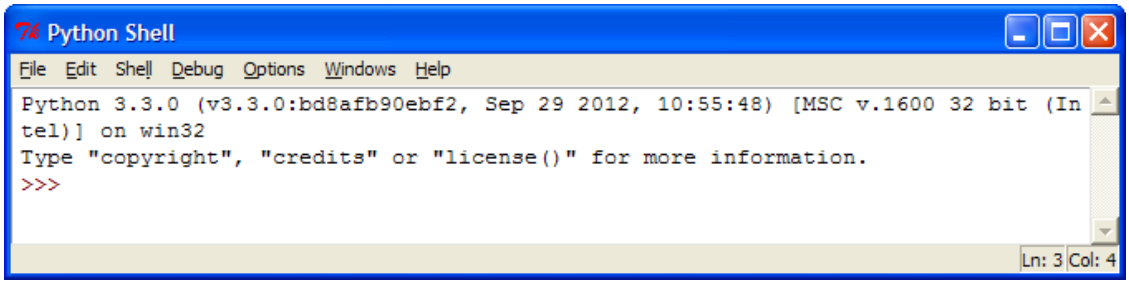
Python è un linguaggio di programmazione interpretato, interattivo e orientato agli oggetti, ideato negli anni novanta; può essere appreso rapidamente; Python è distribuito con licenza Open-Source ed il suo utilizzo è libero e gratuito. Per approfondimenti consultare i siti: www.python.org e www.python.it.

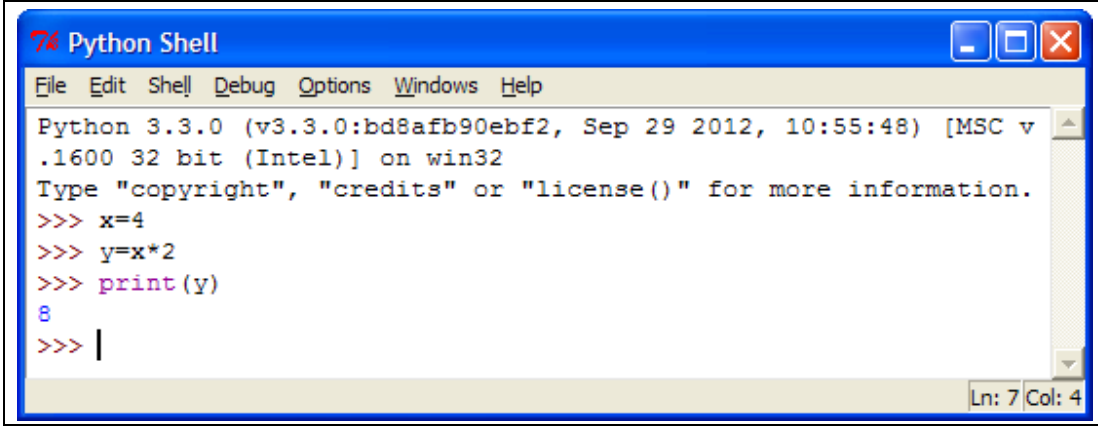
Selezionando l'icona **IDLE (Python GUI)**, si apre la finestra Shell di Python:



Questa è l'interfaccia grafica di Python 3.3.0 che consente di editare, eseguire e testare i programmi.
I caratteri >>> sono il prompt dell'interprete di Python

>>> significa che l'interprete è pronto a ricevere istruzioni

E' possibile definire una **variabile**, semplicemente dandole un nome ed un valore iniziale.

Codice in LINGUAGGIO Python ed esecuzione interattiva	descrizione
	Assegnato ad una variabile di nome x il valore 4, si può assegnare ad una variabile di nome y il risultato della espressione x*2 (x moltiplicato 2) e poi scrivere il valore di y
	PSEUDOCODIFICA
	X ← 4 Y ← x*2 SCRIVI (y)
	OUTPUT
	8

print è l'istruzione che consente di scrivere a video il contenuto di una o più variabili e/o **costanti**

si può far precedere il valore di y da una stringa di testo

PSEUDOCODIFICA	Codice Python	OUTPUT
SCRIVI ("y vale: ", y)	>>> print ('y vale:', y)	y vale: 8

