECDL;
- uso di particolari programmi;
- video, "simulazioni", presentazioni.

L'informatica è molto giovane e in continua evoluzione, non possiamo permetterci di insegnare cose che fra pochi anni saranno cambiate.

In generale il nostro obiettivo non è quello di insegnare "informatica" e meno ancora insegnare a usare il computer. La scommessa è quella di riuscire a usare il computer, o meglio l'informatica, per insegnare cose utili.

1.2.5 Un linguaggio di programmazione

Un linguaggio di programmazione ha diverse potenzialità formative.

- insegna a usare un linguaggio preciso ed efficace .
- Permette di "usare" concetti matematici:

  - variabili,
  - parametri,
  - gradi di libertà,
  - funzioni,
  - algoritmi,
  - ...
- Permette di esplorare "luoghi" della matematica e della geometria:
  - famiglie di funzioni,
  - induzione e ricorsione,
  - frattali,
  - metodi di soluzione di problemi,
  - successioni,
  - serie,
  - ...
- Fornisce agli studenti un "coltellino svizzero" leggero ma comodissimo in mote situazioni.

1.2.6 Imparare

Cosa significa imparare?

Imparare è composto da alcune parti che singolarmente non sono imparare:

- **Capire**: collegare le conoscenze nuove a quelle già presenti);
- **Memorizzare**: saper recuperare le informazioni quando servono;
- **Ri-produrre** (raccontare, utilizzare, ...).

(Maggiori informazioni su: fugamatematica.blogspot.it/2008/10/seconda-lezione-di-matematica-2008-09.html)

1.3 Scelte

1.3.1 Software libero

Cos'è il software libero, cosa non è il software libero.

Le quattro libertà:

0. Libertà di eseguire il programma, per qualsiasi scopo.
1. Libertà di studiare come funziona il programma e di modificarlo in modo da adattarlo alle proprie necessità.
2. Libertà di ridistribuire copie in modo da aiutare il prossimo.
3. Libertà di migliorare il programma e distribuirne pubblicamente i miglioramenti da voi apportati (e le vostre versioni modificate in genere), in modo tale che tutta la comunità ne tragga beneficio.

(vedi: http://www.gnu.org)

1.3.2 Licenze aperte

Cosa sono il le licenze Creative Commons ("copyleft")?

Licenza libera Creative Commons:

- CC: Creative commons;
- BY: Attribuzione;
- SA: Condividi allo stesso modo;
- NC: Non commerciale;
- ND: Non opere derivate.

Attribuzione - Condividi allo stesso modo CC BY-SA

"Questa licenza permette a terzi di modificare, ottimizzare ed utilizzare la tua opera come base, anche commercialmente, fino a che ti diano il credito per la creazione originale e autorizza le loro nuove creazioni con i medesimi termini. Tutte le opere basate sulla tua porteranno la stessa licenza, quindi tutte le derivate permetteranno anche un uso commerciale. Questa è la licenza usata da Wikipedia, ed è consigliata per materiali che potrebbero beneficiare dell'incorporazione di contenuti da progetti come Wikipedia e similari."

vedi: * http://it.wikipedia.org/wiki/Creative_Commons * http://creativecommons.org/licenses/

1.3.3 Linguaggio di programmazione

Quali caratteristiche deve avere il linguaggio nella didattica?

Deve:

- essere trasparente, cioè deve frapporre meno complicazioni possibili tra il problema e la soluzione;
- avere una sintassi semplice;
- essere potente in modo da poter crescere con l'alunno;
- essere sorretto da una progettazione attenta agli aspetti logici;
- avere strutture di dati e costrutti più vicini agli umani che alle macchine;
- permettere di esplorare e usare diversi paradigmi di programmazione;
- essere corredato da materiale didattico libero ed esempi a sorgente aperto;
- avere solide librerie dedicate agli usi più disparati;
- facile da ampliare e adattare ai propri scopi.

1.3.4 La geometria della tartaruga

Negli anni 80 dl secolo scorso, all'MIT Seymour Papert ha modificato un linguaggio dedicato alla soluzione di problemi di intelligenza artificiale per comandare un robottino che aveva una penna e permettere ai bambini di dare le istruzioni per realizzare dei disegni.

La tartaruga è un cursore grafico che può lasciare un segno quando si muove. La geometria della tartaruga è carattreizzata da avere un riferimento intrinseco cioè sono riferiti al cursore stesso e non ad un riferimento esterno.

I comandi base della geometria della tartaruga sono semplici:

- forward (avanti);
- back (indietro);

- right (destra);
- left (sinistra);
- penup (penna su);
- pendown (penna giù);

A partire da questi comandi si possono affrontare problemi con un ampio ventaglio di difficoltà, da quelli elementari a problemi che richiedono conoscenze matematiche elevate.

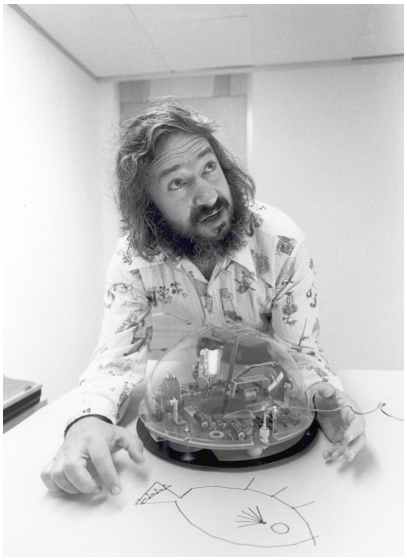


Figura 1.1: Seymour Papert il ceratore di Logo

1.3.5 Possibilità

La mia proposta per questo modulo di informatica per l'insegnamento della matematica nella scuola secondaria di secondo grado è: Giocare con la grafica della tartaruga con un linguaggio di programmazione. Ma quale linguaggio?

- LibreLogo
- Python + pygraph
- Scratch

Vediamo di seguito un esempio sviluppato con questi tre strumenti.

LibreLogo

Il seguente programma scritto nel dialetto di Logo inserito in Libreoffice produce una "spirale" triangolare che ruota su se stesa:

```
to spirale
    # Disegna una spirale.
    repeat 200 [
        forward repcount
        left 121 ]
end

home clearscreen
penup forward 170 pendown

spirale
```

Il risultato è visibile in figura. Si può osservare il disegno stilizzato di una tartaruga in alto a destra.

