

## Standard output

### Comando `print`

- Il modo più semplice per inviare dati allo standard output è tramite il comando `print`
- `print` stampa su standard output una versione stringa dell'oggetto (o degli oggetti) che riceve come argomento:
  1. per ogni oggetto ricevuto come argomento, usa la funzione `str` per convertire l'oggetto in stringa
  2. stampa su standard output le stringhe così ottenute (senza delimitatori) separate da spazi, e termina con un ritorno a capo

```
>>> a, b, c = (1.4, [0,1,2], "aaa")
>>> print a, b, c
1.4 [0, 1, 2] aaa
>>> str(a), str(b), str(c)
('1.4', '[0, 1, 2]', 'aaa')
```

## Standard output

### Eco interattivo

- Nell'uso interattivo, l'interprete Python stampa un "eco" del risultato delle espressioni eseguite (se diverso da `None`)
- L'espressione può anche essere semplicemente il nome di una o più variabili
- A differenza di `print`, l'eco utilizza la funzione `repr` per convertire un oggetto in stringa, ottenendo un formato più aderente all'oggetto come risulta nel codice

```
>>> a, b, c # stampa la tupla
(1.3999999999999999, [0, 1, 2], 'aaa')
>>> a = None
>>> a # non stampa None
>>>
```

## Standard output

### Nota

- Il numero 1.4 viene stampato in forma approssimata dal comando `repr`

```
>>> a = 1.4
>>> print a
1.4
>>> a
1.3999999999999999
```

- Tale forma corrisponde alla rappresentazione decimale della sua codifica binaria finita (provate a convertire 1.4 in binario..)

## Standard input

### Funzione `raw_input`

- La funzione `raw_input` permette di leggere una stringa da standard input
- La funzione legge da standard input una stringa fino al ritorno capo incluso
- Restituisce però la stringa priva del ritorno a capo
- Prende come argomento opzionale una stringa, che stampa in output prima di leggere l'input (in genere per specificare cosa scrivere)

```
>>> p = raw_input("")
una stringa
>>> p
'una stringa'
>>> p = raw_input("termina?(S/N) ")
termina?(S/N) S
>>> p
'S'
```

## Files

### Oggetti file

- Python fornisce un oggetto speciale di tipo `file` per gestire input ed output su file.
- L'oggetto fornisce un'interfaccia tra il programma ed il file, permettendo di aprirlo, chiuderlo, scriverci e leggerci contenuti.
- Sebbene abbastanza diversi dagli oggetti visti finora, i file sono essi stessi oggetti, e vengono quindi creati assegnandoli ad una variabile, che potrà essere copiata, assegnata ad un oggetto diverso, etc.

## Files

### Creazione

- Un oggetto file si crea aprendo un file
- La funzione `open` prende come argomenti:
  - il percorso nel file system del file da aprire (assoluto o relativo alla directory corrente)
  - La modalità con cui aprire il file, tra cui:
    - r** lettura (il file deve esistere già)
    - w** scrittura (il file viene sovrascritto, o creato se non esiste)
    - a** estensione (se il file è già esistente, l'output viene aggiunto in fondo)
  - aggiungendo un **+** ad una delle modalità, si apre il file sia in lettura sia in scrittura

```
>>> f = open('/tmp/prova', "w+") scrittura e lettura
>>> f = open('TODO') # r se non specificata
```

## Files

### Metodi di lettura

`s = f.read()` legge un intero file in una stringa

`s = f.readline()` legge la prossima linea (ritorno a capo incluso) in una stringa

`l = f.readlines()` legge un intero file in una lista di stringhe contenenti le sue linee

## Files

### Metodi di scrittura

`f.write(s)` scrive una stringa nel file

`f.writelines(l)` scrive tutte le stringhe di una lista in un file (non aggiunge ritorni a capo dopo ogni stringa, ci devono già essere)

### Chiudere files

- Dopo aver utilizzato un file (per scriverci o leggerci), deve essere chiuso
- Il metodo `close` permette di chiudere file precedentemente aperti (`f.close()`)

## Files

### Esempi

```
>>> f = open("tmp1", "w")
>>> f.write("prima stringa\n")
>>> f.writelines(["seconda", "terza\n" ,
... "quarta\n"])
>>> f.close()
>>> f = open("tmp1")
>>> f.readline()
'prima stringa\n'
>>> f.readlines()
['seconda\n', 'terza\n', 'quarta\n']
>>> f.readline()
'' # fine del file
```

## Files

### Scrivere e leggere oggetti

- I metodi visti scrivono e leggono stringhe (o liste di stringhe)
- Per poter scrivere e leggere oggetti diversi, devono essere convertiti in stringhe, e viceversa.
- Tale operazione può essere fatta in modi diversi, e il più conveniente dipende dagli oggetti in questione

```

>>> s = "same old string"
>>> x,y,z = -1,4.5,0.2
>>> l = range(5) # crea una lista di interi
>>> l
[0, 1, 2, 3, 4]
>>> d = {3 : 1, 5 : 1} # vettore sparso
>>> f = open("data","w")

```

## Scrivere e leggere oggetti

### Il modulo `pickle`

- Il modo più semplice per leggere e scrivere oggetti complessi su file è tramite il modulo `pickle`
- La scrittura di uno o più oggetti su file viene fatta col metodo `dump`
- La lettura di un oggetto scritto tramite `dump` viene fatta col metodo `load`
- `pickle` converte gli oggetti in delle rappresentazioni stringa interne, non necessariamente facili da leggere se non con i suoi metodi.

## Scrivere e leggere oggetti

### Il modulo `pickle`: esempio scrittura

```

>>> import pickle
>>> f = open("data","w")
>>> pickle.dump(s,f)
>>> pickle.dump((x,y,z),f)
>>> pickle.dump(l,f)
>>> pickle.dump(d,f)
>>> f.close()

```

## Scrivere e leggere oggetti

### Il modulo `pickle`: esempio lettura

```

>>> f = open("data","r")
>>> s = pickle.load(f)
>>> s
'same old string'
>>> t = pickle.load(f)
>>> t
(-1, 4.5, 0.20000000000000001)
>>> l = pickle.load(f)
>>> l
[0, 1, 2, 3, 4]
>>> d = pickle.load(f)
>>> d
{3: 1, 5: 1}
>>> f.close()
>>> f = open("data","r")
>>> f.read()
"S'same old string'\np0\n.(I-1\nF4.5\nF0.20000000000000001\n
tp0\n.(lp0\nI0\naI1\naI2\naI3\naI4\na.(dp0\nI3\nI1\nsI5\nI1\ns."

```