LEONARDO FAVARIO
ANGELO RAFFAELE MEO
AURORA MARTINA
CLOTILDE MORO
MARIO SCOVAZZI

PASSO DOPO PASSO IMPARIAMO A PROGRAMMARE CON PYTHON

PREMESSA

Abbiamo realizzato questa versione del nostro manuale , **dedicata agli studenti della scuola primaria e secondaria di primo grado,** sperando che possa costituire una facile, e accattivante, base per imparare i fondamenti logici della programmazione.

Convinti che i concetti logici della programmazione informatica, opportunamente mediati, possano e debbano far parte del bagaglio culturale dei ragazzi, abbiamo ideato una serie di schede che, passo dopo passo, consentiranno agli alunni "in erba" di avviare un percorso di conoscenza e di esperienza che li porterà, un po' più avanti nel tempo, ad acquisire le capacità necessarie a padroneggiare *in maniera consapevole* il mezzo informatico.

Nella **strutturazione del percorso** di conoscenza, così come nell'organizzazione dei contenuti e nella scelta della veste grafica, sono state adottate soluzioni utili a consentire un uso del manuale quanto più autonomo da parte:

- *degli alunni,* che potranno scaricare liberamente i materiali dalla rete
- *dei docenti* che potranno liberamente disporne per le proprie lezioni

Il manuale è stato creato anche per essere fruito on line sulle piattaforme per la didattica oggi più diffuse.

- **Sul portale FARE (fare.polito.it) sono inoltre disponibili e liberamente scaricabili altri materiali utili ad approfondire i temi qui trattati.**

La scelta del linguaggio di programmazione Python non è casuale. Esso risponde infatti a molti requisiti da noi ritenuti fondamentali. Tra questi:

- libero
- didatticamente adeguato
- attuale
- possibilità di utilizzo e di approfondimento nel tempo
- diffusione nella pratica informatica
- dimensione e importanza delle librerie associate

Questa stesura riporta l'adeguamento del linguaggio alla versione 3.X.

Vi consigliamo di collegarvi direttamente al sito:

fare.polito.it/python

Al fondo del libretto sono riportati i link dei siti dai quali sono tratte le immagini utilizzate nel testo.

Gli Autori

Lezione 1: usare Python come calcolatrice

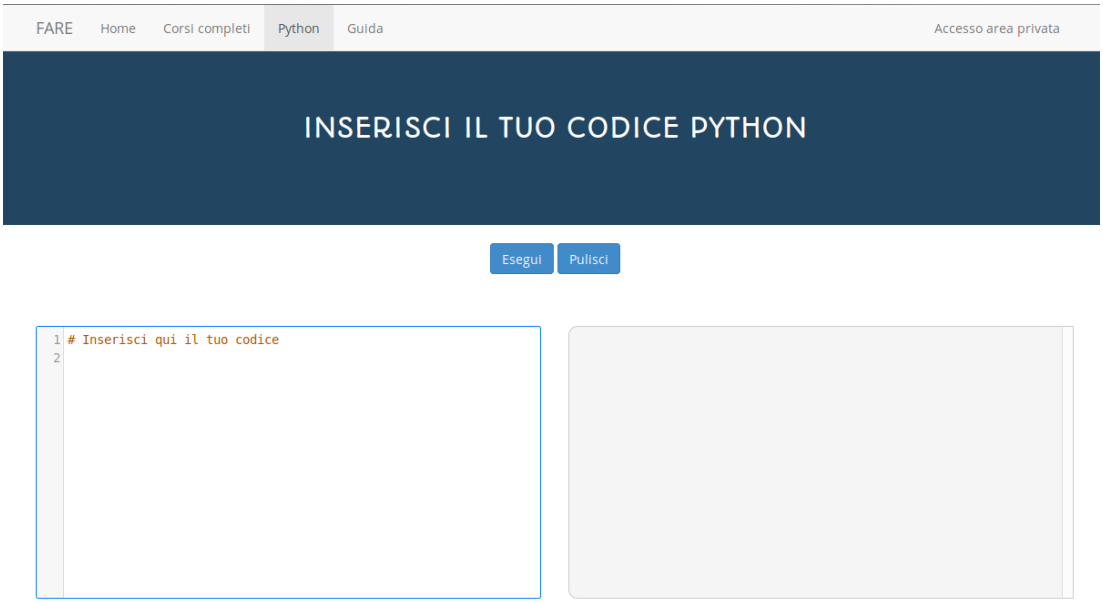
PROGRAMMARE UN COMPUTER è l'arte di far fare a un computer ciò che vogliamo.

Per ottenere questo risultato dobbiamo dare al computer, secondo un certo ordine, delle istruzioni. **Il programma e' l'insieme ordinato di queste istruzioni.**

Per prima cosa dobbiamo saper accendere il computer e "parlare" con lui, in modo da riuscire a comunicargli queste istruzioni e a fargliele eseguire correttamente.

Un modo rapido per chiedere al calcolatore di collegarci a Python consiste nel fare "clic" sulla icona del programma chiamato "browser" che serve per "navigare" sulla Rete Internet, ossia per collegarci a uno delle molte migliaia di siti oggi disponibili e, successivamente, nello scrivere nella riga in alto (nella barra di ricerca): **fare.polito.it/python** che è il nome del sito dove risiede il nostro programma Python.

A questo punto comparirà questa immagine:



Nella finestra di sinistra si scriveranno gli ordini per Python e quindi questa finestra sarà chiamata "finestra degli ordini"; nella finestra di destra, che sarà chiamata "finestra delle risposte", compariranno i risultati del calcolo ed eventuali messaggi nel caso si siano commessi degli errori.
A questo punto possiamo iniziare a lavorare con Python.

Se vuoi chiedergli quanto fa 2+3, devi scrivere: print(2+3) nella finestra degli ordini. Poi clicchi sul tasto "esegui" nella barra dei menù in alto. Il computer ti risponderà facendo comparire il risultato (5) nella finestra delle risposte.
In questo modo puoi fare tutte le operazioni che vuoi, anche le più difficili.
La parola inglese "print" significa "stampa", ma viene usata anche nei programmi per chiedere al calcolatore di "visualizzare" (o farci vedere nella finestra dei risultati) il risultato di un calcolo o di un programma.

Ricorda che:
il simbolo + è il comando per eseguire una somma;
il simbolo − è l'ordine di eseguire una sottrazione;
il simbolo * indica una moltiplicazione;
i simboli / e // indicano la divisione.
Esempio:

print (15-11)
4

print (2*3)
6

print (15/5)
3

Ora impariamo anche l'uso del simbolo **che viene chiamato "**elevato a**" e che ha il significato mostrato nei seguenti esempi:
PRIMO ESEMPIO:
print(23)**
8
(ossia 2*2*2, tre volte)
SECONDO ESEMPIO:
print(24)**
16
(ossia 2*2*2*2, 4 volte)

Adesso prova tu ad esercitarti usando Python. Prova a scrivere:
print(10**2)
print(10**3)
print(3**2)
print(3**3)

poi clicca su "esegui" e controlla il risultato.

Come anticipato, possiamo usare due simboli diversi per la divisione. Ai due simboli diversi corrispondono operazioni un po' diverse, come mostrato nei seguenti esempi:

print (5/2) (e poi esegui) otterrai:
2.5

Verifica ancora:
print (15/4) (e poi esegui) otterrai:
3.75

Se invece scriviamo:

print (5//2)
2

print (5.0//2.0)
2.0
otteniamo la parte intera del risultato della divisione.

In sintesi, Python possiede due divisioni che si ottengono con i due simboli **/** e **//**. La prima divisione genera numeri con virgola, la seconda divisione produce la parte intera del risultato.

Se in una divisione ti interessa solo sapere il resto usa il simbolo **%:**
print (15%12)
3

Fai attenzione. Gli americani, quando usano i numeri decimali, usano il punto (.) e non la virgola. Anche noi dobbiamo usare il punto. Per conoscere il doppio di 2.75 dobbiamo scrivere:
print (2.75*2)

Si possono scrivere nella stessa istruzione tante operazioni una dopo l'altra che verranno eseguite nell'ordine con cui sono scritte, con qualche differenza. Infatti prima bisogna eseguire gli ****** e poi i ***** o **/** e infine i **+** o i **-**. **Ecco alcuni esempi.**

print (4+2*3)
10

print (6/2-1)
2

print(10+3*23)**

34 (Infatti il calcolatore prima calcola 2**3 =8, poi calcola 3*8 =24 e infine calcola 10+24 =34).

Per chiedere al calcolatore di eseguire le operazioni in un ordine ben preciso posso usare le parentesi tonde ( ).
Supponiamo, ad esempio, di voler calcolare il numero delle zampe degli animali che vivono nella casa di Mario, sapendo che ha 2 cani e 3 gatti. Devo scrivere:

print((2+3)*4)

Una coppia di parentesi può stare dentro una frase delimitata da altre parentesi, come quando devo fare un calcolo più complicato. Esempio:
print((20*(4+1)) / (3+1))

Riassumendo: Python segue le stesse regole della matematica per quanto riguarda l'ordine di esecuzione delle operazioni:
Prima le parentesi (partendo da quelle più interne), poi l'elevamento a potenza, poi moltiplicazione e divisione e infine somma e addizione. Quando due operatori hanno la stessa priorità si procede da sinistra verso destra.

Esercitiamoci un po'.

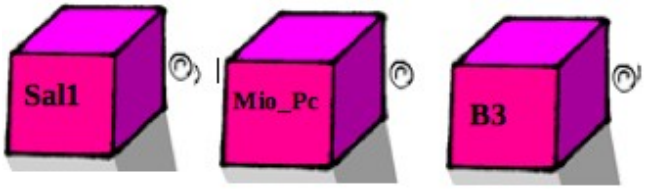
1.1-Esegui il calcolo dell' espressione:
4+4.5-(6*4/2)
1.2-Scrivi l'espressione per calcolare "quanti mesi hai".
1.3-Inventa un'espressione che dia come risultato 48 ed una che dia come risultato 11.
1.4-Supponiamo: per andare da casa di Sandrone a casa di Giulia ci sono 3 km e per andare da casa di Giulia a casa di Clotilde ci sono 4 km.
Scrivi un'espressione che calcoli quanti km deve fare Sandrone per andare a trovare Giulia e Clotilde e tornare a casa ripassando da casa di Giulia.
Poi calcola quanti km ci vogliono per andare a trovare i tuoi migliori amici e per tornare a casa.
1.5-Calcola l'area del ripiano del banco.
1.6-Calcola l'area della tua aula.

Hai dei dubbi o non ti ricordi qualche argomento e ti servirebbe un ripasso del libro?
Facendo "clic" sul pulsante in alto "apri il libro" si aprirà una finestra che chiameremo "finestra del libro" dove potrai liberamente consultare il testo del nostro manuale.
Al termine, clicca sul pulsante "chiudi il libro".

Lezione 2: Le Scatole

Istruire un calcolatore significa essenzialmente creare e usare degli oggetti. Tra questi oggetti quelli di uso più comune sono quelli che chiameremo scatole.

Le scatole vengono usate per contenere numeri, caratteri, parole o frasi.



Le nostre scatole sono del tutto simili a quelle che vediamo nella figura sopra.
Ogni scatola che noi creiamo deve avere un nome.
Il nome che assegniamo alla scatola è l'equivalente dell'etichetta sulla scatola .

Ovviamente dovremo scegliere dei nomi significativi per le nostre scatole per ricordarci a cosa servono.
Ad esempio: SCATOLA1, SCAT1, SAL1, SALAME, PIPPO, PIPPO4C, SCARPONIASI.
Sono validi anche nomi molto corti come: A, B, C, A1, B3 o lunghi come:

ILNOMEPIULUNGOCHEMIVIENEINMENTEPERILMIOCANE

I nomi delle scatole possono contenere lettere, cifre e il sottolineato _, ma non devono iniziare con una cifra. È legale usare sia lettere maiuscole sia minuscole. Ricorda inoltre che per Python i caratteri minuscoli sono diversi dai caratteri maiuscoli.

Quindi le scatole che si chiamano:
Luisa, luisa, LUISA
sono per il calcolatore tre scatole diverse.

Il carattere di sottolineatura (_) può far parte di un nome ed è spesso usato in nomi di scatole composti da più parole. Ad esempio:

il_mio_nome **il_prezzo_del_pane**

In alternativa i nomi possono essere composti usando l'iniziale maiuscola per ciascuna delle parole componenti il nome della variabile, con il resto dei caratteri lasciati in minuscolo come in :

IlMioNome **IlPrezzoDelPane**

Le due operazioni fondamentali che si possono attuare su una scatola sono:

1- L'introduzione di un dato:

scatola1=5+3

(che è l'ordine di eseguire il calcolo 5+3 e di porre il risultato in scatola1)

2- La visualizzazione del contenuto di una scatola:

print (scatola1)

(che è l'ordine di visualizzare il contenuto di scatola1)

Prova a scrivere su Python le due istruzioni precedenti scritte in rosso e poi clicca su esegui. Osserva il risultato.

Non e' possibile invece "battezzare" la nostra scatola con nomi come 1A, 3P, 4SAL perché **il suo nome non può iniziare con un numero**. Assegnando un nome di questo tipo ad una scatola otterremo un messaggio di errore. Il calcolatore ci dirà che il nome che abbiamo usato e' illegale ma non dirà perché è illegale, e quindi dovremo scoprirlo noi.

Vediamo cosa ci dice il calcolatore quando sbagliamo il nome di una scatola:

76strumenti
SyntaxError: invalid syntax

milione$
SyntaxError: invalid syntax

lambda
SyntaxError: invalid syntax

Come mai?
Il nome 76strumenti e' illegale perché non inizia con una lettera,
milione$ e' illegale perché contiene un carattere proibito, il $,
lambda e' illegale perché è una delle parole riservate di Python.

Tutti i linguaggi di programmazione hanno alcune parole riservate *che non possono essere usate come nomi delle scatole.*

Le parole riservate di Python sono 28 e sono le seguenti:

And	**continue**	**else**	**for**
import	**not**	**raise**	**assert**
Def	**except**	**from**	**in**
Or	**return**	**break**	**del**
Exec	**global**	**is**	**pass**
Try	**class**	**elif**	**finally**
If	**lambda**	**print**	**while**

Vediamo se hai capito:

Puoi dare a una scatola il nome 10ART?

Quale di questi nomi è sbagliato?
Cane_M_4 CaneM4 4Cane_M

Puoi dare a una scatola il nome print?

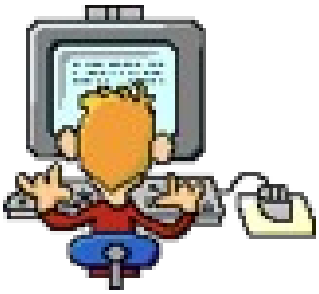
Scrivi di seguito altri esempi inventati da te e controlla se il programma ti da' errore o no quando scrivi istruzioni come: scatola=… oppure print(scatola)

……
……
……
……

Le nostre scatole sono state create per contenere dei dati.
Dobbiamo fare attenzione a non confondere il nome della scatola con quello che mettiamo dentro alla scatola, il suo contenuto.

Il nome di una scatola non cambia mai mentre il suo contenuto cambierà spesso.

Ad esempio la scatola PIPPO potrà contenere, in un certo momento, il numero 8, poi il numero 999 e quindi il numero 2.5 e così via.

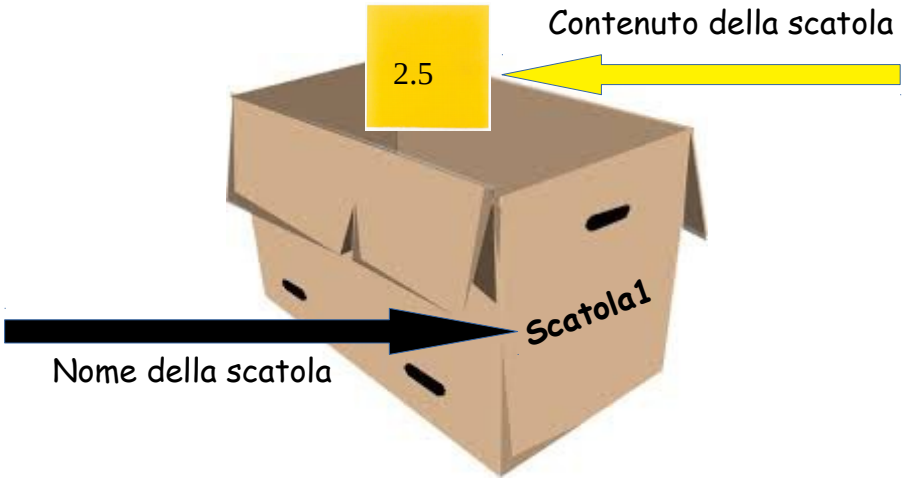


Un programmatore conosce sempre il nome della scatola perché è stato lui a "battezzarla", ma di solito non ne conosce il contenuto, perché può averlo dimenticato oppure perché la scatola è stata utilizzata per calcoli complicati.

Tuttavia il programmatore in qualsiasi momento può aprire la scatola e controllarne il contenuto, scrivendo, ad esempio:

print (scatola1)

Per ricordare meglio la **differenza fra nome e contenuto di una scatola**, immaginiamo di aver scritto il nome della scatola con un pennarello indelebile sulla scatola stessa e che il contenuto sia stato scritto su un foglietto che in qualunque momento può essere tolto dalla scatola e sostituito con un altro foglietto.



Lezione 3: Le Variabili

l'istruzione di assegnazione e l'istruzione di stampa

Il modo più semplice per farsi consegnare una scatola dal calcolatore, darle un nome e riempirla con un numero e' quello di scrivere una frase come la seguente, che è chiamata "**istruzione di assegnazione**":

SCATOLA1=7

Se la scatola di nome SCATOLA1 non è mai stata usata, Python crea una scatola nuova, la chiama SCATOLA1 e pone il numero 7 in quella scatola. Se invece la scatola di nome SCATOLA1 è già stata usata, Python toglie il suo vecchio contenuto e mette il numero 7 come suo nuovo contenuto.

La scatola rappresenta quella che i matematici chiamano variabile.

Questo nome ci ricorda che il contenuto di una scatola può cambiare da un momento all'altro.
Ad essa e' possibile assegnare un valore con l'istruzione di assegnazione.
L'istruzione di assegnazione contiene un ordine per il calcolatore e costituisce un nuovo esempio di "istruzione", ovvero un'operazione che Python capisce ed e' in grado di eseguire.
L'istruzione di assegnazione e' costituita dal nome della scatola seguito dal segno =.

Ad esempio:
SCATOLA1 = 7
SCATOLA2 = 3.14

Il contenuto di una scatola può anche essere un carattere o un messaggio (ossia una sequenza di caratteri).

SCATOLA3 = "come va?"

