

Sollevare degli errori

Informatica@DSS 2023/2024

Massimo Lauria <massimo.lauria@uniroma1.it>
<https://massimolauria.net/informatica2023/>

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

- ▶ gli argomenti passati non vanno bene
- ▶ delle operazioni non sono andate a buon fine

Sintassi

Un errore in python viene *sollevato*, ovvero viene segnalato al sistema python. `raise` è diverso da `return`

```
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):

...

...

ValueError: Lo sconto non è tra zero e cento.

Significato di un errore

Sollevare un errore in una funzione

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

Interruzione del flusso di esecuzione

- ▶ `return` interrompe una funzione e dà un risultato
- ▶ `raise` interrompe il programma (e solleva un errore)
- ▶ `break` interrompe un ciclo
- ▶ `continue` interrompe una ripetizione di ciclo

Alcuni tipi di errore

- ▶ `TypeError` un valore ha tipo sbagliato.
- ▶ `ValueError` argomenti passati non sono accettabili
- ▶ `IndexError` indice non valido in una sequenza
- ▶ `KeyError` chiave non presente in un dizionario
- ▶ `NameError` nome di variabile o funzione non esiste
- ▶ `ZeroDivisionError` divisione per zero
- ▶ `IndentationError` indentazione scorretta
- ▶ `FileNotFoundError` file non trovato

Errori in funzioni annidate (I)

```
def interna():
    print("Inizio della funzione interna")
    raise ValueError("errore provocato")
    print("Fine dalla funzione interna")

def esterna():
    print("Inizio nella funzione esterna")
    interna()
    print("Fine dalla funzione esterna")

esterna()
```

Cosa succede se eseguo esterna? Proviamolo con Thonny

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):

...

...

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 uno di `(tipo1, tipo2, ...)`
- ▶ False altrimenti

Minimo in una lista (I)

- ▶ solleva ValueError se la lista è vuota
- ▶ solleva TypeError se ci sono valori non confrontabili.

```
def minimo(lista): 1
    if len(lista)==0: 2
        raise ValueError("La lista è vuota") 3
    temp_min = lista[0] 4
    for i in range(1,len(lista)): 5
        if temp_min > lista[i]: 6
            temp_min = lista[i] 7
    return temp_min 8
```

```
print(minimo(['canotto', 'ape', 'battello'])) 1
print(minimo([5,1,-3,0,71])) 2
```

```
ape
-3
```

Minimo in una lista (II)

```
print(minimo([]))
```

1

```
Traceback (most recent call last):
```

```
...
```

```
...
```

```
ValueError: La lista è vuota
```

Osservate che il `TypeError` non lo generiamo esplicitamente noi, ma lo fa l'operatore `>`.

```
print(minimo([2,6,"casa",5]))
```

1

```
Traceback (most recent call last):
```

```
...
```

```
...
```

```
TypeError: '>' not supported between instances of 'int' and 'str'
```