Siti di riferimento per LibreLogo:

- extensions.libreoffice.org/extension-center/librelogo
- www.numbertext.org/logo/librelogo.pdf
- librelogo.org
- en.wikipedia.org/wiki/LibreLogo

Python + pygraph

In questo caso il programma è:

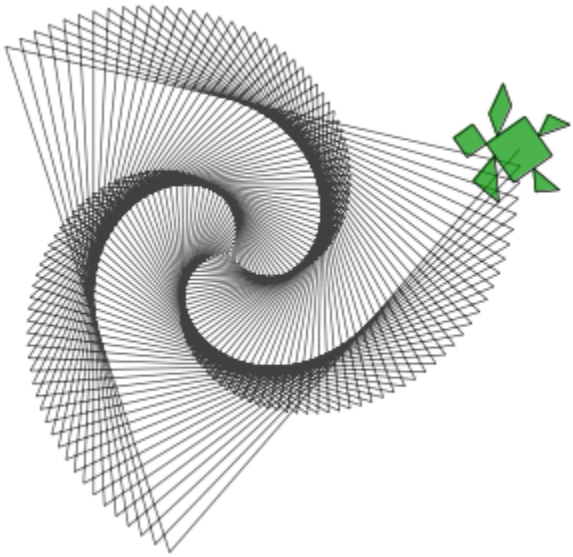


Figura 1.2: “Spirale” creata con LibreLogo.

```
from pyturtle import TurtlePlane, Turtle

def spi():
    """Disegna una "spirale" poligonale."""
    for lato in range(200):
        tina.forward(lato)
        tina.left(121)

tp = TurtlePlane()
tina = Turtle()
spi()

tp.mainloop()
```

Il risultato è quello visibile in figura in questo caso la “tartaruga”, cioè il puntatore grafico è ridotto ad un triangolino.

Siti di riferimento per Python - pygraph:

- www.python.org
- pygraph.readthedocs.org
- bitbucket.org/zambu/pygraph
- http://python4kids.net/

Scratch

Il linguaggio di programmazione Scratch non è del tutto testuale, ogni singolo comando è circondato da un blocco che lo identifica e che visualizza la funzione.

Lo stesso programma realizzato con Scratch è:

Il programmatore non scrive il programma ma assembla i blocchi corrispondenti alle istruzioni. Il risultato è una via di mezzo tra un programma e un diagramma di flusso.

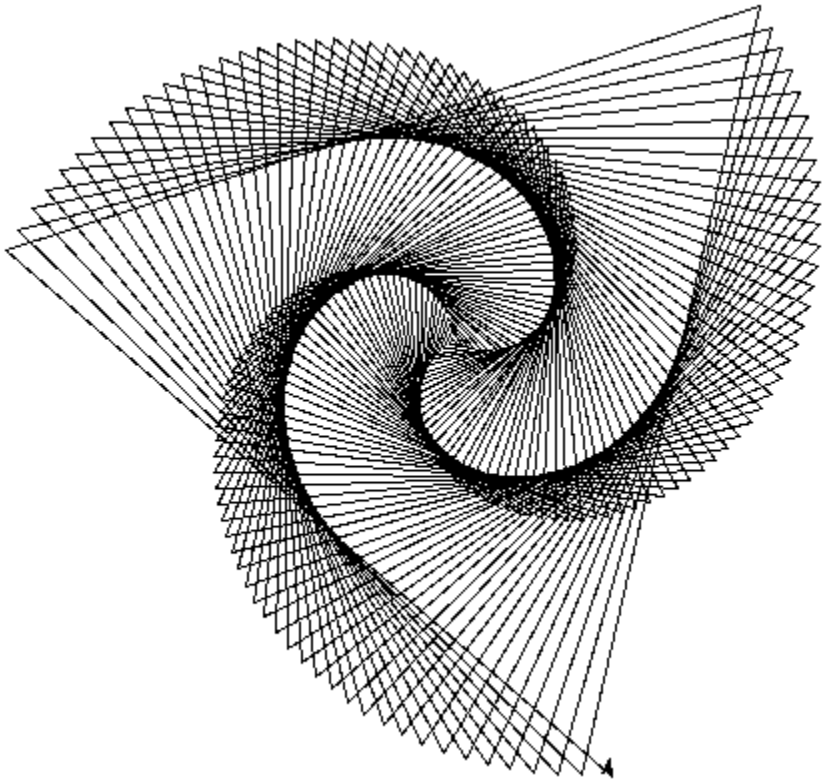


Figura 1.3: “Spirale” creata con pyturtle.

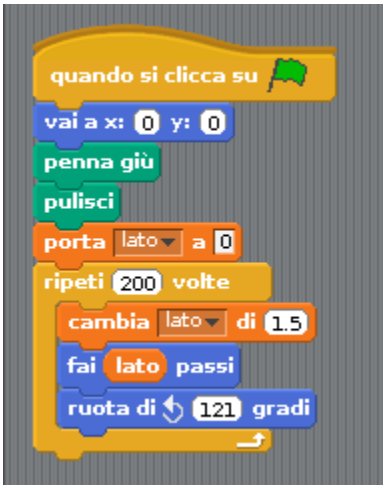


Figura 1.4: Programma scritto in Scratch.

Il risultato è quello riprodotto in figura qui la "tartaruga" è un oggetto *sprite* che può avere diverse rappresentazioni.

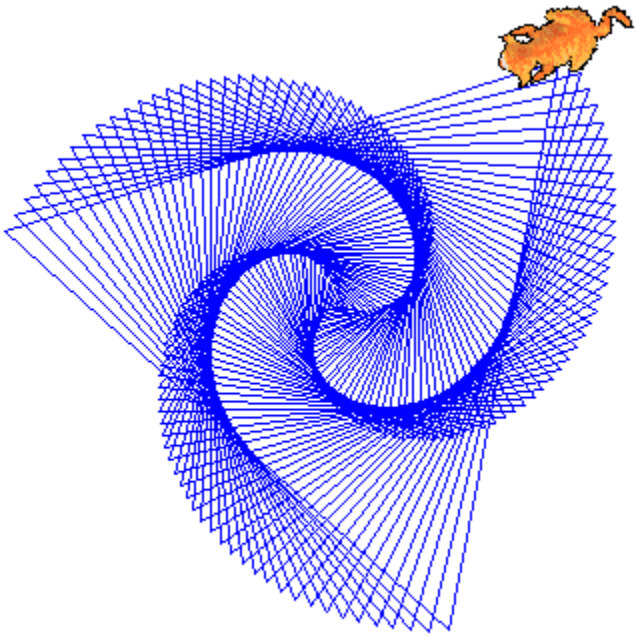


Figura 1.5: "Spirale" creata con Scratch.