Quindi in una scatola posso introdurre numeri con o senza virgola, come nei primi due esempi, oppure messaggi come nel terzo esempio. I messaggi sono fatti da tanti caratteri compresi tra un simbolo " prima del primo carattere e un secondo simbolo " dopo l'ultimo carattere.

Osserva ancora: il simbolo = non ha il significato matematico che conosci. Infatti, nell'istruzione di assegnazione :

SCATOLA1 = 7

il simbolo = significa: nella scatola di nome SCATOLA1 metti il numero 7.

Come abbiamo visto negli esempi precedenti, con l' istruzione di stampa print possiamo ordinare al calcolatore di comunicare il contenuto di una scatola.

Se scriviamo **print(SCATOLA1)** il calcolatore:

- cerca la scatola indicata (in questo caso SCATOLA1)
- la apre e ne legge il contenuto
- presenta sullo schermo il contenuto (in questo caso 7)

Facciamo un esempio:

```
scatola1 = 8+4
print (scatola1)
```

Dopo l'ordine di esecuzione, sul video comparirà il risultato dell'addizione: 12.
Il nome dell'istruzione "print" deve essere sempre scritto minuscolo.
Le parentesi che delimitano il nome della scatola sono obbligatorie.

Qualche esercizio per memorizzare:

3.1 Scrivi in Python questa istruzione per il calcolatore: *dammi la scatola GAIA3 e mettici dentro il numero 1.*

3.2 Visualizza sul calcolatore il risultato di:

```
SCATOLA2 = 10-3
print (SCATOLA2)
SCATOLA3 = 7*4
print (SCATOLA3)
SCATOLA4 = 20/2*5
print(SCATOLA4)
```

3.3 Calcola il risultato delle seguenti istruzioni, utilizzando le corrispondenti istruzioni print:

```
print((3+2)*(3+1))
print(3+2*3+1)
print(3+2**3)
print(3**2*2)
print(3*(2**2))
```

FACCIAMO UNA BREVE INTERRUZIONE.

VOGLIAMO RACCONTARTI LA STORIA DI PYTHON.

Il suo inventore è un geniale signore olandese: **Guido Van Rossum.**

Nel Natale del 1989 Guido decise di passare le vacanze scrivendo un linguaggio di programmazione per il calcolatore che correggesse i difetti che, secondo lui, erano presenti in altri linguaggi. E così fece, creando uno tra i linguaggi di programmazione più moderni e diffusi nel mondo.
Dopo di lui moltissimi altri sviluppatori hanno proseguito ed ampliato il suo lavoro.

A Guido piaceva tantissimo un gruppo di comici inglesi famosi negli anni sessanta del secolo '900: i Monty Python. A loro e alla loro comicità un po' demenziale ha dedicato il suo lavoro. Niente a che vedere, quindi, con il nostro simpatico pitone verde.

Evidentemente i Monty Python dovevano piacere anche a molti altri nell'ambiente, perché non è l'unica volta che questi comici hanno dato il nome a qualcosa di informatico.
Ad esempio, il termine "*spam*", che viene utilizzato per indicare la posta elettronica indesiderata, deriva da un loro sketch, in cui compariva un ristorante nel cui menù erano inseriti tutti piatti ricoperti di spam, un tipo di carne macinata in scatola, particolarmente disgustosa.

Guido, dopo tanta fatica, ha deciso di donare a tutti il suo lavoro, così oggi noi possiamo disporre liberamente e gratuitamente di questo linguaggio. Un regalo di Natale molto speciale.

Nel mondo dell'informatica queste cose succedono. Ne riparleremo.

Per adesso grazie a Guido e a tutti coloro che hanno continuato, e continuano, la sua opera.
Per noi, un motivo in più per impegnarci a fondo ad imparare la programmazione.

ESERCIZI

In base a quanto hai imparato sin qui, crea semplici programmi secondo le istruzioni che seguono:

Esercizio n°1:
Il mago Silvan fa tanti giochi di magia: dal suo cappello a cilindro escono tre conigli bianchi e due neri. Quanti conigli sono nascosti nel cappello?

Esercizio n°2:
Al mago Berri invece piace fare le magie con le maxi-carte: sono così grandi che quasi non stanno sul tavolo!
Se ciascuna carta è lunga cm. 45 e larga cm. 30, quanto è grande la superficie di ciascuna carta?

Esercizio n°3:
Quale superficie del tavolo occupano le carte con i quattro Assi usati dal mago Berri per i suoi giochi di magia, affiancati per il lato più lungo?

Esercizio n°4:
Il mago Gian ha un bellissimo mantello di seta nera ricamato con tante stelle argentate. Per farlo il sarto ha utilizzato ben 5 metri di stoffa. Se la stoffa costava 13 € al metro, quanto ha speso per comprarla?

Esercizio n°5:
Se un mantello costa 80 €, un cappello a cilindro 45 €, una bacchetta magica 20 €, un mazzo di maxi-carte 13 €, quanto costa l'attrezzatura per fare il mago? Se tutti i 37 alunni della scuola di magia sono vestiti come il mago, quanto è costata la loro attrezzatura?

Esercizio n°6: (difficilissimo, se riesci a farlo sei --- un mago!)
Nella classe 3D della scuola ci sono 8 maschi e 10 femmine. Se Mario è alto m. 1.55, Fabio, Matteo e Luca sono alti m. 1.60, Andrea, Aldo, Giovanni e Giacomo m. 1.50, qual è l'altezza media dei maschi della classe?
Se Marta, Giovanna, Elisabetta e Francesca sono alte come Mario, mentre Stefania, Chiara e Simonetta sono alte m. 1.50, Daria e Domitilla sono 5 cm più piccole di Arianna che è alta m. 1,68, qual è l'altezza media delle femmine della classe?

Lezione 4: Dati e tipi di Dati

Fino ad ora abbiamo lavorato quasi sempre su **numeri**.
Abbiamo visto, ad esempio, che una scatola può contenere numeri interi come 27, 251, 1100050, oppure numeri decimali, ossia numeri con virgola che Python vuole che siano scritti con il simbolo "**.**", come :
27.35
251.5
3.1415
In questa sezione inizieremo a parlare anche di **caratteri e stringhe.**

Come già accennato esistono diversi tipi di dati:

- **i numeri**
- **i caratteri**
- **le stringhe**

Dei numeri abbiamo già parlato nel capitolo precedente, mentre **un carattere è un simbolo** come, ad esempio, la lettera "a" minuscola, la "Q" maiuscola, la cifra "7", il simbolo "+", e qualunque altro simbolo che compaia sulla tastiera del tuo PC (preceduto e seguito da virgolette").
Una stringa è un qualsiasi messaggio, quindi una serie di caratteri, numeri, lettere o altri simboli presenti sulla tastiera, compresi tra virgolette, che il computer può stampare tali e quali sul video.
In Python le stringhe possono presentarsi in diversi modi:

- tra virgolette semplici: 'ciao'
- tra virgolette doppie: "ciao"
- tra virgolette doppie ripetute tre volte: """ciao"""

Ecco alcuni esempi di stringhe composte da:

Una serie di lettere	"**ciao**"
Una serie di lettere che formano un messaggio	"**viva la Juve**"
Una serie di simboli	"**#@&%**"
Anche i numeri tra virgolette vengono interpretati come simboli e stampati tali e quali	"**8-5=3**"

Prova a scrivere:

print (“Ciao”)

Il calcolatore visualizzerà tutti i caratteri compresi tra le virgolette:

Ciao

Ora scrivi (anche se ami un’altra squadra):

print("Viva la Juve")

Anche in questo caso il calcolatore visualizzerà tutti i caratteri racchiusi tra virgolette:

Viva la Juve

Se scriviamo l’istruzione: print (“8+4”)
il calcolatore visualizzerà: 8+4
senza fare nessun calcolo. Perché?
Perché hai dato al computer il seguente ordine: *visualizza la stringa costituita da tre caratteri: 8 + 4.*

I numeri 8 e 4 sembrano effettivamente dei numeri ma essendo chiusi tra virgolette vengono interpretati come i caratteri di un messaggio qualunque e quindi come…”una stringa”!

Se poi scrivi l’istruzione: print (“8+4=13”) il calcolatore scriverà: 8+4=13 senza accorgersi dell’errore di calcolo, perché l’ordine che gli hai dato è solo quello di scrivere la stringa composta dai caratteri: 8+4=13

Ovviamente possiamo mettere la stringa in una scatola, come abbiamo fatto con i numeri, usando l'istruzione di assegnazione.

Ad esempio:
SCATOLA1 = "**scuola elementare**"
print (SCATOLA1)
Il risultato sarà:

scuola elementare

Eseguendo i seguenti comandi:

```
print ("MATEMATICA")
print ("Via Roma 100")
print ("Area")
print ("++++****++++")
```

visualizzeranno rispettivamente:

```
MATEMATICA
Via Roma 100
Area
++++****++++
```

Proviamo ora a scrivere:

```
print ("8+4  =",  8+4)
```

Il calcolatore prima visualizzerà la stringa "8+4 =" e poi il risultato dell'operazione 8+4. Quale sarà il risultato? Vedremo comparire nella finestra delle risposte il messaggio: **8+4 = 12**

Facciamo ancora due esempi :

```
print  ("3+5=  ", 3+5)
```

Avrà come risultato:

```
3+5=  8
```

```
print  ("4*4= ",  4*4)
```

Avrà come risultato:

```
4*4=  16
```

Sono stringhe	Sono numeri
"3+5= " "4*4= "	3+5 4*4

Nell'esempio: **print ("3 per5 " , "è uguale a ",3*5)** osserva che la print è composta da due stringhe e poi da un numero, separati da due virgole.

Se vuoi fare visualizzare più oggetti sulla stessa riga devi separarli con una virgola.

Nel tuo caso hai detto a Python: visualizzami la stringa "5*3= " così com'è e subito dopo hai aggiunto: visualizzami il risultato della moltiplicazione tra i numeri 5 e 3.

Nell'esempio successivo cambia solo l'ordine con cui devono essere eseguiti gli ordini, in questo caso prima l'operazione su numeri e poi la stringa:

print (100-10, " = 100 - 10")
Il risultato sarà:
90 = 100-10

Facciamo due esempi simili utilizzando le variabili:

scatola2 = 14+8
print ("14+8= ", scatola2)
Il risultato sarà: **14+8= 22**

scatola3=5*3
print ("5*3= ", scatola3)
Il risultato sarà: **5*3= 15**

Prova a rispondere ai seguenti quesiti:

1. In quale di questi due comandi i numeri sono numeri e non simboli?
print ("6 + 3") e print (6+3)

2. Quale sarà il risultato dei due comandi?

3. Quale sarà il risultato del comando:
print ("9 * 5 = ", 9 * 5)

4. Quale risultato darà l'istruzione seguente?
print ("Auguri a " + "tutti")

Dati e tipi di dati (operazioni sulle stringhe)

Numeri interi e decimali possono essere utilizzati per operazioni matematiche. Viceversa, se una stringa contiene l'indicazione di un'operazione aritmetica, questa non viene eseguita ma trascritta carattere per carattere.
Scrivere: "ciao"/12 oppure "18"+5
e' sbagliato e genera un SYNTAX ERROR

Tuttavia gli operatori **+** e ***** , e soltanto questi, possono essere utilizzati anche con le stringhe ma in questo caso hanno un significato diverso da quello della somma e del prodotto.
Se ad una stringa viene sommata un'altra stringa l'effetto che si ottiene e' il ***concatenamento****, cioè la seconda stringa si aggiunge al fondo della prima.*

Ad esempio:
print("casa"+" dolce "+"casa") genera a video:
casa dolce casa
Gli spazi prima e dopo la parola "dolce" fanno parte della stringa e sono necessari, altrimenti a video le stringhe concatenate sarebbero attaccate come un'unica parola, come succede scrivendo: print("casa"+"dolce"+"casa").

Se invece vogliamo ripetere tante volte la stessa stringa (questo effetto si chiama ***ripetizione****) possiamo moltiplicarla per un numero intero usando l'operatore* *******.*

Ad esempio:
"ciao"*3 diventa ciaociaociao
"ciao "*3 diventa ciao ciao ciao
"Ha, "*5 diventa Ha, Ha, Ha, Ha, Ha,
(Nota bene: la barretta colorata indica lo spazio che va inserito nel secondo e nel terzo esempio)

Possiamo anche mettere una stringa in una scatola, ossia assegnare una stringa ad una variabile e poi applicare le operazioni possibili sulle stringhe.

Ad esempio:
SCATOLA1 = "casa "
SCATOLA2= "dolce casa"
print (SCATOLA1 + SCATOLA2)
casa dolce casa

Esercitiamoci un po'

4.1 Scrivi le istruzioni necessarie per far comparire sul video la scritta:

Cinque per tre e' uguale a 15.

(Puoi ottenere questo risultato con o senza scatole, ossia variabili. Trova le varie soluzioni).

4.2 Scrivi tutte le sequenze di istruzioni possibili per visualizzare il messaggio "Buon Compleanno" cinque volte.

4.3 Scrivi la sequenza di istruzioni per visualizzare il tuo nome e cognome in due stringhe separate.

4.4 Scrivi la sequenza di istruzioni per ottenere il messaggio seguente utilizzando le variabili: l'area del rettangolo e' uguale a 50.

4.5 Scrivi la sequenza di istruzioni per ottenere il perimetro e l'area di un rettangolo che abbia i lati di cm 9 e cm 5.

4.6 Prova a verificare il programma:

```
print ('  O   O ')
print ('     |     ')
print ('   \_/   ')
```

Adesso scrivine uno tu, inventando un nuovo disegno.

4.6 Trova l'errore:

a- print (ciao+4)

b- print ("ciao"+4)

4.7 Scrivi le istruzioni per visualizzare cinque volte il messaggio "In classe è arrivato un nuovo compagno".

4.8 Scrivi le istruzioni per ottenere che nella scatola1 ci sia 12, nella scatola2 ci sia anni e che il computer stampi: *' domani compirai 12 anni'* .

Lezione 5: primi Programmi

La COPIATURA

Iniziamo a vedere come fare a passare da una scatola all'altra.

Cosa succede quando diamo al computer un'istruzione come:

PIPPO = PLUTO

Il computer fa le seguenti operazioni:

1. mette a disposizione la scatola di nome PIPPO, che potrebbe essere nuova oppure essere già stata utilizzata.
2. apre la scatola di nome PLUTO
3. legge il contenuto (vi ricordate il foglietto nella scatola?) di PLUTO lo ricopia su un altro foglietto che mette nella scatola PIPPO.

Al termine di queste operazioni le due scatole PIPPO e PLUTO contengono lo stesso numero 15.

Ecco come si presentano le scatole prima dell'esecuzione dell'istruzione:

Ecco come si presentano le scatole dopo dell'esecuzione dell'istruzione:

Osserva: il contenuto di Pluto (la scatola che si trova a destra del segno =) viene trasferito in Pippo e non viceversa e Pluto non perde il suo contenuto.

Vediamo cosa succede invece se scriviamo:

PLUTO = PIPPO,

quando

PIPPO = 5 e PLUTO = 15

Le due scatole conterranno il numero 5.

Vediamo come scrivere i programmi corrispondenti a queste due istruzioni:

Programma1 PIPPO = PLUTO	Il risultato sarà:
PIPPO=5 PLUTO=15 print ("PIPPO = ", PIPPO) print ("PLUTO = ", PLUTO) PIPPO = PLUTO print ("Dopo l'esecuzione dell'istruzione PIPPO = PLUTO") print(" PIPPO = ", PIPPO) print ("PLUTO = ", PLUTO)	PIPPO = 5 PLUTO = 15 Dopo l'esecuzione dell'istruzione PIPPO = PLUTO PIPPO = 15 PLUTO = 15
Programma 2 PLUTO = PIPPO	Il risultato sarà:
PIPPO=5 PLUTO=15 print ("PIPPO = ", PIPPO) print ("PLUTO = ", PLUTO) PLUTO = PIPPO print ("Dopo l'esecuzione dell'istruzione PLUTO = PIPPO") print("PLUTO = ", PLUTO) print ("PIPPO = ", PIPPO)	PIPPO = 5 PLUTO = 15 Dopo l'esecuzione dell'istruzione PLUTO = PIPPO PIPPO = 5 PLUTO = 5

L' INCREMENTO

Prova a immaginare cosa fa il computer per eseguire la seguente istruzione:

PIPPO = PLUTO + 3

supponendo che PLUTO contenga il numero 15.

Il computer farà così:

- Metterà a disposizione una scatola di nome PIPPO
- Cercherà PLUTO e ne leggerà il contenuto.
- Aggiungerà 3 al contenuto di PLUTO e metterà in PIPPO il risultato dell'operazione.

A questo punto, che cosa conterrà PIPPO e che cosa conterrà PLUTO?

PIPPO conterrà il numero 18 e PLUTO conterrà sempre il numero 15.

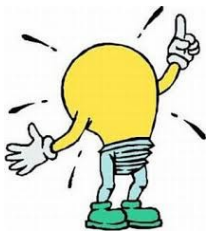
In pratica il computer inizia a lavorare sull'operazione a destra dell'uguale e il risultato viene messo nella scatola a sinistra.

Osserva ancora che il nome della scatola a destra dell'uguale indica il suo contenuto, mentre il nome della scatola a sinistra precisa la scatola che conterrà il risultato.

Lo SCAMBIO

Un po' più complicata è l'operazione di scambio del contenuto di due scatole.

Ad esempio se MINNI = 10 e MICKEY = 12 come posso scambiare il contenuto di MINNI e MICKEY, cioè inserire 12 in MINNI e 10 in MICKEY?



E'come scambiare il contenuto di due bicchieri uno pieno di Coca Cola e l'altro pieno di aranciata. In quel caso occorre un terzo bicchiere.

Nel nostro caso **serve una terza scatola!**

Una scatola che possiamo chiamare, ad esempio, PARK, nella quale riponiamo il contenuto di una delle due scatole. Cosa fa il computer?

1. Apre le due scatole già disponibili di nome MINNI e di nome MICKEY.
2. Mette a disposizione una scatola di nome PARK e ci inserisce il contenuto di MINNI.
3. Legge il contenuto di MICKEY e lo mette in MINNIE.
4. Legge il contenuto di PARK (che era quello di MINNIE) e lo mette in MICKEY.

Rifletti e prova a rispondere: cosa contengono le scatole della colonna **arancione**?

MINNIE =10 ; MICKEY = 12	
PARK = MINNIE	**PARK** =..... **MINNIE** =.....
MINNIE = MICKEY	**MINNIE** =..... **MICKEY** .....
MICKEY = PARK	**MINNIE** =..... **PARK**=.....

Prima di proseguire prova a eseguire questo esercizio:

Se CLICK1 = 24 e SLAM1 = 32 come faccio per copiare il contenuto di CLICK1 in SLAM1? E quando l'ho copiato come faccio per rimettere nelle due scatole il contenuto originale? Prova a illustrare i vari passaggi attraverso i quali il calcolatore copia il contenuto di una scatola in un'altra.

