Laurea in Ingegneria Gestionale

Corso di Fondamenti di Informatica
A.A. 2017/2018

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Elementi principali del linguaggio Python

Slide tratte dal lavoro di Giorgio Fumera
(Univ. degli Studi di Cagliari) e riadattate da Raffele Nicolussi
(Sapienza) e Domenico Lembo (Sapienza)

Istruzioni, espressioni, variabili

Come per la maggior parte dei linguaggi di alto livello, un programma Python è composto da una sequenza di **istruzioni.**

Tre sono le istruzioni fondamentali:

- istruzione di assegnamento
- istruzione condizionale
- istruzione iterativa

Ogni istruzione contiene **espressioni** che producono valori da elaborare (per esempio, espressioni aritmetiche).

Sono inoltre disponibili espressioni e istruzioni per l'**acquisizione** dei dati da elaborare e per la **stampa/memorizzazione** dei risultati attraverso le periferiche (tastiera, memoria secondaria, schermo, ecc.).

Nei programmi è inoltre possibile (e, di norma, necessario) memorizzare i dati da elaborare e i risultati nelle celle di memoria del calcolatore, alle quali si fa riferimento attraverso nomi simbolici detti **variabili.**

Istruzioni, espressioni, variabili

Di seguito si descrivono la **sintassi** e la **semantica** delle principali istruzioni ed espressioni del linguaggio Python

- sintassi: insieme di regole che indicano come le istruzioni devono essere **scritte** dal programmatore, ovvero quali sono le istruzioni corrette o valide del linguaggio
- semantica: insieme di regole che definiscono il **significato** delle istruzioni valide, ovvero come tali istruzioni devono essere **eseguite** dal calcolatore

La sintassi verrà descritta usando:

- il colore nero, `e questo tipo di carattere`, per le parti da scrivere in modo testuale
- il colore rosso per indicare le parti da sostituire come spiegato caso per caso

Un suggerimento per lo studio di Python

La comprensione della sintassi e della semantica di un linguaggio di programmazione è un prerequisito **indispensabile** per essere in grado di codificare algoritmi (e quindi scrivere programmi) in tale linguaggio.

A questo scopo si consiglia di sfruttare l'interattività del linguaggio Python (una delle caratteristiche principali dei linguaggi interpretati), cioè la possibilità di scrivere ed eseguire **singole** istruzioni nella *shell*.

Variabili e istruzione di assegnamento

Nei programmi in linguaggio macchina i dati d'ingresso e i risultati delle elaborazioni devono essere memorizzati in **celle di memoria** scelte dal programmatore, alle quali ci si riferisce tramite il loro **indirizzo**.

Anche i linguaggi di alto livello prevedono la memorizzazione di dati e risultati nelle celle di memoria. Questo avviene attraverso l'istruzione di assegnazione, una delle istruzioni fondamentali e comuni alla maggior parte di tali linguaggi.

Contrariamente al linguaggio macchina, le celle di memoria vengono indicate nei linguaggi di alto livello attraverso **nomi simbolici** scelti dal programmatore.

Tali nomi simbolici sono detti variabili, poiché i valori memorizzati nelle celle corrispondenti possono essere modificati da successive istruzioni di assegnamento.

Variabili e istruzione di assegnamento: sintassi

Sintassi: variabile = espressione

- non devono esserci **spazi** prima del nome della variabile
- variabile deve essere un nome simbolico scelto dal programmatore (con le limitazioni descritte più avanti)
- espressione indica il valore (per es., un numero) che si vuole associare alla variabile

Esempi (il significato diventerà chiaro più avanti):

```
x = -3.2
messaggio = "Buongiorno"
y = x + 1
```

Variabili e istruzione di assegnamento: sintassi

Limitazioni sui nomi delle variabili:

- possono essere composti da uno o più dei seguenti caratteri:
  - lettere minuscole e maiuscole
  - cifre
  - il carattere _ (*underscore*)

  esempi: `x`, `somma`, `Massimo_Comun_Divisore`, `y_1`
- **non** devono iniziare con una cifra; esempio:
  `12abc` **non** è un nome ammesso
- **non** devono coincidere con i nomi predefiniti delle istruzioni e di altri elementi del linguaggio, come per esempio:
  `input`, `print`, `if`, `while`

Variabili e istruzione di assegnamento: semantica

L'istruzione di assegnazione viene **eseguita** in due fasi:

1. l'interprete valuta espressione, cioè ne determina il valore
2. il valore di espressione viene assegnato (o associato) a variabile

Dopo l'assegnamento, il nome della variabile potrà essere usato in espressioni all'interno di altre istruzioni: nella valutazione di tali espressioni l'interprete sostituirà al nome della variabile il valore a essa associato (si veda più avanti).

L'effetto concreto di un'istruzione di assegnamento é la memorizzazione del valore di espressione all'interno di una o più celle di memoria.

Se un'istruzione di assegnamento si riferisce a una varilabile alla quale in precedenza era già stato assegnato un valore, il nuovo valore **sostituisce** quello precedente.

Tipi di dati ed espressioni

I principali tipi di valori, o **tipi di dati**, che possono essere rappresentati ed elaborati dai programmi Python, sono:

- numeri interi
- numeri frazionari
- sequenze di caratteri

Le espressioni Python che producono valori appartenenti a tali tipi di dati, e che possono contenere opportuni **operatori**, sono le seguenti:

- espressioni aritmetiche
- stringhe (sequenze di caratteri)

Espressioni aritmetiche

La più semplice espressione aritmetica è un singolo numero.
I numeri vengono rappresentati nelle istruzioni Python con diverse notazioni, simili a quelle matematiche, **espressi in base dieci**.

Python distingue tra **due tipi** di dati numerici:

- numeri interi *(int)*, codificati nei calcolatori in complemento a due; esempi: `12`, `-9`
- numeri frazionari (*float*), codificati in virgola mobile (floating point), e rappresentati nei programmi:
  - come parte intera e frazionaria, separate da un **punto** (notazione anglosassone); esempi: `3.14`, `-45.2`, `1.0`
  - in notazione esponenziale, $m \times b^e$, con base b pari a dieci, ed esponente introdotto dal carattere `E` oppure `e`;
    esempi: `1.99E33` ($1,99 \times 10^{33}$), `-42.3e-4` ($-42,3 \times 10^{-4}$), `2E3` (2×10^3)

NOTA: i numeri espressi in notazione esponenziale sono sempre considerati numeri **frazionari**

Espressioni aritmetiche

Espressione aritmetiche più complesse si ottengono combinando numeri attraverso **operatori** (addizione, divisione, ecc.), e usando le parentesi **tonde** per definire la precedenza tra gli operatori.

