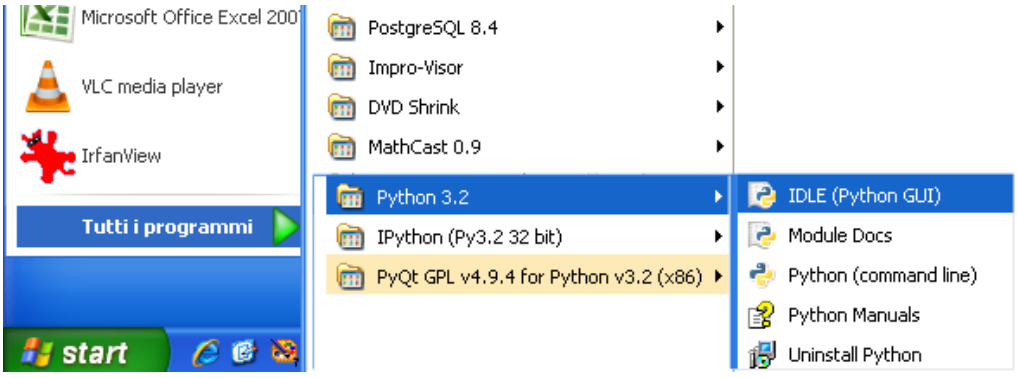
INDICE

Creiamo una infrastruttura di lavoro

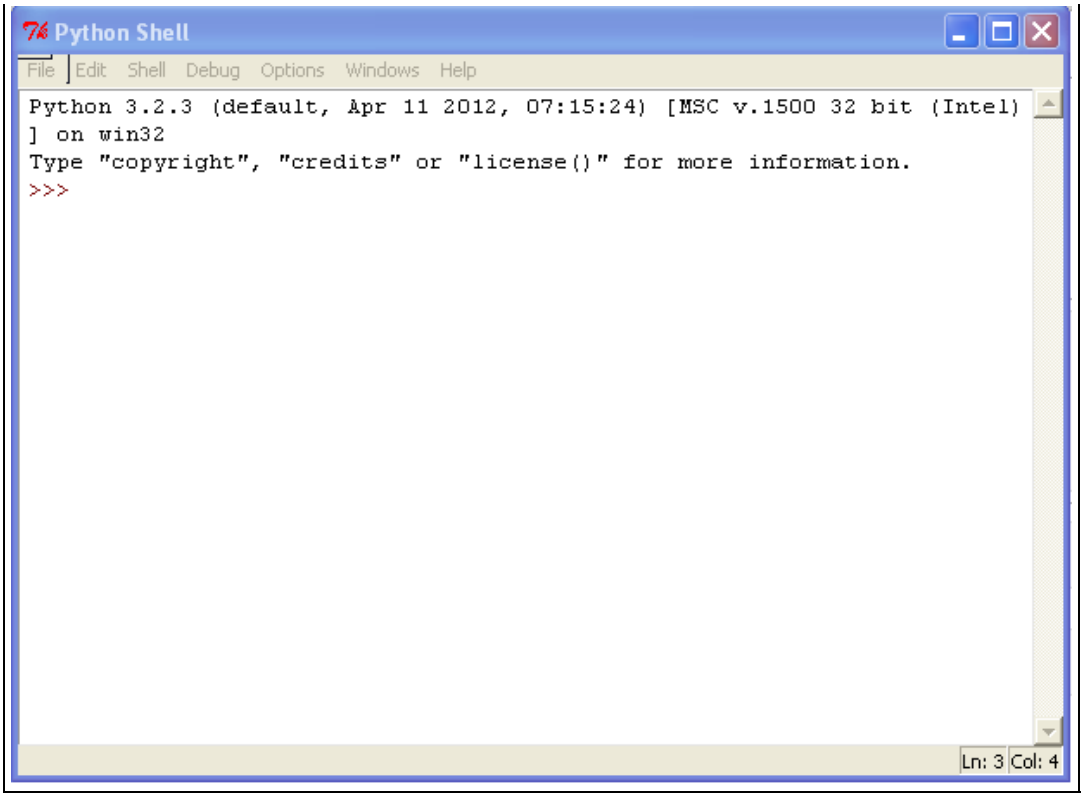
- creiamo un sistema di cartelle per il materiale che useremo durante il corso.
- create una cartella principale, io la ho chiamata 'materialexcorsopython' voi chiamatela come volete ...
- all'interno della cartella principale create le seguenti cartelle (per le sotto-cartelle rimaniamo allineati sui nomi)
  - lezioni
  - esercizi
  - work

