[FIM] FONDAMENTI DI INFORMATICA
per medicina e chirurgia high tech

P07: Eccezioni

Dott. Giorgio De Magistris

demagistris@diag.uniroma1.it

Corso di Laurea in Medicina e Chirurgia High Tech

Sapienza
Università di Roma

I3S Facoltà di Ingegneria dell'Informazione, Informatica e Statistica

DIAG Dipartimento di Ingegneria Informatica, Automatica e Gestionale

Errori

- In un programma possiamo distinguere due tipi di errori

Errori di sintassi

Vengono identificati dal parser PRIMA che il codice venga eseguito. Alcuni errori di sintassi comuni sono:

- Indentazione sbagliata
- Mismatch tra parentesi aperte e chiuse
- Dimenticarsi i due punti

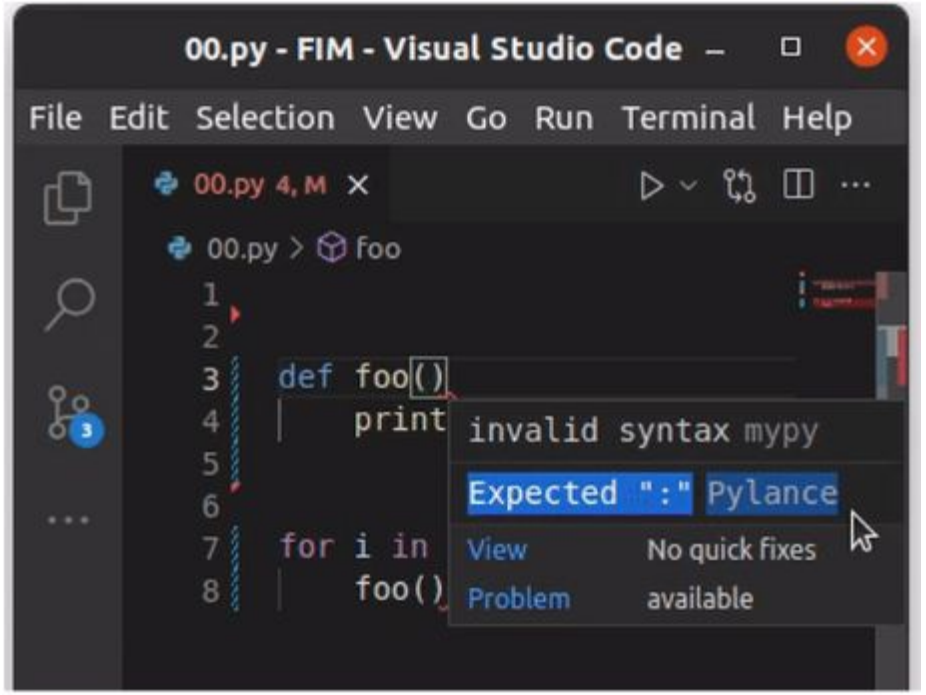
Eccezioni

Errori che si verificano DURANTE l'esecuzione del programma. Alcuni esempi sono:

- Chiamare una funzione non definita
- Riferimento a variabile non dichiarata
- Divisione per zero
- Index out of range

Errori di Sintassi

- Gli errori di sintassi vengono evidenziati direttamente dall' IDE
- Se eseguiamo uno script con errori di sintassi il parser visualizza la dicitura SyntaxError insieme al numero della riga dell'errore
- Gli errori di sintassi vanno corretti per poter eseguire il programma