Gli operatori disponibili nel linguaggio Python sono i seguenti:

simbolo	**operatore**
+	somma
–	sottrazione
*	moltiplicazione
/	divisione
//	divisione intera
%	modulo (resto di una divisione)
**	elevamento a potenza

Espressioni aritmetiche: esempi

Alcuni esempi di istruzioni di assegnazione contenenti espressioni aritmetiche.

Notare che ciò che viene assegnato a una variabile è il **valore** dell'espressione corrispondente, indicato in neretto sulla destra.

```
x = -5                                  -5
y = 1 + 1                               2
z = (1 + 2) * 3                         9
circonferenza = 2 * 3.14 * 3            18.84
q1 = 6 / 2                              3.0
q2 = 7.5 / 3                            2.5
q3 = 5 // 2                             2
resto = 10 % 2                          0
```

Espressioni aritmetiche: osservazioni

- Se **entrambi** gli operandi di `+`, `-` e `*` **sono interi**, il risultato è rappresentato come **intero** (senza parte frazionaria); altrimenti è rappresentato come numero **frazionario**.
  Esempi:
  `1 + 1` → `2`, `2 - 3.1` → `-1.1`, `3.2 * 5` → `16.0`
- Indipendentemente dal fatto che gli operandi di `/` siano interi o frazionari, il quoziente restituito è sempre frazionario

  Esempi:

  `6 / 2` → `3.0`, `6.0 / 2` → `3.0`, `2 / 5` → `0.4`,
  `2 / 5.0` → `0.4`, `-2 / 3` → `-0.6666666666666666`
- L'operatore `//` produce **sempre** il più grande **intero** non maggiore del quoziente, rappresentato come intero se entrambi gli operandi sono interi, altrimenti come numero **frazionario**.

  Esempi: `6 // 2` → `3`, `6.0 // 2` → `3.0`, `2 // 5` → `0`,
  `2 // 5.0` → `0.0`, `-2 // 3` → `-1`

Operatori di assegnamento composti (+=, -=, …)

E’ anche possibile scrivere enunciati che effettuano una assegnazione e svolgono allo stesso tempo un calcolo.

Esempi:

```
i -= 1 # equivalente a i = i - 1

i *= 2 # equivalente a i = i * 2

i /= 3 # equivalente a i = i / 3

i **= 4 # equivalente a i = i ** 4

i %= 2 # equivalente a i = i % 2
```

Espressioni: Funzioni matematiche predefinite

Una funzione è una sequenza di istruzioni a cui è associato un nome e degli argomenti di input e che restituisce un valore (o più valori).

E' possibile invocare l'esecuzione della sequenza di comandi definiti dalla funzione, usando il suo nome della e definendo i valori in input.
L'invocazione di una funzione può comparire ovunque è ammissibile un valore del tipo restituito dalla funzione.

Python ha molte funzioni predefinite e (come vedremo più avanti) permette la definizione di nuove funzioni in un qualsiasi programma.

Funzioni matematiche predefinite

Funzione	**Restituisce**
`abs(x)`	valore assoluto di `x`
`round(x)`	valore di `x` arrotondato all'intero più vicino ad `x`
`round(x,n)`	valore di `x` arrotondato ad `n` cifre decimali
`min(`x_1`,..,`x_n`)`	valore minore fra quelli presenti come argomenti
`max(`x_1`,..,`x_n`)`	valore maggiore fra quelli presenti come argomenti

Espressioni: stringhe (1\3)

I programmi Python possono elaborare testi rappresentati come **sequenze di caratteri** (lettere, numeri, segni di punteggiatura) racchiuse tra apici singoli o doppi, dette stringhe.

Esempi:

```
"Esempio  di  stringa"
'Esempio di stringa'  "A"
'qwerty, 123456'
""  (una stringa vuota)
```

È quindi possibile assegnare una stringa a una variabile.

Esempi:

```
testo = "Esempio di stringa"
carattere = "a"
messaggio = "Premere un tasto per continuare"
t = ""
```

Espressioni: stringhe (2\3)

Il linguaggio Python prevede alcuni operatori anche per il tipo di dato *stringa*. Uno di questi è l'operatore di **concatenazione**, che si rappresenta con il simbolo **+** (lo stesso dell'addizione tra numeri) e produce come risultato una **nuova** stringa ottenuta concatenando due altre stringhe.

Esempi:

- `parola = "mappa" + "mondo"`
  assegna alla variabile `parola` la stringa `"mappamondo"`

- `testo = "Due" + " " + "parole"`
  assegna alla variabile `testo` la stringa `"Due parole"`
  (notare che la stringa `" "` contiene solo un carattere di spaziatura)

Espressioni: stringhe (3\3)

Altre espressioni ammissibili sulle stringhe

- Moltiplicazione di stringa con un intero
  - ✓ `"bla" * 5` restituisce `"blablablablabla"` cioè concatena 5 volte la stringa con se stessa. Ovviamente vale per qualunque numero n "moltiplicato" per una qualunque stringa

  - ✓ Funziona anche invertendo gli operandi. Ad esempio,
    `5 * "bla" = "bla" * 5`

Operatori di assegnamento composti su stringhe

Esempi:

```
x = 'mappa'

x += 'mondo' # equivalente a x = x + 'mondo'
```

`x` è pari alla stringa `'mappamondo'`

```
x = 'bla'

x *= 3 # equivalente a x = x * 3
```

`x` è pari alla stringa `'blablabla'`

Espressioni: osservazioni sulle stringhe

- Per inserire in una stringa un apice singolo o doppio è necessario racchiuderla tra apici dell'altro tipo, oppure far precedere l'apice dal carattere `\` (detto *backslash*), senza spazi tra i due.
  Esempi:

```
testo1 = "L'apostrofo" testo2 =
'L\'apostrofo'
testo3 = 'Doppi "apici" in una stringa'
testo4 = "Doppi \"apici\" in una stringa"
```

- Per rappresentare un'interruzione di riga ("a capo", *newline*) all'interno di una stringa si usa la sequenza di caratteri `\n` (si veda più avanti l'istruzione `print`).
  Esempio: `testo5 = "Prima riga.\nSeconda riga."`

- Dato che il carattere `\` ha un significato particolare in una stringa, per farlo comparire testualmente si deve scrivere `\\` (si veda più avanti l'istruzione `print`).
  Esempio: `testo6 = "abc\\def"`

Alcune Sequenze di escape

Come abbiamo detto, \ è un carattere di escape e, insieme al carattere che lo segue, assume un significato particolare (sequenza di escape).

