# Errori ed eccezioni

November 5, 2020

## 1 Errori di sintassi ed eccezioni

In python possono capitare due tipi di errori principalmente: - Errori di sintassi : errori dovuti alla scrittura sbagliata di comandi nel codice - eccezioni : errori la cui natura è logica e non simile alla precedente Fortunatamente python fornisce dei possibili metodi per risolvere questi tipi di problemi e gestirli.

### 1.1 Errori di sintassi

Questi purtroppo devono essere risolti, il motivo è semplice: abbiamo sbagliato a scrivere male un comando, abbiamo dimenticato una parentesi o altro, in tal caso dobbiamo soltanto rimediare all'errore, facciamo un esempio:

[1]:
```
val = 5  #tutto ok
val += 1 #tutto ok
pint('Il valore è:', val) #qua c'é un errore
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-1-737a63aec688> in <module>
      1 val = 5  #tutto ok
      2 val += 1 #tutto ok
----> 3 pint('Il valore è:', val) #qua c'é un errore

NameError: name 'pint' is not defined
```

L'errore è che ho sbagliato a scrivere `print` e ho scritto `pint`, un altro esempio è:

[2]:
```
val = 0 #ok
while True: #ok
    val += 1 #ok
    if (val == 5) break; # non ok
```

```
  File "<ipython-input-2-165a5c94f1c4>", line 4
    if (val == 5) break; # non ok
                 ^
SyntaxError: invalid syntax
```

In questo caso ci siamo dimenticati di mettere dopo le parentesi : e questo ha creato un SintaxError.

## 1.2 eccezioni

Le eccezione sono invece dovute a un errore nella logica ed in questo caso python ci permette di gestire i problemi, vediamo un esempio:

[3]:
```
1/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-3-9e1622b385b6> in <module>
----> 1 1/0

ZeroDivisionError: division by zero
```

Python ci avvisa già che l'errore che stiamo commettendo è la divisione di un numero per zero, che è impossibile da fare, in tal caso possiamo decidere di gestire l'eccezione.

### 1.2.1 Try e Except

Partiamo prima dal come catturare l'eccezione e definire cosa fare in tal caso, il primo passo è indentare il nostro codice dopo aver usato la parola chiave `try:` infine scriviamo il codice interessato e qualora un'eccezione capitasse nel blocco usiamo `expect` per catturarla e definire cosa fare.

[4]:
```
try: #prova a interpretare il codice
    val = 1/0
except Exception: #cattura l'eccezione
    #questo codice verrà interpretato solo se
    #un'eccezione è capitata
    print('Qualcosa è andato storto')
```

```
Qualcosa è andato storto
```

Come possiamo notare ora non abbiamo un errore come prima, ma abbiamo stampato una stringa qualora una **qualunque eccezione** fosse capitata, infatti `Exception` si riferisce a una qualsiasi eccezione senza essere specificata, qualora volessimo essere più precisi dobbiamo definire che tipo di eccezione considerare per far invocare il nostro codice.

[5]:
```
try:
    val = 1/0
except ZeroDivisionError:
    #solo nel caso dividiamo un numero diviso per
    #zero questo blocco verrà interpretato
    print('Ho diviso un numero per zero')
```

```
Ho diviso un numero per zero
```

Ovviamente qualora non ci fosse nessun errore il blocco di except non verrà intepretato e verrà intepretato solo il blocco di `try`. In genere è possibile aggiungere più tipi di eccezioni scrivendo `except(ZeroDivisionError, TypeError, ecc..)`. Qualora volessimo usare anche le informazioni delle eccezioni possiamo definire un acronimo per la nostra eccezione con `as`.

[6]:
```
try:
    val = 1/0
#ho definito ZeroDivisionError come error usando as
except ZeroDivisionError as errore:
    #solo nel caso dividiamo un numero diviso per
    #zero questo blocco verrà interpretato
    print('Ho diviso un numero per zero')
    print('errore originale:',errore)
```

```
Ho diviso un numero per zero
errore originale: division by zero
```

### 1.3 Raise

Qualora fosse necessario creare un eccezione possiamo usare `raise` per creare un eccezione a nostro piacimento, riprendiamo un esempio fatto in funzioni:

[7]:
```
def somma_interi(x = 0, y = 0):
    '''somma due numeri interi'''
    #qualora uno dei due valori o entrambi non siano interi
    #creiamo un eccezione di tipo TypeErro
    if (type(x)!= int) or (type(y)!=int):
        #invoca l'eccezione con un messaggio personalizzato
        raise TypeError('entrambi i numeri devono essere interi!')
    else:
        #qua va tutto bene
        return x + y
```

[8]:
```
somma_interi(3.5, 4)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-8-9c73290ad20a> in <module>
----> 1 somma_interi(3.5, 4)

<ipython-input-7-db9e8a2f103c> in somma_interi(x, y)
      5     if (type(x)!= int) or (type(y)!=int):
      6         #invoca l'eccezione con un messaggio personalizzato
----> 7         raise TypeError('entrambi i numeri devono essere interi!')
      8     else:
      9         #qua va tutto bene
```

```
TypeError: entrambi i numeri devono essere interi!
```

Come possiamo notare `raise` impedisce la corretta interpretazione della funzione poiché abbiamo creato un eccezione di tipo TypeError **con un messaggio personalizzato in maniera tale da specificare la motivazione per cui è stato creato questo errore** questo può risultare utile soprattutto per evitare che queste funzioni abbiano un comportamento indesiderato, ovviamente in genere si possono inserire molte altre eccezioni di natura specifica o generale in modo da evitare lo scorretto uso di funzioni. Ovviamente di solito il comando `raise` viene usato con un blocco try e catch. Qualora invece noi mettessimo i valori desiderati in tal caso la funzione non verra interpretata senza alcun tipo di problema,

[9]:
```
somma_interi(3,4)
```

[9]: 7

### 1.4 Finally

`Finally` è invece un comando che si usa per definire un blocco di codice che viene eseguito **sempre dopo un blocco try, catch e verrà eseguito sempre dopo di esso anche qualora sia presente un eccezione**.

[10]:
```
try:
    1/0
except ZeroDivisionError as e:
    print('Un errore è sorto:',e)
finally:
    #questo blocco verrà sempre interpretato
    #indipendentemente dal fatto che ci sia un eccezzione o no
    print('Finito il blocco try catch')
```

```
Un errore è sorto: division by zero
Finito il blocco try catch
```

[11]:
```
try:
    1/1 #non crea un eccezione
except ZeroDivisionError as e:
    print(e)
finally:
    #questo blocco verrà sempre interpretato
    #indipendentemente dal fatto che ci sia un eccezzione o no
    print('Finito il blocco try catch')
```

```
Finito il blocco try catch
```

---

COMPLIMENTI AVETE COMPLETATO LA LEZIONE SUGLI ERRORI ED ECCEZIONI!