Programming Lab

Parte 5

La gestione degli errori:
le eccezioni ed il flusso try-except

Laura Nenzi

Many slides by Stefano Russo

Le eccezioni

In Python gli errori si chiamano “eccezioni”:

```
1
2   my_var = 'Ciao'
3   print(mia_variablie)
4
```

```
$ python esercizio.py
Traceback (most recent call last):
  File "esercizio.py", line 2, in <module>
 print(mia_variablie)
          ^^^^^^^^^^^^
NameError: name 'mia_variablie' is not defined
```

```
1
2   my_var = 'Ciao'
3   float(my_var)
4
```

```
$ python esercizio.py
Traceback (most recent call last):
  File "esercizio.py", line 2, in <module>
    float(my_var)
ValueError: could not convert string to float: 'Ciao'
```

Cosa sono le eccezioni

- Sono una qualsiasi interruzione del normale flusso del codice
- Segnalano la presenza di un problema che il programma deve gestire esplicitamente.
- Non tutte le eccezioni sono errori, ma tutti gli errori gestiti da Python sono rappresentati come eccezioni.

Cosa sono le eccezioni

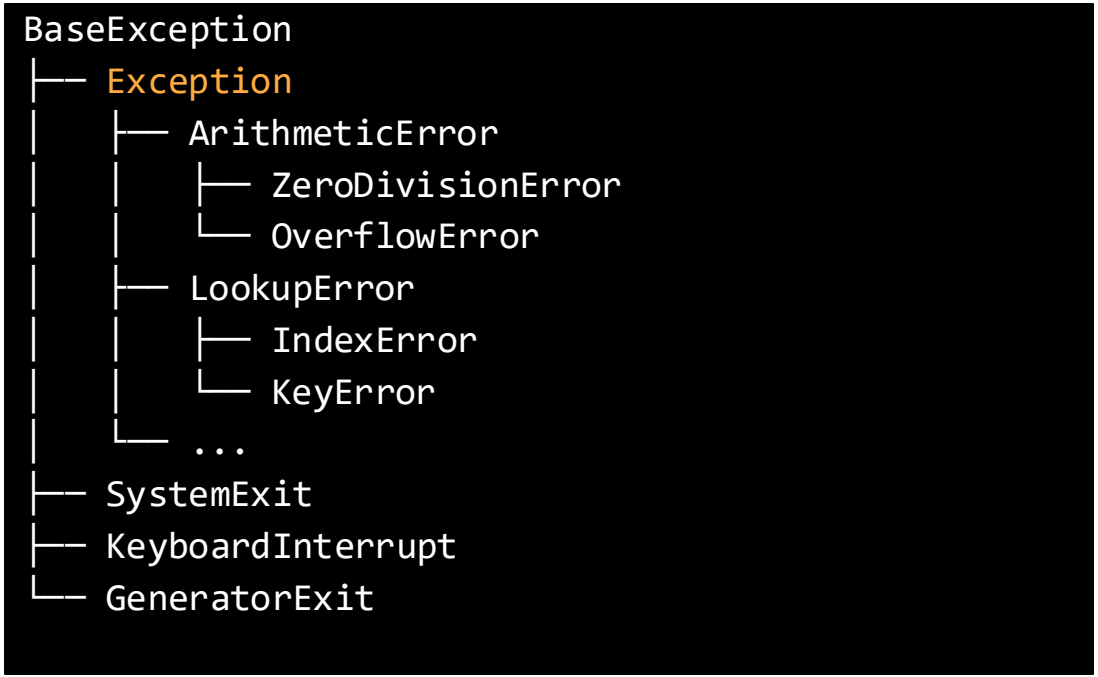
Le eccezioni sono oggetti. Come tutto in Python.

Ci sono diversi tipi di eccezioni → *estendono tutti la classe base “Exception”*

Esempi di Built-in Exceptions:

```
Exception
    ArithmeticError
        FloatingPointError
        ZeroDivisionError
    AttributeError
    SyntaxError
    NameError
    TypeError
    ValueError
```