Primi passi in modalita' interattiva

Utilizzare lo shell IDLE, installato insieme all'interprete:



Useremo python in modalita' interattiva, dove >>> e' il simbolo del prompt python



Digitare:

```
In [ ]: print("Ciao mondo")
```

print e' una funzione che in questo caso accetta come parametro una stringa e la 'stampa' sullo shell interattivo.

Le funzioni help e print

```
In [ ]: help(print)
```

NOTE:

- Abbiamo utilizzato Python in modalita' interattiva
- Abbiamo visto due esempi di funzione, cioe' delle procedure che possono essere 'chiamate', che possono accettare degli argomenti e restituire di valori e che, più in generale, fanno qualcosa e rilasciano il controllo all'interprete una volte che hanno finito.
- Abbiamo utilizzato la funzione print e ne abbiamo letto parte della documentazione utilizzando la funzione help
- Le funzioni help e print sono due delle cosiddette funzioni predefinite di Python e sono in un certo senso integrate nell'interprete (fanno parte del linguaggio)

definizione di variabili

Quindi digitare le seguenti istruzioni:

```
In [ ]: a=1
        b=2*a
        c=3.0
        d='questa è una stringa'
        print(a)
        print(b)
        print(c)
        print(d)
```

- Il codice si struttura in righe (senza necessita' di segnalare il fine riga con un carattere)
- Nel caso di utilizzo interattivo di Python, quando premo INVIO l'istruzione della riga appena scritta ha effetto e l'eventuale output 'stampato'

subito dopo.
- Abbiamo definito le variabili a, b, c, d senza averne specificato preventivamente le caratteristiche ed in particolare senza averne 'dichiarato' prima il tipo
- Le variabili sono state definite mediante l'`operatore =` detto anche `operatore di assegnamento`. In realta' con *l'espressione* `a=1` abbiamo fatto 2 cose:

  I. Abbiamo creato un numero intero (che vale 1).
  II. Gli abbiamo assegnato il nome 'a'.

  Sulla assegnazione dei nomi in Python torneremo nel seguito perche' e' un aspetto di fondamentale importanza.

Se provo ad usare una variabile z che non esiste (o che non e' visibile nel contesto in cui lavoro, ne parleremo poi), l'interprete segnala un `NameError`:

In [ ]:
```
#z*a
```

le funzione type ed un primo incontro con alcuni tipi di base

In [ ]:
```
a=3
b=2*a
print("tipo di b = ",type(b))
```

In [ ]:
```
c=3.0
print("tipo di c = ",type(c))
```

In [ ]:
```
d='questa è una stringa'
print("tipo di d = ",type(d))
```

- Python assegna un tipo a ciascun 'oggetto' nel momento della sua inizializzazione, come abbiamo potuto verificare con la funzione `type` (*si dice che Python è un linguaggio fortemente tipizzato ma che non richiede la dichiarazione esplicita dei tipi*)

```
<li>i tipi <b>int</b>, <b>float</b> e <b>string</b>, che abbiamo appena incontrato, sono detti <b>tipi
predefiniti</b> (<b>tipi di base</b> o <b>builtin types</b>) e sono <b>integrati nell'inteprete</b> (in
altre parole fanno parte della definizione stessa del linguaggio di programmazione Python)</li>
```