Siti di riferimento per Scratch:

- scratch.mit.edu
- wiildos.wikispaces.com/scratch

Snap

Snap è un linguaggio a blocchi derivato da Scratch. Supera alcune pesanti limitazioni di quest'ultimo risultando molto più interessante per un uso nella didattica.

Permette di realizzare funzioni con parametri e è in grado di interfacciarsi con vari dispositivi esterni come:

- Lego NXT package by Connor Hudson
- Nintendo Wiimote package by Connor Hudson
- Finch and Hummingbird robots package by Tom Lauwers
- Parallax S2 robot package by Connor Hudson
- LEAP Motion by Connor Hudson
- speech synthesis by Connor Hudson
- Arduino package by Alan Yorinks

Siti di riferimento per Scratch:

- snap.berkeley.edu/snapsource/snap.html

1.3.6 Proposta

Concludendo, la proposta per questo laboratorio riguarda l'uso di:

- software libero,
- multi piattaforma,
- semplice da installare,
- linguaggio di programmazione:
  - il più possibile "trasparente",
  - adatto per un uso didattico,
- geometria della tartaruga.

Concretamente proporrò gli esempi che propongo possono essere realizzati in LibreLogo, Python + pygraph o Snap.

Nel resto di questo documento gli esempi saranno presentati in **Python + pygraph**.

CAPITOLO 2

pyturtle

2.1 La geometria della tartaruga

2.1.1 un possibile percorso didattico

Di seguito presento un possibile percorso di uso didattico della geometria della tartaruga realizzata con *Python* e *pyturtle*.

2.2 La geometria della tartaruga: procurarsi gli strumenti

Per prima cosa dobbiamo procurarci gli strumenti necessari:

- *Python*
- la libreria *pygraph*

2.2.1 Installare Python

Per prima cosa deve essere installato nel proprio sistema l'interprete del linguaggio di programmazione **Python**.

Per chi usa Windows, dal sito

www.python.org

ci si scarica l'ultima versione e la si installa.

Per gli altri, si chiede al proprio sistema di installare:

- *python*;
- *idle*;

2.2.2 Installare *pygraph*

Dal sito:

bitbucket.org/zambu/pygraph

si fa il dowload del pacchetto e lo si scompatta in una cartella.

Poi si copia la cartella *pygraph* e il file *pygraph.pth* nella cartella:

syte-packages del proprio Python.

Per Windows la cartella dovrebbe trovarsi nel percorso:

C:python3.3Libsyte-packages

2.2.3 I comandi di base

Idle

Il modo più semplice per scrivere un programma in Python è quello di usare l'interfaccia *Idle*.

Per cui dal menu-programmi-Python, si avvii *Idle*.

Idle ci permette di dare dei comandi e di vederne il risultato alla pressione del tasto <*Invio*>.

Ad esempio possiamo dare il comando:

```
fa qualcosa!
```

```
  File "<ipython-input-3-0ab3e73963c7>", line 1
    fa qualcosa!
              ^
SyntaxError: invalid syntax
```

In questo caso otteniamo un errore.

Un comando che dovrebbe capire è:

```
print(5)
```

```
5
```

questa volta è andato...

Al posto di 5 possiamo scrivere un'espressione complessa quanto vogliamo. Se è corretta verrà eseguita e verrà stampato il risultato.

Provate.

```
print(2**100)
```

```
1267650600228229401496703205376
```

Possiamo anche osservare che l'aritmetica dei numeri interi di Python prevede numeri limitati solo dalle capacità del computer. Pyton è in grado di calcolare anche 2^1000.

```
print(2**1000)
```

```
10715086071862673209484250490600018105614048117055336074437503883703510511249361224931983788156958581
```

Oltre ai numeri interi Python opera anche con altri oggetti primitivi:

- interi;
- numeri con la virgola;
- stringhe;
- insiemi;
- tuple;

- liste;
- ...

Giusto per curiosità possiamo anche vedere che Python è in grado di fare operazioni piuttosto strane:

```
print('casa')
print('matta')
print('casa' + 'matta')
print('ciao')
print('ciao ' * 7)
```

```
casa
matta
casamatta
ciao
ciao ciao ciao ciao ciao ciao ciao
```

2.2.4 Altri problemi

1. Calcola la somma dei primi 20 numeri naturali.
2. Calcola il prodotto dei naturali dall'uno al venti.
3. Calcola l'area di un trapezio che ha: B = 15.3, b = 11.4, e h = 21.3
4. Confronta l'area ottenuta al punto precedente con le aree che ottieni diminuendo di un decimo o aumentando di un decimo le misure.
5. Fa calcolare a Python la soluzione di un problema di geometria.

2.3 La geometria della tartaruga: il primo programma

Ma basta divagazioni, presa confidenza con qualche comando Python passiamo a scrivere il nostro primo programma con la geometria della tartaruga.

Dal menu File sciegliamo New window.

Viene creata una finestra vuota dove scrivere il nostro programma.

Per prima cosa salviamo il programma con un nome **che termini con '.py'**.

Poi eseguiamolo premendo il tasto <*F5*>.

Se non compare nessuno strano messaggio vuol dire che non ci sono errori (e come potrebbero essercene dato che non abbiamo ancora scritto niente?).

Bene, incominciamo a riempire il nostro programma.

2.3.1 I commenti

Come prime istruzioni scriviamo qualcosa che non faccia assolutamente niente. In ogni buon programma devono esserci dei commenti e noi iniziamo il programma scrivendo, come commento, la data, il nostro nome e un titolo del nostro lavoro:

"Primo programma"

Per indicare a Python che la parte di riga che segue è un commento, si usa il carattere: *#*.

```
# 6 giugno 2014
# Daniele Zambelli
# Primo programma del corso Pas
```

Come prima eseguiamo il programma premendo <*f5*>. Se tutto fila liscio non dovrebbe succedere niente.