Esempi di sequenze di escape

\\	Backslash (\)
\'	Carattere di quotatura singola (')
\"	Carattere di quotatura doppia (")
\n	ASCII Linefeed (LF) – sposta il cursore all'inizio della linea successiva
\t	ASCII Tab orizzontale (TAB)
\f	ASCII Formfeed (FF) – interruzione di pagina

Espressioni contenenti nomi di variabili

Se ad una variabile è già stato assegnato un valore, il suo nome potrà essere usato come **operando** all'interno di un'espressione.
Quando l'interprete valuterà l'espressione, sostituirà al nome della variabile il valore a essa associato.
Esempi:

- eseguendo in sequenza le due istruzioni

```
x = 5
y = x
```

alla variabile `y` sarà assegnato il valore `5`

- eseguendo le istruzioni

```
raggio = 2
circonferenza = raggio * 6.28
```

alla variabile `circonferenza` sarà assegnato il valore `12.56`

- eseguendo le istruzioni

```
parola1 = "mappa"
parola2 = parola1 + "mondo"
```

alla variabile `parola2` sarà assegnato il valore `"mappamondo"`

Espressioni contenenti nomi di variabili

Un'espressione contenente il nome di una variabile alla quale **non** sia stato ancora assegnato nessun valore è sintatticamente errata.

Per esempio, assumendo che alla variabile `h` non sia ancora stato assegnato nessun valore, scrivendo nella *shell* l'istruzione

```
x = h + 1
```

si otterrà il seguente messaggio d'errore:

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

Conversione tra numeri e stringhe (1\2)

L'enunciato `str(1729)` converte l'intero `1729` in una stringa

Esempio:

```
Name = "Agent" + str(1729)
```

Assegna alla variabile `Name` il valore `Agent1729`

Viceversa, per trasformare in valore numerico una stringa contenente un numero si usano `int` e `float` come nell'esempio seguente

```
id = int("1729")
price = float("17.29")
```

Se la stringa *data in input* a `int` o `float` non è convertibile in un numero, viene restituito un errore. Ad esempio, `float("12x3")` restituisce l'errore `could not convert string to float`

`str`, `int` e `float` sono funzioni unarie.

Conversione tra numeri e stringhe (2\2)

`int` può anche convertire valori frazionari in interi, ma nel farlo **tronca** (cioè toglie) la parte decimale.

```
>>> int(3.99999)
3
>>> int(-2.3)
-2
```

Nota: `int("3.99999")` restituisce invece l'errore

```
invalid literal for int() with base 10
```

Similmente, la funzione float può anche convertire numeri interi in numeri in numeri frazionari:

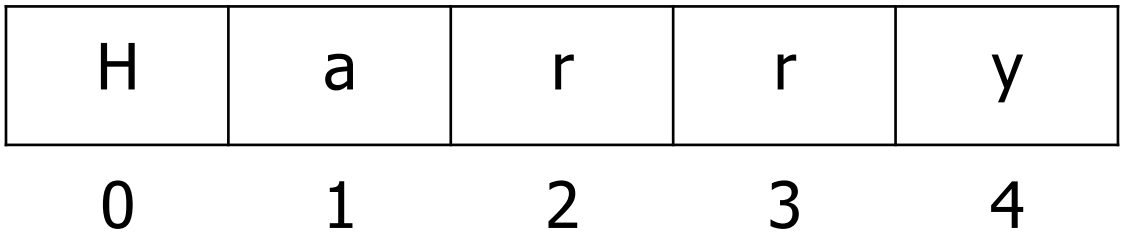
```
>>> float(32)
32.0
```

Può sembrare strano il fatto che Python distingua il valore intero 1 dal corrispondente valore frazionario 1.0. Questi rappresentano effettivamente uno stesso numero ma appartengono **a tipi differenti** (rispettivamente intero e frazionario) **e quindi vengono rappresentati in modo diverso** all'interno della memoria del computer.

Stringhe e caratteri

Le stringhe sono sequenze di caratteri (in questo senso sono analoghe alle liste, che studieremo più avanti, che sono sequenze di valori e/o oggetti).

La posizione di un carattere in una stringa è denotata da un indice



Per accedere ad un carattere si usa l'indice. Ad esempio,

```
name = "Harry"
first = name[0]
last = name[4]
```

a `first` viene assegnato il valore `H` ed a `last` il valore `y`.

`Name[i]` con i diverso dal range `0..4` causa un

```
IndexError: index out of range
```

Funzioni e metodi su stringhe

- La *funzione* `len` restituisce un intero pari alla lunghezza della stringa in input
  Esempio:
  `len("Harry")` è pari a `5`

- Il *metodo* `upper()` (risp. `lower()`) invocato su una stringa `s` restituisce una stringa ottenuta da `s` trasformando le lettere minuscole in maiuscole (risp. maiuscole in minuscole)

```
n = "John Smith"
m = n.upper()
```

  dopo l'esecuzione delle istruzioni `m` è pari alla stringa `"JOHN SMITH"`

Nota1: in Python non c'è un tipo di dati per i caratteri, che sono semplicemente stringhe di dimensione 1

Nota2: la *distinzione fra metodo e funzione* sarà più chiara in seguito. Per il momento si noti la sintassi diversa usata nei due casi

Acquisizione di valori attraverso la tastiera

In molti casi è utile scrivere programmi che acquisiscano **durante l'esecuzione** i dati che dovranno elaborare, attraverso periferiche d'ingresso come la tastiera o la memoria secondaria.

Nei programmi Python l'acquisizione di dati attraverso la tastiera si ottiene tramite l'espressione `input`, che richiede all'utente di immettere un valore.

Se un programma Python viene eseguito in un ambiente come `IDLE`, l'immissione dei dati dovrà avvenire nella finestra della *shell*.

L'espressione `input`

Questa espressione può essere scritta in due forme. Il suo uso all'interno di un'istruzione di assegnamento è il seguente.

Sintassi e semantica:

- variabile = `input()`
  l'interprete resta in attesa che l'utente inserisca nella *shell*, attraverso la tastiera, una sequenza di caratteri che dovrà essere conclusa dal carattere `RETURN`; successivamente si assegna a variabile la sequenza di caratteri inserita, interpretandola come stringa.

- variabile = `input(`messaggio`)`
  l'interprete stampa messaggio, che deve essere una **stringa**, nella *shell*, poi procede come indicato sopra.