Operatori sui tipi di base

- Python definisce anche degli operatori che lavorano sui tipi di base (nell'esempio la somma e la moltiplicazione per gli interi e per le stringhe, sono definiti nella `Python standard library`)

In [ ]:
```
print(a*b)
```

In [ ]:
```
b='ciao '
print(b+b)
```

In [ ]:
```
print(b*4)
```

Alcuni link utili sui tipi di base e sulle funzioni standard

- Funzioni standard library
- Tipi di base standard library
- Sempre sui tipi di base ma dalla reference guide di Python

Cosa mette a disposizione l'interprete 'nudo e crudo'?

per capirlo introduciamo una nuova funzione.

La funzione dir

Facciamo adesso la conoscenza con la funzione predefinita dir:

In [ ]:
```
help(dir)
```

dir()

Come dice la documentazione, la funzione dir, se invocata senza argomenti, indica i nomi definiti nel contesto (scope) corrente: La funzione dovrebbe restituire una 'lista' che contiene i seguenti nomi:

- '__builtins__' (nell'ambiente che sto usando per la presentazione e' __builtin__)
- '__doc__'
- '__name__'
- '__package__'
- 'a'
- 'b'
- 'c'

Dove i nomi a,b e c sono quelli delle variabili che abbiamo definito noi.
Per capire meglio a cosa sono associati questi nomi possiamo utilizzare sia le funzioni help e type, gia' viste in precedenza, sia la funzione dir stessa

In [ ]:
```
#__name__
type(__name__)
```

In [ ]:
```
print(__name__)
```

__name__ e' quindi il nome di una stringa che ha come contenuto "__main__"
In questo particolare caso "__main__" e' il nome del contesto di lavoro principale dell'interprete nel quale si opera in modalita' interattiva
Piu' in generale "__main__" e' il nome assegnato al programma principale eseguito dall'interprete (questa cosa sara' + chiara in seguito).

dir(oggetto)

dir e' una funzione molto utile per capire quali sono le proprieta' (attributi) che caratterizzano una generica variabile (oggetto):

In [ ]:
```
a=1 #creo un intero
print(type(a))
print(dir(a))
```

il modulo builtins

Passiamo a __builtins__:

In [ ]:
```
#builtin
print( type(__builtin__))
```

Si tratta di un modulo cioe' di un pacchetto software di python che integra nuove funzioni o anche nuovi tipi di oggetti.
Nel caso di __builtins__ , si tratta dell'unico modulo che viene sempre caricato dall'interprete quando questo e' usato e definisce i tipi e le funzioni predefinite

Se uso la funzione dir passandogli come parametro il nome __builtins__ ottengo una lista in cui ci sono tutti i nomi (di funzioni e tipi di dato ad esempio) definiti nel modulo

In [ ]:
```
print(dir(__builtin__))
```

Verifichiamo che le funzioni ed i tipi di dato che abbiamo usato fino ad ora siano effettivamente definiti in __builtins__:

In [ ]:
```
print('int' in dir(__builtin__))
print('float' in dir(__builtin__))
print('str' in dir(__builtin__))
print('help' in dir(__builtin__))
print('dir' in dir(__builtin__))
print('type' in dir(__builtin__))
print('-'*20)
print('xxx' in dir(__builtin__))
```

Importazione di moduli

Oltre alle funzioni ed ai tipi predefiniti ci sono migliaia di `moduli` aggiuntivi che estendono le funzionalita' del linguaggio.
I moduli possono essere suddivisi sostanzialmente in due categorie:

- `moduli standard`: sono quelli che vengono distribuiti ed installati insieme all'interprete e fanno praticamente parte integrante del linguaggio
- `moduli non standard`: sono moduli che ci dobbiamo procurare ed installare separatamente dall'interprete (numpy e matplotlib ad esempio).