Gerarchia delle Eccezioni



Quando si verifica una eccezione …

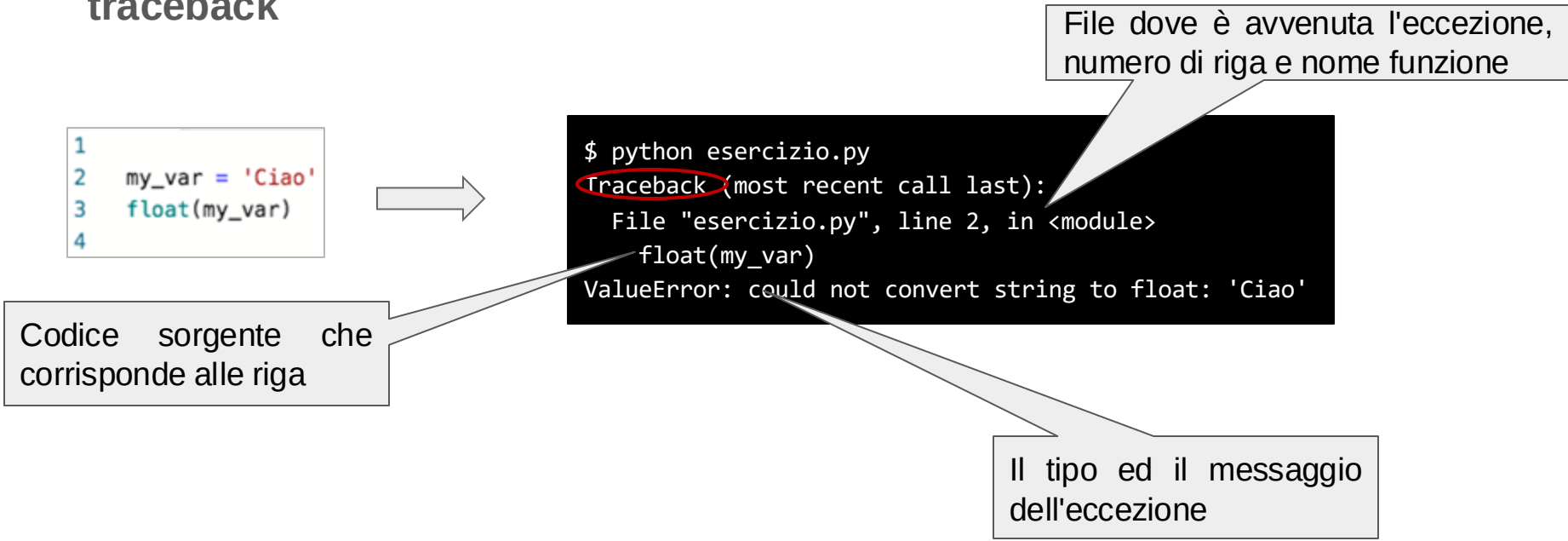
- Python interrompe l'esecuzione del codice dove è avvenuta e stampa il **traceback**

```
1
2   my_var = 'Ciao'
3   float(my_var)
4
```

```
$ python esercizio.py
Traceback (most recent call last):
  File "esercizio.py", line 2, in <module>
    float(my_var)
ValueError: could not convert string to float: 'Ciao'
```

Quando si verifica una eccezione …

- Python interrompe l'esecuzione del codice dove è avvenuta e stampa il **traceback**



Il Traceback

- È un “report” automaticamente generato da Python
- Serve ad identificare dove è avvenuta l’eccezione.
- Quello che fa è proprio di rintracciare all’indietro le chiamate delle funzioni che hanno portato al suo verificarsi.

Il costrutto try-except

Permette di gestire le eccezioni (errori).

```
try:
    # Blocco di codice in cui possono verificarsi eccezioni
except [EccezioneTipo] [as e]:
    # Blocco di codice per gestire l'eccezione specificata
else:
    # Blocco di codice che viene eseguito se NON si verifica alcuna eccezione
finally:
    # Blocco di codice che viene sempre eseguito, sia in caso di eccezione che no
```

Il costrutto try-except

Permette di gestire le eccezioni (errori).

```
1
2   my_var = 'Ciao'
3
4   try:
5       my_var = float(my_var)
6   except:
7       print('Non posso convertire "my_var" a valore numerico!')
8
```

```
> python lezione5.py
Non posso convertire "my_var" a valore numerico!
```

Il costrutto try-except

Permette di gestire le eccezioni (errori).

```
1
2   my_var = 'Ciao'
3
4   try:
5       my_var = float(my_var)
6   except:
7       print('Non posso convertire "my_var" a valore numerico!')
8       print('Uso il valore di default "0.0" per "my_var"')
9       my_var = 0.0
10
```

```
> python lezione5.py
Non posso convertire "my_var" a valore numerico!
Uso il valore di default "0.0" per "my_var"
```

