Programming Lab

Parte 6

Controllo degli input e sanitizzazione

Laura Nenzi

Many slides by Stefano Russo

Cosa sono gli input

Gli input sono qualsiasi cosa “entri” da qualche parte. Ad esempio:

- gli argomenti delle funzioni, incluse quelle degli oggetti
- i dati caricati da file
- gli input dell’utente (da linea di comando o da richiesta del programma)
- i dati caricati da database (non li vedremo)
- i dati che arrivano dai socket di rete (non li vedremo)
- etc.

Input dell’utente

In Python esiste una funzione predefinita chiamata `input` che sospende il programma ed attende che l’utente scriva qualcosa

```
>>> testo = input()
Cosa stai aspettando
>>> testo
'Cosa stai aspettando'
```

Prima dell’inserimento dei dati, è buona norma visualizzare un messaggio, chiamato prompt, che informa l’utente di ciò che deve inserire.

```
>>> nome = input('Come...ti chiami?\n')
Come...ti chiami?
Artù, Re dei Bretoni!
>>> nome
'Artù, Re dei Bretoni!'
```

Non fidarsi degli input

- Non ci si deve mai fidare della bontà degli input, perché non si ha il controllo su chi o cosa li genera.

- Se i dati sono in formato sbagliato, danneggiati, fuori range o in generale non adatti per l'uso che se ne deve fare, il programma può:
  - andare in errore
  - oppure, peggio, non mostrare nessun segno di problema ma dare risultati sbagliati.

Cosa vuol dire che un’input è corretto

→ **Di base, *dipende da cosa volete farci***

Ad esempio, se state scrivendo una funzione che legge i dati di un file CSV e volete solo le prime “**n**” righe, questo parametro dovrà:

1. essere un numero
    - oppure una stringa convertibile a numero, dipende se lo volete o meno
2. essere un numero maggiore di zero

Se invece volete specificare un range di righe (**inizio**, **fine**), chiaramente l’inizio dovrà essere antecedente alla fine, ovvero inizio < fine.

Prevedere gli *edge cases*

- Non ci si può basare solo su scenari idealizzati, ma bisogna essere pragmatici e preparati.

- È importante considerare scenari inaspettati, come input vuoti, valori estremamente grandi o piccoli, formati inaspettati.

- È di solito sulle eventuali condizioni rare che in genere sorgono la maggior parte dei problemi.

Prevenire o correggere?

- In genere, i linguaggi tipizzati (come C e Java) preferiscono prevenire tutto
- In Python, che non è un linguaggio tipizzato, è invece in genere preferita la logica del "prima provo, poi chiedo scusa".
- Non si cerca sempre di prevenire, ma piuttosto di correggere gli errori (cioè, gestendo le eccezioni) laddove si verifichino.
- Tuttavia, anche in Python può spesso avere molto senso controllare prima alcune cose. E.g. lavorare con un parametro che deve essere dentro ad un certo range.

Come si controlla che gli input siano corretti

Operatori classici

- `>`, `<`, `>=`, etc.

Controllo di non nullità

- `is None`
- `is not None`
- `== ''` (stringa vuota)

Come si controlla che gli input siano corretti

Valori entro un insieme predefinito

- `sesso in ['M', 'F']`

Tipi, oggetti, classi

- `type(nome_variabile)`
- `isinstance(nome_variabile, Classe)`
- `issubclass(nome_super_classe, nome_sotto_classe)`

Cosa fare se l’input non è corretto

Opzioni possibili:

1) stampare l’errore ed usare un default o aggirarlo (errore “recoverable”)
2) stampare l’errore ed uscire (errore non “recoverable”)

Il punto 2 si fa SOLO se si sta scrivendo il corpo di un programma interattivo.

→ Ovvero, **MAI** nelle **funzioni**, **MAI** negli **oggetti**.

..ma esiste anche

3) generare (sollevare) una nostra eccezione
   → Nelle funzioni e negli oggetti si fa questo!

Come si “solleva” una eccezione

```
raise Exception('Messaggio di errore')
```

Come si “solleva” una eccezione

```
raise Exception('Messaggio di errore')
```

Questa è la stringa che viene poi stampata a schermo quando l'eccezione sale fino all’interprete Python, o che stampate a schermo esplicitamente voi quando le catturate.

Come si “alza” una eccezione

```
raise Exception('Messaggio di errore')
```

..ma MOLTO meglio fare qualcosa del tipo:

```
raise Exception('Ho avuto un errore, ecco il parametro che
                lo ha generato: "{}"'.format(parametro))
```

Come creare una vostra eccezione

```
class Errore(Exception)
    pass
```

..e poi semplicemente:

```
raise Errore('Mona!')
```

Come creare una vostra eccezione

Esempio di generazione di eccezione:

```
parametro = -5
if parametro < 0:
    raise  ValueError('Il parametro è minore di zero')
```

Esempio di definizione e generazione di una eccezione:

```
class InvalidParameter(Exception):
    pass

parametro = -5
if parametro < 0:
    raise  InvalidParameter('Il parametro è minore di zero')
```