Una terza categoria e' costituita dai moduli che eventualmente creeremo noi.

Un esempio di modulo standard : math

Il modulo math, come tutti gli altri moduli standard, e' distribuito insieme all'interprete

In [ ]:
```
import math
help(math)
```

Oltre alla funzione help e' possibile utilizzare la funzione dir passandogli come argomento il nome del modulo ed otterremo l'elenco di tutto cio' che e' definito nel modulo stesso:

In [ ]:
```
print(dir(math))
```

Per utilizzare una delle funzioni definite nel modulo math si usa la sintassi ***math.funzione***

In [ ]:
```
math.cos(0.0)
```

In [ ]:
```
math.cos(math.pi/2.0) #in questo caso abbiamo usato anche la costante math.pi
```

Verifichiamo che il nome math e' stato aggiunto al contesto di lavoro corrente.

In [ ]:
```
'math' in dir()
```

Primi passi con gli script

Fino ad ora abbiamo usato l'interprete in modalita' interattiva. Vediamo adesso cosa e' uno script.

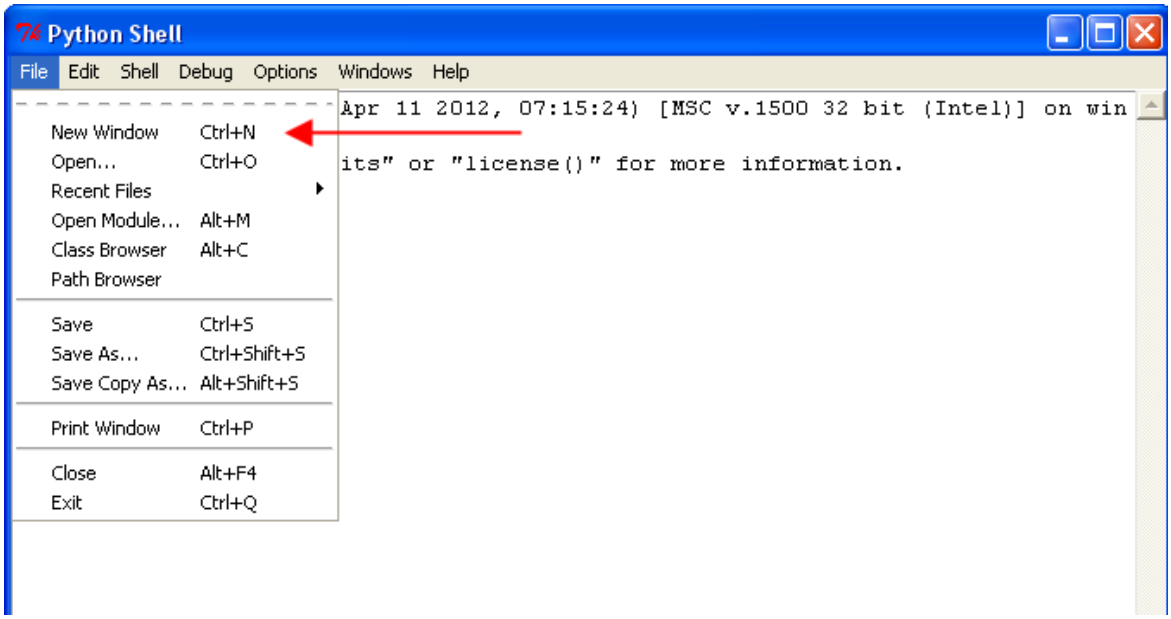
Uno script e' un file di testo che contiene istruzioni di python e che puo' essere eseguito dall'interprete.