Se appaiono delle scritte rosse dobbiamo cercare di capire cosa Python tenta di dirci leggendo l'ultima linea.

2.3.2 Lettura delle librerie

Python è un programma di uso generale, ma noi vogliamo fargli prodirre la geometria della tartaruga. Per ottenere questo dobbiamo dire al programma di leggere la libreria che contiene gli oggetti negessari:

```
import pyturtle as pt
```

Eseguiamo il programma: <*F5*> e per prima cosa controlliamo che non siano apparsi strani messaggi rossi.

Ora cerchiamo di capire l'istruzione:

Viene letta la libreria e viene associata all'abbreviazione *pt*.

Il programma funziona, ma è piuttosto deludente: non appare ancora nulla!

Il nostro secondo comando produrrà un effetto visibile sullo schermo:

Vogliamo che venga creato un piano su cui far muovere delle tartarughe. A questo piano vogliamo dare un nome per poterci riferire a lui in seguito.

2.3.3 Creazione degli elementi di base

```
foglio = pt.TurtlePlane()
```

Eseguendo il programma (al solito con la pressione del tasto *F5*>, ma non lo dirò più), Appar una finestra vuota.

La finestra non risponde ai comandi del mouse, non si riesce neppure a chiudere.

Per renderla attiva dobbiamo comandarglielo:

```
foglio.mainloop()
```

Questo comando dovrà restare l'ultimo del nostro programma per cui ci portiamo tra questi due comandi e riprendiamo a scrivere...

Per poter far disegnare una figura alla tartaruga abbiamo bisogno di... una **tartaruga**!

Dovremo anche darle un nome, come abbiamo fatto per il piano. E allora creiamola. Il programma diventa:

```
import pyturtle as pt       # legge la libreria 'pyturtle' chiandola 'pt'
foglio = pt.TurtlePlane()   # crea un paese delle tartarughe
tina = pt.Turtle()          # crea una tartaruga

foglio.mainloop()           # rende attiva la finestra
```

2.3.4 I primo problema

Se tutto è andato bene, nessuno strano messaggio rosso, siamo pronti per concentrarci sulla geometria della tartaruga.

I comandi di base della geometria della tartruga sono:

- *forward(<num>)*;
- *back(<num>)*;
- *left(<num>)*;
- *right(<num>)*;
- *penup()*;
- *pendown()*;

Come primo problema facciamo disegnare a tina: un quadrato;

```
import pyturtle as pt       # legge la libreria 'pyturtle' chiandola 'pt'
foglio = pt.TurtlePlane()   # crea un paese delle tartarughe
tina = pt.Turtle()          # crea una tartaruga

tina.forward(50)        # va avanti di 50 passi
tina.left(90)           # gira a sinistra di 90 gradi
tina.forward(50)        # ...
tina.left(90)
tina.forward(50)
tina.left(90)
tina.forward(50)
tina.left(90)

foglio.mainloop()           # rende attiva la finestra
```

2.3.5 Altri problemi

1. Disegna le seguenti figure:
2. una bandierina;
3. una casetta;
4. una barchetta.
5. Inventa un semplice disegno e realizzalo.
6. Realizza alcuni semplici disegni nello stesso foglio.

2.4 La geometria della tartaruga: strutture di controllo

Nel primo programma che abbiamo costruito abbiamo utilizzato la struttura base di ogni programma: la **sequenza**.

Ogni programma può essere pensato come una sequenza di istruzioni, ma ci sono situazioni nelle quali la sequenza non è comoda.

In questo capitolo vedremo altre due strutture di controllo:

- l'iterazione,
- le funzioni,
- la selezione.

2.4.1 L'iterazione

Se voglio disegnare un quadrato devo ripetere 4 volte una coppia di comandi, ma se invece di un quadrato volessi far disegnare un "ottantagono"? La cosa non sarebbe più complessa, ma più noiosa sì!

Gli informatici, che sono ancora più pigri dei matematici, hanno inventato dei modi per evitare di ripetere istruzioni. Sono le struttre di *iterazione*.

In Python come negli altri linguaggi ci sono diverse struttre di controllo, ma noi vedremo l'istruzione *for*.

La sua sintassi è:

```
for <variabile> in range(<numero>):
    <istruzioni>
```

Un esempio forse può chiarire meglio il suo funzionamento:

```
for contatore in range(5):
    print(contatore)
```

La variabile *contatore* assume tutti i numeri compresi tra *0* e *5*, con *5* escluso, e ogni volta viene eseguito il blocco di istruzioni che segue il carattere *:*.

In questo esempio:

- *contatore* è il nome della variabile,
- *5* è il numero di ripetizioni,
- *print(contatore)* è il blocco di istruzioni che viene ripetuto.

Nota: in realtà le cose sono un po' più complesse, ma, per ora, possiamo accontentarci di questa spiegazione.

Vogliamo scrivere un programma che disegni un quadrato usando l'iterazione.

Per prima cosa dobbiamo scrivere i vari comandi di base...

Ma invece di riscriverli (la pigrizia!), usiamo un barbatrucco:

1. salviamo il programma precedente con un nuovo nome, ricordandoci che deve terminare con *.py*
2. svuotiamolo da dutte le istruzioni tranne quelle di base;
3. modifichiamo il titolo: "poligoni";
4. eseguiamolo per vedere se è tutto a posto;

Ora scriviamo un programma che disegni un quadrato usando l'iterazione di Python:

```
import pyturtle as pt       # legge la libreria 'pyturtle' chiandola 'pt'
foglio = pt.TurtlePlane()   # crea un paese delle tartarughe
tina = pt.Turtle()          # crea una tartaruga

for cont in range(4):       # ripete 4 volte il blocco seguente
    tina.forward(20)        # va avanti di 20 passi
    tina.left(90)           # gira a sinistra di 90 gradi

foglio.mainloop()           # rende attiva la finestra
```

2.4.2 Le funzioni

Se in un programma avessi bisogno di più quadrati, dovrei copiare in più punti le stesse tre righe. Questo è contrario all'etica della pigrizia, gli informatici hanno inventato un modo per associare un blocco di istruzioni ad un nome.