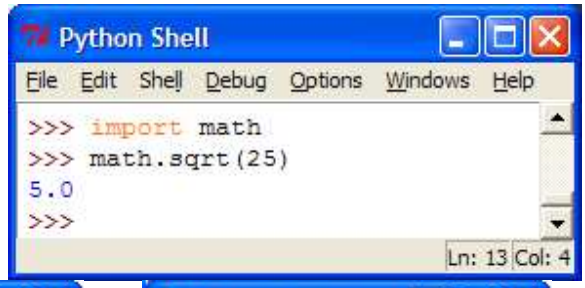
Gli **operatori aritmetici** utilizzabili in una espressione sono: + - * / ** //
(rispettivamente: piu meno per diviso potenza div.intera)

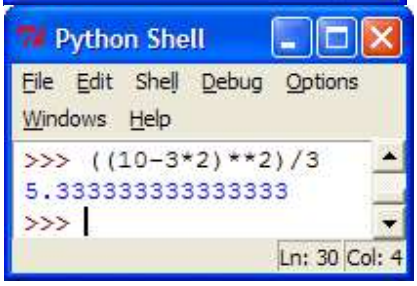
Le **parentesi tonde** si usano per alterare l'ordine consueto di esecuzione delle operazioni (vengono eseguite prima le potenze, poi moltiplicazioni e divisioni, infine addizioni e sottrazioni).

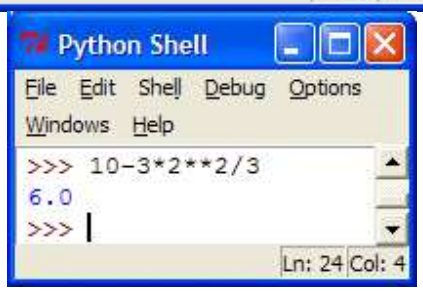
L'interprete di Python qui è usato come una semplice calcolatrice.

Il risultato di un'espressione con operatore / è un numero reale; gli altri operatori se operano su dati interi determinano risultato intero, altrimenti un valore reale.

Alcune funzioni matematiche di uso comune possono essere richiamate includendo il modulo *math* mediante il comando **import**.

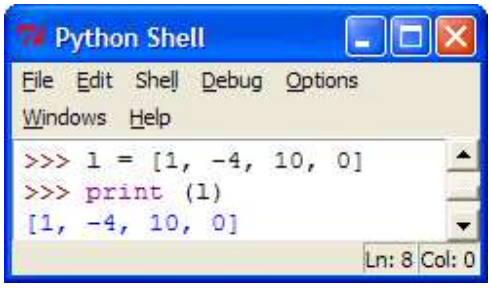




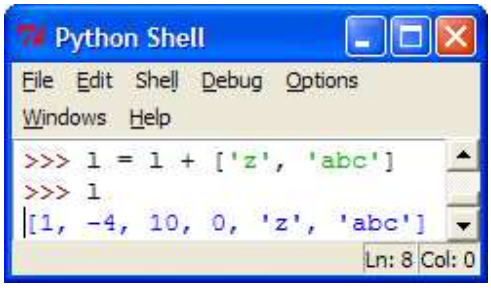


Oltre a dati elementari di tipo numerico **interi** e **floating point** (per contenere rispettivamente valori interi e reali) Python prevede anche dati strutturati, ad esempio le liste.

Una **lista** è un elenco ordinato di elementi anche di tipo diverso (numeri, stringhe di testo, etc.).
Utilizzando l'operatore + è possibile accodare elementi ad una lista.



A sinistra: definizione di una variabile `l` *(elle)* di tipo lista a cui viene assegnato l'elenco di valori 1, -4, 10 e 0 *(zero)*.
La lista `l` viene poi stampata.
A destra: si accodano altri 2 elementi ad `l`; i valori accodati sono le stringhe 'z' e 'abc'.
Poi si visualizza il contenuto di `l`.



Per vedere il contenuto di una variabile si può semplicemente scrivere il nome della variabile dopo il prompt:
`>>> l` l'interprete espone il valore (strutturato) che la lista `l` *(elle)* assume in quel momento.

Questa modalità è quella già utilizzata per ottenere dall'interprete il risultato di un'espressione aritmetica; se dopo il prompt digitiamo il nome di una variabile elementare come y dell'esempio precedente:

```
>>> y
8
```

possiamo vedere il contenuto della variabile y in quel momento.

Nella lista `l` di 6 elementi, il primo valore della lista occupa la posizione 0 (zero), l'ultimo la posizione 5.