Ad uno script e più in generale ad un programma scritto in python si associa l'estensione `.py`

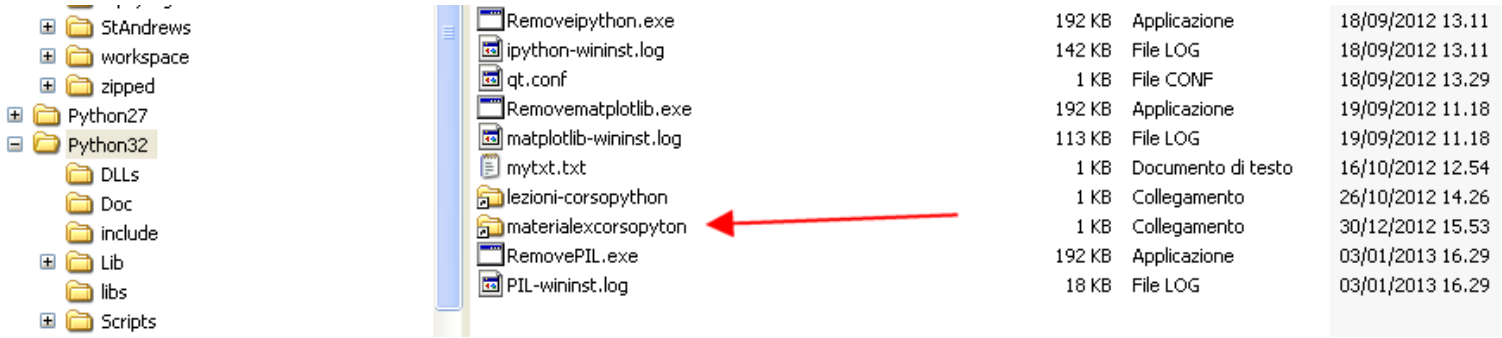
creo uno script

Usiamo IDLE (ma va bene qualsiasi editor di testo):

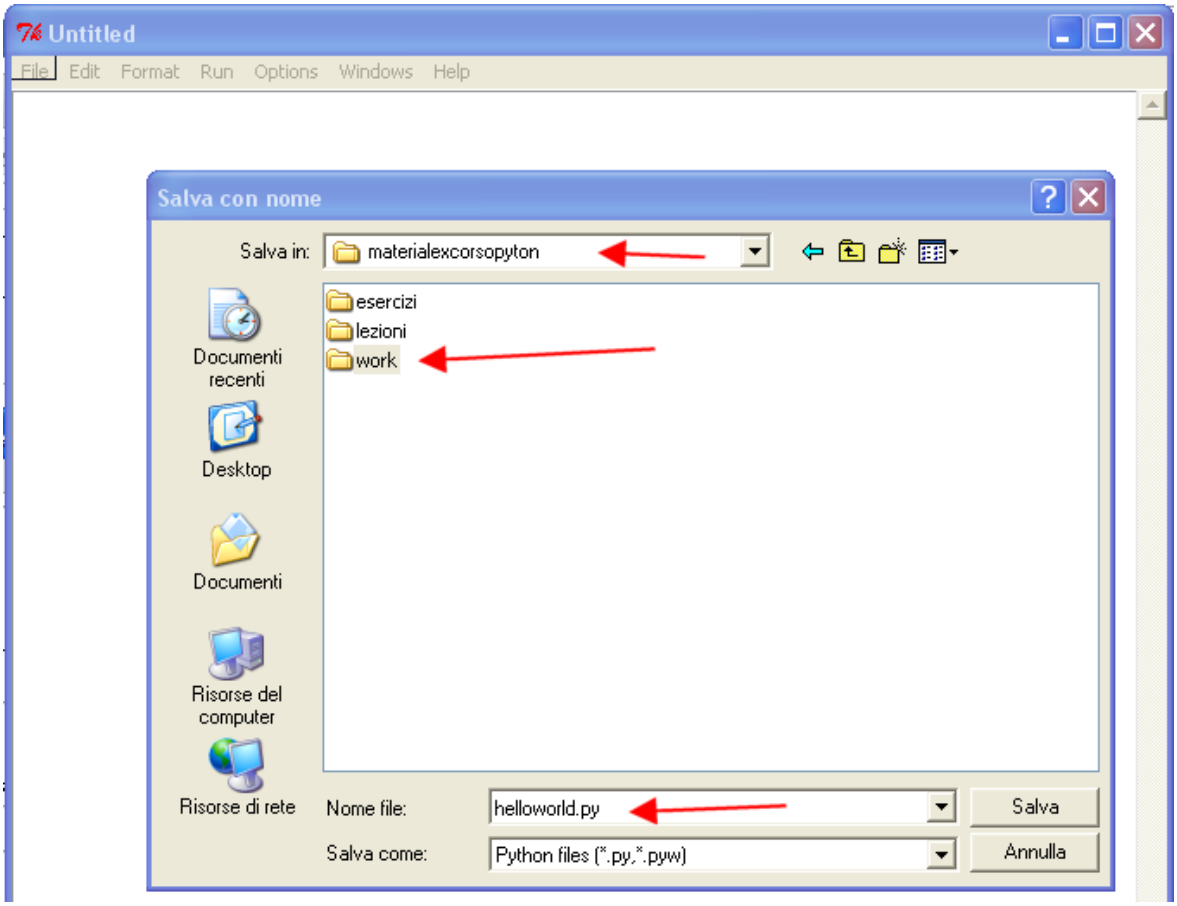
- Menù file -> New Window
- Nella nuova finestra : Menù file -> Save