INPUT

Finora abbiamo visto come inserire un numero o una stringa in una scatola, cioè un dato in una variabile utilizzando le istruzioni di assegnazione del tipo:

SCATOLA1 = 37.5 oppure SCATOLA1 = "Viva la Juve"

Oltre a questo, esiste un altro modo, molto importante, per introdurre un numero o una stringa in una scatola, rappresentato dall'istruzione input.

Se scriviamo SCATOLA1 = 37.5 il valore 37.5 viene introdotto in SCATOLA1 nel momento in cui quell'istruzione viene eseguita. Invece, con l'istruzione **input** Python chiede a chi sta utilizzando il programma di introdurre un dato che potrà essere diverso nei diversi momenti in cui si utilizzerà quel programma.

INPUT, che significa letteralmente "ingresso", si usa nel modo seguente:

scatola = input(prompt)

dove:
scatola è il nome della scatola nella quale inserirò un nuovo dato;
input è il comando che diamo al computer e che serve a inserire un dato qualunque nella scatola. Quel dato è indicato dall'utilizzatore del programma attraverso la tastiera.

prompt è un messaggio che diamo all'utilizzatore perché sappia quale dato deve inserire da tastiera.

Ad esempio con:

ANNI_DI_ROSI = input ("Quanti anni hai? ")
chiediamo all'utilizzatore di indicare i propri anni (scrivendolo in quel momento da tastiera).

Il computer leggerà il dato e lo inserirà nella scatola di nome ANNI_DI_ROSI.

Quando il computer legge la parola input, si ferma e attende che l'operatore inserisca un dato dalla tastiera.
Per far capire al computer quando il numero e' finito, l'operatore preme il tasto Invio (o Enter). A questo punto il programma riprende e input interpreta ciò che l'operatore ha inserito da tastiera come una stringa di caratteri e lo mette nella scatola indicata.

Il programma prosegue poi con le istruzioni successive.

Finora abbiamo sempre inserito tutti i dati prima dell'esecuzione di un programma e poi abbiamo eseguito il programma stesso; con input, invece, i dati possono essere inseriti durante l'esecuzione. Vediamo in dettaglio cosa succede nel programma seguente quando usiamo la "funzione" input:

Anni = input ("Quanti anni hai? ")

print ("Tu hai ", Anni, " anni")

`Anni = input ("Quanti anni hai?")`	1. il computer mette a disposizione la scatola chiamata "Anni", se questa scatola è stata già utilizzata; oppure una scatola nuova alla quale dà il nome "Anni". 2. si ferma nell'attesa che venga inserito un dato dalla tastiera 3. inserisce il dato nella scatola indicata
`print ("Tu hai ", Anni, " anni")`	Stampa prima la stringa "Tu hai ", poi il contenuto della scatola Anni e infine la stringa " anni".

Vediamo un altro esempio. Il programma seguente:

```
s = input ("Come ti chiami? ")

print ("Ciao ", s)
```

Ricordati sempre di lasciare lo spazio

darà come risultato la parola "Ciao " seguita dal contenuto della scatola chiamata "s"
Se manderai in esecuzione il programma tante volte inserendo sempre un nome diverso il risultato sarà:
Ciao Paola
Ciao Alda
Ciao Marco

Il dato inserito da tastiera è sempre considerato da Python come una stringa di caratteri. Ad esempio, se il programma precedente dove si chiede "quanti anni hai?" proseguisse con l'istruzione:

print("Io ho ", Anni *2,"anni")

scritta per dire "il doppio dei tuoi anni", si vedrebbe il dato introdotto dall'utilizzatore scritto due volte di seguito. Se l'utilizzatore avesse scritto "8", il messaggio sul video sarebbe:

"Io ho 88 anni"

Prova a verificare scrivendo il programma su Python.

Se con l'istruzione input aspettiamo un numero intero dobbiamo tradurre la stringa introdotta dall'operatore scrivendo, ad esempio:

Anni = int(input("Quanti anni hai?"))

Se aspettiamo invece un numero con virgola dobbiamo scrivere:

Anni= float(input("Quanti anni hai?"))

Ricorda che la parola "float" è iniziale di "floating point", ossia "punto mobile" o "virgola Mobile".

Prova a descrivere la sequenza di operazioni fatte dal calcolatore per eseguire il programma seguente che serve a calcolare il triplo di un numero :

```
numero = int (input ("Introduci un numero "))
numero = numero * 3
print ("Il triplo del numero introdotto è : ", numero)
```

Prova ora ad inserire dei caratteri che non rappresentino un numero e osserva quale sarà il nuovo risultato. Sfortunatamente se i caratteri inseriti dall'operatore non rappresentano un numero, il programma stampa un messaggio d'errore e si blocca perché int(input()) e float(input()) funzionano soltanto con i numeri.

Come facciamo a far in modo che l'interprete accetti qualunque carattere immesso dall'utilizzatore?
Usiamo semplicemente il comando "input" senza specificare nulla.

Prova a scrivere su Python gli esempi seguenti e osserva il risultato:

```
print ("Alt! ")
s = input ("Chi va la'? ")
print ("Passa pure ", s)

num = int (input ("Scrivi un numero "))
print ("num = ", num)
print ("num * 2 = ", num * 2)
```

Esercitiamoci un po'

Ci sono più soluzioni possibili per ognuno degli esercizi proposti;

sta a te trovarle e, soprattutto, provarle.

5.1 Scrivi un programma che chiede un numero e ne calcola il quadrato e il cubo e li visualizza sullo schermo.

5.2 Scrivi un programma che aggiunge 7 a qualunque numero inserito e visualizza il risultato sullo schermo.

5.3 Scrivi un programma che chiede due numeri, li somma e visualizza il risultato.

5.4 Scrivi il programma per calcolare l'area di qualunque rettangolo chiedendo all'utilizzatore la base e l'altezza.

5.5 Scrivi il programma che chieda tre numeri e ne visualizzi sia la somma sia il prodotto.

5.6 Scrivi il programma che calcola la metà e il doppio di qualunque numero inserito dall'utente e visualizza i risultati.

5.7 Scrivi il programma che chiede la misura del lato di un quadrato e ne calcola l'area, poi visualizza il risultato.

5.8 Scrivi il programma che calcola il perimetro del cortile della scuola che è un rettangolo i cui lati misurano rispettivamente 45 m e 65 m e visualizza il risultato. Quindi calcola il perimetro di ogni rettangolo per il quale l'operatore inserisca la misura della base e dell'altezza.

5.9 Scrivi un programma che chieda tre numeri, ne calcola la somma, la somma dei quadrati e il quadrato della somma. Infine, visualizza i risultati.

Concediamoci un momento di pausa per giocare un po'.
Giochiamo a:

CACCIA ALL'ERRORE!

Regole del gioco: In ogni programma è inserito un errore.

Leggi attentamente ciascun programma, prova a digitarlo utilizzando Python, scopri e correggi l'errore.

5.10 Stampa il nome del tuo cantante preferito.

```
cantante = input ("Scrivi il nome del cantante preferito: ")

print ("Il mio cantante preferito e' ", cantant)
```

5.11 Input di numeri e stringhe

```
Primonumero= int(input ("Scrivi il primo numero:  "))

Secondonumero= int(input ("Scrivi il secondo numero: "))

Nome = input ("Scrivi il tuo nome:  ")

Cognome = input ("Scrivi il tuo cognome:  ")

print nome , cognome, "primonumero", "per", secondonumero, "uguale",
primonumero*secondonumero
```

5.12 Domanda di filosofia

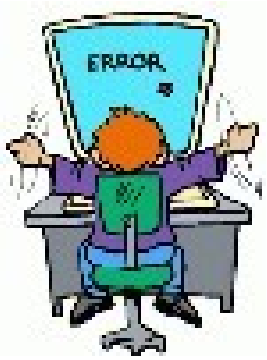
```
print (" Sai in quale anno e' nato Socrate?")

print ("Nell'anno 469 prima di Cristo")
```

5.13 Disegno un quadrato

```
Print ("*****")

print ("*     *")

print ("*   *")

print ("*   *")

print ("*****")
```

5.14 Divisione con resto

```
primo = float (input ("Inserisci il primo numero"))

secondo = float (input ("Inserisci il secondo numero"))

print (primo, "diviso", secondo,"fa", primo/secondo)

print "il resto della divisione e' ", primo % secondo
```



Lezione 6: le Istruzioni Condizionali if e if...else

Supponiamo di voler mandare un messaggio gentile a chi usa il nostro programma. Nello scrivere il programma ci accorgiamo che il messaggio e' diverso a seconda che il nostro interlocutore sia una femmina oppure un maschio.
Allora gli chiediamo se e' femmina o maschio e poi scriviamo un messaggio se e' una femmina, un messaggio diverso se e' un maschio. Come fare? Usiamo l'istruzione:

if

L'istruzione che ci permette di scegliere cosa fare si chiama if, che in inglese significa SE.

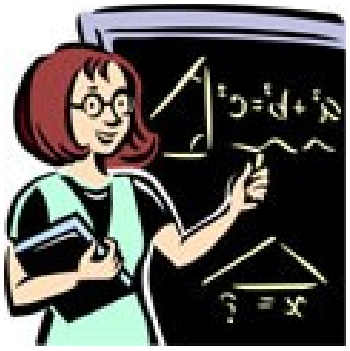
Questo e' il codice del programma che descrive l'esempio appena fatto :

```
nome = input ("Scrivi il tuo nome ")
utente = input ("Sei femmina? ")
if utente == "si":
    print ("Cara ", nome, ", sei bravissima!")
if utente == "no":
    print ("Caro ", nome, ", sei bravissimo!")
```

(NOTA BENE: i rettangolini colorati evidenziano gli spazi che devi lasciare.
Questo metodo di scrittura si chiama INDENTAZIONE).

Osserva nell'esempio precedente che occorre scrivere prima la parola if, poi la condizione poi i : (due punti), e infine l'elenco delle cose da fare, scritte un poco a destra nella riga successiva, secondo questo schema:

```
if e poi una condizione :
    azione 1
    azione 2
    ……..
```



Hai notato che nell'esempio precedente abbiamo usato un nuovo carattere? E' il segno di uguaglianza == (doppio uguale) che utilizziamo per indicare "uguale a" .
Questo perché in Python il simbolo di uguale (=) ha il significato: "metti nella scatola".
Oltre al segno di uguaglianza esistono i segni di disuguaglianza. Li utilizziamo quando occorre indicare una condizione di confronto come indicato nella tabella seguente:

simbolo	significato	esempio:
==	uguale a	5 = 5
>	maggiore di	6 > 3
<	minore di	3 < 7
>=	maggiore o uguale a	x >= y
<=	minore o uguale a	Y <= x
<>	diverso da	5 <> 1
!=	diverso da	5 !=

Adesso che abbiamo imparato i nuovi segni, vediamo due esempi di programmi:

```
voto = int (input  "che voto hai preso? ")
if voto >= 6 : print ("promosso!")
```

che é equivalente a scrivere:

```
if voto >= 6: print("promosso")
```

```
if tuoi_soldi < 3: print ("Non puoi comprare la pizza!")
```

La frase che segue la if si chiama: condizione

Quando la condizione e' soddisfatta, si esegue l'istruzione che segue i due punti (:), altrimenti non si fa nulla.

Consideriamo l'esempio:
Il voto e' maggiore o uguale a 6?
Allora sei promosso!

L'istruzione if viene scritta abitualmente nel modo seguente:

struttura	esempio
if <condizione> :	if voto >= 6:
<istruzione>	print ("promosso")

Come già accennato, l'istruzione deve stare più all'interno della prima riga (hai presente i margini del foglio? Se la prima riga è allineata al margine sinistro, le linee successive devono stare più a destra). Si dice che devono essere "indentate".

Dopo la "condizione" anziché una sola istruzione, come negli esempi precedenti, possono essere scritte due o più istruzioni:

struttura	esempio
if <condizione> :	if voto >= 6:
<istruzione1>	print ("promosso")
<istruzione2>	print ("mamma contenta")

Non c'è un limite al numero di istruzioni che possono comparire nel corpo di un'istruzione if ma deve sempre essercene almeno una.

La prima riga di istruzioni che non sta più all'interno (che non e' più "indentata") segnala al computer la fine del blocco di istruzioni e non ne fa parte.

Prova a scrivere questo programma:

```
if 5 > 10:
    print ("sun")
print ("moon")
```

il computer scriverà "moon" perchè la linea non fa più parte dell'istruzione if. Ma se la linea print ("moon") viene indentata, essa non viene più stampata, come puoi verificare scrivendo:

```
if 5 > 10:
    print ("sun")
    print ("moon")
```

Vediamo ora un esempio in cui e' un po' più complicato prendere una decisione perché consideriamo più condizioni insieme:

Hai più di sette anni?
Sei bravo a giocare a calcio?
Se sì puoi iscriverti alla scuola calcio del Milan.

```
ragazzo = input ("Hai più di sette anni?")
campione = input ("Sei bravo a giocare a calcio?")
if ragazzo == "sì":
        if campione == "sì":
                print ("Puoi iscriverti alla scuola calcio del Milan")
```

Per semplificare la scrittura di questo ultimo programma, possiamo usare tre nuovi operatori chiamati **or, and, not:**

Operatore	Significato
or	"oppure"
and	"e inoltre"
not	"non"

Usando gli operatori logici, l'ultimo programma che abbiamo scritto diventa molto semplice:

```
if ragazzo == "sì" and campione == "sì":
        print ("Puoi iscriverti alla scuola calcio del Milan")
```

Vediamo alcuni esempi concreti in cui questi nuovi operatori sono necessari.

and

SE hai fatto i compiti E c'è il sole puoi andare in bicicletta.

```
if compiti == "si" and sole == "si"
   print ("puoi andare in bicicletta")
```

SE i tre lati di un triangolo sono uguali, il triangolo è equilatero

```
if lato1 == lato2 and lato1 == lato3:
   print ("triangolo equilatero")
```

SE un numero è maggiore di 10 E minore di 20 allora è compreso fra 10 e 20
if num > 10 and num < 20:
    print ("il numero", num, "è compreso fra 10 e 20")
NOTA BENE! Tutte e due le condizioni richieste devono essere soddisfatte, sia quella dopo if sia quella dopo and. Il messaggio viene visualizzato solo in questo caso.

or

SE in un triangolo la lunghezza del lato1 e quella del lato2 sono uguali OPPURE sono uguali le lunghezze del lato1 e quella del lato3, oppure sono uguali le lunghezze del lato 2 e del lato 3, allora il triangolo è isoscele.

```
if lato1 == lato2 or lato1 == lato3 or lato2 ==lato3:
        print "triangolo isoscele"
```

*NOTA BENE: con l'operatore **or** non è necessario che siano soddisfatte contemporaneamente tutte le condizioni in esame, ma è sufficiente che, fra tante condizioni, ne sia verificata solo una.*

not

SE NON hai compiti da fare puoi andare a giocare.
if not fatto_compiti == "si":
      print ("Non puoi andare a giocare")

Spesso ci capita di voler fare una cosa se la condizione è vera e un'altra se la condizione è falsa. In questo caso dobbiamo utilizzare l'istruzione:

IF…ELSE

Il blocco di programma:
```
if condizione:
    istruzione
    istruzione
    …
```
può essere arricchito introducendo la parola inglese "else" che significa "altrimenti" nel modo seguente:
```
if condizione:
    istruzione
    istruzione
    …
```

```
else:
    istruzione
    istruzione
    ...
```

Vediamo alcuni semplici esempi:

```
if x < 3:
        print ("x è minore di 3")
else:
        print ("x non è minore di 3")

ring = input ("Suonano alla porta? ")
if ring == "si" :
         print ("vado ad aprire")
else:
         print ("continuo a leggere")

Se voto e' >= 6 stampo "promosso", se voto e' < 6 stampo "bocciato"
voto = int(input("che voto hai preso? "))
if voto >= 6 :
          print ("promosso!")
else:
          print ("bocciato!")

fatto_compiti = input ( "hai fatto i compiti ? ")
if not fatto_compiti ==  "si ":
    print ( "Non puoi andare a giocare ")
else:
    print ( "Bravo! Vai a giocare ")

nome = input ("Come ti chiami? ")
femmina = input ("Sei femmina? ")
if femmina == "si":
         print ("Cara ", nome,)
         print (", sei bravissima!")
else:
         print ("Caro ", nome)
         print (", sei bravissimo!")
```

```
risposta = input("Vuoi sapere come calcolare l'area del rettangolo? (S/N) ")
if risposta == "S":
        print ("devi moltiplicare la base per l'altezza")
else:
        print ("Spero che tu lo sappia")


risposta1 = input ("chi e' l'autore dei Promessi Sposi? ")
if risposta1 == "Manzoni":
        print ("la risposta e' esatta")
        print ("bravo!!")
else:
        print ("risposta errata!")
        print ("la risposta esatta e': Manzoni")
        print ("Studia di più")

risposta2 = input ("chi e' l'autore della Divina Commedia? ")
if risposta2 == "Dante":
        print ("la risposta e' esatta")
        print ("bravo!!")
else:
        print ("risposta errata!")
        print ("la risposta esatta e': Dante")
        print ("Ripassa la lezione.")
```

if...elif...

Purtroppo molte volte ci sono più di due possibilità di scelta e quindi abbiamo bisogno di un'altra soluzione. Usiamo l'operatore:

elif

elif è l'abbreviazione di "else if", che in inglese significa "altrimenti se".
In modo formale questa istruzione viene definita: condizioni in serie.
Non c'è alcun limite al numero di istruzioni elif.

Vediamo subito qualche esempio:

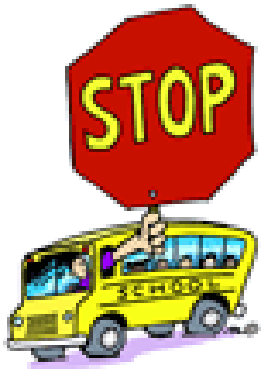
```
x = int(input("indica il numero x "))
y = int(input("indica il numero y "))
if x < y:
        print (x, "e' minore di", y)
elif x > y:
        print (x, "e' maggiore di", y)
else:
        print (x, "e", y, "sono uguali")
```

```
nome = input("qual è il tuo nome? ")
if nome == "Carlo":
    print ("il tuo onomastico è il 4 novembre")
elif nome == "Francesca":
     print ("il tuo onomastico è il 9 marzo")
elif nome == "Anna":
     print ("il tuo onomastico è il 26 luglio")
elif nome == "Andrea":
     print ("il tuo onomastico è il 30 novembre")
elif nome == "Stefano":
     print ("il tuo onomastico è il 26 dicembre")
else:
     print ("non so quando è il tuo onomastico ma ti
            auguro che ogni giorno sia la tua festa!! ")
```

```
numero = 78
indovina = 0
print ("indovina il numero che ho in mente")
indovina = int (input ("inserisci il numero:"))
if indovina > numero:
        print ("troppo alto")
elif indovina < numero
        print ("troppo basso")
else:
        print ("GIUSTO!")
```

Qualche volta non è necessario l'else finale, come in questo esempio:

```
print ("Stampa se un numero è pari o dispari")
numero = int (input ("scrivi un numero: "))
if numero %2 == 0:
        print (numero, " è pari")
elif numero%2 == 1:
        print (numero, " è dispari")
```



Esercitiamoci un po'

6.1 Che cosa significano le parole if, else, elif?

6.2 Che cosa fa il computer quando non e' soddisfatta la condizione introdotta da if?

6.3 Scrivi un esempio di uso dell'operatore if in cui compaia l'operatore and, una in cui compaia or e uno in cui compaia not.