L'espressione `input`: esempi

- Dopo aver scritto nella *shell* l'istruzione:

  ```
  n = input("Inserire un numero: ")
  ```

  l'interprete mostrerà (sempre nella *shell*) il messaggio

  ```
  Inserire un numero:
  ```

  seguito da un cursore lampeggiante; si inserisca a questo punto il valore `25` e si prema il tasto `RETURN`: il risultato sarà l'assegnazione del **valore di tipo stringa** 25 alla variabile `n`

Valutazione di espressioni nella *shell*

Nella *shell* è anche possibile scrivere una qualsiasi espressione invece che un'istruzione: in questo caso l'interprete valuterà l'espressione e ne mostrerà il **valore**, sempre nella *shell*.

Questo consente, per esempio, di usare la *shell* di Python come una calcolatrice, e di visualizzare il valore associato a una variabile.

Come nel caso delle istruzioni, anche le espressioni **non** devono essere precedute da **spazi**.

Valutazione di espressioni nella *shell* : esempi

Scrivendo nella shell l'espressione: `1`
l'interprete mostrerà il valore: `1`

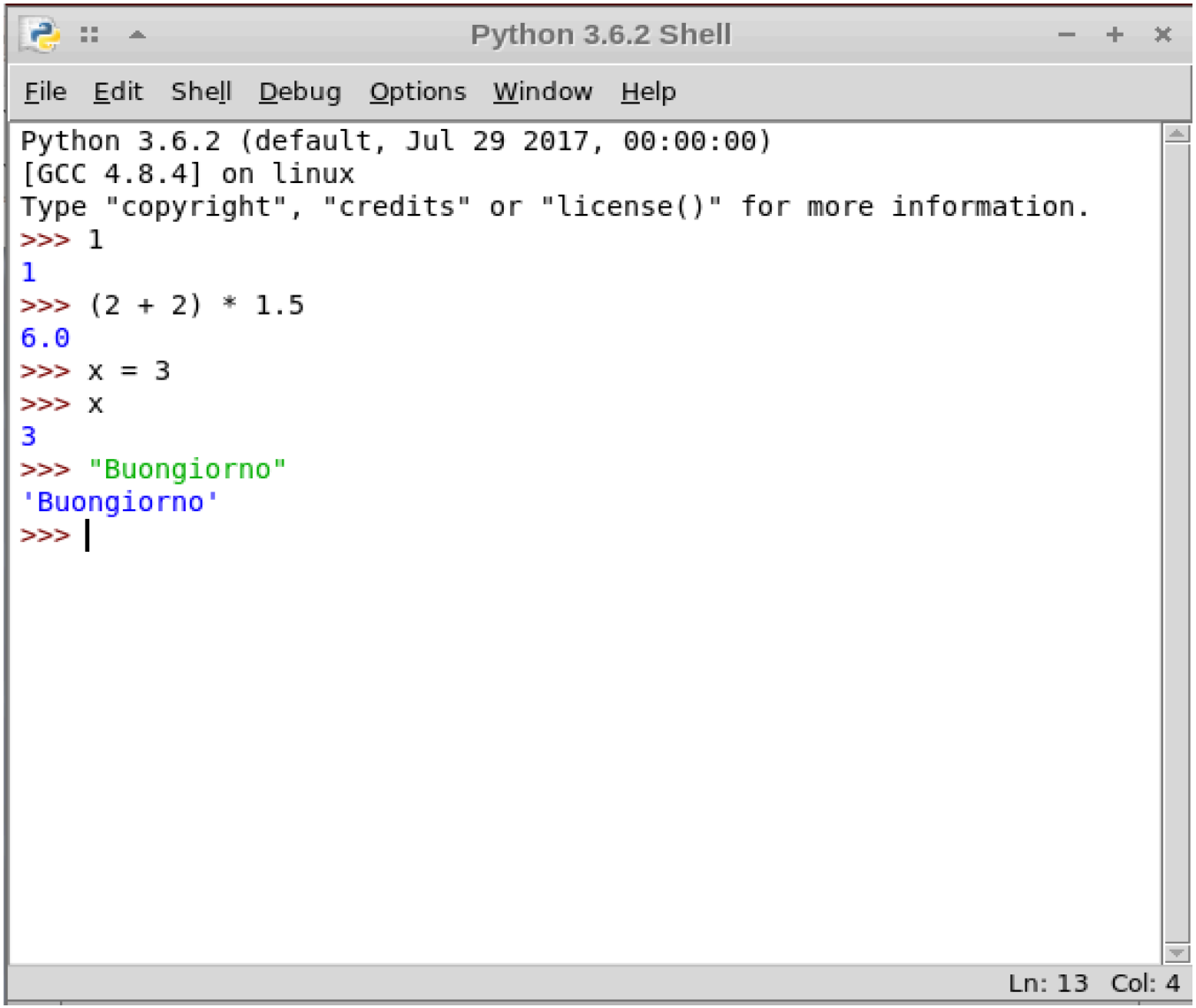
Scrivendo nella shell l'espressione: `(2 + 2) * 1.5`
l'interprete mostrerà il valore: `6.0`

Scrivendo nella shell l'assegnamento: `x = 3`
e successivamente l'espressione: `x`
l'interprete mostrerà il valore. `3`

Scrivendo nella shell l'espressione: `"Buongiorno"`
l'interprete mostrerà il valore `'Buongiorno'` (si noti che i valori di tipo stringa vengono mostrati includendo gli apici singoli)

Valutazione di espressioni nella *shell* : esempi

Il contenuto della *shell* dopo gli esempi mostrati in precedenza:



Istruzione di assegnamento: esempio conclusivo

Scrivere nella *shell* la seguente sequenza d'istruzioni (premendo il tasto `RETURN` al termine di ciascuna di esse), e inserire il valore 4 quando l'interprete eseguirà la seconda istruzione (si ricordi di **non** inserire **spazi** prima di ciascuna istruzione):

```
pi_greco = 3.14
raggio = float(input("Inserire il raggio di un cerchio:"))
circonferenza = 2 * pi_greco * raggio
```

Dopo l'esecuzione di queste istruzioni:

- la variabile `pi_greco` conterrà il valore 3.14
- la variabile `raggio` conterrà il valore 4
- la variabile `circonferenza` conterrà il valore 25.12

Istruzione di assegnamento: esempi di errori

Alcuni esempi di errori che si possono commettere nella scrittura di istruzioni ed espressioni:

- Scrivendo nella *shell* una qualsiasi istruzione preceduta da uno o più spazi, per es. (si noti la posizione del *prompt*):

  ```
  >>>        x = 1
  ```

  verrà mostrato un messaggio d'errore come il seguente:

  ```
  SyntaxError: unexpected indent
  ```