Se dal menù file proviamo a salvare (il file vuoto), su windows IDLE propone di default la cartella di installazione di Python (C:/Python32/ nel mio caso).
E' comodo creare nella cartella di installazione un collegamento alla nostra cartella principale in modo da velocizzare le operazioni di apertura/salvataggio dei files.



Creare un file helloworld.py nella cartella work.

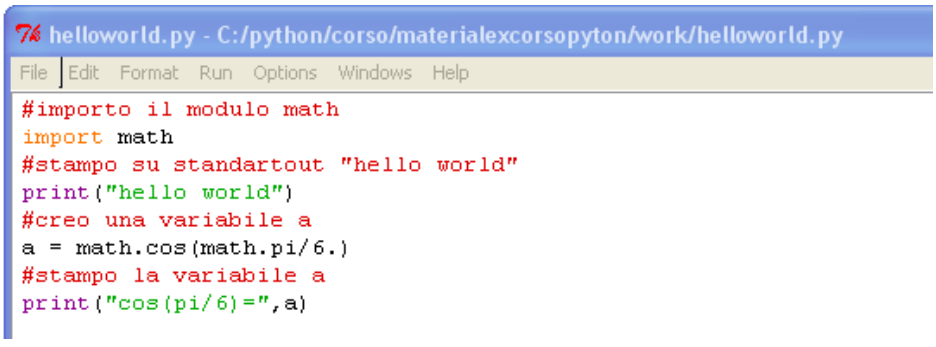


scrivo nel file alcune delle semplici istruzioni che abbiamo gia' visto intervallandole a commenti

i commenti

è il carattere di commento ed e' valido anche in modalita' interattiva oltre che quando scriviamo script o programmi su file di testo.