6.4 Spiega il significato delle seguenti istruzioni.

a)
```
if numero <> 20:
        print (numero)
```

b)
```
if qui <20:
        quo = 30
```

6.5 Scrivi il programma che controlla il risultato di una addizione, dati due numeri.

6.6 Scrivi un programma che dati due numeri, li visualizza in ordine crescente (o decrescente).

6.7 Scrivi un programma che dia consigli per i vestiti se piove, se nevica e se fa freddo.

6.8 Scrivi il programma che chiede di indicare l'autore di un libro, se e' sbagliato stampa "risposta errata", se e' corretto stampa "risposta esatta" e prosegue a chiedere un altro autore per un altro libro (puoi ripeterlo quante volte vuoi)
Per esercitarti un po' di più puoi scrivere lo stesso programma per i seguenti argomenti: calciatori e squadre di calcio, nazioni e capitali, città e nome degli abitanti.

Lezione 7: le Funzioni

Impariamo insieme :

- cos'è una funzione
- quando è utile usare una funzione
- come si definisce una funzione
- come si fa a chiamare una funzione

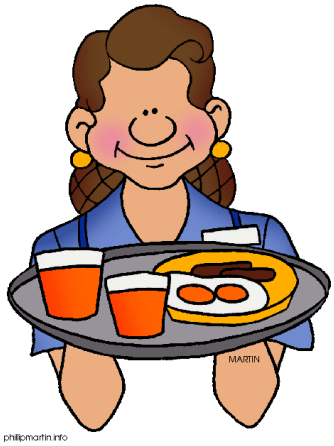
Riprendiamo in esame l'esempio degli "scrittori" esaminato nella lezione precedente:

```
risposta1 = input ("Chi e' l'autore dei Promessi Sposi? ")
if risposta1 == "Manzoni":
        print ("La risposta e' esatta")
        print ("Bravo!!")
else:
        print ("Risposta errata!")
        print ("La risposta esatta e': Manzoni")
        print ("Studia di piu'")
risposta2 = input ("Chi e' l'autore della Divina Commedia? ")
if risposta2 == "Dante":
        print ("La risposta e' esatta")
        print ("Bravo!!")
else:
        print ("Risposta errata!")
        print ("La risposta esatta e': Dante")
print ("Ripassa la lezione")
```

Hai osservato come il programmatore nel trattare la seconda risposta ripete molte cose che aveva già scritto per la prima ? Questo non è né comodo né veloce. Fortunatamente esiste in Python, come in molti altri linguaggi di programmazione, un meccanismo semplice per specificare una sola volta le operazioni ricorrenti e per richiamare quelle operazioni ogni volta che ci servono.

Questo meccanismo è la **"funzione"**

Una funzione è, in sostanza, un pezzo di programma cui viene dato un nome, una sequenza di istruzioni per eseguire una determinata sequenza di operazioni, che può essere utilizzata tante volte nel corso di un programma.



Possiamo immaginare la funzione come un "cameriere" al quale si insegnano certe mansioni da svolgere in casa.

Ad esempio, al cameriere si spiega come "preparare la colazione" nel modo seguente:

1. preparare i croissant
2. fare il caffè
3. scaldare il latte

Dopo averlo spiegato una volta non è più necessario ripetere tutte le volte gli ordini 1, 2 e 3 ma basta chiedere al cameriere di "preparare la colazione".

Nel caso del nostro esempio del programma degli "scrittori" si deve chiedere al cameriere, ossia alla funzione, di eseguire le seguenti operazioni:

- porre la domanda e acquisire la risposta
- verificare se la risposta data è uguale a quella corretta
- se la risposta è corretta, visualizzare il messaggio di congratulazioni
- se la risposta è errata, visualizzare il messaggio di rimprovero e il messaggio per ricordare la risposta corretta.

Il nostro "servente", ossia la funzione, dovrà eseguire proprio queste operazioni, ogni volta che porremo una nuova domanda. Al cameriere, ossia alla funzione, sarà sufficiente indicare la domanda e la risposta esatta.

In pratica dobbiamo scrivere un breve programma che esegua le operazioni che abbiamo elencato, dopo aver dato il nome alla funzione e alle scatole (ossia alle variabili) di cui la funzione avrà bisogno.
Nel nostro caso, INTERROGAZIONE sarà il nome della funzione.

DOMANDA sarà il nome della scatola contenente la domanda.
RISPOSTA_ESATTA sarà la scatola destinata a contenere la risposta esatta.

Tutto questo andrà scritto su una sola riga, che sarà la prima linea della funzione.

DEFINIZIONE DI FUNZIONE

def INTERROGAZIONE (domanda, risposta_esatta):

Questa linea indica a Python che vogliamo definire un blocco di istruzioni che potrà essere eseguito a richiesta in un qualunque altro punto del programma semplicemente scrivendo il suo nome (interrogazione), dopo aver indicato la domanda e la risposta corretta.

Quindi la prima linea della funzione dovrà contenere:

la parolina def (abbreviazione di define, definisci)	**DEF**
il nome che ho deciso di dare alla funzione	**INTERROGAZIONE**
i nomi delle scatole su cui lavorerà la funzione, separati da virgole e racchiusi da parentesi	**(DOMANDA, RISPOSTA_ESATTA)**

Ricordati di mettere sempre i due punti dopo le parentesi e di indentare nel modo corretto, come puoi vedere negli esempi seguenti. La prima linea del programma non più indentata segnala a Python la fine della funzione.

Dopo la definizione, occorre scrivere la funzione, cioè il programma che lavorerà sulle scatole indicate. Nel nostro caso:

```
def interrogazione(domanda, risposta_esatta):
risposta_allievo = input (domanda)
if risposta_esatta == risposta_allievo:
        print ("La risposta e' esatta")
        print ("Bravissimo")
else:
        print ("Risposta errata")
        print ("La risposta esatta e' ", risposta_esatta)
        print ("studia di piu'!")
```

Confrontiamolo ora con il pezzo di programma da cui siamo partiti e che non volevamo ripetere tante volte.

Notiamo subito che il codice è diventato molto più generale perchè tutte le risposte sono scritte dentro le variabili che potranno assumere valori diversi in situazioni diverse.
Mentre il programma valeva solo per domande sugli scrittori, la funzione può essere usata in un programma che pone domande di storia, di geografia, di calcio ecc.
La seguente tabella mostra come è nata la funzione a partire dal programma base.

Programma	Funzione

if risposta == "Manzoni":
  print ("la risposta è esatta")
  print ("bravo!!")
else:
  print ("risposta errata!")
print ("la risposta esatta è: Manzoni")
  print ("Studia di più")
	def interrogazione (domanda,risposta_esatta):
	  risposta_allievo = input (domanda)
	  if risposta_esatta == risposta_allievo:
	  print ("La risposta è esatta")
	  print ("Bravissimo")
	  else:
	  print ("Risposta errata")
	  print (" La risposta esatta è ",risposta_esatta)
	  print ("studia di più!")

Nota bene: puoi usare qualsiasi nome per una funzione, tranne le parole riservate di Python.

CHIAMATA DI FUNZIONE

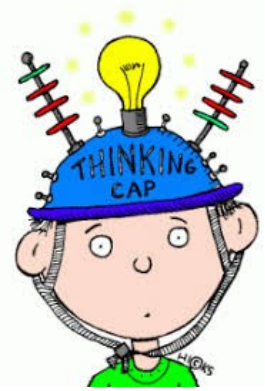
Le istruzioni all'interno di una definizione di funzione non sono eseguite finché la funzione non viene chiamata.
Ora dobbiamo imparare a chiamare la funzione, ossia ordinare alla funzione stessa di eseguire le operazioni in essa contenute.
In generale, la chiamata di una funzione viene effettuata scrivendo il nome della funzione, seguita dai nomi delle scatole su cui lavorare separati da virgole e racchiuse da parentesi.
Nota bene: in linguaggio informatico, i nomi delle variabili su cui opera una funzione sono chiamati "*argomenti della funzione*"

Nel caso del programma che avevamo scritto avremo:

```
domanda1 = "chi e' l'autore dei Promessi Sposi? "
risposta_esatta1 = "Manzoni"
interrogazione (domanda1, risposta_esatta1)
domanda2 = "chi e' l'autore della Divina Commedia? "
risposta_esatta2 = "Dante"
interrogazione (domanda2, risposta_esatta2)
```

PRIMA CHIAMATA

SECONDA CHIAMATA



Fermiamoci a riflettere e a riassumere ciò che abbiamo detto finora a proposito di programmi e del flusso di esecuzione delle singole istruzioni.

L'esecuzione di una funzione inizia sempre alla prima riga del programma e le istruzioni sono eseguite una alla volta dall'alto verso il basso.

Una funzione deve essere definita prima del suo uso perché Python deve sapere che la funzione esiste e cosa fa.

La definizione di funzione non altera il flusso di esecuzione del programma e le istruzioni all'interno della funzione non sono eseguite finché questa non viene chiamata (questo vuol dire che, se definisco la funzione all'inizio del programma, Python legge la definizione ma non fa nulla e prosegue nell'esecuzione del programma finché non trova la chiamata della funzione).

La chiamata della funzione è una deviazione nel flusso di esecuzione: invece di proseguire con l'istruzione successiva, l'esecuzione salta alla prima riga della funzione chiamata ed esegue tutte le sue istruzioni; alla fine della funzione il flusso riprende dal punto dov'era stato deviato dalla chiamata di funzione. Fortunatamente Python è sufficientemente intelligente da ricordare dove il flusso di esecuzione viene via via interrotto e sa dove riprendere quando una funzione è conclusa

Quando il flusso del programma giunge all'ultima istruzione, dopo la sua esecuzione il programma è terminato. Ricorda che la definizione della funzione termina con l'ultima istruzione indentata.

Vediamo ora il nostro programma scritto con e senza l'uso della funzione "interrogazione":

senza	con

risposta1 = input ("chi e' l'autore dei Promessi Sposi? ")
if risposta1 == "Manzoni":
print ("la risposta e' esatta")
print ("bravo!!")
else:
print ("risposta errata!")
print ("la risposta esatta e': Manzoni")
print ("Studia di piu'")
risposta2 = input ("chi e' l'autore della Divina Commedia? ")
if risposta2 == "Dante":
print ("la risposta e' esatta")
print ("bravo!!")
else:
print ("risposta errata!")
print ("la risposta esatta e': Dante")
print ("Studia di piu'")
	def interrogazione (domanda,risposta_esatta):
	risposta_allievo = input (domanda)
	if risposta_esatta == risposta_allievo:
	print ("La risposta e' esatta")
	print ("Bravissimo")
	else:
	print ("Risposta errata")
	print ("La risposta esatta e' ", risposta_esatta)
	print ("Studia di piu'!")
	
	domanda1 = "chi e' l'autore dei Promessi Sposi? "
	risposta_esatta1 = "Manzoni"
	interrogazione (domanda1, risposta_esatta1)
	domanda2 = "chi e' l'autore della Divina Commedia? "
	risposta_esatta2 = "Dante"
	interrogazione (domanda2, risposta_esatta2)

Nota bene: i nomi delle scatole usati nella istruzione di chiamata della funzione sono diversi dai nomi delle scatole usati nella definizione.

def interrogazione (domanda,risposta_esatta): ⟶ DEFINIZIONE
interrogazione (domanda1, risposta_esatta1) ⟶ CHIAMATA

La prima cosa che fa la funzione quando viene chiamata è prendere le scatole usate nella istruzione con cui è stata chiamata, scoperchiarle e mettere il loro contenuto nelle corrispondenti scatole usate all'interno della funzione.
Il contenuto di domanda1 viene scritto in domanda e il contenuto di risposta_esatta1 viene scritto in risposta_esatta.
Perché è necessaria questa operazione?
Le scatole all'interno della funzione sono "invisibili" all'esterno e sono utilizzate solo dentro il corpo della funzione.
In questo modo la semplice funzione che abbiamo scritto potrà essere utilizzata da voi per altri programmi o da qualche vostro amico per costruire programmi più complicati. Questo è il secondo motivo, forse il più importante, per cui si scrivono le funzioni.
I programmi attuali sono diventati cosi complicati che nessun programmatore, per quanto bravo, riuscirebbe a scriverli da solo.
Qualunque programma è sempre composto per una piccola parte da nuove istruzioni e per la maggior parte da funzioni già scritte in precedenza.
Ora che sai cosa sono le funzioni, devi sapere anche che Python è provvisto di numerose e importanti "librerie" di funzioni già scritte, che ricoprono le esigenze dei ricercatori di tutte le aree della scienza.
Ad esempio, una libreria chiamata "**math**" consente di fare calcoli matematici anche molto complicati.
Prima di poter usare le funzioni fornite da math, devi dire a Python di caricare quelle funzioni che ti servono in memoria.
Questa operazione, che si chiama "importazione", può essere ottenuta scrivendo:

```
>>> import math
```

che significa: "Caro Python, ho bisogno di alcune funzioni contenute in una collezione di funzioni matematiche chiamata math".
Si noti che in alcuni contesti questa collezione è chiamata "modulo". In termini più generali, la libreria standard - chiamata "standard Python library – è vista come un insieme di sottolibrerie o moduli, ciascuno dei quali contiene alcune funzioni.
Ricordati di dire sempre a Python in quale sottolibreria o modulo dovrà cercare la funzione che ti serve. Altrimenti, Python si lamenterà di non conoscere il nome della funzione chiamata.

Ti propongo un esercizio basato sull'utilizzo delle funzioni: **inserisci un numero e stampa la sua radice quadrata.**
(Dove, ovviamente, math. è il nome del modulo o sottolibreria , sqrt è il nome della funzione che serve a calcolare la radice quadrata).

```
import math

numero = float(input("Scrivi un numero "))

print ("La radice quadrata di", numero,'è', math.sqrt(numero))
```

ANCORA FUNZIONI. L'ISTRUZIONE RETURN

In tutti gli esempi precedenti, la funzione chiamata eseguiva completamente una certa attività.
Spesso la funzione chiamata esegue dei calcoli il cui risultato serve al programma chiamante.
Vediamo un esempio:

```
def doppio(numero):
    numero_per_due = numero * 2
    return numero_per_due
```

L'istruzione return numero_per_due ordina alla funzione doppio di trasmettere al programma chiamante il valore di numero_per_due.

Nel programma chiamante non si dovrà scrivere numero_per_due perché questa è una variabile della funzione e le variabili delle funzioni sono invisibili all'esterno, ma si scriverà un'istruzione del tipo: pippo = 7 + doppio(5)

```
#definizione di funzione
def doppio(numero):
  numero_per_due = numero * 2
  return numero_per_due

#programma principale
numero1 = float(input("Inserisci un numero "))
pippo = 7 + doppio(numero1)
print ("Se al numero 7 aggiungo il doppio di ",numero1)
print ("il risultato sarà: ", pippo)
```

In sostanza tutto avviene come se il contenuto della scatola della funzione numero_per_due fosse trasferito nella scatola del programma principale doppio (numero1).

Funzioni che chiamano funzioni

Nel corpo di una funzione si può anche scrivere una o più istruzioni che chiamino altre funzioni. Al limite, una funzione può anche chiamare se stessa.
Supponiamo di voler calcolare il fattoriale (prodotto dei successivi numeri interi da 1 fino a un numero dato) del numero 7, ossia

fatt = 7*6*5*4*3*2*1

La funzione che calcola il fattoriale di un numero **n** dovrà essere scritta come **n** moltiplicato per il fattoriale di **(n-1)**;

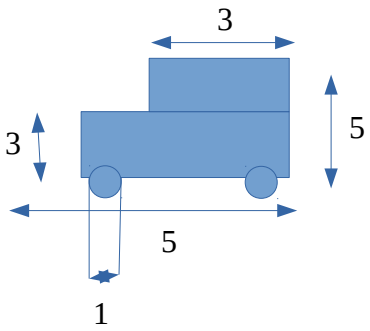
ad esempio, se a **n** attribuiamo il valore **7**:

fatt(7) = 7 * fatt(6)

La funzione sarà:	**Il programma sarà:**

def fatt(n):
  if n == 1:
  return 1
  else:
  return n* fatt(n-1)
	def fatt(n):
	  if n == 1:
	  return 1
	  else:
	  return n * fatt(n-1)
	n = int(input ("Scrivi un numero "))
	print ("Il fattoriale di ", n, " è ",
	  fatt(n))

In questo esempio calcoliamo l'area della figura:



Scriviamo una funzione che consenta di calcolare l'area di un rettangolo:

```
def area_rettangolo (base, altezza):
    return base * altezza
```

Scriviamo una funzione che calcoli l'area di un cerchio di raggio r:

```
def area_cerchio (raggio):
    return 3.14 * raggio**2
```

Utilizziamo ora le due funzioni che abbiamo scritto per calcolare l'area della figura azzurra:

```
# programma che calcola l'area della figura azzurra

def area_rettangolo (base, altezza):
    return base * altezza
def area_cerchio (raggio):
    return 3.14 * raggio**2
area_figura = area_rettangolo (3,2) + area_rettangolo(5,3) + 2  * area_cerchio(1)/2
print ("L'area della figura azzurra è ", area_figura)
```

Quest'ultimo esempio mostra come calcolare la potenza di un numero elevato a un numero intero, usando una funzione definita da noi.

```
def potenza(numero, esponente):
  if esponente == 1:
    return numero
  else:
    return potenza(numero,esponente-1) * numero
numero = float(input("Inserisci un numero "))
esponente = int(input("Inserisci l'esponente "))
print (potenza(numero, esponente))
```

Esercitiamoci un po'

7.1 Scrivi un programma utilizzando la funzione
che chieda due numeri e poi li sommi tra loro.

7.2 Scrivi un programma utilizzando la funzione che concateni due stringhe (per esempio due nomi Sandro e Silvia).

7.3 Trova l'errore:

```
def facciamofesta(musica):
    torte = 5
    print ("Stasera ci sono",torte,"torte, e la musica di", musica)

musica = input("Qual'è il tuo cantante preferito?")
facciamofesta()
```

7.4 Scrivi un programma utilizzando la funzione che chiede qual è il tuo primo piatto preferito e il programma risponde "A me piacciono i peperoni in carpione" e poi chiede qual è il secondo preferito e risponde sempre "A me piacciono i peperoni in carpione".