- Scrivendo un'espressione con una sintassi errata, per es.:

  ```
  n = 2 +
  ```

  si otterrà un messaggio d'errore come il seguente:

  ```
  SyntaxError:  invalid syntax
  ```

Esercizi

Determinare il valore delle seguenti **espressioni** Python (verificare la propria risposta scrivendole nella *shell*):

- `1.0 + 2`
- `(1 + 2) / 2`
- `2 ** 3`
- `3 // (4 * 3)`
- `256 % 2`
- `2 % 256`
- `1 / 0`
- `2 + 2`
- `"2 + 2"`
- `"2" + "2"`

Esercizi: soluzione

Determinare il valore delle seguenti **espressioni** Python (verificare la propria risposta scrivendole nella *shell*):

- `1.0 + 2 -> 3.0`
- `(1 + 2) / 2 -> 1.5`
- `2 ** 3 -> 8`
- `3 // (4 * 3) -> 0`
- `256 % 2 -> 0`
- `2 % 256 -> 2`
- `1 / 0 ->` **`ZeroDivisionError`**
- `2 + 2 -> 4`
- `"2 + 2" -> "2 + 2"`
- `"2" + "2" -> "22"`

Esercizi

Quali saranno i valori delle variabili `x` e `y` dopo la seguente sequenza di istruzioni? (verificare la propria risposta scrivendo le istruzioni nella *shell*, e valutando poi le espressioni `x` e `y`)

```
y = 2
x = y +1
y = 0
```

Qual è il valore della variabile `c` dopo la seguente sequenza di istruzioni?

```
a = "1"
b = "2"
c = a + b
```

Qual è il valore della variabile `k` dopo la seguente sequenza di istruzioni?

```
k = 0
k = k + 1
```

Esercizi

Quali saranno i valori delle variabili `x` e `y` dopo la seguente sequenza di istruzioni? (verificare la propria risposta scrivendo le istruzioni nella *shell*, e valutando poi le espressioni `x` e `y`)

```
y = 2
x = y +1
y = 0
```

x=3, y=0

Qual è il valore della variabile `c` dopo la seguente sequenza di istruzioni?

```
a = "1"
b = "2"
c = a + b
```

C='12'

Qual è il valore della variabile `k` dopo la seguente sequenza di istruzioni?

```
k = 0
k = k + 1
```

k=1

Esercizi

Qual è il valore della variabile `p` dopo la seguente sequenza di istruzioni, se il valore inserito attraverso la tastiera nel momento in cui viene valutata l'espressione `input` è 4?

```
m = 2.5
n = int(input("Inserire un numero: "))
p = (m * n) + 1
```

Qual è il valore delle variabili `r`, `s` e `t` dopo la seguente sequenza di istruzioni, se i valori inseriti attraverso la tastiera nel momento in cui vengono valutate le due espressioni `input` sono rispettivamente 1 e 2?

```
r = int(input("Inserire un numero: "))
s = int(input("Inserire un altro numero: "))
t = r + s
```

Esercizi

Qual è il valore della variabile `p` dopo la seguente sequenza di istruzioni, se il valore inserito attraverso la tastiera nel momento in cui viene valutata l'espressione `input` è 4?

```
m = 2.5
n = int(input("Inserire un numero: "))
p = (m * n) + 1
```

p=11.0

Qual è il valore delle variabili `r`, `s` e `t` dopo la seguente sequenza di istruzioni, se i valori inseriti attraverso la tastiera nel momento in cui vengono valutate le due espressioni `input` sono rispettivamente 1 e 2?

```
r = int(input("Inserire un numero: "))
s = int(input("Inserire un altro numero: "))
t = r + s
```

t=3

L'istruzione `print`

L'istruzione `print` consente di mostrare nella finestra della *shell* il **valore** di una qualsiasi **espressione** Python.

Essa costituisce un semplice meccanismo attraverso il quale i risultati delle elaborazioni svolte da un programma **possono essere comunicati agli utenti del calcolatore**.

L'istruzione `print`: sintassi e semantica

Sintassi: `print` (espressione)

- **non** devono esserci spazi prima del simbolo `print`
- espressione deve essere una qualsiasi espressione valida del linguaggio Python

Semantica: viene mostrato nella *shell* il **valore** di espressione.

È anche possibile mostrare con **una sola** istruzione `print` i valori di **un numero** qualsiasi di espressioni, con la seguente sintassi:

`print` (espressione$_1$, espressione $_2$, . . .)

In questo caso i valori delle espressioni vengono mostrati su una stessa riga, separati da un carattere di spaziatura.

L'istruzione `print`: esempi

Negli esempi seguenti il valore mostrato nella *shell* è indicato al di sotto di ciascuna istruzione (si assume che alle variabili `x` e `p` siano già stati assegnati i valori `5` e "*Mappa*": in caso contrario si otterrà un messaggio d'errore):

```
print (-3.2)
-3.2
print (2 + 2)
4
print(x)
5
```

L'istruzione `print`: esempi

- `print ((x + 1) / 4.0)`
  `1.5`
- `print ("Ciao")`
  `Ciao` (si noti che non vengono mostrati gli apici)

- `print (p + "mondo")` `Mappamondo`
- `print ("Prima riga.\nSeconda riga.")`
  `Prima riga.`
  `Seconda riga.`
  (si noti l'effetto del carattere *newline* `\n`)
- `print ("Il risultato è", x)`
  `Il risultato è 5`

Uso dell'istruzione `print`

Nella *shell* il valore di un'espressione può essere mostrato scrivendo direttamente l'espressione stessa: questo rende spesso superfluo l'uso dell'istruzione `print` nella *shell*.

L'istruzione `print` è invece **l'unico strumento utilizzabile all'interno di un programma** per mostrare nella *shell* il valore di una qualsiasi espressione.

Esercizi

Che cosa viene mostrato nella *shell* dopo l'esecuzione di ciascuna delle seguenti istruzioni?

- `print (2 - 1)`
- `print ("2 - 1")`
- `print ("2\n1")`

Che cosa viene mostrato nella *shell* dopo l'esecuzione della seguente sequenza di istruzioni?

```
b = 2
B = 3
h = 2.5
print ("Area del trapezio: ", (b + B) * h / 2.0)
```

Soluzione

Che cosa viene mostrato nella *shell* dopo l'esecuzione di ciascuna delle seguenti istruzioni?

- `print (2 - 1)` -> **1**
- `print ("2 - 1")` -> **2-1**
- `print ("2\n1")` -> **2**
  1

Che cosa viene mostrato nella *shell* dopo l'esecuzione della seguente sequenza di istruzioni?

```
b = 2
B = 3
h = 2.5
print ("Area del trapezio: ", (b + B) * h / 2.0)
```

Area del trapezio: 6.25

Scrittura ed esecuzione di programmi

Un programma Python è costituito da una **sequenza** di istruzioni che devono essere scritte attraverso un *editor* di testi e memorizzate in un *file* di testo.