In Python la sintassi è:

```
def <none funzione>():
    <blocco di istruzioni>
```

In pratica possiamo associare alla parola "quadrato" le istruzioni per disegnare un quadrato:

```
def quadrato():
    """Fa disegnare un quadrato a tina."""
    for cont in range(4):       # ripete 4 volte il blocco seguente
        tina.forward(20)        # va avanti di 20 passi
        tina.left(90)           # gira a sinistra di 90 gradi
```

Si può osservare che la seconda riga della funzione è costituita da una stringa detta "docstring". Questa stringa serve per documentare la funzione, non è obbligatoria, ma è fortemente raccomandata.

Il programma precedente può dunque diventare:

```
import pyturtle as pt       # legge la libreria 'pyturtle' chiandola 'pt'
foglio = pt.TurtlePlane()   # crea un paese delle tartarughe
tina = pt.Turtle()          # crea una tartaruga

def quadrato():
    """Fa disegnare un quadrato a tina."""
    for cont in range(4):       # ripete 4 volte il blocco seguente
        tina.forward(20)        # va avanti di 20 passi
        tina.left(90)           # gira a sinistra di 90 gradi

foglio.mainloop()           # rende attiva la finestra
```

Ma questo programma non disegna niente!

Cosa è successo?

Se lo osserviamo bene possiamo accorgerci che abbiamo definito come disegnare un quadrato, ma non gli abbiamo mai detto di disegnarlo.

dobbiamo aggiungere l'istruzione:

quadrato()

2.4.3 Struttura di un programma

Approfittiamo per ristrutturare il programma e dargli la forma che hanno di solito i programmi seri:

1. Intestazione;
2. lettura delle librerie;
3. definizioni;
4. programma principale.

```
# 6 giugno 2014
# Daniele Zambelli
# Primo programma del corso Pas

# lettura delle librerie
import pyturtle as pt       # legge la libreria 'pyturtle' chiandola 'pt'

# definizione delle funzioni
def quadrato():
```

```
    """Fa disegnare un quadrato a tina."""
    for cont in range(4):       # ripete 4 volte il blocco seguente
        tina.forward(20)        # va avanti di 20 passi
        tina.left(90)           # gira a sinistra di 90 gradi

# programma principale
foglio = pt.TurtlePlane()       # crea un paese delle tartarughe
tina = pt.Turtle()              # crea una tartaruga
quadrato()                      # disegna un quadrato

foglio.mainloop()               # rende attiva la finestra
```

Possiamo richiamare una funzione anche dall'interno di un'altra funzione. Per esempio potremmo definire *bandierina* come un'asta seguita da un quadrato:

```
def bandierina():
    """Disegna una bandierina quadrata."""
    tina.forward(40)
    quadrato()
    tina.back(40)
```

Aggiungi questa funzione al programma (dove?) e fa disegnare una bandierina.

Poi usando l'iterazione fa disegnare una rosa di bandierine.

2.4.4 La selezione

Alle volte in un programma bisogna scegliere in base a una qualche condizione, se eseguire un blocco di codice o un altro.

La struttura per fare ciò è la selezione. La sua sintassi è:

```
if <condizione>:
    <istruzioni se vera>
[else:
    <istruzioni se falsa>]
```

Nota: la parte tra parentesi quadre è facoltativa.

Anche qui un esempio può chiarire:

```
numero = int(input('scrivi un numero: '))
if (numero % 2) == 0:
    print(numero, 'è pari')
else:
    print(numero, 'è dispari')
```

Questa struttura di controllo ci sarà utile, nella geometria della Tartaruga, quando costruiremo funzioni ricorsive.

2.4.5 Altri problemi

1. Senza cancellare i poligoni precedentemente disegnati, disegna anche altri poligoni regolari:
2. un triangolo;
3. un pentagono;
4. un esagono;

5. un ettagono;
6. ...
7. Disegna un quadrato formato da quattro quadrati.
8. Disegna il simbolo di pericolo radiazioni.
9. Ripeti più volte un percorso a casaccio.
10. Realizza bandierine triangolari o con altre forme.
11. Disegna una fila di bandierine.
12. Definisci una funzione che disegni un percorso a casaccio, poi richiamala all'interno di un ciclo.
13. Scrivi un programma che chiede un numero compreso tra 2 e 5 e disegna un poligono con il numero di lati digitato.

2.5 La geometria della tartaruga: i parametri

Abbiamo visto un modo per scrivere una porzione di codice e riutilizzarla in punti diversi del programma. Ma nelle situazioni concrete sorge spesso l'esigenza di eseguire una porzione di codice con qualche piccola variazione.

Ad esempio noi potremmo avere bisogno di disegnare quadrati grandi e piccoli.

Scrivamo un programma che disegni due quadrati, uno grande, "quadratone" e uno piccolo, "quadratino".

```
# 6 giugno 2014
# Daniele Zambelli
# Programma che disegna quadrati di diverso lato

# lettura delle librerie
import pyturtle as pt      # legge la libreria 'pyturtle' chiandola 'pt'

# definizione delle funzioni
def quadratone():
    """Fa disegnare un quadratone a tina."""
    for cont in range(4):      # ripete 4 volte il blocco seguente
        tina.forward(100)      # va avanti di 100 passi
        tina.left(90)          # gira a sinistra di 90 gradi

def quadratino():
    """Fa disegnare un quadratino a tina."""
    for cont in range(4):      # ripete 4 volte il blocco seguente
        tina.forward(20)       # va avanti di 20 passi
        tina.left(90)          # gira a sinistra di 90 gradi

# programma principale
foglio = pt.TurtlePlane()      # crea un paese delle tartarughe
tina = pt.Turtle()             # crea una tartaruga
quadratone()                   # disegna un quadratone
quadratino()                   # disegna un quadratino

foglio.mainloop()              # rende attiva la finestra
```

Funziona, ma le due funzioni sono quasi uguali e questo contraddice il principio di pigrizia.

Osservate le funzioni e evidenziate le differenze: a parte il nome l'unica differenza è il numero che indica la lunghezza del lato del quadrato.

Gli informatici hanno inventato un meccanismo per far disegnare quadrati grandi o piccoli alla stessa funzione.