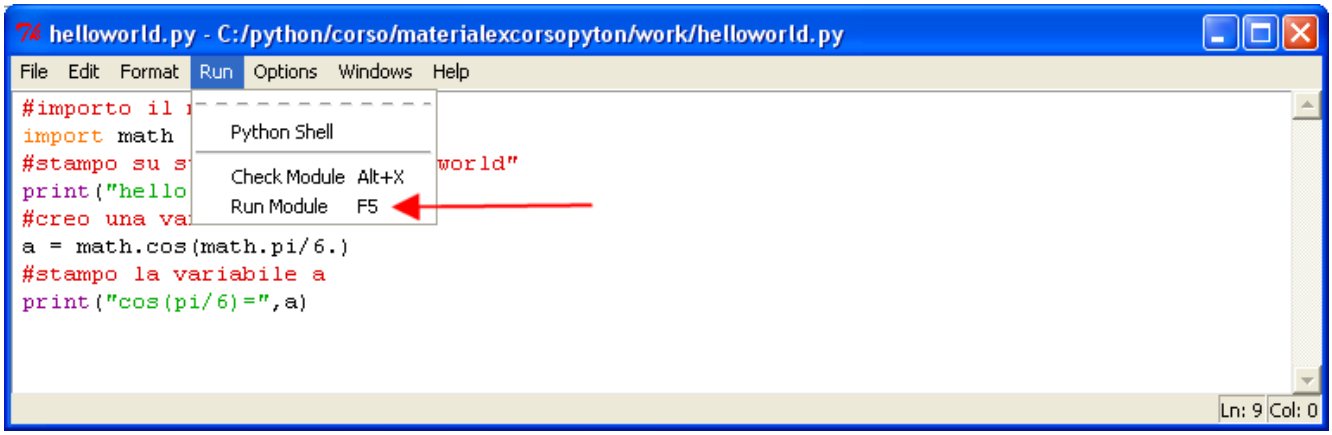
In [ ]:
```
#commento a riga intera
import math #commento sullo
#stampo su standartout "hello world"
print("hello world")
#creo una variabile a
a = math.cos(math.pi/6.)
#stampo la variabile a
print("cos(pi/6)=",a)
```



Eseguire uno script

da IDLE

Se si utilizza IDLE basta utilizzare Menù run --> run module (o tasto F5)

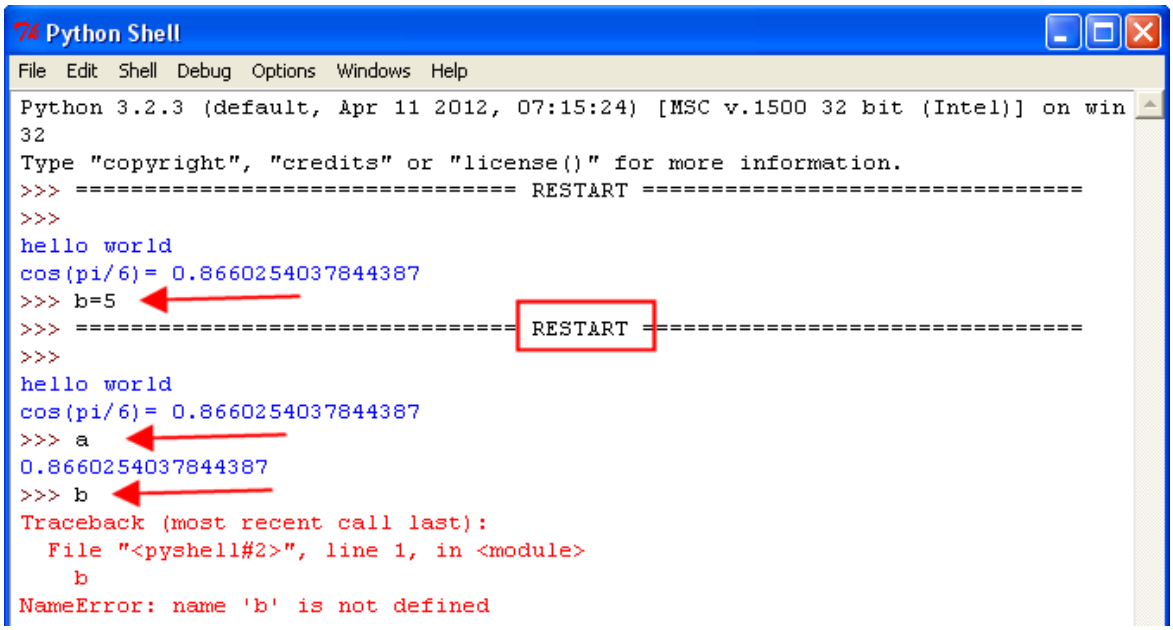


Come risultato nella finestra interattiva dell'interprete viene stampato l'output del codice.

```
Python 3.2.3 (default, Apr 11 2012, 07:15:24) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
hello world
cos(pi/6)= 0.8660254037844387
>>>
```