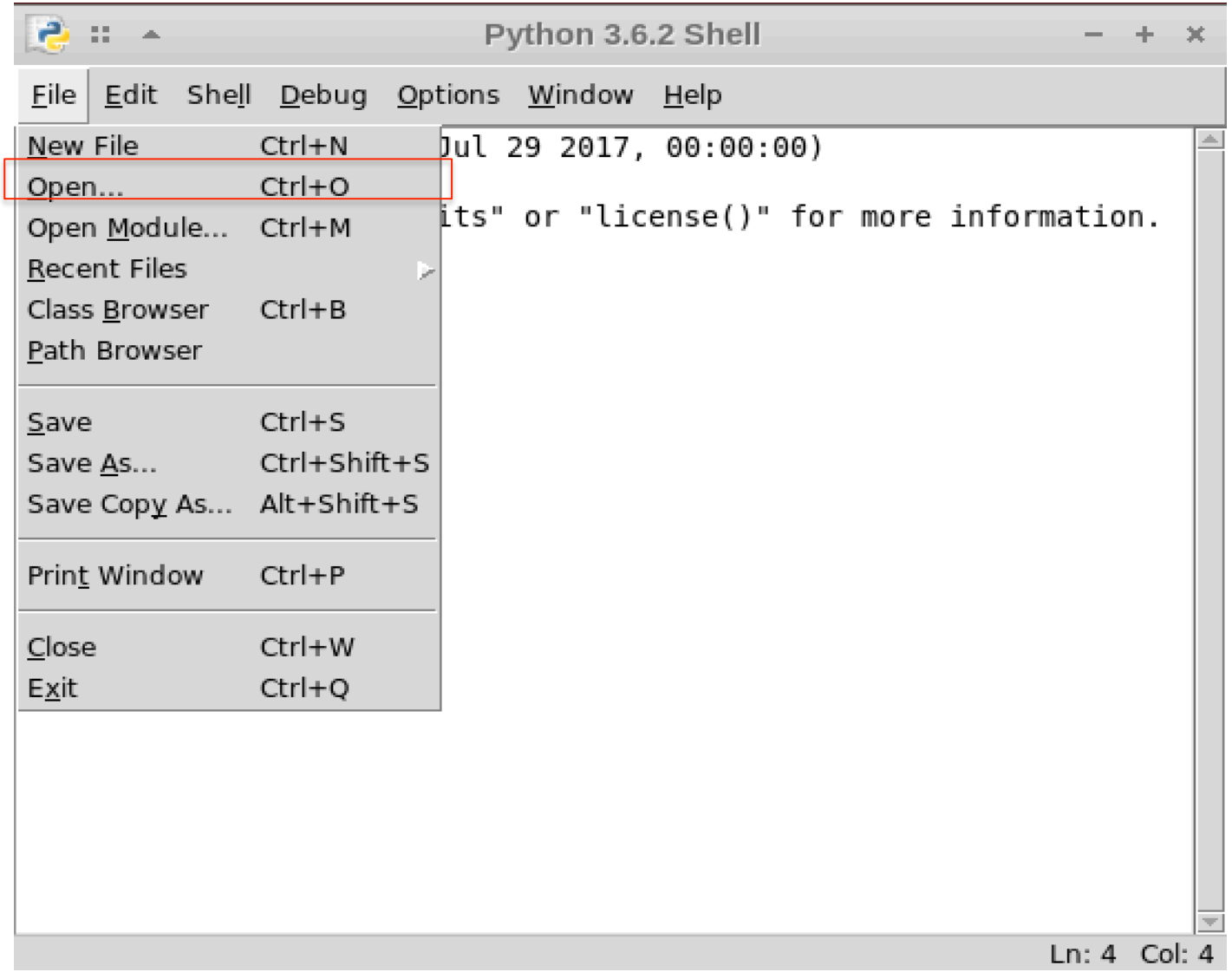
Il programma potrà **successivamente** essere eseguito all'interno di un ambiente di programmazione come `IDLE`, oppure in una *finestra dei comandi* del sistema operativo (purché sia stato installato e configurato un interprete Python).

I nomi dei *file* che contengono programmi Python devono avere estensione `.py` (per esempio, `prova.py`).

Come ogni ambiente di programmazione, `IDLE` comprende un *editor* di testi che facilita la scrittura di programmi, per esempio evidenziando con colori diversi le diverse componenti sintattiche (nomi delle istruzioni, numeri, stringhe, ecc.).