Al posto del numero che cambia scriviamo un nome:

```
def quadrato():
    """Fa disegnare un quadratone a tina."""
    for cont in range(4):       # ripete 4 volte il blocco seguente
        tina.forward(lato)      # va avanti di lato passi
        tina.left(90)           # gira a sinistra di 90 gradi
```

Ma non basta, la funzione deve sapere che ha bisogno di conoscere la lunghezza del lato. Il meccanismo usato dagli informatici è quello aggiungere dei parametri alla funzione, la sintassi è:

```
def <nome funzione>(<elenco dei parametri>):
    <istruzioni>
```

Nel nostro caso, aggiungiamo alla funzione *quadrato* il parametro *lato*:

```
def quadrato(lato):
    """Fa disegnare un quadratone a tina."""
    for cont in range(4):       # ripete 4 volte il blocco seguente
        tina.forward(lato)      # va avanti di lato passi
        tina.left(90)           # gira a sinistra di 90 gradi
```

Se ora chiamiamo la funzione quadrato come nei programmi precedenti otteniamo un errore che ci dice più o meno che quadrato ha un parametro e che deve essere chiamato passandogli un argomento, cioè un valore da associare al parametro.

La chiamata di questa funzione dovrà essere qualcosa di simile a:

```
quadrato(37)
```

La lunghezza del lato del quadrato viene così decisa non quando viene definita la funzione, ma quando viene chiamata. La funzione *quadrato* disegnerà quindi un numero enorme di quadrati diversi a seconda del valore dell'argomento passato alla funzione.

Il programma precedente diventa quindi:

```
# 6 giugno 2014
# Daniele Zambelli
# Programma che disegna quadrati di diverso lato

# lettura delle librerie
import pyturtle as pt      # legge la libreria 'pyturtle' chiandola 'pt'

# definizione delle funzioni
# Invece di cancellare le seguenti due funzioni, ormai inutili,
# le ho commentate così da tenere traccia dell'evoluzione
# del programma.
#
#def quadratone():
#    """Fa disegnare un quadratone a tina."""
#    for cont in range(4):      # ripete 4 volte il blocco seguente
#        tina.forward(100)      # va avanti di 100 passi
#        tina.left(90)          # gira a sinistra di 90 gradi
#
#def quadratino():
#    """Fa disegnare un quadratino a tina."""
#    for cont in range(4):      # ripete 4 volte il blocco seguente
#        tina.forward(20)       # va avanti di 20 passi
#        tina.left(90)          # gira a sinistra di 90 gradi
```

```
def quadrato(lato):
    """Fa disegnare a tina un quadrato dato il lato."""
    for cont in range(4):       # ripete 4 volte il blocco seguente
        tina.forward(lato)      # va avanti di 20 passi
        tina.left(90)           # gira a sinistra di 90 gradi

# programma principale
foglio = pt.TurtlePlane()       # crea un paese delle tartarughe
tina = pt.Turtle()              # crea una tartaruga
quadrato(100)                   # disegna un quadratone
quadrato(20)                    # disegna un quadratino

foglio.mainloop()               # rende attiva la finestra
```

2.5.1 Altri problemi

1. Disegna tre quadrati diversi in tre posizioni dello schermo.
2. Disegna 15 quadrati con lato da 10 a 150 uno dentro l'altro.
3. Disegna una spirale di quadrati.
4. Disegna una fila di quadrati. Ricordati di far tornare la tartaruga nella posizione iniziale.
5. Disegna una "coda" di quadrati, cioè una fila di quadrati non disposti in linea retta.
6. Scrivi le procedure che disegnano un triangolo e un quadrato con lato variabile. Confrontale. Scrivi la procedura che, dati *numlati* e *lato*, disegni un qualunque poligono regolare.
7. Usa la procedura precedente per realizzare una composizione grafica a fantasia.

2.6 La geometria della tartaruga: risolvere un problema

Risolvere problemi è una delle principali attività della nostra vita. Insegnare metodi efficaci per risolvere probremi dovrebbe essere uno dei principali obiettivi del nostro insegnamento.

2.6.1 Metodi di soluzione di problemi

Risolvere problemi è un'attività complessa quindi non esiste **il metodo** di soluzione dei problemi.

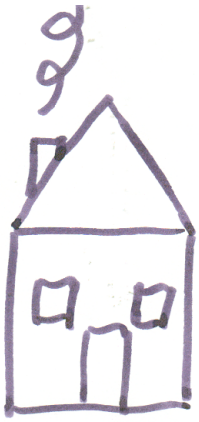
La geometria della tartaruga può aiutarci a imparare i metodi "top down" e "bottom up".

2.6.2 Top Down

È il metodo più razionale, va dalla soluzione del problema generale alla soluzione delle sue componenti. È adatto alla ricerca delle soluzioni di un problema complesso, si presta bene all'uso in gruppi dove il lavoro viene suddiviso in parti. Ma partiamo con un esempio.

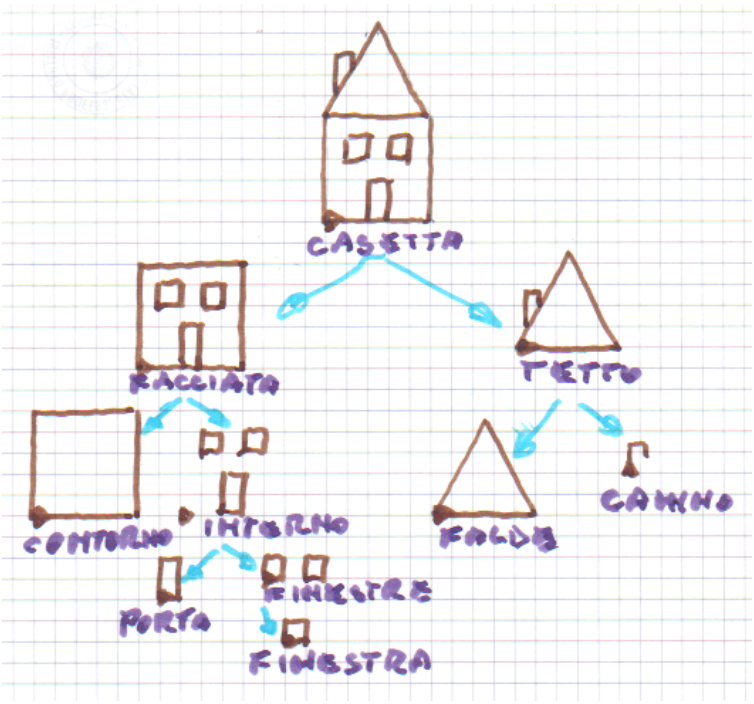
2.6.3 Il problema

Voglio far disegnare a Tartaruga una casetta con porte finestre e camino. Qualcosa che assomigli a questo:



2.6.4 L'analisi

Prima di scrivere un programma che disegni la casetta, dobbiamo analizzarlo. L'analisi consiste nel suddividere il problema in parti più semplici e ripetere questa operazione in modo ricorsivo finché si arriva a parti elementari. Nel nostro caso una possibile analisi è:



Nell'analisi dobbiamo anche precisare: * Le dimensioni delle varie parti, nel nostro caso: un quadretto = 10 passi; * La posizione dove parte e dove arriva Tartaruga, normalmente queste due posizioni devono coincidere. * Un nome per ogni parte.