7.5 Scrivi un programma utilizzando la funzione che chiede 2 numeri e visualizzi la somma, il valore medio e visualizzi il minimo tra i due.

7.6 Scrivi un programma utilizzando una funzione che calcoli l'area e il volume di un parallelepipedo rettangolo.

Lezione 8 : I CICLI

Impariamo insieme a :

- ad usare il ciclo "loop"
- a creare cicli annidati
- quando è opportuno utilizzare l'istruzione while
- oppure usare l'istruzione for
- a conoscere e utilizzare meglio le stringhe

Come già sai una variabile può assumere valori diversi nel corso di un programma.

Ad esempio:

```
Num = 3
print (Num)
Num = 9
print (Num)
```

Il risultato di questo programma è la visualizzazione di 3 9 su due righe diverse, perché la prima volta il valore di Num e` 3, la seconda 9.

Prova a scrivere il seguente programma e analizza cosa succede.

```
a =1
while a < 4 :
  print (a)
  a = a + 1
```

Prova a leggere il programma con l'istruzione while come fosse scritto in un linguaggio naturale (while significa: finchè, fino a quando, fintantoche`).
Otterrai:

"Poni a=1. Finché (while) a è minore di 4, stampa il valore di a e poi aumentalo di 1. Quando arrivi a 4 esci perché a deve essere minore di 4".

Se non ci riesci prosegui a leggere: troverai i due programmi analizzati in dettaglio più avanti.

Come hai capito questo comando viene usato per eseguire le azioni ripetitive e i computer sono bravissimi a ripetere le azioni tante volte. Per risolvere questo tipo di problema si usano i "cicli" o "loop".

Ci sono due tipi di "cicli" e tra i due c'è una differenza molto importante.
Il primo, chiamato **while**, viene usato quando non si sa prima quante volte deve ripetere l'azione.

Il secondo, chiamato **for**, è usato quando si sa in anticipo quante volte dovrà ripetere l'azione.

Tralasciamo per un attimo il ciclo for, che studieremo nella prossima lezione, per imparare bene ad utilizzare:

while <relazione>:

Riprendiamo ora il semplice programmino di prima:

```
a =1
while a < 4 :
    print (a)
    a = a + 1
```

Avete visto che il programma visualizza:

1
2
3

Nota bene:

L'istruzione **while:** significa: ***"fai tutto quello che segue fintantoché` la relazione scritta fra la parola while e i due punti è soddisfatta".***

I due punti sono obbligatori.

In modo più formale ecco il flusso di esecuzione di un'istruzione while:

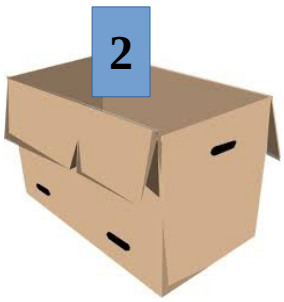
1. Valuta la condizione controllando se essa è vera o falsa.
2. Se la condizione è falsa esci dal ciclo while e continua l'esecuzione dalla prima istruzione che lo segue.
3. Se la condizione è vera esegui tutte le istruzioni nel corpo del while e torna al passo 1.

Nel nostro caso il programma esegue le istruzioni:

```
print (a)
a = a + 1
```

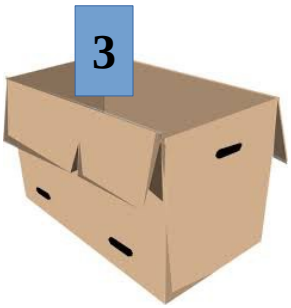
tre volte, cioè finchè a è minore di 4.

Ecco in dettaglio quello che succede in quelle tre volte.



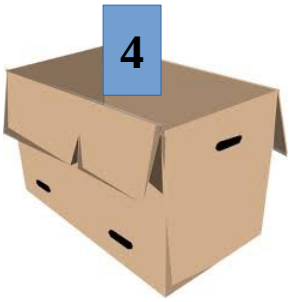
LA PRIMA VOLTA

a vale 1
La relazione a < 4 è soddisfatta
Il programma esegue prima print a
e visualizza 1
poi esegue a = a + 1
e pone il valore 2 nella scatola a



LA SECONDA VOLTA

a vale 2
a < 4 è ancora soddisfatta
Il programma esegue print a e visualizza il valore di a,
cioè 2 e
pone il valore 2 + 1 = 3 nella scatola a



LA TERZA VOLTA

a vale 3
a < 4 è soddisfatta
Il programma visualizza il valore di a, cioè 3,
ed esegue a = a + 1 ponendo 4 nella scatola a.

LA QUARTA VOLTA

a vale 4
Il programma esce dal ciclo.

Nota bene: Tutte le istruzioni che seguono la

while <relazione> :

devono essere indentate se fanno parte delle cose da fare quando <relazione> è soddisfatta.

Se la condizione è falsa al primo controllo, le istruzioni del corpo non saranno mai eseguite.

Il corpo del ciclo dovrebbe cambiare il valore di una o più variabili così che la condizione possa prima o poi diventare falsa e far così terminare il ciclo. In caso contrario il ciclo si ripeterebbe all'infinito e il calcolatore non si fermerebbe mai.

Esamina ora il seguente programma.

```
a = 3
while a > 2:
  print (a)
  a = a + 1
```

Il programma stampa prima 3, poi 4, poi 5 e cosi via, e non si ferma mai.
Perché?

Se volete evitare che il vostro PC visualizzi la successione di numeri naturali (ossia 3,4,5,6…) per molti anni, sulla tastiera premete contemporaneamente i tasti CTRL e C.

ESERCITIAMOCI UN PO'

8.1 ***Prova ad eseguire questi due programmi:***

Programma 1

```
a = 1
while a < 100 :
a = a+1
```

Programma 2
a = 1
while a < 100000 :
a = a + 1

Ora rifletti. *Cosa fanno questi due programmi?*
Perché il secondo dura più del primo?
A cosa possono servire questi programmi?

Qualche volta, nel corso di un programma complesso, può essere opportuno fermarsi un po' di tempo per consentire all'utilizzatore del programma di riflettere prima di proseguire.
I due programmi precedenti servono proprio a questo, ossia a perdere tempo.

Quanto tempo?
Dipende dalla velocità del calcolatore. Infatti, un calcolatore molto veloce può eseguire una delle due istruzioni:

```
while a < 100:
    a = a + 1
```

in un decimo di milionesimo di secondo.

Prova a rispondere a questa domanda: fa perdere più tempo il programma 1 oppure il programma 2?

8.2 Visualizza tutti i numeri compresi fra 3 e 9.

```
i = 3
while i <= 9 :
        print (i)
        i = i + 1
```

8.3 Visualizza tutti i numeri compresi fra 0 e 100.

```
i = 0
while i <= 100 :
        print (i)
        i = i + 1
```

8.4 Visualizza in ordine decrescente i numeri da 200 a 100.

```
i = 200
while i >= 100 :
        print (i)
        i = i - 1
```

8.5 Scrivi il programma "Conto alla rovescia" che - finché (while) n è più grande di 0 - stampa il valore di n e poi lo diminuisce di 1. Quando arriva a 0 stampa la stringa "Pronti…VIA!"-.

```
n = 10
while n > 0 :
        print (n)
        n = n - 1
print ("Pronti ...VIA!")
```

8.6 Eseguendo il programma precedente, scopri che la visualizzazione dei numeri è troppo veloce. Come fai per rallentarla?
(la soluzione l'hai già trovata in questo STEP!)

8.7 Visualizza la tabellina del sette stampando i numeri di seguito.

```
i = 1
while i <= 10 :
        print (7*i)
        i = i + 1
```

8.8 Scrivi un programma che attende finchè non viene inserita la password corretta.
(La password la decidi tu, ad es. "chicchiricchi").

```
password = "python"
while password != "chicchiricchi":
   password = input("Password:")
print ("Entra pure")
```

8.9 Visualizza tutti i numeri dispari compresi fra 1 e 100.

```
i = 1
while i < 100 :
   print (i)
   i = i + 2
```

8.10 Stampa la frase "Aiuto! Non mi fermo più" infinite volte usando il ciclo while.
(Per fermare il programma devi ...)

```
i=1
while i != 0 :
    print ("Aiuto! Non mi fermo più")
    print
    i = i + 1
```

e poi stampala di seguito, senza saltare una riga e senza andare a capo.
(Per fermare il programma devi ….)

```
i=1
while i != 0 :
    print ("Aiuto! Non mi fermo più")
    i = i + 1
```

8.11 *** Prova a trovare i divisori di un numero usando il ciclo while.
(Se non riesci, la soluzione è illustrata nelle pagine seguenti)

```
i=1
n = int(input ("qual è il numero di cui vuoi trovare i divisori? "))
print (n, " è divisibile per ")
while i < n :
    if n%i == 0 :
        print (i)
    i = i + 1
```

8.12 *** Scrivi un programma che chieda il risultato dell'operazione 15 x 17
tante volte sino a quando non viene indicata la soluzione corretta.
(Se non riesci, la soluzione è illustrata nelle pagine seguenti)

```
corretto = "no"
while corretto == "no":
    risposta = float(input ("quanto vale 15 x 17? "))
    if risposta == 15*17:
        corretto = "si"
        print ("Brava!")
    else:
        corretto = "no"
```

SECONDA PARTE: I CICLI ANNIDATI

Precedentemente abbiamo risolto il programma n. 10.5 ma quando lo eseguiamo scopriamo che il conteggio alla rovescia è molto veloce.
Troppo.
Avete trovato la soluzione per rallentarlo?
Se si, bene! Leggete comunque come abbiamo illustrato la soluzione qui di seguito.

Per rallentarlo introduciamo il programma che abbiamo già scritto e che serviva a perdere tempo (esercizio 10.1)

```
a = 1
while a < 100 :
        a = a+1
```

e nella relazione da verificare decidiamo quanto aspettare. La soluzione che conta lentamente non è altro che la fusione dei due programmi.

```
n = 10
while n > 0 :
        print (n)
        a = 1
        while a < 1000000 :
                a = a + 1
        n = n - 1
print ("Pronti ...VIA!")
```

Nota bene: Quest'ultimo programma contiene due cicli while, di cui il secondo, quello introdotto per "perdere tempo", è **ANNIDATO** entro il primo.

Notate bene anche come è stato scritto il programma:
l'istruzione while a < 1000000 è indentata rispetto a while n > 0
mentre l'istruzione a = a + 1 è indentata rispetto a while
a < 1000000

Quante volte viene eseguita l'istruzione a = 1?
Quante volte viene eseguita l'istruzione a = a + 1?

Riprendiamo adesso due esercizi fatti in precedenza (quelli con ***).
La soluzione del primo e' un esempio di annidamento dove l'istruzione annidata è una if. La soluzione è la seguente:

```
i=1
n = int(input ("qual è il numero di cui vuoi trovare i divisori? ")
print (n, " è divisibile per ")

while i < n :
    if n%i == 0 :
        print (i)
    i = i + 1
```

Nel secondo la if annidata è più complessa.

```
corretto = "no"
while corretto == "no":
    risposta = int(input ("quanto vale 15 x 17? "))
    if risposta == 15*17:
        corretto = "si"
        print ("Brava!")
    else:
        corretto = "no"
```

Fai attenzione all'indentazione con cui sono scritte le istruzioni perché è fondamentale per avere un'esecuzione corretta del programma!

Vediamo qualche altro esempio:

Scrivi un programma per stampare il quadrato e il cubo dei numeri da 1 a 10.

```
i = 1
while i <= 10 :
    e = 2
    while e <= 3 :
        print (i ** e, " ",)
   e = e + 1
    print
    i = i + 1
```

Scrivi un programma che chiede una sequenza di numeri da aggiungere ad una somma. Per terminare inserisci 0.

```
a = 1
somma = 0
print ('Inserisci i numeri da aggiungere alla somma ')
print ('Quando hai finito inserisci 0')
while a != 0 :
        print ('La somma è:',somma)
        a = float(input('Numero? '))
        somma = somma + a
print ('Totale =',somma)
```

Scrivi un programma per stampare la Tavola Pitagorica.

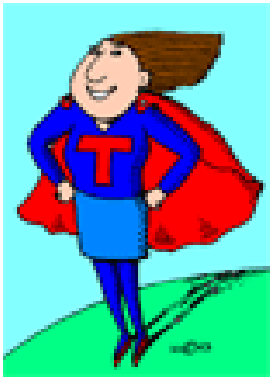
Attenzione: la stampa dei numeri non risulterà bene in colonna.
Prova trovare il modo per stampare bene le colonne.

```
print (" TAVOLA PITAGORICA")
riga = 1
while riga <= 10 :
    colonna = 1
    while colonna <= 10 :
        print ('\t', riga * colonna)
        colonna = colonna + 1
    riga = riga + 1
```

Indovina un numero! Scrivi un programma per indovinare un numero

```
numero = 27
indovina = 0
while indovina!= numero :
    indovina = float(input ("Indovina un numero: "))
    if indovina > numero:
        print ("Troppo grande")
    elif indovina < numero :
            print ("Troppo piccolo")

print ("BRAVO!!")
```

Se non sei riuscito a stampare in maniera ordinata le colonne?



Nessun problema.
Adesso ti spiego un "trucco" per formattare
le stampe.

Il carattere di backslash '\' (barra inlinata rovesciata) indica l'inizio di quella che viene chiamata una "sequenza di escape ".
Le sequenze di escape sono usate per rappresentare caratteri "speciali" e invisibili come la tabulazione ('\t') e il ritorno a capo ('\n') e possono comparire in qualsiasi punto di una stringa.
Prova ad esempio a stampare una stringa unica che produca questo risultato:

vengo
subito
a casa

print ("vengo \n \t subito \n \t \t a casa")

….ancora sulle STRINGHE

Sinora abbiamo visto che una stringa è una serie di caratteri, cifre, lettere o altri simboli che si trovano sulla tastiera, cioè un messaggio qualunque. Inoltre, sappiamo che una stringa per essere considerata tale deve essere racchiusa tra virgolette (semplici o doppie) e che si possono fare alcune operazioni con le stringhe.
Possiamo scrivere:
"ciao" * 3 oppure "ciao" + "ciao" + "ciao" oppure "ciao" * 2 + "ciao"
invece
"ciao"/4 oppure "ciao" + 5 oppure "18" + 8
sono sbagliate e generano un syntax error.
Vi ricordate come si chiamano le operazioni sulle stringhe?

Che cosa hanno di diverso le stringhe dagli altri tipi di dati (interi, floating point)?
Le stringhe sono qualitativamente diverse dagli altri tipi di dati perchè sono composte di unità più piccole: i caratteri.
Per questo le stringhe sono dati "composti" in alternativa ai dati "semplici" che sono gli interi e i floating point.

Questo ci consente di trattare una stringa come fosse una singola entità oppure di agire sulle sue singole parti (i caratteri) a seconda di ciò che stiamo facendo. Come si fa ad agire sui singoli caratteri?

Secondo te quale sarà il risultato di questo programma?
squadra = "Juventus"
messaggio = squadra[0] + squadra [1] + squadra [2] + squadra [3] + squadra [4] + squadra [5] + squadra [6] + squadra [7]
print (squadra)
print (messaggio)

Come si individuano i singoli caratteri.
Nella variabile squadra abbiamo messo la stringa tutta intera mentre nella variabile messaggio l'abbiamo inserita un carattere per volta.
squadra[i] e` una stringa fatta di un solo carattere e non e` altro che il carattere di Juventus che occupa la posizione i + 1.

L'espressione tra parentesi quadrate seleziona i caratteri della stringa. Quindi squadra[0] seleziona il primo carattere, [1] il secondo e cosi via.
Ricordati che i calcolatori iniziano sempre a contare da 0.
L'espressione tra parentesi quadrate è chiamata indice.
Un indice individua un particolare elemento di una stringa e può essere una qualsiasi espressione intera.
Un'espressione aritmetica negativa come –i fra parentesi quadre [-i] indica il carattere posto nella posizione (i + 1) a partire dal fondo della stringa.
Cosi, ad esempio:
squadra[-0] è uguale al carattere "s"
squadra[-2] è uguale al carattere "t"

Cosa otterrai scrivendo il seguente programma?
squadra = "Juventus"
messaggio = "VIVA" + " " + squadra[0] + squadra[1] + squadra[2] + squadra[3]
print (squadra)
print (messaggio)

>>>Juventus
>>>VIVA Juve

Lezione 9: LE LISTE

In questo step impareremo:

- cos'è una lista
- quando è utile utilizzare le liste
- a creare liste annidate
- ad esaminare tutti gli elementi di una lista utilizzando il ciclo "for"

Cos'è una lista? È come quella che si fa quando bisogna fare la spesa? Come quando si fa l'elenco degli amici da invitare alla festa per il compleanno? Esatto!

Una lista è un insieme ordinato di valori di qualunque tipo, proprio come la lista della spesa.

Vediamo qualche esempio:

`[3,6,9,12,15]` lista di tutti numeri interi

`["Luigi","Mario","Nicola","Giuseppina"]` lista di tutte stringhe

`["pane","latte","zucchero",1,15,230,"bicchieri",1.5,2.5]` lista mista:

3 stringhe, 3 numeri interi, 1 stringa, 2 numeri decimali

I valori della lista sono chiamati "elementi" .
Le parentesi quadrate [ ] iniziano e finiscono la lista e la virgola separa un elemento della lista dall'altro.
Come abbiamo visto in uno degli esempi precedenti, non è obbligatorio che gli elementi di una lista siano tutti dello stesso tipo, tutti numeri o tutte stringhe. Una lista può essere non omogenea.

Esiste poi una lista speciale chiamata "lista vuota" e si indica con [ ].

La lista vuota non contiene alcun elemento.

Altri esempi:

```
["pane","latte","zucchero",1,15,230,"bicchieri",1.5,2.5]
[1,"due",3.0]
[]
["tabellina del cinque",5,10,15,20,25]
```

Ovviamente, alle liste dobbiamo dare un nome:

```
spesa=["pane","latte","zucchero",1,15,230,"bicchieri",1.5,2.5]
vocabolario = ["bicicletta","casa","scuola"]
dati_di_buffon = ["Buffon", "Gianluigi", 1978]
lista_vuota = []
```

Prova a visualizzare sul video il contenuto delle liste precedenti.

Le liste sono un tipo di variabile. Variabili "composte".

Le variabili ordinarie immagazzinano un singolo valore.
Le liste sono variabili che possono contenere più di un valore. Possiamo pensarle come armadi composti da una serie di cassetti e scatole.