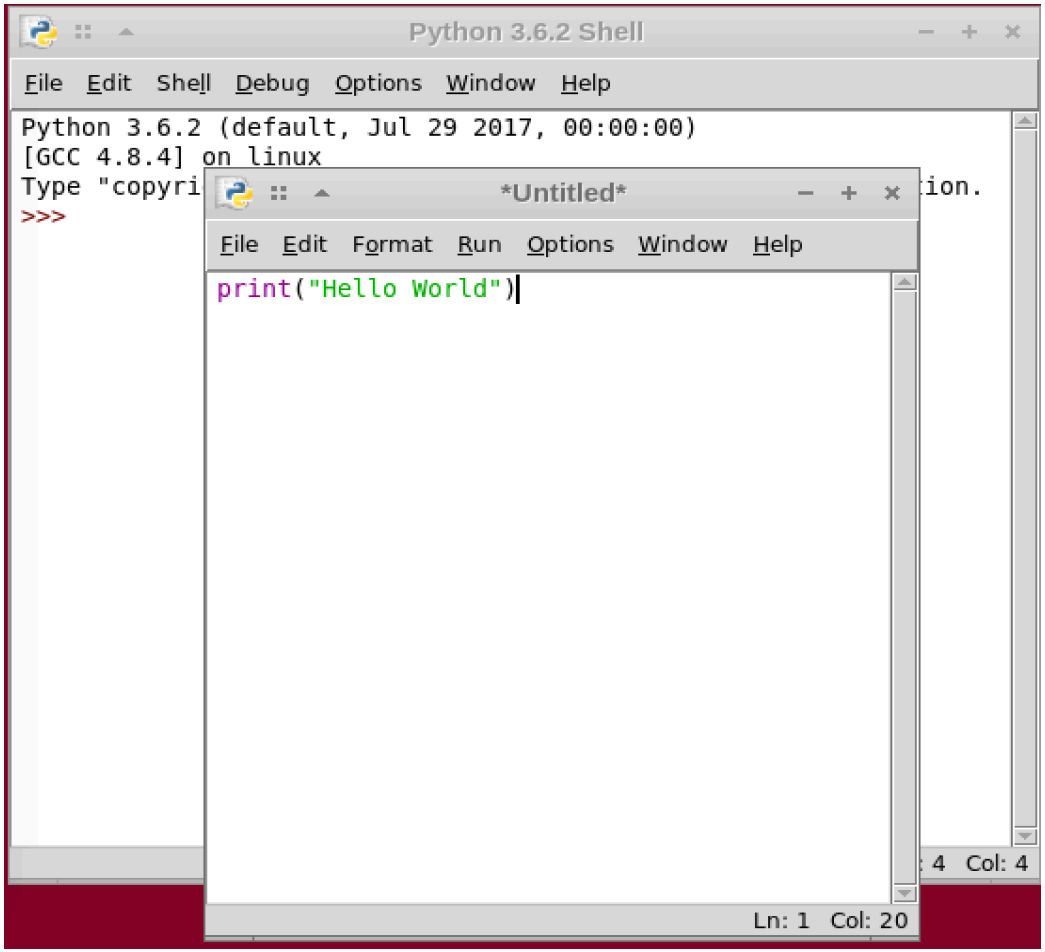
Scrittura ed esecuzione di programmi

Per aprire una nuova finestra dell'*editor* di `IDLE`, scegliere la voce `New File` del menu `File` della *shell* :



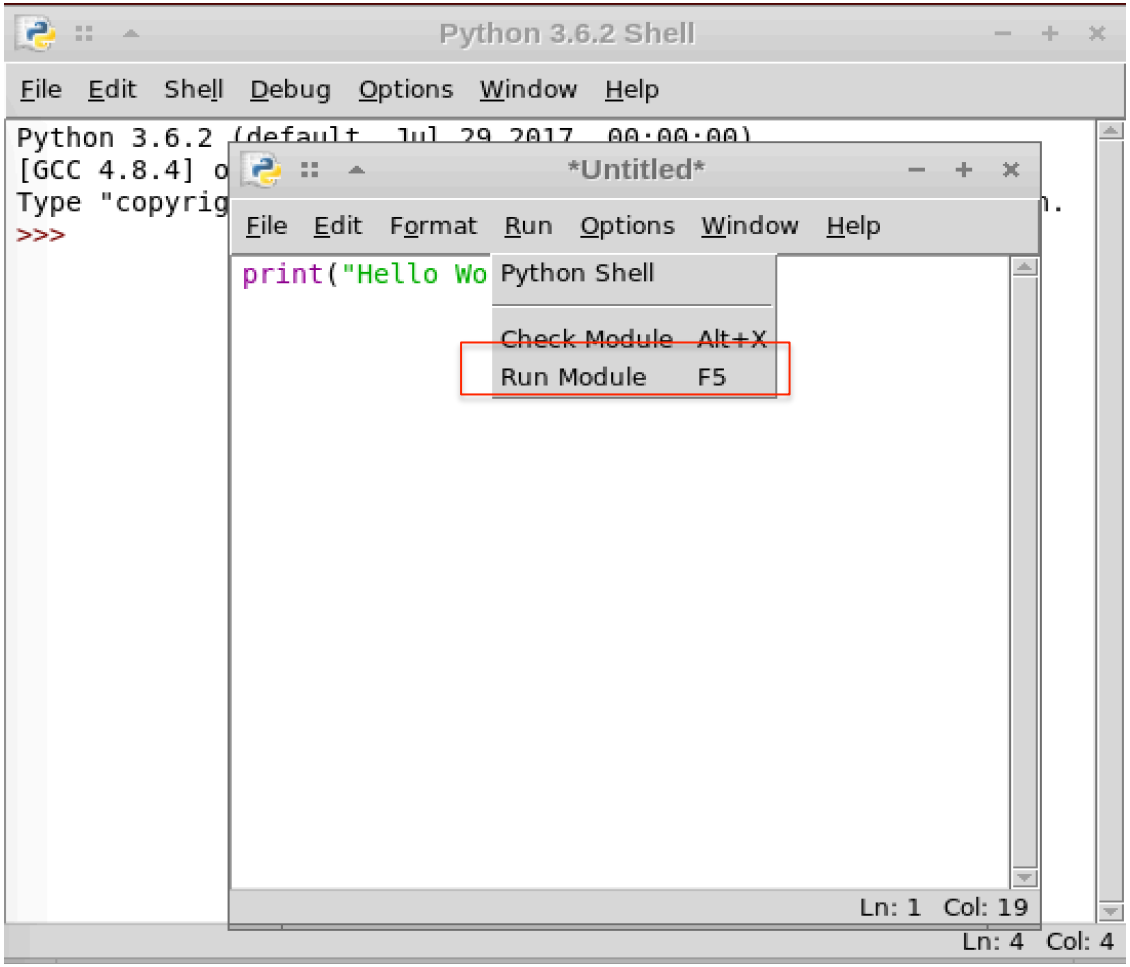
Scrittura ed esecuzione di programmi

Ogni istruzione di un programma deve essere scritta in una riga **distinta**, e **senza spazi** all'inizio (con l'eccezione delle istruzioni `if-else`, `while` e `for`, descritte più avanti). Si noti che le istruzioni scritte nell'*editor* **non** vengono immediatamente eseguite, come avviene invece per quelle scritte nella *shell*.



Scrittura ed esecuzione di programmi

Dopo aver memorizzato il programma in un *file* (il cui nome abbia estensione `.py`), esso potrà essere eseguito scegliendo la voce `Run module` del menu `Run` nella finestra dell'*editor*, oppure premendo il tasto `F5` quando la finestra dell'*editor* è **attiva**:



Scrittura ed esecuzione di programmi

L'esecuzione di un programma comporta il **riavvio** (*restart*) della *shell*, che **ha come conseguenza l'eliminazione delle eventuali variabili definite in precedenza**.

Le eventuali espressioni `input` e istruzioni `print` presenti nel programma mostreranno nella *shell* i valori corrispondenti; analogamente, i valori richiesti dalle espressioni `input` dovranno essere inseriti (attraverso la tastiera) nella finestra della *shell*.