Se digitiamo:	vediamo il valore:	se digitiamo:	vediamo il valore:
`>>> l[1]`	`-4`	`>>> l[5]`	`'abc'`
`>>> l[0]`	`1`	`>>> l[3]`	`0`

Per le liste invece di print si può anche usare il comando **list**:

```
>>> list (l)
[1, -4, 10, 0, 'z', 'abc']
```

Il costrutto **range** individua particolari liste: l'elenco di valori interi compresi tra il primo ed il secondo valore (estremo inferiore incluso, estremo superiore escluso).

```
>>> list (range(-3,9))
[-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8]
```

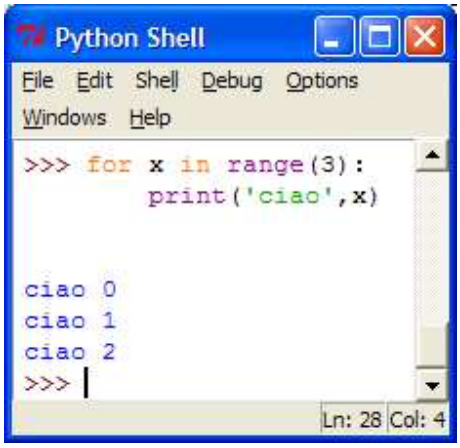
```
>>> list (range(9))
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

Il costrutto range è utilizzato per controllare la **ripetizione** di azioni (**Ripetizione con contatore**).

Per scrivere 3 volte la stringa di testo 'ciao' si può usare l'istruzione **for** come nell'esempio riportato a fianco.

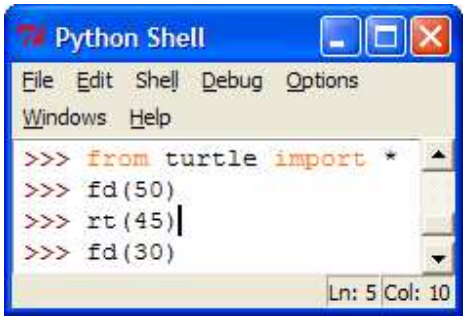
La variabile x serve solo a controllare (contare) quante volte eseguire l'istruzione print.

Nell'esempio, oltre a scrivere 3 volte la stringa 'ciao', solo per evidenziare il ruolo della variabile x ne viene stampato il valore.



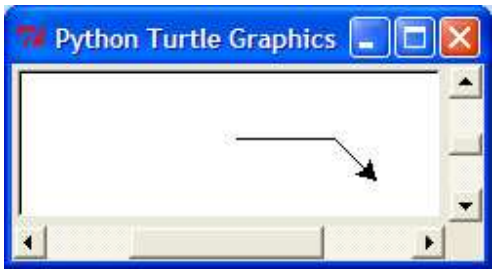
In Python è caratteristico l'uso dell'**indentazione** per la definizione dei blocchi di istruzioni; per indicarne inizio e fine invece di usare parentesi o parole chiave, in Python si usa l'indentazione stessa (si può usare sia una tabulazione, sia un numero arbitrario di spazi bianchi, ma lo standard è di 4 spazi bianchi).
Nell'esempio precedente, il primo blocco (for) inizia a colonna 4, le istruzioni che lo compongono (print) iniziano a colonna 8. Al secondo invio si chiude il blocco e l'interprete esegue il codice.
Ad ogni invio è l'interprete a decidere dove posizionare il cursore.

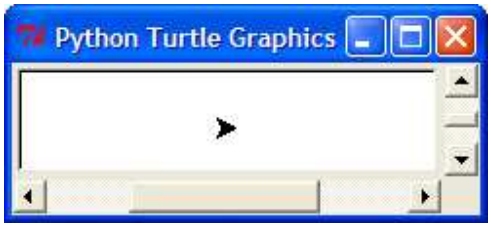
Di seguito alcuni esempi di algoritmi con uso della struttura di ripetizione utilizzando il pacchetto di funzioni grafiche TURTLE (tartaruga). Il modulo **turtle** va incluso mediante comando *import*. L'esecuzione di funzioni di turtle determinano l'apertura di una finestra di output al cui centro compare la tartaruga (rappresentata dalla punta di una freccia ➤) che lascia una scia quando si sposta (come richiesto) nel piano cartesiano.