Come si controlla che gli input siano corretti

Esempio: Controllo a priori del tipo di input di una funzione che somma tutti gli elementi di una lista

```
def somma_lista(lista_input):
    if not type(lista_input) == list:
        raise TypeError('Il tipo di lista_input non è lista ' +
            'ma "{}"'.format(type(lista_input)))
    somma = 0
    for elemento in lista_input:
        somma += elemento
    return somma
```

Come si controlla che gli input siano corretti

Esempio: Controllo a priori del tipo di input di una funzione che somma tutti gli elementi di una lista accettando anche le sottoclassi del tipo dati lista.

```
def somma_lista(lista_input):
    if not isinstance(lista_input, list):
        raise TypeError('Il tipo di lista_input non è lista, ' +
                        'ma "{}"'.format(type(lista_input)))
    somma = 0
    for elemento in lista_input:
        somma += elemento
    return somma
```

Sanitizzare gli input

Spesso potete provare a “sanitizzare” gli input, per prevenire potenziali errori.

Esempio 1: supponiamo vi arrivi la variabile **sesso** con il valore “m”.

Se avete fatto il check con:

```
if sesso not in ['M', 'F']:
    # avvisa dell’errore
```

...allora il check non passa. Per evitare questo problema, potete fare:

```
sesso_sempre_maiuscolo = sesso.upper()
if sesso_sempre_maiuscolo not in ['M', 'F']:
    # avvisa dell’errore
```

Sanitizzare gli input

Spesso potete provare a “sanitizzare” gli input, per prevenire potenziali errori.

Esempio 2: supponiamo vi arrivi la variabile **sesso** con il valore “M ”.

Notate lo spazio

Se avete fatto il check con:

```
if sesso not in ['M', 'F']:
    # avvisa dell’errore
```

...allora il check non passa. Per evitare questo problema, potete fare:

```
sesso_pulito = sesso.strip()
if sesso_pulito not in ['M', 'F']:
    # avvisa dell’errore
```

Messaggio da portare a casa

Ragionate sempre in logica “*cosa succede se*”...

Ad esempio:

- non trovo il file che dovrei leggere
- mi passano una stringa e non un numero
- mi chiedono di leggere le righe di un file dalla 100 alla 200 ma il file di righe ne ha solamente 17
- mi passano un valore non ammissibile

Ora potete raggiungere il 10 sull'esercizio Parte 4

Create un oggetto **CSVFile** che rappresenti un file CSV, e che:

1) venga inizializzato sul nome del file csv, e
2) abbia un attributo "name" che ne contenga il nome
3) abbia un metodo "get_data()" che torni i dati dal file CSV come lista di liste, ad es: `[ ['01-01-2012', '266.0'], ['01-02-2012', '145.9'], ... ]`

Provatelo sul file "shampoo_sales.csv".

Poi, scaricate il vostro script Python e testatelo su autograding

Esercizio

Modificate l’oggetto **CSVFile** della lezione precedente in modo che alzi un'eccezione se il nome del file non è una stringa (nell’ `__init__()`).

Poi, modificate la funzione **get_data**() di **`CSVFile`** in modo da leggere solo un’ intervallo di righe del file*, aggiungendo gli argomenti `start` ed `end` *opzionali*, controllandone la correttezza e sanitizzandoli se serve.

```
get_data(self, start=None, end=None)
```

*inclusa l’intestazione, estremi inclusi, e partendo da “1”.

Alla fine ricordatevi di committare tutto

Esercizio

Nota: se avete eseguito correttamente l'esercizio della lezione precedente, anche `NumericalCSVFile` ha quasi automaticamente ereditato queste migliorie.

"Quasi" perchè mentre il controllo del nome è stato effettivamente ereditato, per il range di righe bisogna ricorrere ad una "formula magica" che "inoltra" gli argomenti della funzione figlia al padre, a prescindere da quali essi siano:

```
class NumericalCSVFile(CSVFile):
            def get_data(self, *args, **kwargs):
            csv_data = super().get_data(*args, **kwargs)
                        ...
```

Altri Esercizi su gestione dell'input

1. Scrivete un programma che riceva una data di nascita come input e visualizzi l’età dell’utente e il numero di giorni, ore, minuti e secondi che mancano al prossimo compleanno.

2. Scrivete un programma che chieda all'utente di inserire un numero intero. Se l'utente inserisce un valore valido, il programma deve stampare il quadrato del numero. Se l'utente inserisce un valore non valido, il programma deve visualizzare un messaggio di errore e richiedere nuovamente l'input.

3. Create un programma che mostri un menù all'utente con tre opzioni:
   1. Calcolare la somma di due numeri.
   2. Calcolare la differenza tra due numeri.
   3. Uscire.

   Il programma deve:
   1. Chiedere all'utente di scegliere un'opzione (1, 2 o 3).
   2. Eseguire l'operazione corrispondente se l'input è valido.
   3. Gestire input non validi mostrando un messaggio di errore.
   4. Continuare a mostrare il menù fino a quando l'utente sceglie di uscire (opzione 3).