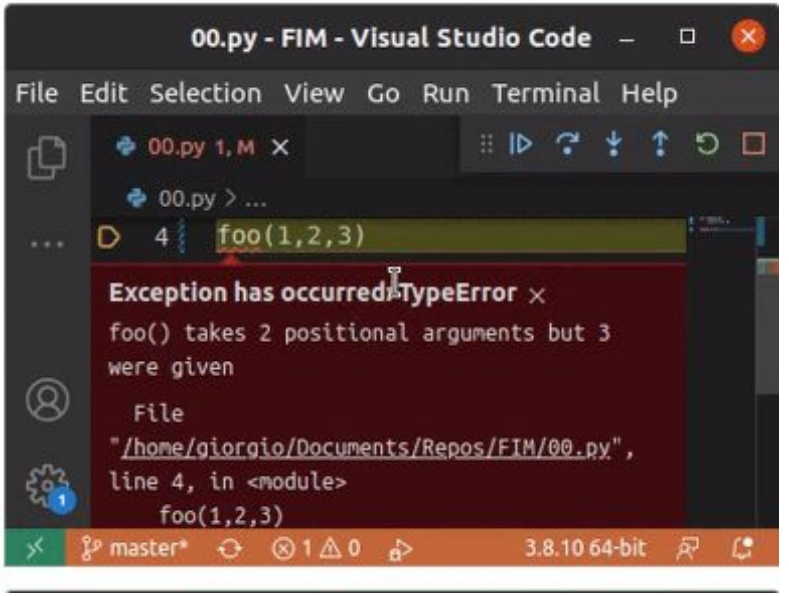


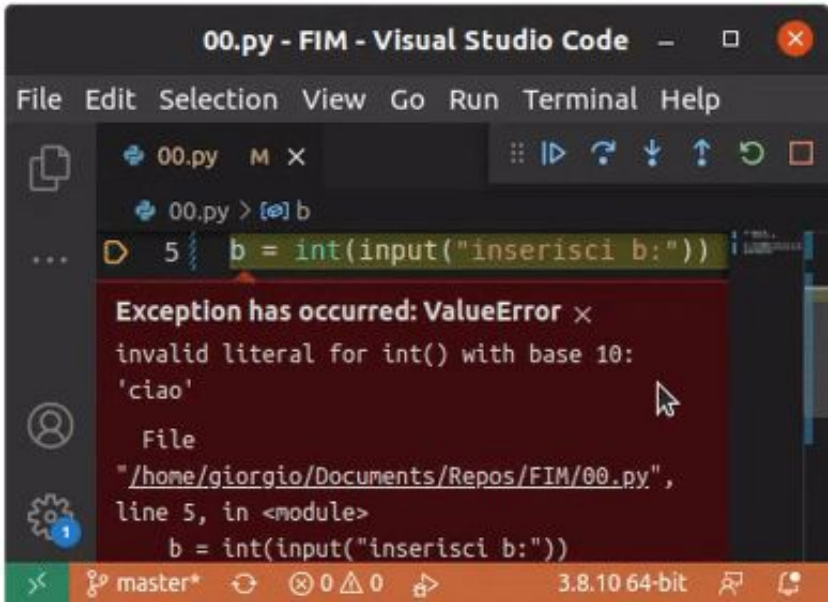

```
  File "/home/giorgio/Documents/Repos/FIM/00.py", line 3
    def foo()
            ^
SyntaxError: invalid syntax
```

Eccezioni

- Quando viene generata un'eccezione il programma termina stampando:
  - Il nome dell'eccezione che ha causato la terminazione del programma
  - Un'eventuale descrizione
  - Lo stack trace, ovvero l'intero call stack, in cui l'ultima funzione è quella più recente
  - La riga che ha generato l'eccezione

