Programming Lab

Parte 6

Controllo degli input e sanitizzazione

Stefano Alberto Russo

Cosa sono gli input

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

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

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.

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)`

Come si controlla che gli input siano corretti

Ricordatevi che potete anche “prima provare e poi chiedere scusa”. Esempi:

```
try:
    float(numero)
except:
    # gestite l’errore
```

```
try:
    my_string.split(',')
except:
    # gestite l’errore
```

Nota bene: *queste slides non sono esaustive!* sono solo degli degli esempi.

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 (alzare) una nostra eccezione
   → Nelle funzioni e negli oggetti si fa questo!

Come si “alza” una eccezione

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

Come si “alza” 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!')
```

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

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)
        ...
```