2.6.5 Grafo ad albero

Osservazione: l'analisi fatta in questo modo ha prodotto un oggetto che è studiato dalla matematica: un **grafo ad albero**.

Le varie parti della casetta costituiscono i *nodi*.

Le frecce costituiscono i *rami*.

Il nodo da cui parte tutto l'albero si chiama *radice*.

I nodi da cui non parte alcun ramo si chiamano *foglie*.

2.6.6 La soluzione

Fatta l'analisi possiamo iniziare a scrivere il nostro programma:

```
# 13 giugno 2014
# Daniele Zambelli
# Casetta

# lettura delle librerie
import pyturtle as pt          # legge la libreria 'pyturtle' chiandola 'pt'

# programma principale
foglio = pt.TurtlePlane()      # crea un paese delle tartarughe
tina = pt.Turtle()             # crea una tartaruga

foglio.mainloop()              # rende attiva la finestra
```

Lo eseguiamo per assicurarci che non ci siano errori, ma già ci aspettiamo che non faccia un gran ché.

Passiamo a costruire la casetta, avendo deciso di seguire il metodo Top - Down, partiamo dal problema generale per scendere verso le varie componenti.

Quindi per prima cosa scriviamo la procedura casetta. Dall'analisi vediamo che è composta da due parti: * *facciata*, * *tetto*,

La funzione dovrà disegnare la facciata, poi spostarsi e disegnare il tetto, ma non è finita, poi ritornare dove era partita:

```
def casetta():
    """Disegna una casetta."""
    facciata()
    # sposta tina nella posizione adatta per disegnare il tetto
    tetto()
    # rimetti a posto tina
```

Guardando l'analisi non è difficile trovare quali istruzioni vanno scritte al posto dei commenti.

Il programma diventa:

```
# 13 giugno 2014
# Daniele Zambelli
# Casetta

# lettura delle librerie
import pyturtle as pt          # legge la libreria 'pyturtle' chiandola 'pt'

def casetta():
    """Disegna una casetta."""
    facciata()
    tina.left(90)
    tina.forward(50)
    tina.right(90)
    tetto()
```

```
    tina.left(90)
    tina.back(50)
    tina.right(90)

# programma principale
foglio = pt.TurtlePlane()       # crea un paese delle tartarughe
tina = pt.Turtle()              # crea una tartaruga

foglio.mainloop()               # rende attiva la finestra
```

Lo eseguiamo... che delusione, non disegna proprio niente. Come mai?

Abbiamo definito casetta ma non abbiamo chiesto a Python di eseguire questa funzione, facciamolo:

```
# 13 giugno 2014
# Daniele Zambelli
# Casetta

# lettura delle librerie
import pyturtle as pt           # legge la libreria 'pyturtle' chiandola 'pt'

def casetta():
    """Disegna una casetta."""
    facciata()
    tina.left(90)
    tina.forward(50)
    tina.right(90)
    tetto()
    tina.left(90)
    tina.back(50)
    tina.right(90)

# programma principale
foglio = pt.TurtlePlane()       # crea un paese delle tartarughe
tina = pt.Turtle()              # crea una tartaruga
casetta()                       # disegna una casetta

foglio.mainloop()               # rende attiva la finestra
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)

<ipython-input-12-9945567f031a> in <module>()
     20 foglio = pt.TurtlePlane()       # crea un paese delle tartarughe
     21 tina = pt.Turtle()              # crea una tartaruga
---> 22 casetta()                       # disegna una casetta
     23
     24 foglio.mainloop()               # rende attiva la finestra


<ipython-input-12-9945567f031a> in casetta()
      8 def casetta():
      9     """Disegna una casetta."""
---> 10     facciata()
     11     tina.left(90)
     12     tina.forward(50)


<ipython-input-6-1639e4a1c7a9> in facciata()
```

```
     40     """Disegna la facciata della casetta."""
     41     contorno()
---> 42     interno()
     43
     44 def contorno():


NameError: name 'interno' is not defined
```

Accidenti, otteniamo un errore!

L'ultima riga del messaggio di errore ci dà un'indicazione: *facciata* non è definita... Già, potevamo aspettarcelo, gli abbiamo insegnato a fare *casetta* ma non a fare *facciata*.

Rimettiamoci al lavoro e definiamo la nuova funzione. *facciata* risulta più semplice dato che le due parti da cui è composta hanno la stessa partenza, quindi non servono spostamenti intermedi. Aggiungiamo questa nuova funzione ed eseguiamo il programma:

```
# 13 giugno 2014
# Daniele Zambelli
# Casetta

# lettura delle librerie
import pyturtle as pt          # legge la libreria 'pyturtle' chiandola 'pt'

def casetta():
    """Disegna una casetta."""
    facciata()
    tina.left(90)
    tina.forward(50)
    tina.right(90)
    tetto()
    tina.left(90)
    tina.back(50)
    tina.right(90)

def facciata():
    """Disegna la facciata della casetta."""
    contorno()
    interno()

# programma principale
foglio = pt.TurtlePlane()      # crea un paese delle tartarughe
tina = pt.Turtle()             # crea una tartaruga
casetta()                      # disegna una casetta

foglio.mainloop()              # rende attiva la finestra
```

Facendoci guidare dagli errori che man mano otteniamo, possiamo procedere a completare il programma.

Prima di procedere possiamo aggiungere 3 funzioni di supporto: *quadrato(lato)*, *triangolo(lato)*, *rettangolo(base, altezza)*:

Otteniamo ancora un errore, ma finalmente *tina* si è degnata di disegnare qualcosa.