Vediamo ora come si fa ad accedere ai singoli elementi di una lista.
Prova a scrivere il seguente programma e analizza attentamente il risultato:

```
Squadre = ["Juventus", "Inter", "Milan", "Roma"]
i= 0
while i < 4:
  print (Squadre [i])
  i=i+ 1
```

Questo programma assegna alla lista `squadre` alcuni valori e con il ciclo `WHILE` esamina tutti gli elementi della lista e ne visualizza un elemento alla volta.

Per farlo usa un indice (nel nostro caso "i"). `Squadre[i]` individua l'elemento i della lista `squadre`.

In questo modo è possibile accedere separatamente a ciascun elemento della lista.

Nota bene: Fai attenzione al fatto che gli indici di una lista, cosi come gli indici di una stringa, iniziano sempre da zero; può sembrarti strano, ma i calcolatori preferiscono iniziare a contare da zero.
Quindi per accedere al primo elemento di una lista dobbiamo scrivere [0] e non [1].

Esercitati a selezionare gli elementi di tutte le liste che abbiamo usato come esempio e prova a scrivere un ciclo while per esaminare e visualizzare gli elementi su video, come nell'esempio seguente, con la lista: `[3,6,9,12,15]:`

1. Innanzitutto dobbiamo dare un nome alla lista:

```
tabellina = [3,6,9,12,15]
```

2. Poi scrivere il ciclo per attraversare tutti gli elementi della lista:

```
i= 0
while i < 5:
    print (tabellina [i])
    i=i+ 1
```

Il programma sarà dato dalla somma di 1 più 2.

Indici

Si accede agli elementi di una lista in modo simile a quello già visto per accedere ai caratteri di una stringa.

`print (tabellina [1])` Visualizza il secondo elemento della lista: tabellina

Una qualsiasi espressione che produca un intero può essere un indice. Ad esempio:
`tabellina[5-1]` visualizza: 15

Prova a usare un numero decimale come indice e analizza il messaggio di errore che compare: `TypeError: sequence index must be integer`

Prova a leggere o a modificare un elemento che non esiste. Ad esempio:

```
tabellina[7] = 2
```

L'interprete cerca di assegnare a `tabellina [7]` il valore 2, ma tabellina [7] non esiste.

Anche qui analizza attentamente il messaggio di errore:

```
IndexError: list assignment index out of range
```

Cosa succede se l'indice ha valore negativo? Esempio: `tabellina [-2]`

Se un indice ha valore negativo il conteggio parte dalla fine della lista. Per verificarlo prova a scrivere: `tabellina[-3]`. Questo metodo di usare l'indice negativo si rivela molto comodo quando non si conosce la lunghezza della lista. Sappiamo che se scriviamo [-1] accederemo all'ultimo elemento, [-2] al penultimo e così via.

Lista speciale

È la lista composta da tutti numeri interi, talmente comune che Python fornisce un

modo speciale per crearla. La funzione `list(range)` è lo strumento usato per crearla. Infatti, scrivere `[0,1,2,3,4]` è equivalente a scrivere `list(range(5))`.

Cioè, la funzione `list(range)` crea una lista che parte da 0 e arriva fino al numero indicato dall'indice -1.

`list(range (10))` è equivalente a `[0,1,2,3,4,5,6,7,8,9]`

Con `list(range)` si può scrivere: `list(range (1,5))` che è equivalente a `[1,2,3,4]`

La funzione `list(range)` in questo caso legge dalla lista di interi gli elementi che partono dal primo indice incluso e arrivano all'ultimo indice escluso.

Nota Bene: La funzione list`(range)` ha sempre le parentesi tonde.

Con list(range) possiamo scrivere anche espressioni del tipo: `list(range (1,10,2))` che è equivalente a `[1,3,5,7,9]`

Il terzo indice si chiama "passo" e indica con quale intervallo leggere i numeri dalla lista di interi partendo da 1 e arrivando a 10 escluso. In questo caso legge il primo, il terzo, il quinto e cosi via fino a 10-1, ottenendo una lista di numeri dispari.

Che lista ottieni con questa funzione? list(range(2,10,2))

Lunghezza di una lista

La funzione `len` applicata ad una lista produce il numero di elementi di una lista, come nelle stringhe.

```
allievi=["Luigi","Marco","Filippo","Paola","Gabriella","Silvia"]
i= 0
while i < len(allievi):
          print (allievi[i])

          i= i + 1
Luigi Marco Filippo Paola Gabriella Silvia
```

Operazioni sulle liste

Come per le stringhe, l'operatore + concatena le liste:

```
allievi_3A=["Luigi","Marco","Filippo","Paola","Gabriella","Silvia"]
allievi_4E = ["Gabriele","Alessandro","Anna","Michela","Antonio"]
allievi = allievi_3A + allievi_4E
print (allievi)

['Luigi', 'Marco', 'Filippo', 'Paola', 'Gabriella', 'Silvia',
'Gabriele', 'Alessandro', 'Anna', 'Michela', 'Antonio']
```

L'operatore * ripete una lista un dato numero di volte.

1° esempio: Se **numeri** é la lista [3,6,9,12,15], numeri* 3 é la lista
[3, 6, 9, 12, 15, 3, 6, 9, 12, 15, 3, 6, 9, 12, 15,].

2° esempio: print(allievi_4E * 2) produce:

```
['Gabriele', 'Alessandro', 'Anna', 'Michela','Antonio','Gabriele',
'Alessandro', 'Anna', 'Michela', 'Antonio']
```

Sezione di lista

Una sezione di lista è l'analogo di una sezione di stringa e ha la stessa logica di funzionamento. Infatti, come per le stringhe, si adotta l'operatore porzione :

Se amici é la lista [“Gianni”,”Luigi”,”Carlo”]

```
print (amici [0:3])

produce:

[“Gianni” “Luigi” “Carlo”]

print (amici [1:3])
produce:
[“Luigi” “Carlo”]
```

Se **allievi_4E** é la lista
["Gabriele","Alessandro","Anna","Michela","Antonio"]

```
print(allievi_4E [1:3])
produce:
[“Alessandro,Anna”]
```

Ricordati: il primo indice incluso, il secondo indice escluso. Con l' operatore [] possiamo eliminare uno o più elementi di una lista assegnando alla corrispondente sezione una stringa vuota.

```
allievi_4E = ["Gabriele","Alessandro","Anna","Michela","Antonio"]
allievi_4E [1:3] = []
print (allievi_4E)
["Gabriele","Michela","Antonio"]
```

Ma non è così pratico ed è facile sbagliare. Con la funzione del è molto più facile cancellare un elemento da una lista.

```
Squadre = ["Juventus", "Inter", "Milan", "Roma"]
del Squadre [1]
print (Squadre)
["Juventus","Milan","Roma"]
```

Analogamente possiamo inserire uno o più elementi in una lista inserendoli in una sezione vuota nella posizione desiderata.

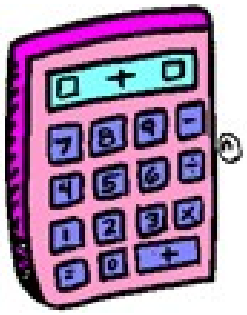
1° esempio: se **allievi_4E** é la stringa
["Gabriele","Alessandro","Anna","Michela","Antonio"]
l'istruzione print (allievi_4E [2:2])
produrrà: []
2° esempio: se
allievi_4E [2:2] = ["Sandra", "Andrea"]
l'istruzione print (allievi_4E) produrrà:
["Gabriele","Alessandro","Sandra","Andrea","Anna","Michela",
"Antonio"]

Cosa ottengo se scrivo quanto segue?

```
allievi_4E =
["Gabriele","Alessandro","Anna","Michela","Antonio"]

print (allievi_4E  [:4])
print (allievi_4E  [3:])
print (allievi_4E [:])
```

Se non viene specificato il primo indice la porzione parte dall'inizio della stringa. Senza il secondo indice la porzione finisce con il termine della stringa. Se mancano entrambi gli indici si intende tutta la lista.

Liste annidate



Un elemento di una lista può essere a sua volta una lista.

```
tabellina = [3,6,9,12,15,[10,20,30]]
print (tabellina [5])
[10 20 30]
```

Il sesto elemento della lista `tabellina` è a sua volta una lista.

Nell'esempio seguente la lista `allievi_4E` è diventata una lista di liste:

```
allievi_4E=[["Bianchi","Gabriele"],["Verdi","Alessandro"],
["Rossi","Anna"],["Neri","Michela"],["Viola","Antonio"]]
print (allievi_4E[2])
["Rossi", "Anna"]
```

Cosa ottengo se scrivo `print (allievi_4E[2][0])` ?

Ottengo il risultato: `Rossi`, che è l'elemento 0 della sottolista 2.

Posso estrarre un elemento da una lista annidata con due metodi differenti.

Il primo è il seguente:

```
allievo = allievi_4E [2]
allievo [0]
```

Creo una lista che si chiama `allievo` e prendo il primo elemento di quella nuova lista.

Il secondo mi permette di scrivere direttamente: `allievi_4E [2][0]`

Questa espressione deve essere letta da sinistra verso destra: trova il terzo elemento [2] della lista `allievi_4E`
ed essendo questo elemento a sua volta una lista, ne estrae il primo elemento [0].

Ancora una domanda non troppo difficile. Qual è la lunghezza della lista seguente?

```
allievi_4E =
[["Bianchi","Gabriele"],["Verdi","Alessandro"],["Rossi","Anna"],
["Neri","Michela"],["Viola","Antonio"]]
```

Il ciclo FOR

(ancora più semplice esaminare tutti gli elementi di una lista una lista)

Riprendiamo l'esempio:

```
allievi =["Luigi","Marco","Filippo","Paola","Gabriella","Silvia"]
i= 0
while i < len(allievi):
          print (allievi[i])
          i=i+ 1
Luigi
Marco
```

```
Filippo
Paola
Gabriella
Silvia
```

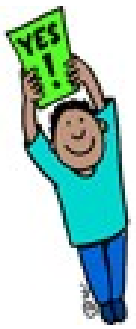
Questo ciclo può essere scritto anche in questo modo:

```
allievi = ["Luigi","Marco","Filippo","Paola","Gabriella","Silvia"]
for allievo in allievi:
   print ("Ora stampo l'allievo: ", allievo)
```

Possiamo leggere quasi letteralmente: "Fai quello che segue per ciascun allievo nella lista allievi, lavorando la prima volta sul primo elemento, la seconda sul secondo, e cosi via. Stesso risultato, ma il programma è più breve. In termini generali devo scrivere:

```
for <nome di variabile> in <nome di lista> :
```

L'istruzione `for` significa: per ciascuna variabile della lista fai tutto quello che segue. In modo più formale la lista viene analizzata dal primo elemento sino all'ultimo. La prima volta:



1. alla variabile `allievo` assegna il valore `allievi[0]`
2. esegue i comandi che seguono i due punti
3. ritorna all'istruzione `for`

e prosegue fin quando trova un elemento nella lista.
Negli esempi che seguono cerchiamo di capire a cosa può servire il ciclo `for`.

Stampiamo il quadrato dei primi 11 numeri interi. Il programma seguente:

```
for numero in list(range(11)):
   print ("il quadrato di ", numero, " è ", numero * numero)
```

produce:

```
il quadrato di 0 è  0
il quadrato di 1 e` 1
il quadrato di 2 e` 4
il quadrato di 3 e` 9
il quadrato di 4 e` 16
il quadrato di 5 e` 25
```

```
il quadrato di 6 e` 36
il quadrato di 7 e` 49
il quadrato di 8 e` 64
il quadrato di 9 e` 81
il quadrato di 10 e` 100
```

Misuriamo ora la lunghezza di alcune parole. Il programma seguente:

```
vocabolario=['alba','banana','carro','dodici','giacca','finestra']
for parola in vocabolario:

  print (parola, len(parola))
```

produce:

```
alba 4
banana 6
carro 5
dodici 6
giacca 6
finestra 8
```

ESERCIZI

9.1 Scrivi un programma che analizzi le seguenti stringhe e per ognuna visualizzi le lettere una ad una: "banana", "cipolla", "bambino", "finestra", "girotondo".

9.2 Scrivi un programma che sommi tutti gli elementi di una lista data di numeri.

9.3 Scrivi un programma che stampi la "successione di Fibonacci". Cioè, la somma di due elementi definisce l'elemento successivo (1,2,3,5,8,13.....)

9.4 Scrivi un programma per trovare i numeri primi in una lista di numeri che va da 2 a 20.

9.5 Scrivi un programma che conta il numero di volte in cui la lettera 'a' compare in una stringa, usando un contatore.

9.6 Scrivi un programma che operi su una lista di numeri e conti il numero di volte in cui un valore cade in un determinato intervallo.

ADESSO GIOCHIAMO A TRIS !

Dopo tanto studio ti meriti di giocare un po'.

Il programma che segue non solo ti permette di giocare a "Tris" ma è anche un esempio di programma che puoi realizzare grazie a quanto hai imparato sinora.

Per chi avesse voglia e molto tempo libero per affrontare un problema più difficile, descriviamo sommariamente il codice del gioco.

Buon divertimento!

Il programma è caratterizzato da numerose funzioni per facilitarne la comprensione.
La prima funzione, chiamata *scacchiera* consente la visualizzazione della scacchiera sullo schermo. Si ricorda che il carattere '\t' (carattere di tabulazione) serve, in sostanza, a saltare un numero predefinito di spazi nella riga.
La prima istruzione di questa funzione

```
print ('\t 1 \t 2 \t 3 \n')
```

serve a visualizzare i caratteri "1", "2" e "3" nella prima riga delle caselle di gioco e successivamente, in virtù del carattere "\n" , ad andare a capo.
La seconda riga visualizzerà il carattere "a" e, dopo le tabulazioni, le tre indicazioni relative alle tre caselle della prima riga del gioco. Ad esempio, a1 può assumere tre valori distinti:

- "x" che indica che la casella è ancora libera;
- "N" che indica il nome del primo giocatore;
- "R" che rappresenta il nome del secondo giocatore.

La terza e la quarta riga hanno gli stessi valori relativi alla seconda e terza riga della tastiera.
Ovviamente, la funzione *scacchiera* visualizza la scacchiera con i valori aggiornati ad ogni chiamata (vedremo che sarà chiamata diverse volte nel programma).

```
def scacchiera ():
    print ('\t 1 \t 2 \t 3 \n')
    print ('a \t', a1, '\t', a2, '\t', a3, '\n')
    print ('b \t', b1, '\t', b2, '\t', b3, '\n')
    print ('c \t', c1, '\t', c2, '\t', c3, '\n')
```

E quindi il risultato sarà il seguente:

	1	2	3
a	x	x	x
b	x	x	x
c	x	x	x

Vediamo ora sommariamente le altre istruzioni del programma.
La funzione *vittoria* effettua il controllo della situazione attuale ad ogni chiamata e quindi indicherà l'esistenza o meno di un vincitore.

```
def vittoria (n):
    if ((a1 == n and a2 == n and a3 == n)   or
       (b1 == n and b2 == n and b3 == n) or
       (c1 == n and c2 == n and c3 == n)  or
       (a1 == n and b1 == n and c1 == n)   or
       (a2 == n and b2 == n and c2 == n) or
       (a3 == n and b3 == n and c3 == n) or
       (a1 == n and b2 == n and c3 == n)  or
       (a3 == n and b2 == n and c1 == n)):
         EsisteVincitore = 'si'
    else:
        EsisteVincitore = 'no'
    return EsisteVincitore
```

Le prossime istruzioni permettono di scrivere il carattere "x" nelle nove caselle di gioco.

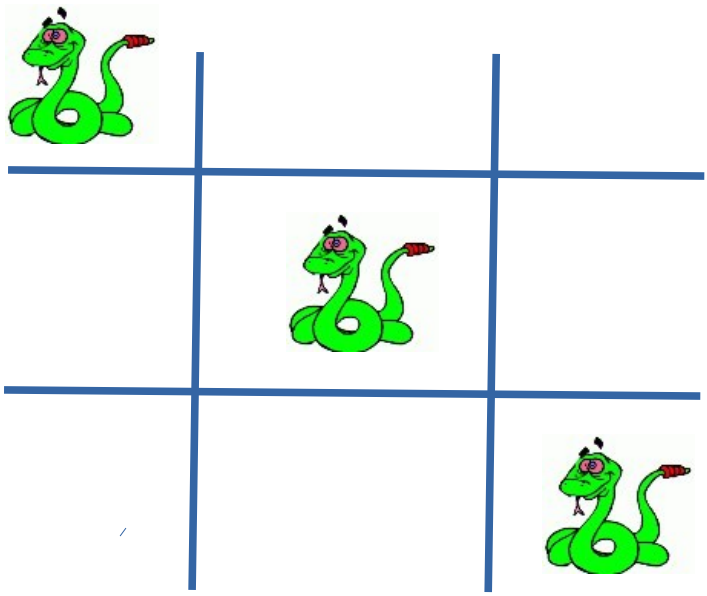
```
a1 = 'x'
a2 = 'x'
a3 = 'x'
b1 = 'x'
b2 = 'x'
b3 = 'x'
c1 = 'x'
c2 = 'x'
c3 = 'x'
```