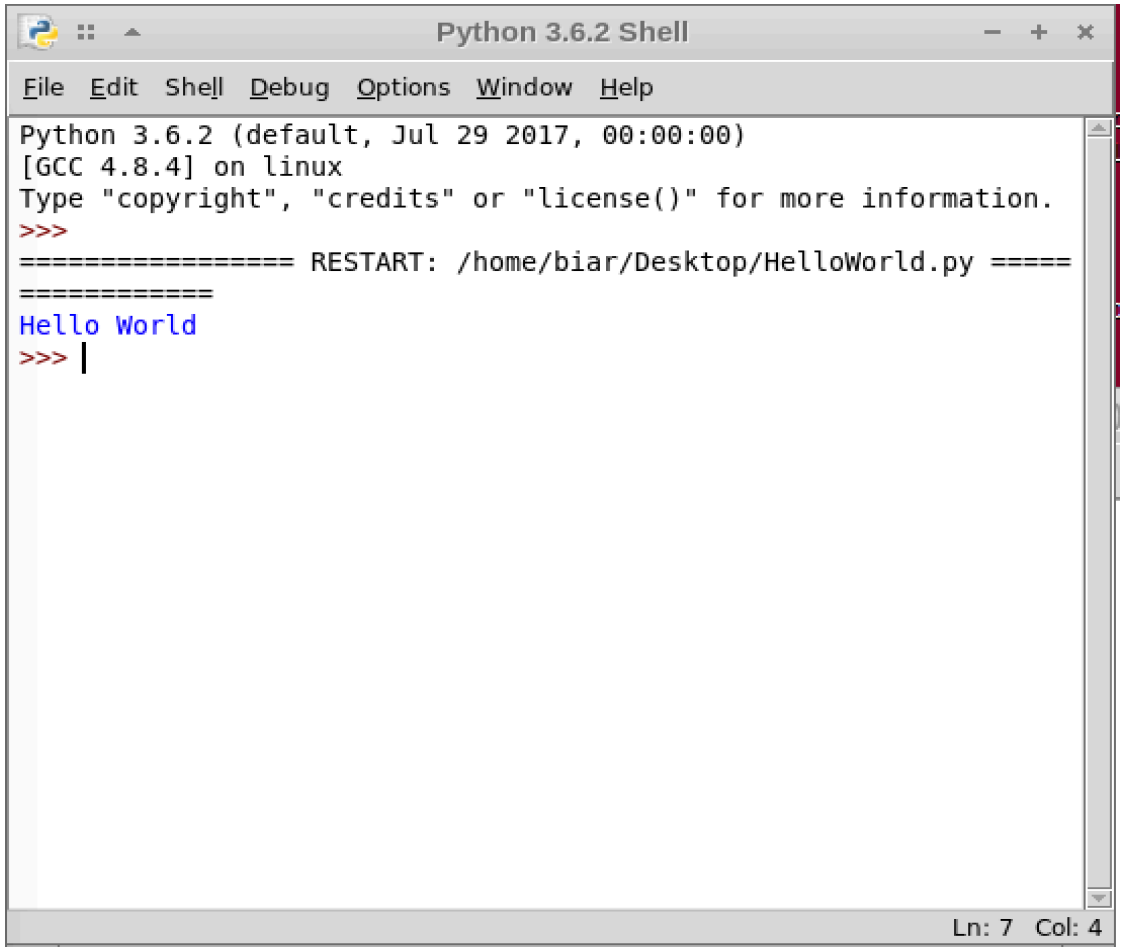
Si noti che **gli unici effetti visibili** dell'esecuzione di un programma, oltre al riavvio della *shell*, sono quelli prodotti dalle espressioni `input` e dalle istruzioni `print`.

Le istruzioni di assegnamento e le espressioni (con l'esclusione di `input`) **non producono nessun effetto visibile**. Tuttavia, dopo l'esecuzione di un programma le eventuali variabili definite al suo interno sono accessibili da successive istruzioni nella *shell*.

Scrittura ed esecuzione di programmi

Ecco il contenuto della *shell* dopo l'esecuzione del programma mostrato negli esempi precedenti.

Si noti il messaggio che indica il riavvio (*restart*) della *shell*.



Scrittura ed esecuzione di programmi

Qualche precisazione sull'apertura di *file* contenenti programmi Python.

Su alcuni sistemi operativi un "doppio *click* " sull'icona di un tale *file* fa sì che il programma venga aperto in una nuova finestra dell'*editor* di `IDLE` (consentendone la modifica o l'esecuzione, come già visto).

In altri sistemi operativi questa azione comporta invece l'**esecuzione** del programma in una **finestra dei comandi** dello stesso sistema operativo, che si chiuderà non appena il programma sarà terminato. Se il programma non contiene espressioni `input`, di solito la sua esecuzione è talmente veloce da essere impercettibile all'utente.

Se si desidera aprire e/o eseguire un programma all'interno dell'ambiente `IDLE`, si consiglia di aprire il *file* che lo contiene usando la voce `Open` dal menu `File` dell'ambiente `IDLE`, oppure di selezionare l'icona del *file* con il tasto destro del *mouse* e scegliere tra le varie opzioni l'apertura dello stesso *file* con il programma `IDLE`.

Mettere in pausa un programma Python

Per mettere in pausa l'esecuzione del programma ed evitare che la finestra si chiuda istantaneamente si può usare

```
time.sleep(secs)
```

che mette in pausa l'esecuzione per `secs` secondi

Es.:

```
import time

# Wait for 5 seconds
time.sleep(5)

# Wait for 300 milliseconds
# .3 can also be used
time.sleep(.300)
```

Commenti all'interno dei programmi

È sempre utile documentare i programmi inserendo commenti che indichino quale operazione viene svolta dal programma, quali sono i dati di ingresso e i risultati, qual è il significato delle variabili o di alcune sequenze di istruzioni, ecc.

Nei programmi Python i commenti possono essere inseriti in qualsiasi riga, preceduti dal carattere **#** ("cancelletto").

Tutti i caratteri che seguono il cancelletto, **fino al termine della riga**, sono considerati commenti e vengono trascurati dall'interprete.

Primi esempi di programmi Python

Un programma che acquisisce attraverso la tastiera il valore del raggio di un cerchio, e ne calcola la circonferenza.

File: `circonferenza.py`

```
raggio = float(input("Inserire il valore del raggio: "))
circonferenza = 2 * 3.14 * raggio
print ("La lunghezza della circonferenza è",circonferenza)
```

Primi esempi di programmi Python

Un programma che calcola l'area di un trapezio dopo aver acquisito le lunghezze delle basi e l'altezza.

File: `area_trapezio.py`

```
base_minore = float(input("Base minore del trapezio: "))
base_maggiore = float(input("Base maggiore: "))
altezza = float(input("Altezza: "))
area_trapezio = (base_minore + base_maggiore)*altezza/2
print ("L'area è", area_trapezio)
```