L'interno della facciata è il più complesso sarà qualcosa di questo tipo:

```
def interno():
    """Dsegna l'interno della facciata."""
    # spostati in avanti
```

```
    # disegna la porta
    # spostati indietro e in alto senza tracciare segni
    # disegna le finestre
    # ritorna dove eri partita senza tracciare segni
```

Facendociguidare dagli errori, completiamo il disegno della casetta.

2.6.7 Il programma completo

```
# 13 giugno 2014
# Daniele Zambelli
# Casetta

# lettura delle librerie
import pyturtle as pt           # legge la libreria 'pyturtle' chiandola 'pt'

def quadrato(lato):
    """Fa disegnare a tina un quadrato dato il lato."""
    for cont in range(4):       # ripete 4 volte il blocco seguente
        tina.forward(lato)      # va avanti di lato passi
        tina.left(90)           # gira a sinistra di 90 gradi

def triangolo(lato):
    """Fa disegnare a tina un triangolo equilatero dato il lato."""
    for cont in range(3):       # ripete 3 volte il blocco seguente
        tina.forward(lato)      # va avanti di lato passi
        tina.left(120)          # gira a sinistra di 120 gradi

def rettangolo(base, altezza):
    """Fa disegnare a tina un rettangolo dato i lati."""
    for cont in range(2):
        tina.forward(base)
        tina.left(90)
        tina.forward(altezza)
        tina.left(90)

def casetta():
    """Disegna una casetta."""
    facciata()
    tina.left(90)
    tina.forward(50)
    tina.right(90)
    tetto()
    tina.left(90)
    tina.back(50)
    tina.right(90)

def facciata():
    """Disegna la facciata della casetta."""
    quadrato(5)                 # contorno
    interno()

def interno():
    """Dsegna l'interno della facciata."""
    tina.forward(20)            # spostati in avanti
    rettangolo(10, 20)          # disegna la porta
    tina.up()                   # spostati indietro e in alto
```

```
    tina.back(10)               # senza tracciare segni
    tina.left(90)
    tina.forward(30)
    tina.right(90)
    tina.down()
    finestre()                  # disegna le finestre
    tina.up()                   # ritorna dove eri partita
    tina.back(10)               # senza tracciare segni
    tina.right(90)
    tina.forward(30)
    tina.left(90)
    tina.down()

def finestre():
    """Disegna la porta della casetta."""
    quadrato(10)                # una finestra
    tina.up()                   # spostamento
    tina.forward(20)
    tina.down()
    quadrato(10)                # l'atra finestra
    tina.up()                   # ritorna dove eri partito
    tina.back(20)
    tina.down()

def tetto():
    """Disegna il tetto."""
    triangolo(50)               # falde del tetto
    tina.left(60)
    tina.forward(10)
    tina.left(30)
    camino()                    # camino
    tina.right(30)
    tina.back(10)
    tina.right(60)

def camino():
    """Disegna il camino."""
    tina.forward(20)            # avanti
    tina.right(90)
    tina.forward(7)
    tina.right(90)
    tina.forward(5)
    tina.back(5)                # ritorna per la stessa strada
    tina.left(90)
    tina.back(7)
    tina.left(90)
    tina.back(20)

# programma principale
foglio = pt.TurtlePlane()       # crea un paese delle tartarughe
tina = pt.Turtle()              # crea una tartaruga
casetta()                       # disegna una casetta

foglio.mainloop()               # rende attiva la finestra
```

2.6.8 Bottom up

Il metodo Bottom-up parte dalla stessa analisi, arriva allo stesso risultato (programma), ma segue un percorso diverso.

Invece che scrivere la funzione principale (radice) e farsi guidare dagli errori, si può partire dalla diverse componenti (foglie) realizzarle una alla volta ontrollando che rispondano alle specifiche dell'analisi e metterle insieme per ottenere il risultato desiderato.

2.6.9 Top down e problemi di matematica

Normalmente per risolvere problemi di matematica, nella scuola, si propone il meodo bottom up: si parte dai dati, si trova qualcosa di utile e via via cisi avvicina all'incognita. Questo metodo risulta di poco aiuto nella *ricerca* della soluzione.

Il metodo top down può fornire una guida preziosa nella soluzione dei problemi.

Prendiamo come esempio un problema di geometria solida:

Il volume di una piramide è $1000cm^3$ e la base è un rombo il cui perimetro è $52cm$ e una diagonale è di $24cm$. Calcola la misura dell'altezza.

Soluzione

$altezza = \frac{3*volume}{sup.base}$

$sup.base = \frac{diag.1*diag.2}{2}$

$diag.1 = 2 * semid.1$

$semid.1 = \sqrt{lato^2 + semid.2^2}$

$semid.2 = \frac{diag.2}{2} = \frac{24cm}{2} = 12cm$

$lato = \frac{perimetro}{4} = \frac{52}{4} = 13cm$

$semid.1 = \sqrt{lato^2 + semid.2^2} = \sqrt{13^2 + 12^2} = 5cm$

$diag.1 = 2 * semid.1 = 2 * 5cm = 10cm$

$sup.base = \frac{diag.1*diag.2}{2} = \frac{10cm*24cm}{2} = 120cm^2$

$altezza = \frac{3*volume}{sup.base} = \frac{3*1000cm^3}{120cm^2} = 25cm$

2.6.10 Altri problemi

1. Disegna una casetta con un'altra forma.
2. Disegna un albero.
3. Disegna un fiore.
4. Disegna una farfalla.
5. Riunisci alcuni elementi definiti sopra in un unico disegno.
6. Disegna una casetta a più piani.
7. Disegna una casetta con un numero variabile di piani: un grttacielo.

conclusione

3.1 Conclusione

Di seguito riporto cosa è stato trattato in questo corso e il tema della verifica in itinere.

3.1.1 Argomenti svolti

- matematica, informatica e didattica;
- libertà di insegnamento e software libero, conoscenze libere e licenze Creative Commons;
- introduzione a un linguaggio di programmazione con applicazioni alla geometria.

3.1.2 Verifica in itinere

Ricordando che in questo corso abbiamo parlato:

- di matematica, informatica e didattica,
- di libertà della conoscenza e licenze libere,
- dell'uso di un linguaggio di programmazione,

Produci una mappa o un breve testo in cui questi concetti sono connessi ad una attività didattica.

CAPITOLO 4

Indici e tavole

- *genindex*
- *modindex*
- *search*