Le prossime istruzioni permetto di visualizzare la scacchiera e di fare la prima mossa.
Nota bene: i commenti nel codice, che iniziano con il simbolo "#" permettono di capire quali istruzioni saranno eseguite nelle righe successive.

```
# Visualizzo la scacchiera
scacchiera ()

# inizializzo la variabile che indica quando la partita e' finita
partitafinita = 'no'

# inizializzo la variabile nmosse che conta il numero delle mosse fatte dai giocatori
nmosse = 0


# Prima mossa
print ('Il primo giocatore sarà  N \n')
# Ciclo while, valido finché il numero di mosse è minore di 9
while nmosse < 9 :
        print ('Muova N \n')
        mossafatta = 'no'
        scelta = input ('Dove vuoi scrivere N: a1, a2, a3, b1, b2, b3, c1, c2, c3? \n')
        if scelta == 'a1' and a1 == 'x':
            a1 = 'N'
            mossafatta = 'si'
        elif scelta == 'a2' and a2 == 'x':
            a2 = 'N'
            mossafatta ='si'
        elif scelta == 'a3' and a3 == 'x':
            a3 = 'N'
            mossafatta ='si'
        elif scelta == 'b1' and b1 =='x':
            b1 = 'N'
            mossafatta ='si'
        elif scelta == 'b2' and b2 == 'x':
            b2 = 'N'
            mossafatta ='si'
        elif scelta == 'b3' and b3 == 'x':
            b3 = 'N'
            mossafatta ='si'
        elif scelta == 'c1' and c1 == 'x':
            c1 = 'N'
            mossafatta ='si'
        elif scelta == 'c2' and c2 == 'x':
            c2 = 'N'
            mossafatta ='si'
```

```
elif scelta == 'c3' and c3 == 'x':
                c3 = 'N'
                mossafatta ='si'

            # Controllo la validità della mossa
            if mossafatta == 'no':
                print ('SQUALIFICATO!')
                partitafinita = 'si'
                nmosse = 10

            scacchiera ()
            nmosse = nmosse + 1
            partitafinita = vittoria ('N')


            # Se la partita è finita, indico il vincitore
            if partitafinita == 'si':
                print ('Ha vinto N \n')
                nmosse = 10
            elif   nmosse == 9:
                print ('PAREGGIO')
                nmosse = 10

            # I seguenti comandi sono uguali a quanto visto prima
            # ma sono per il secondo giocatore, “R”
            if partitafinita == 'no' and nmosse < 9:
                nmosse = nmosse +1
               mossafatta = 'no'
                print ('La mossa tocca a R \n')
                scelta = input ('Dove vuoi scrivere R: a1, a2, a3, b1, b2, b3, c1, c2, c3? \n')
                if scelta == 'a1' and a1 == 'x':
                    a1 = 'R'
                    mossafatta = 'si'
                elif scelta == 'a2'and a2 == 'x':
                    a2 = 'R'
                    mossafatta = 'si'
                elif scelta == 'a3'and a3 == 'x':
                    a3 = 'R'
                    mossafatta = 'si'
                elif scelta == 'b1'and b1 == 'x':
                    b1 = 'R'
                    mossafatta = 'si'
```

```
elif scelta == 'b2'and b2 == 'x':
                    b2 = 'R'
                    mossafatta = 'si'
                elif scelta == 'b3'and b3 == 'x':
                    b3 = 'R'
                    mossafatta = 'si'
                elif scelta == 'c1'and c1 == 'x':
                    c1 = 'R'
                    mossafatta = 'si'
                elif scelta == 'c2'and c2 == 'x':
                    c2 = 'R'
                    mossafatta = 'si'
               elif x == 'c3'and c3 == 'x':
                    c3 = 'R'
                    mossafatta = 'si'

                if mossafatta == 'no':
                    print ('SQUALIFICATO!')
                    partitafinita = 'si'
                    nmosse = 10

                scacchiera ()
                partitafinita = vittoria ('R')
                if partitafinita == 'si':
                    print ('Ha vinto R \n')
                    nmosse = 10
```

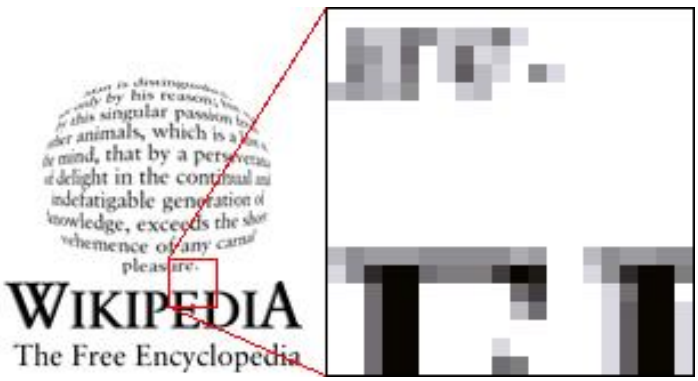


Lezione 10: La Grafica

In questo passo imparerai a scrivere programmi in Python che ti consentiranno di produrre semplici disegni e animazioni sul video.
Per comprendere bene come funziona il video di un personal computer e come scrivere poi le istruzioni che ti consentiranno di disegnare sul video, devi prima studiare i paragrafi seguenti.

La codifica delle immagini.

Osserva attentamente le immagini della televisione o del video del PC. Esse sono formate da una serie di quadratini molto piccoli, quasi puntini, uno vicino all'altro.
Ingrandendo l'immagine i quadratini si vedranno molto bene. Ad esempio, se ingrandiamo un pezzo della scritta Wikipedia riconosciamo i quadratini che compongono l'immagine.

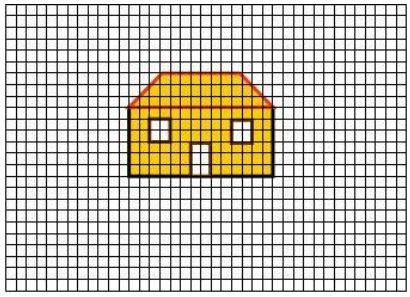


I pixel

Nella TV i quadratini sono circa 500 su ogni riga e ci sono circa 300 righe sullo schermo, per un totale di circa 150.000 quadratini.
Questi quadratini si chiamano **"pixel"** (**picture element** o "elemento di un'immagine"), e sia sul PC che sulla TV possono essere in bianco e nero o colorati. Adesso comincia ad analizzare i puntini in bianco e nero perché quelli colorati sono un poco più complicati.

Inoltre, supporrai quasi sempre che un quadratino sia bianco oppure nero. Nella grafica più raffinata un quadratino può essere non soltanto bianco e nero ma anche grigio più o meno scuro.

Prendi ora l'immagine di una casetta e pensa di rappresentarla sul monitor del calcolatore. Sovrapponi poi al disegno una griglia di quadratini ognuno dei quali rappresenta un "pixel". Lascia il quadratino in bianco se nel quadratino c'è più bianco che nero, oppure annerisci il quadratino se c'è più nero (o colore).
Ora prova a farlo con due griglie diverse: la prima ha i quadratini di 1 cm e la seconda di 0,5 cm.



Verificherai che se i quadratini sono più piccoli il disegno verrà meglio; in linguaggio tecnico si dice che è più definito.

La definizione di un'immagine si chiama risoluzione ed è il numero di quadratini che ci sono in una data area.
Un'immagine ha una risoluzione più alta quando i quadratini per area sono tanti, ha una risoluzione bassa se i quadratini sono pochi.
La risoluzione di uno schermo si misura moltiplicando il numero di pixel di ogni riga per il numero di righe.
Ad esempio, la risoluzione dello schermo televisivo, che è 500 x 300, corrisponde a 300 righe ciascuna delle quali è composta da 500 pixel.
La risoluzione dello schermo TV è molto bassa; sui video dei PC usati per applicazioni di grafica si arriva a risoluzioni di 4096 x 3300 pixel.

Come hai appena visto, un'immagine viene rappresentata con quadratini e la perfezione del disegno dipende dal numero dei quadratini e quindi dai bit. Servono tantissimi bit per rappresentare bene un'immagine.
L'enorme quantità di bit necessaria per rappresentare bene un'immagine è il motivo per cui ci vuole tanto tempo per "scaricare" un'immagine da Internet.
Poiché le immagini sul video sono composte da tanti quadratini, uno vicino all'altro, o se

preferisci da tanti bit, anche un'immagine può essere rappresentata sul calcolatore come una lunga fila di bit. Si dice che i pixel possono essere accesi (quando il quadratino è nero o colorato) o spenti (quando il quadratino è bianco) e l'immagine si può rappresentare specificando quali pixel sono accesi e quali spenti.
Questo si può dindicare con un codice, ad esempio 0 = spento, 1 = acceso.

Lo schermo in bianco e nero

Per gli schermi a colori il discorso è un po' più complesso; per ora quindi cerca di comprendere come funziona lo schermo in bianco e nero. Inoltre supponi che il puntino possa essere soltanto bianco o nero, ma non grigio.

L'immagine della casetta si trasforma in una lunga fila di 1 e 0 (di bit). Se pensi questi bit come dei quadratini fosforescenti, puoi dire che = 1 significa quadratino acceso, =0 significa quadratino spento (in parole povere, luce o non luce su un certo quadratino dell'immagine).
Ricorda che con 150.000 bit puoi descrivere un'immagine in bianco e nero su uno schermo televisivo, ma ne hai bisogno di 500.000 per descrivere la stessa immagine sul PC.
Questo perché il video del PC ha molte più righe e molte più colonne; anzi sul PC puoi decidere tu quante righe e colonne usare. Partendo da 800 colonne x 600 righe, puoi arrivare fino ai più potenti video che hanno 14096 colonne x 2160 righe.

L'immagine nel calcolatore sarà una lunga sequenza di 0 e 1, che saranno usati proprio come comandi per decidere se accendere o spegnere i pixel da una apposita "memoria" del PC; infatti il video del PC ha una memoria che si chiama "memoria video", dove sono scritte le sequenze di 0 e 1 che comandano la luce sui pixel.

Quindi la memoria video, in cui è descritta l'immagine come una sequenza di 0 e 1, trasmette al video i comandi da eseguire per rappresentare l'immagine.

L'operazione di trasformazione di un'immagine in una sequenza di bit si chiama anche "**discretizzazione**".
"Immagine digitale", "TV digitale", "macchina fotografica digitale" ecc.; con queste espressioni indichiamo tutti gli apparati le cui immagini sono rappresentate con numeri (che specificano la luminosità dei singoli quadratini).

VOCABOLARIO ELETTRONICO	**PIXEL** = picture element = elemento di un'immagine. **RISOLUZIONE** = numero di pixel per unità di lunghezza, si misura in DPI (dot X inch, punti per pollice). **IMMAGINE DIGITALE** = immagine rappresentata con 0 e 1 (quadratini o pixel). **MEMORIA VIDEO** = memoria apposita del PC per il comando del video. **DISCRETIZZARE** = trasformare un'immagine in bit (0 e 1). Si può dire anche "numerizzare" o "digitalizzare".

I moduli e le funzioni

In conclusione, potresti produrre un disegno sul video scrivendo un programma fatto da molte istruzioni che precisino i punti dello schermo che vuoi illuminare uno alla volta. Ma occorrerebbero troppe istruzioni e molto tempo per produrre disegni anche molto semplici.
Fortunatamente altri programmatori prima di te hanno scritto dalcune funzioni che ti consentono di disegnare parti di un disegno con uno sforzo molto piccolo.
Queste funzioni sono organizzate in moduli.

Il modulo che useremo per disegnare è chiamato: `turtle` (tartaruga)

Il primo disegno!

Scriviamo insieme un primo semplice programma che servirà a disegnare un palloncino.
Il programma inizierà con un'istruzione un po' strana:

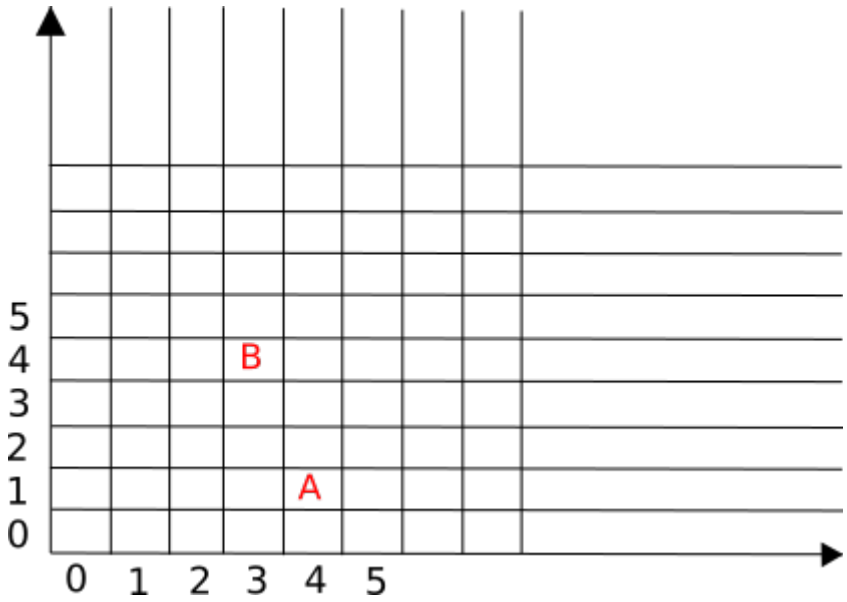
```
from turtle import *
```

che significa:
"*Caro Python, ho bisogno delle funzioni del modulo turtle. Per favore mettimi a disposizione ("`import`") tutte le funzioni di questo modulo.*

Adesso sei in grado di disegnare il tuo palloncino, ma per continuare è importante sapere cosa sono le "coordinate". Se ancora non lo sai, chiedi alla tua insegnante di matematica di spiegarti cosa sono le coordinate.

Abbiamo detto che i pixel del video sono individuati dalle coordinate (riga e colonna),

come nella figura che vediamo sotto.



Il pixel A ha "ascissa" 4, perché sta nella colonna 4 e "ordinata" 1, perché sta nella riga 1 e la sua posizione si indica con A(4,1).

Invece, il pixel B avrà ascissa uguale a 3 e ordinata uguale a 4, e quindi B(3,4).

Iniziamo a disegnare

Il nostro programma è per ora : `from turtle import *`

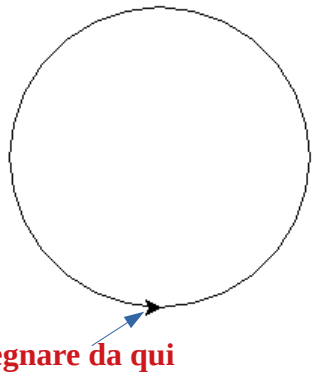
Dopo l'esecuzione su FARE dell'istruzione `from turtle import *`, automaticamente si apre a video una terza finestra che sarà chiamata "finestra dei disegni". Infatti tutti i disegni che noi produrremo compariranno in questa terza finestra.
Per capire la logica dei disegni che intendiamo realizzare, dobbiamo immaginare una tartaruga piccolissima, come un pixel, che inizialmente si trova al centro della finestra dei disegni.

Ora disegnamo il palloncino e per questo usiamo la funzione: **circle**
Chi sa un po' l'inglese avrà già capito che questa funzione serve a disegnare un cerchio ("circle").

L'istruzione che devi scrivere è facilissima: `circle(30)`

Se **30** è il raggio del cerchio che vogliamo disegnare per rappresentare il palloncino (30 quadratini o pixel), il nostro programma completo sarà:

```
from turtle import *
circle(30)
```



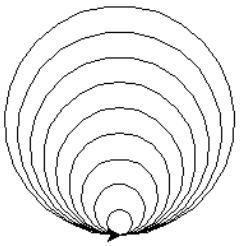
Come puoi verificare facilmente a video, è la tartaruga che, proprio come una matita, disegna il cerchio di raggio 30, partendo dal centro della finestra dei disegni.
Se ad esempio cambiamo il valore del raggio da 30 a 100 otteniamo un palloncino più grande.

La tartaruga inizia a disegnare da qui

Giocando con le istruzioni che abbiamo imparato possiamo ottenere tanti disegni. Ad esempio, con il programma :

```
i=1
while i<10:
        circle (10*i)
        i=i+1
```

produciamo il disegno a lato.



Se vuoi disegnare il cerchio in un'altra area della finestra dei disegni, devi spostare la tartaruga, ricordando che il punto occupato dalla piccolissima tartaruga rappresenta il punto più basso del cerchio.
Per spostare la posizione della tartaruga dobbiamo utilizzare l'istruzione:

`setposition(x,y)` che porrà la tartaruga nel punto di ascissa `x` e coordinata `y`.

Quando tu disegni normalmente su un foglio, alzi e abbassi la matita per spostarti da un punto all'altro del disegno e così dobbiamo fare anche noi. Poichè la tartaruga rappresenta la matita per disegnare, prima di eseguire l'istruzione `setposition` dobbiamo inserire l'istuzione `up()` che significa "solleva la matita dal foglio" e dopo la stessa istruzione dobbiamo scrivere `down()` che significa "appoggia la matita sul foglio"
Ad esempio:

```
up()
setposition(20,20)
down()
```

Esercitati a disegnare, giocando con la dimensione e la posizione del palloncino. Cambia le coordinate della tartaruga e la dimensione del raggio. Inizia con questo semplice

programma che disegnerà una conchiglia:


```
for i in range(10):
        up()
        setposition(10*i,10*i)
        down()
        circle(i*10)
```

Giochiamo con i colori

Tutto avviene come se avessi tre matite colorate: una rossa, una verde e una blu. Ad esempio la scritta (50,50,200) indica un colore fatto con un po' di rosso, un po' di verde e molto blu.

Per chiedere alla tartaruga di utilizzare una matita rossa sul disegno, posso scrivere l'istruzione:

```
color("red")
```

Se invece voglio disegnare con la matita blu, o verde, debbo utilizzare l'istruzione color("blue") oppure color("green"), rispettivamente.

E' anche possibile usare un modo più raffinato per specificare la sfumatura di colore desiderata, basato sulla *notazione standard* (Red,Green,Blue) spesso chiamata (RGB), ove ogni componente può assumere un valore compreso fra 0 e 255.

Ad esempio, il colore (255,0,0) indica il rosso vivo; il colore (0,255,0) indica il verde scuro; il colore (0,100,100) indica la miscela di verde e blu molto intenso senza alcuna componente di rosso.

Per specificare un colore facendo riferimento a questa notazione, occorre scrivere: colormode (255) seguito, ad esempio, da color (165,42,42). In questo caso otterremo un bel colore marrone. Per utilizzare invece la notazione color ("red") o color ("green") o color ("blue") occorre premettere l'istruzione colormode (1).

Di seguito sono indicati i programmi per disegnare tre palloncini di colore rosso, verde e blue, nei due modi: colormode(255) e colormode (1).

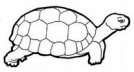
```
colormode (255)
color (255,0,0)
circle (40)
color (0,255,0)
circle (60)
color (0,0,255)
circle (80)
```

```
colormode (1)
color ("red")
circle (40)
color ("green")
circle (60)
color ("blue")
circle (80)
```

La tabella dei colori disponibili è molto ampia e può essere consultata all'indirizzo: https://www.tcl.tk/man/tcl8.4/TkCmd/colors.htm

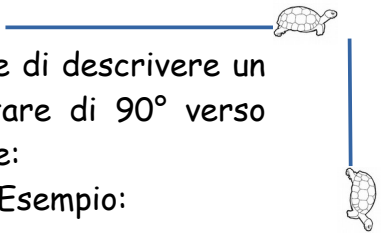
Il disegno dei segmenti

Nell'istante iniziale che si definisce automaticamente dopo l'istruzione `from turtle import*`, la tartaruga occupa una posizione predefinita (il centro della finestra del disegno) ed è orientata in una posizione prestabilita. Tutto avviene come se la sua corazza fosse orientata orizzontalmente con la testa rivolta verso destra.



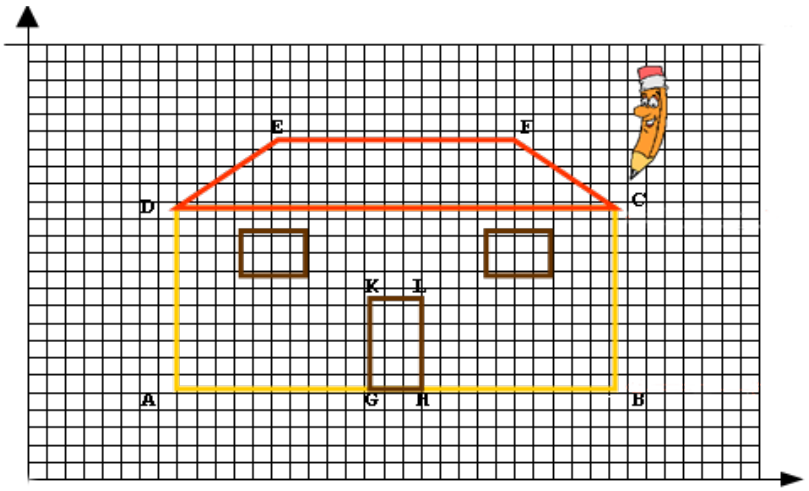
Così se scriviamo: `forward (150)`

come prima istruzione la tartaruga descrive un segmento di 150 pixel nella direzione orizzontale da sinistra a destra.