NOTA BENE:

- Tutte le volte che eseguo uno scripto con il comando run di idle l'interprete viene RIAVVIATO e cio' implica che tutte le variabili eventualmente istanziate prima dell'esecuzione vadano perse (vedi variabile b nell'esempio sotto)
- Sempre con riferimento allo script visto sopra, dopo l'esecuzione dello script, la variabile a e' disponibile ed accessibile nell'ambiente interattivo (...)



da un generico shell interattivo di python

```
In [ ]: exec(open('C:/python/corso/materialexcorsopyton/work/helloworld.py').read())
```

per ora non si capisce
Nelle prossime lezioni vedremo tutte le funzioni usate (exec, open ed il metodo read dell'oggetto file)
vedi anche qui per una conversazione su un forum a proposito.

NOTA: il path completo dello script non e' necessario se lo script e' nella la *cartella di lavoro* di python (in genere quella dalla quale si e' invocato

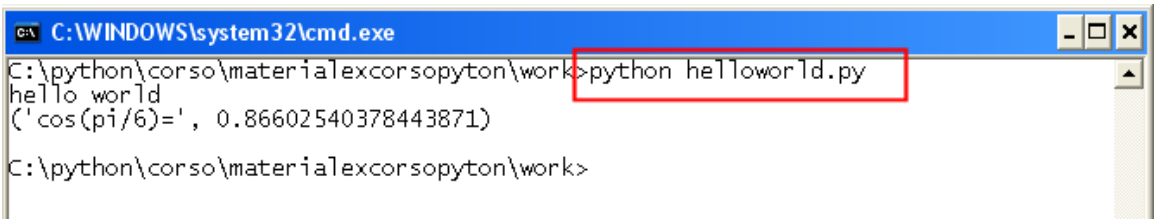
l'interprete).

NOTA: così facendo una variabile b istanziata prima dell'esecuzione dello script e' accessibile anche dopo l'esecuzione dello script stesso:

```
>>> ================================ RESTART ================================
>>> b=444
>>> exec(open('C:/python/corso/materialexcorsopyton/work/helloworld.py').read())
hello world
cos(pi/6)= 0.8660254037844387
>>> a
0.8660254037844387
>>> b
444
>>>
```

da prompt dei comandi

Da un generico prompt dei comandi del sistema operativo:



- python nomescript se la cartella di lavoro e' quella dove e' lo script e uso la versione di python standard per il sistema operativo
- python pathscript se la cartella di lavoro NON e' quella dove e' lo script e uso la versione di python standard per il sistema operativo
- pathpython pathscript se la cartella di lavoro NON e' quella dove e' lo script e NON uso la versione di python standard per il sistema operativo

```
C:\python\corso\materialexcorsopyton>c:/python32/python ./work/helloworld.py
hello world
cos(pi/6)= 0.8660254037844387

C:\python\corso\materialexcorsopyton>
```

Nota per utenti wondows: come aprire uno shell MS-DOS in una specifica cartella

Per aprire uno shell MS-DOS in una specifica cartella (senza necessita' di cambiare cartella con il comando chdir) è utile la funzione 'Open command prompt here' di windows:

- per Windows XP
- per Windows Vista e 7

Alternativamente, fissata la cartella di interesse (ad esempio work) creare un file con estensione bat che contiene una sola riga:

```
cmd.exe "%1"
```

doppio clic sull'icona

per evitare che il prompt si chiuda immediatamente aggiungere la chiamata a funzione
`input("ciao ciao")`
alla fine dello script

script di python eseguibili in unix

vedi qui