

Gestione degli errori

Informatica@DSS 2019/2020 — Il canale

Massimo Lauria <massimo.lauria@uniroma1.it>
<https://massimolauria.net/courses/informatica2019/>

Segnalare un errore

In caso di errore l'esecuzione del programma si interrompe ed python vi comunica cosa è andato storto.

```
5 / 0
```

```
1
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
ZeroDivisionError: division by zero
```

```
2+"ciao"
```

```
1
```

```
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Causare un errore volontariamente

Una funzione può causare volontariamente una situazione di errore per segnalare, ad esempio, che

- i parametri passati non vanno bene
- delle operazioni non sono andate a buon fine

Sintassi

```
raise NomeDellErrore 1  
raise NomeDellErrore("Messaggio opzionale") 2  
raise NomeDellErrore("Messaggio opzionale") 3
```

Esempio

```
def scontato(prezzo,sconto):          1
    if not ( 0 <= sconto <= 100 ):    2
        raise ValueError("Lo sconto non è tra zero e cento.") 3
                                        4

    return prezzo*(1.0 - sconto/100.0 ) 5
                                        6

print( scontato(500, 20) )            7
print( scontato(500,-10) )           8
```

400.0

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "/tmp/babel-HMbt3B/python-ninti2", line 8, in <module>

print(scontato(500,-10))

File "/tmp/babel-HMbt3B/python-ninti2", line 3, in scontato

raise ValueError("Lo sconto non è tra zero e cento.")

ValueError: Lo sconto non è tra zero e cento.

Significato di un errore

Sollevarre un errore in una funzione

- interrompe il programma
- l'errore viene segnalato all'utente

Esiste la possibilità di far gestire al programma stesso gli errori che produce. Noi non vedremo questa cosa.

Interruzione del flusso di esecuzione

- ▶ `break` interrompe un ciclo
- ▶ `return` interrompe una funzione
- ▶ `raise` interrompe il programma

Alcuni tipi di errore

- ▶ `TypeError` un valore ha tipo sbagliato.
- ▶ `ValueError` valori in passato non sono accettabili
- ▶ `NameError` nome di variabile o funzione non esiste
- ▶ `ZeroDivisionError` divisione per zero
- ▶ `IndentationError` indentazione scorretta

e molti altri.

Ancora con scontato (I)

```
def scontato(prezzo,sconto):           1
    if not isinstance(prezzo, (int,float) ): 2
        raise TypeError("prezzo deve essere di tipo numerico") 3
    4
    if not isinstance(sconto, (int,float) ): 5
        raise TypeError("lo sconto deve essere di tipo numerico") 6
    7
    if not ( 0 <= sconto <= 100 ): 8
        raise ValueError("Lo sconto non è tra zero e cento.") 9
    10
    return prezzo*(1.0 - sconto/100.0 ) 11
```

Ancora con scontato (II)

```
print( scontato(500,20) )      1
print( scontato(500,'ciao') ) 2
```

400.0

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

File "/tmp/babel-HMbt3B/python-kcSbhU", line 2, in <module>

print(scontato(500,'ciao'))

File "/tmp/babel-HMbt3B/python-HAzew5", line 6, in scontato

raise TypeError("lo sconto deve essere di tipo numerico")

TypeError: lo sconto deve essere di tipo numerico

Type checking e `isinstance`

```
isinstance( expr, tipo )
```

1

restituisce

- ▶ True se `expr` ha valore del tipo richiesto
- ▶ False altrimenti

```
isinstance( expr, (tipo1, tipo2, ... ) )
```

1

restituisce

- ▶ True se tipo di `expr` è in `(tipo1, tipo2, ...)`
- ▶ False altrimenti