La funzione fd (50)
fa spostare la tartaruga di 50 passi in avanti verso destra.
forward (distance)

La funzione rt (45)
fa ruotare la tartaruga di 45 gradi verso destra
right (angle)





La funzione **reset()** pulisce la finestra e pone turtle al centro

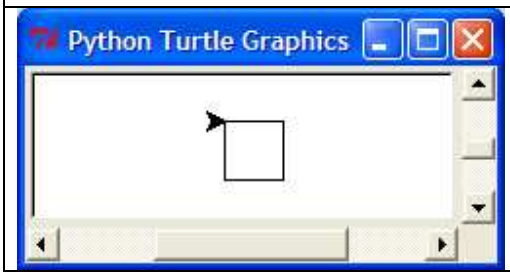
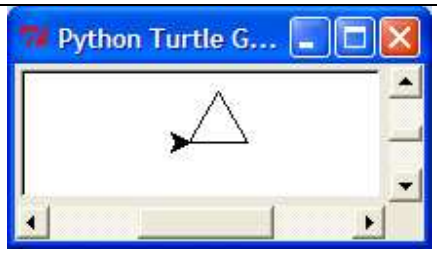
La funzione **left (angle)** fa ruotare turtle di *angle* gradi verso sinistra

La funzione **color(s)** modifica colore della tartaruga e della scia

ESEMPI di **Ripetizione con contatore**

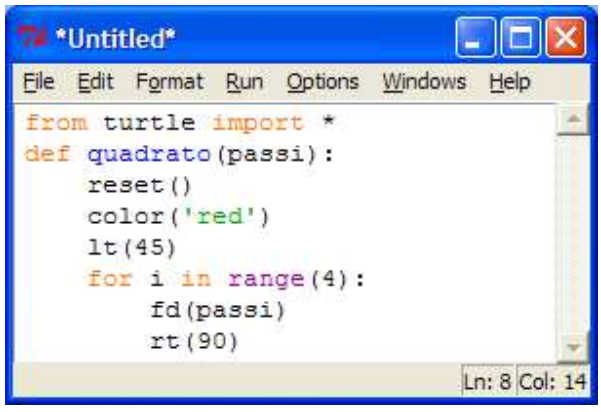
QUADRATO di 30 passi		TRIANGOLO equilatero di 30 passi

In basso l'esito finale ottenuto digitando nella finestra shell di Python:
`>>> for t in range(4):`
`  fd(30)`
`  rt(90)`
	PER contatore
	DA inizio A fine PASSO 1
	ESEGUI
	istruzioni
	RIPETI
		`>>> reset()`
		`>>> for y in range(3):`
		`  fd(30)`
		`  lt(360/3)`

	Per t che va dal valore iniziale (zero) a quello finale (4-1=3) con incremento 1 (cioè per t=0, t=1, t=2 ed infine t=3, in totale 4 volte) eseguiamo due istruzioni di turtle. Le chiediamo: - di fare 30 passi, e poi - di ruotare a destra di 90 gradi	

Porzioni di codice possono essere generalizzate e rese parametriche per essere richiamate più volte per valori diversi di un parametro; si possono cioè creare nuove **funzioni** e richiamarle successivamente.

Le funzioni si possono salvare in file con estensione **.py** detti **moduli**.

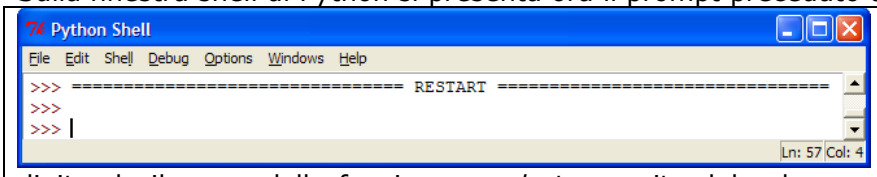


Per creare un nuovo modulo dal menu *File* selezionare *New Window* si apre una finestra *Untitled* dove è possibile scrivere codice Python.

Per **salvare** il codice su disco: dal menu *File* della nuova finestra selezionare *Save as…* selezionare la cartella dati opportuna e dare un **nome** al modulo, per esempio *quadrato.py*

Per **eseguire** il codice: dal menu *Run* della finestra/modulo appena salvata, selezionare *Run module.*

Sulla finestra shell di Python si presenta ora il prompt preceduto dalla indicazione == RESTART ==:



digitando il nome della funzione *quadrato* seguito dal valore scelto per il parametro *passi:*