Il costrutto try-except

...e posso avere l'eccezione dentro l' except:

```
1
2   my_var = 'Ciao'
3
4   try:
5       my_var = float(my_var)
6   except Exception as e:
7       print('Non posso convertire "my_var" a valore numerico!')
8       print('La variabile "my_var" valeva: "{}"'.format(my_var))
9       print('Ed ho avuto questo errore: "{}"'.format(e))
10
```

```
> python lezione5.py
Non posso convertire "my_var" a valore numerico!
La variabile "my_var" valeva: "Ciao"
Ed ho avuto questo errore: "could not convert string to float: 'Ciao'"
```

Il costrutto try-except

...e posso avere l'eccezione dentro l' except:

```
1
2   my_var = 'Ciao'
3
4   try:
5       my_var = float(my_var)
6   except Exception as e:
7       print('Non posso convertire "my_var" a valore numerico!')
8       print('La variabile "my_var" valeva: "{}"'.format(my_var))
9       print('Ed ho avuto questo errore: "{}"'.format(e))
10
```

Per usare l'eccezione dentro l'except, devo sempre specificare che eccezione voglio gestire, se le voglio gestire tutte allora uso la classe base Exception

```
> python lezione5.py
Non posso convertire "my_var" a valore numerico!
La variabile "my_var" valeva: "Ciao"
Ed ho avuto questo errore: "could not convert string to float: 'Ciao'"
```

Il costrutto try-except

Posso infatti anche gestire solo eccezioni specifiche:

```
try:

    my_var = float(my_var)

except ValueError:
    print('Non posso convertire "my_var" a valore numerico!')
    print('Ho avuto un errore di VALORE. "my_var" valeva "{}".'.format(my_var))

except TypeError:
    print('Non posso convertire "my_var" a valore numerico!')
    print('Ho avuto un errore di TIPO. "my_var" era di tipo "{}".'.format(type(my_var)))

except Exception as e:
    print('Non posso convertire "my_var" a valore numerico!')
    print('Ho avuto un errore generco: "{}"'.format(e))
```

L'`else` ed il `finally`

Sono utili per "fare pulizia"

```
try:
    file_parametro = open(nome_file_parametro)
    parametro_come_stringa = file_parametro.read()
    parametro_come_float = float(parametro_come_stringa)
except IOError:
    print('Non posso leggere il file!')
    print('Userò il vaore di default per il parametro')
except ValueError:
    print('Non posso convertire il parametro a valore numerico!')
    print('Errore di valore, il parametro valeva "{}".'.format(parametro_come_stringa))
    print('Userò il valore di default per il parametro')
except Exception as error:
    print('Errore generico: "{}".'.format(error))
    print('Userò il valore di default per il parametro')
else:
    parametro = parametro_come_float
finally:
    file_parametro.close()
```

Ora potete raggiungere il 10 sull’esercizio Parte 3

Scrivete una funzione `sum_csv(file_name)`

che sommi tutti i valori delle vendite degli

- shampoo del file passato come argomento

Provatelo sul file “shampoo_sales.csv”.

Poi, scaricate il vostro script e testatelo su Autograding

Esercizio (a)

Modificate l'oggetto **`CSVFile`** della lezione precedente in modo che stampi a schermo un messaggio di errore* se si cerca di aprire un file non esistente.

Potete fare questo controllo:

a) nella funzione `get_data()`, oppure

b) nell' `__init__()` (basta che leggiate la prima riga per vedere se il file esiste)

**il messaggio di errore deve contenere la parola "Errore" per essere rilevato dai test di autograding.*

Esercizio (b)

Estendete l'oggetto **`CSVFile`** chiamandolo **NumericalCSVFile** e facendo in modo che converta automaticamente a numero `float` *tutte le colonne* tranne la prima (della data). Chiamate la `get_data` originale con `super().get_data()`, poi converite tutto a float.

A questo punto, aggiungete a mano questi due campi al file "shampoo_sales.csv":

```
01-01-2015,
01-02-2015,ciao
```

e gestite gli errori che verranno generati in modo che le linee vengano saltate senza bloccare il programma ma che venga stampato a schermo l'errore*

**il messaggio di errore deve contenere la parola "Errore" per essere rilevato dai test di autograding.*