Se dopo aver descritto questo segmento la tartaruga decide di descrivere un segmento verticale verso il basso, il suo corpo deve ruotare di 90° verso destra. L'istruzione che ordina questa rotazione è la seguente:
`right(angolo)` dove l'angolo deve essere espresso in gradi. Esempio:
`right (90).`

Per imporre una analoga rotazione a sinistra occorrerà scrivere: `left(angolo)`. Esempio: `left(50)`



Ad esempio, per disegnare i segmenti CB,BA e AD della casetta disegnata qui sopra, occorrerà scrivere le seguenti istruzioni:

```
from turtle import *

def pos(x,y):
    up()
    setposition(x,y)
    down()
pos(150,200)
```

```
color("yellow")
right(90), forward(150), right(90), forward(400), right(90),
forward(150)
```

Si noti la funzione `pos (x,y)` *che consente di saltare nella posizione (x,y) senza precisare up() e down() ad ogni chiamata. Nel nostro caso il disegno parte dal punto C (vedi immagine della matita a destra della casetta) come specificato nell'istruzione* `pos(150,200)` *.*

Successivamente la tartaruga riceve l'ordine di ruotare di 90° verso destra e di descrivere un segmento di 150 pixel (segmento giallo CB). Quindi la tartaruga ruota una seconda volta verso destra di 90° e disegna un segmento di 400 pixel (segmento BA). Infine, la tartaruga ruoterà una terza volta verso destra di 90° e disegnerà un segmento verticale di 150 pixel (segmento AD).
Per disegnare il tetto della casetta (segmenti DC,CF,FE e ED) occorrerà scrivere le seguenti istruzioni:

```
color("red")
right(90), forward(400), right(225), forward(200), right(315),
forward(120), right(315), forward(200)
```

Infine è facile verificare che le finestre e la porta possono essere disegnate con il seguente codice (nota nel programma l'uso dell'istruzione `color` che serve a cambiare il colore dei segmenti):

```
color("brown")
pos(100,160)
right(45), forward(30),  right(90), forward(30), right(90),
forward(30), right(90), forward(30), right(90)
pos(-160,160)
forward(30), right(90), forward(30), right(90), forward(30),
right(90), forward(30), right(90)
color("brown")
pos(-30,50)
forward(30), right(90), forward(70), right(90), forward(30),
right(90), forward(70), right(90)
```

Segmenti, casette, palloncini...non è facile ma considera che è la prima volta che ti cimenti in un lavoro così impegnativo.
Adesso prova ad inventare qualche nuovo disegno per impratichirti un po'. Un suggerimento? Leggi la pagina seguente e guarda cosa puoi ottenere scrivendo poche righe di codice:

DISEGNIAMO UN ESAGONO

programma	*descrizione*
from turtle import * lati = 6	Introduco nella variabile *lati* il numero dei lati della figura geometrica.
lunghezza = 70	Numero dei pixel di ciascun lato della figura.
angolo = 360.0/lati	Angolo compreso tra due lati.
for i in range (lati):	Pongo i = 0,1,2,3,4,5
forward (lunghezza) right (angolo)	Traccia un segmento di 70 pixel e ruota la tartaruga di 60°
write (“Ecco un esagono”)	

Prova ora a cambiare il numero dei lati del poligono (esempio: lati = 8) e la scritta indicando il giusto nome del poligono che hai disegnato (triangolo, pentagono, ottagono….).

Lezione 11: l'Animazione e il Fumetto

In questo passo impareremo:

- come far volare un palloncino
- più in generale, come programmare la creazione di immagini in movimento

Il Palloncino

Nel modulo turtle, il cursore principale che indica la posizione ed il verso della punta della nostra "penna" è graficamente rappresentato dal simbolo ">". Questo simbolo si può cambiare utilizzando una funzione dedicata chiamata ***shape*** (dall'inglese "forma"). I programmatori del modulo turtle hanno messo a disposizione diverse forme richiamabili grazie ai loro nomi inglesi, quali ad esempio:

- "arrow", per indicare la "freccia";
- "turtle", per la "tartaruga";
- "circle", per il "cerchio";
- "square", per il "quadrato";
- "triangle", per rappresentare un "triangolo".

Prova anche tu a cambiare la forma del cursore semplicemente chiamando la funzione ***shape*** e fornendo come primo ed unico parametro una tra le stringhe sopracitate (ricordati che sono stringhe, dunque devono essere inserite tra doppi apici "").

Ad esempio, per utilizzare una tartaruga, possiamo scrivere:

```
shape("turtle")
```

Questa funzione è fondamentale per il proseguo del libro. Finora abbiamo visto come il cursore può disegnare un palloncino ma, siccome questo capitolo è dedicato all'animazione, vediamo ora come farlo volare verso l'alto!

Per far ciò, ci serve innanzitutto cambiare la forma del cursore dal semplice ">" ad un cerchio. Per farlo, utilizzeremo la funzione shape che abbiamo appena visto con il parametro "*circle*":

```
shape("circle")
```

A questo punto avremo un piccolo cerchio posizionato al centro della finestra grafica. Per cambiare il colore di questa figura possiamo usare la funzione già vista in

precedenza chiamata color specificandone il colore tra parentesi (sempre in inglese). Ad esempio, *color*("*red*"), colorerà il nostro cerchio di rosso.

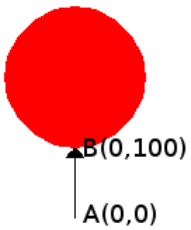
Questo sarà l'output in un primo momento.

Siccome questo puntino non può rappresentare un vero e proprio palloncino, possiamo facilmente allargarne il diametro utilizzando la funzione *shapesize*, dall'inglese "dimensione della forma", che abbiamo introdotto poco fa.
Provando con *shapesize*(5) riusciamo ad ottenere questo risultato:

Prova a cambiare la dimensione cambiando il numero passato come argomento alla funzione *shapesize*. Più si alza il numero e più incrementa il diametro!

A questo punto non ci resta che far muovere il cursore.
Conosciamo già le funzioni che permettono di muovere il cursore all'interno della finestra grafica, quale ad esempio *forward*(). In questo caso impariamo ad utilizzare un'altra funzione che si chiama ***goto***, dall'inglese "vai a". Questa funzione ci permette di indicare le coordinate, indicate come ascisse e ordinate, finali dove vogliamo spostare il nostro cursore. Ricordiamoci che la posizione di partenza è il punto di ascissa 0 e ordinata 0, ovvero il centro della nostra finestra grafica per turtle.

E' tutto pronto per far muovere il cursore. Rimane però un problema, ovvero come facciamo per evitare che venga tracciata una linea mentre il palloncino si muove? Semplice, usiamo la funzione ***penup***, sempre dall'inglese "penna alzata". Quindi ricordiamoci di "alzare la penna" prima di impartire delle nuove coordinate al nostro cursore.
A questo punto serve un po' di geometria: se vogliamo far volare il palloncino verso l'alto, quali nuove coordinate dovremo fornire alla funzione goto?
Una traslazione verso l'alto implica una variazione sull'asse delle ordinate ma non delle ascisse, quindi la funzione goto riceverà sempre delle coordinate fisse per le ascisse (in questo caso 0) e aggiornate per le ordinate (siccome ci muoviamo verso l'alto dobbiamo sommare dei numeri positivi alla posizione precedente).
Ad esempio, la funzione ***goto(0,100)*** produrrà uno spostamento dalla posizione verso l'alto di 100 pixel, ovvero dalla posizione (0,0) a quella (0,100).
Vedi in figura:



Invece, una variazione sull'asse delle ascisse mantenendo le ordinate fisse produrrebbe una traslazione in orizzontale, ovvero la funzione ***goto(100,0)*** produce uno spostamento di 100 pixel verso destra dal punto iniziale.

Detto questo, non ci resta che provare!
Il codice è il seguente:

```
from turtle import  *
# Imposto il colore
color("red")

# Selezioniamo la forma da utilizzare, in questo caso un cerchio
shape("circle")
# Selezioniamo le dimensioni
shapesize(3)
# Ora “alzo la penna” siccome non voglio tracciare una linea durante il movimento
penup()
# Sposto il cerchio in alto di 100 pixel
goto(0,100)
```

Ottimo! Il nostro palloncino si è spostato verso l’alto.

Manca ancora qualcosa? Si, ovvero l’effetto di animazione.

Per ottenere un’animazione è necessario ripetere questo movimento più volte. Ricordi l’istruzione in Python che permette di ripetere diverse volte un’istruzione? Se non te lo ricordi, te lo suggeriamo noi, è il costrutto *while.*

Il nostro programma dovrà eseguire queste istruzioni:

“sposta il palloncino verso l’alto di qualche pixel finché...”

Come finiamo la frase? Siccome sappiamo che costrutto while deve avere un limite per evitare di costruire un programma che non termina mai, dobbiamo impostare un termine a questa iterazione. Nel nostro caso, vogliamo fermare il palloncino quando questo raggiunge il soffitto! Quindi all’interno del nostro ciclo while controlleremo che la coordinata y sia ancora valida, ovvero che il palloncino non abbia ancora sbattuto contro il soffitto.

Vediamo come:

```
x = 0
y = 0

# Inizio il ciclo
while y < 200:
  # Aggiorno solo le ordinate ad esempio di 10
   y = y+10
   goto(x,y)
```

A questo punto il palloncino volerà verso l'altro fino al raggiungimento del soffitto e a quel punto si fermerà e il programma terminerà!

Sei riuscito a far volare il palloncino rosso?
ALLORA...RAGGIUNTA LA META !

Tutto quello che abbiamo visto finora, però, è una rappresentazione di un palloncino e non un palloncino vero e proprio.
Come possiamo fare per far volare verso l'alto un vero palloncino? Semplice, possiamo cercare un'immagine di un palloncino e usarla come "forma" del nostro cursore! In questo modo, possiamo far diventare il nostro puntino rosso un vero e proprio palloncino, come questo:

Come utilizzare un'immagine

Spesso è utile utilizzare nei nostri programmi immagini trovate sulla rete Internet.
Per chi usa il Firefox o altri navigatori ("browser") la soluzione più semplice è la seguente:

1. Si cerca l'immagine desiderata su un motore di ricerca. (Si consiglia di effettuare la ricerca con indicazioni del tipo: "immagini Paperino, gif". O, al limite, "Paperino, gif". Infatti Turtle tratta solo immagini in formato gif.)
2. Si seleziona l'immagine scelta.
3. Con il tasto destro del mouse si impartisce l'ordine "Copia indirizzo"

A questo punto si può usare quell'immagine dentro ad un programma come ad esempio il seguente per far volare un palloncino.
Si inizia scrivendo due istruzioni misteriose (per il momento):

```
screen = Screen()
```

che serve a caricare una serie di funzioni che utilizzeremo per interagire con le componenti dello schermo in un sottoinsieme che chiamiamo "screen"
In altri termini, Screen è una classe, che possiamo immaginare come una famiglia di funzioni: screen è un insieme di dati che noi assembliamo utilizzando le funzioni di Screen per sviluppare il nostro programma.

```
screen.addshape("indirizzo dell'immagine")
```

con questo comando aggiungiamo alla lista dlle immagini predefinite (da "arrow" a "triangle" indicate all'inizio di questo capitolo)l'immagine specificata attraverso il suo indirizzo. Successivamente si scriveranno le due seguenti istruzioni:

```
shape("indirizzo dell'immagine")
```

che serve a creare una nuova forma che si aggiunge alla lista di forme vista all'inizio di questo capitolo e

```
shapesize(1) o shapesize(d)
```

per specificare le dimensioni dell'immagine che sarà utilizzata. (Valore "1" per precisare che si accetta la dimensione attuale, oppure valore "d" per eventualmente aumentarne le dimensioni di un fattore d).
In sostanza, con le ultime due istruzioni, si crea un'immagine prelevandone il campione dalla nuova lista e modificandone le dimensioni in funzione delle esigenze.
Siamo ora in grado di far volare un'immagine del palloncino più raffinata di un semplice cerchio.

Il codice sorgente del programma sarà:

```
from turtle import *
# Importo l'immagine come forma
screen = Screen()
screen.addshape("palloncino.gif")

# Uso la forma
shape("palloncino.gif")
# Dichiaro le dimensioni
shapesize(1)

# Posizione iniziale
x = 0
y = 0
# Alzo la "penna"
penup()

# Muovo verso l'alto
while y < 200:
    y = y+10
    goto(x,y)
```

Si consiglia, dopo aver identificato l'indirizzo di un'immagine come "palloncino.gif", di introdurre una nuova variabile con un nome nuovo scrivendo, ad esempio:
palla = "palloncino.gif".

Ricordati che quello che hai imparato negli step precedenti costituisce solo un inizio per saper programmare e pertanto, se vuoi, dovrai approfondire l'argomento in tempi successivi.
Intanto consolida le tue conoscenze ed esercitati utilizzando quello che hai appreso finora.

Come spostare un'immagine utilizzando la tastiera

Supponiamo di aver creato una nuova immagine sul video operando nel modo sopra descritto.
Vi è la possibilità - invero difficile, per cui si sconsiglia la lettura di questo paragrafo a chi non abbia molto tempo – di modificare quell'immagine attraverso spostamenti ed

eventuali aggiunte di segmenti di testo.
Ad esempio, possiamo spostare l'immagine appena creata premendo il tasto "↑" sull'insieme delle quattro frecce direzionali della tastiera. Questo spostamento sarà determinato da un segmento di codice, posto, come vedremo, al fondo del programma. Questo sarà basato sulla funzione:

```
onkey(su, "Up")
```

A sua volta onkey chiamerà la seguente funzione:

```
def su():
  setheading(90)
  forward(5)
```

Il programma dapprima varierà di 90° in senso antiorario l'angolo di direzione dello spostamento rispetto all'asse orizzontale e, in un secondo momento, muoverà di 5 pixel l'oggetto grafico nella nuova direzione.
In modo analogo sarà possibile assegnare agli altri tasti del tastierino numerico funzioni per effettuare diverse operazioni. Nel nostro caso, le seguenti funzioni sono state definite:

```
def sinistra():
    setheading(180)
    forward(5)

def destra():
    setheading(0)
    forward(5)

def giu():
    setheading(270)
    forward(5)
```

e quindi alla pressione delle frecce sulla tastiera corrisponderà un movimento analogo sul monitor.
Ecco il codice completo:

```
from turtle import *
screen = Screen()
screen.addshape("palloncino.gif")
```

```
penup()
# Uso la forma
shape("palloncino.gif")
 # Dichiaro le dimensioni
shapesize(1)

# Dichiaro le quattro funzioni di movimento
def su():
  setheading(90)
  forward(5)

def sinistra():
    setheading(180)
    forward(5)

def destra():
    setheading(0)
    forward(5)

def giu():
    setheading(270)
    forward(5)

# Registro le funzioni
onkey(su, 'Up')
onkey(sinistra, 'Left')
onkey(destra, 'Right')
onkey(giu, 'Down')

listen()
```

Si noti che è obbligatorio chiudere il codice con la chiamata della funzione listen.

La creazione di un fumetto

Vuoi divertirti a creare semplici fumetti e scenette? Facile!

Negli esempi precedenti abbiamo visto come si fa a trasferire un'immagine selezionata in rete nell'ambiente turtle.
Ora possiamo utilizzare questo sistema per i nostri fumetti.

Innanzitutto cerchiamo in rete un'immagine per lo sfondo e scriviamo:
`screen.bgpic` ("indirizzo dell'immagine di sfondo").
Ove la funzione bgpic serve proprio per creare uno sfondo.

Poi proviamo a sovrapporre un'immagine allo sfondo, procedendo come nel caso del palloncino visto precedentemente. Ad esempio:
`screen.addshape` ("indirizzo dell'immagine")

Infine per aggiungiamo un segmento di testo (abbiamo scelto il colore bianco); sarà necessario scrivere:
`color("white")`
`write` ("testo")

Il testo apparirà adiacente all'immagine, come nel nostro fumetto.

In questo secondo esempio, per posizionare la scritta dove vogliamo, occorre spostare la tartaruga usando le frecce direzionali.
Una volta scelta la posizione preferita, si deve schiacciare il tasto "P" della tastiera, che eseguirà la funzione:

```
def p():
   color (black)
   write ("testo")
```

Il codice completo sarà:

```
import turtle

screen = turtle.Screen()
screen.bgpic("matrimonio.gif")

move_speed = 10

def forward():
  turtle.setheading(90)
  turtle.forward(move_speed)

def backward():
   turtle.setheading(270)
   turtle.forward(move_speed)
```

```
def left():
  turtle.setheading(180)
  turtle.forward(move_speed)

def right():
  turtle.setheading(0)
  turtle.forward(move_speed)

def p():
  turtle.color("black")
  turtle.write("Minni: \n quando ci sposiamo?", False, align="center",
font=("Arial", 20, "bold"))

def a():
  turtle.color("black")
  turtle.write("Topolino: \n quando questo libro sarà finito", False,
align="center", font=("Arial", 20, "bold"))

turtle.penup()
turtle.speed(0)
turtle.home()

screen.onkey(forward, "Up")
screen.onkey(backward, "Down")
screen.onkey(right, "Right")
screen.onkey(left, "Left")
screen.onkey(p, "P")
screen.onkey(a, "A")
screen.listen()
```

LINK SITI IMMAGINI

http://www.clipartguide.com/_pages/0511-1111-0813-0133.html

http://clipartwork.com/clip-art/download-laptop-clipart-

3194/http://www.clipartpanda.com/categories/thinking-cap-images

http://www.massere.it/didattica/85-geometria/98-il-cubo.html

http://clipartix.com/wp-content/uploads/2017/07/Computer-clip-art-free-download-clipart-images-3-3.png

http://cliparting.com/free-question-mark-clip-art-7622/

http://moziru.com/explore/Woman%20clipart%20computer%20programmer/#gal_post_4346_software-clipart-computer-programming-12.png

http://school.discovery.com/clipart/category/tchr0002.html

www.free-graphics.com/clipart/

http://www.fotosearch.com/clip-art/box_2.html

http://www.awesomeclipartforeducators.com/

http://www.theteachersguide.com/Freebies.html

www://freepix.com

www.phillipmartin.info/

INDICE