```
>>> quadrato(30)
```

si ottiene il disegno:

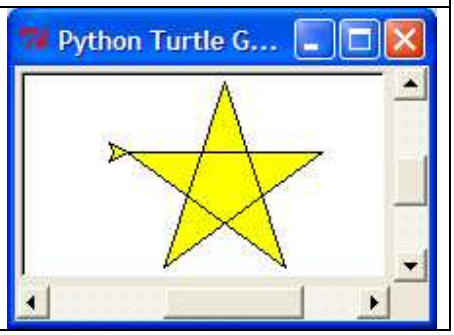
modulo *stella.py*

```
from turtle import *
def stella():
    fillcolor('yellow')
    begin_fill()
    arg=144
    for i in range(5):
        fd(100)
        rt(arg)
    end_fill()
```

eseguendo la funzione con *Run module* e digitando su IDLE il nome della funzione stella seguita solo dalle parentesi tonde (perché non prevede parametri in input) :

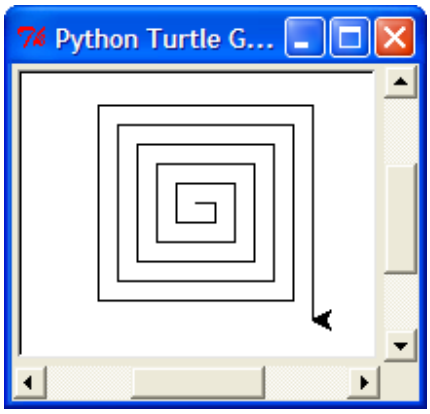
```
>>> stella()
```

si ottiene il disegno a fianco:



Le funzioni possono a loro volta richiamare altre funzioni (**funzioni nidificate**).
Per ottenere le seguenti tracce di turtle occorrono 2 funzioni.

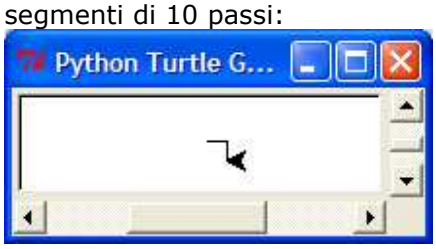
```
>>> spirale (10)
```



modulo *spirale.py*

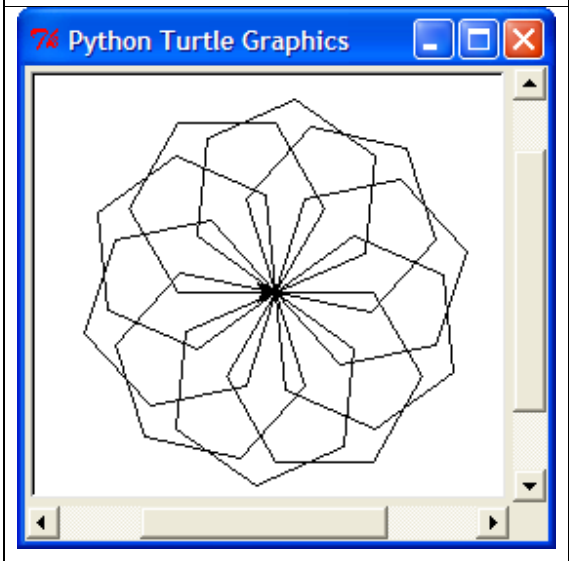
```
from turtle import *

def duelati(z):
    fd(z)
    rt(90)
    fd(z)
    rt(90)

def spirale(x):
    reset()
    for i in range (11):
        y=(1+i)*x
        duelati(y)
```

La funzione *duelati(10)* disegna due segmenti di 10 passi:



La funzione *spirale(10)* richiama 11 volte la funzione *duelati* dopo aver incrementato y ad ogni iterazione (ripetizione) di un valore pari ad x=10. Per i=0, ad y viene assegnato il valore di x; per i=1, y varrà 2*x cioè 20, per i=2, y sarà 3*x cioè 30, etc.

```
>>> pol_rotante(6)
```



modulo *polRotante.py*

```
from turtle import *

def poligono(n):
    for i in range(n):
        fd(50)
        rt(360/n)

def pol_rotante(n):
    reset()
    for i in range (10):
        poligono(n)
        rt(360/10)
```

eseguendo *poligono(3)* si ha:

eseguendo *pol_rotante(3)si ha:*