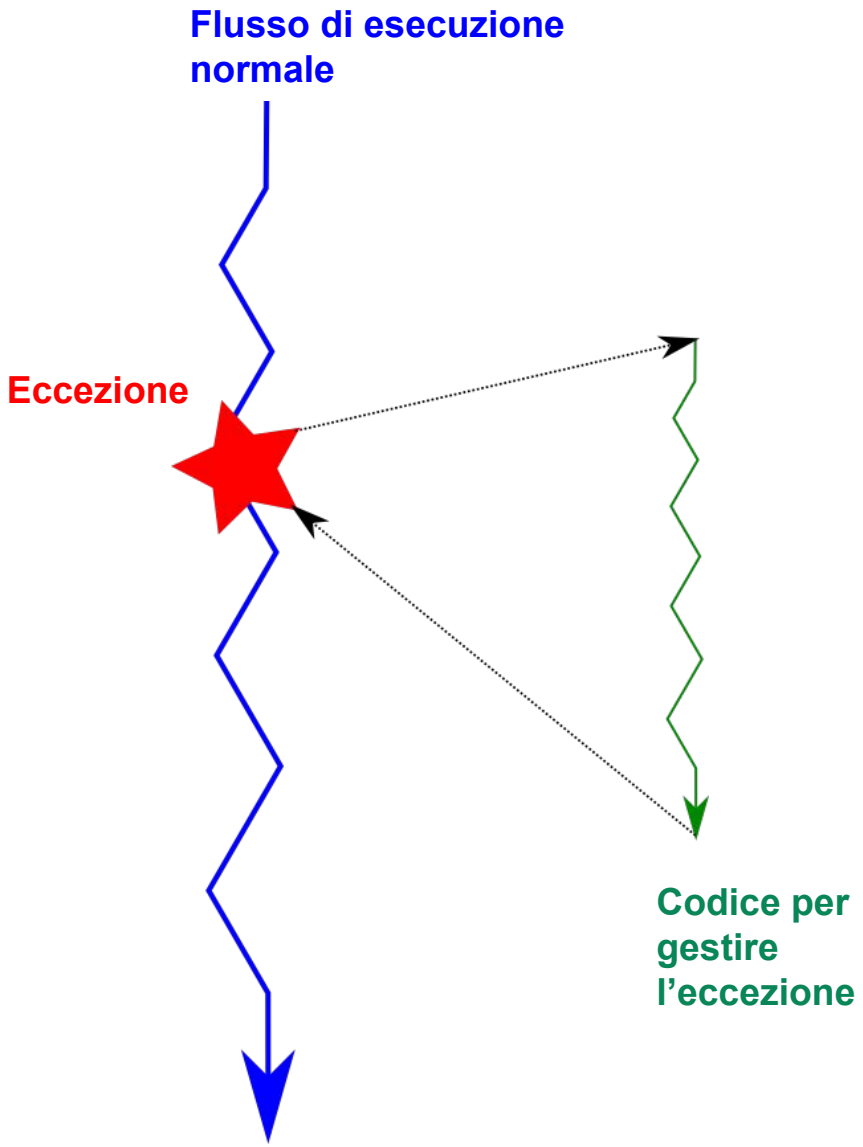
- I due esempi in figura sono molto diversi tra loro, provate a spiegare perchè . . .





Gestire le Eccezioni

- Le eccezioni si possono gestire
- Possiamo pensare al programma come ad un flusso di istruzioni
- Quando si verifica un'eccezione automaticamente si passa dal flusso di esecuzione normale al codice per la gestione delle eccezioni
- Dopo che l'eccezione è stata gestita il programma riprende il flusso normale dell'esecuzione



Gestire le Eccezioni

```
while True:
    try:
        a = int(input("Inserisci un intero: "))
        break
    except ValueError:
        print("Input non valido, riprova")
print(f"Hai inserito: {a}")
```

Il codice che può generare eccezioni va racchiuso nella clausola **try**

Il codice per gestire l'eccezione va racchiuso nella clausola **except**, seguita dal **nome dell'eccezione** che si vuole gestire. Una clausola try può essere seguita da più clausole except (se si vogliono gestire diverse eccezioni). Nel caso di più clausole catch, la prima clausola che corrisponde all'eccezione da gestire viene eseguita. Se nessuna clausola corrisponde all'eccezione generata il programma termina con un messaggio di errore (come nel caso di eccezione non gestita)

Gestire le Eccezioni

- Cosa c'è di sbagliato in questo programma ?

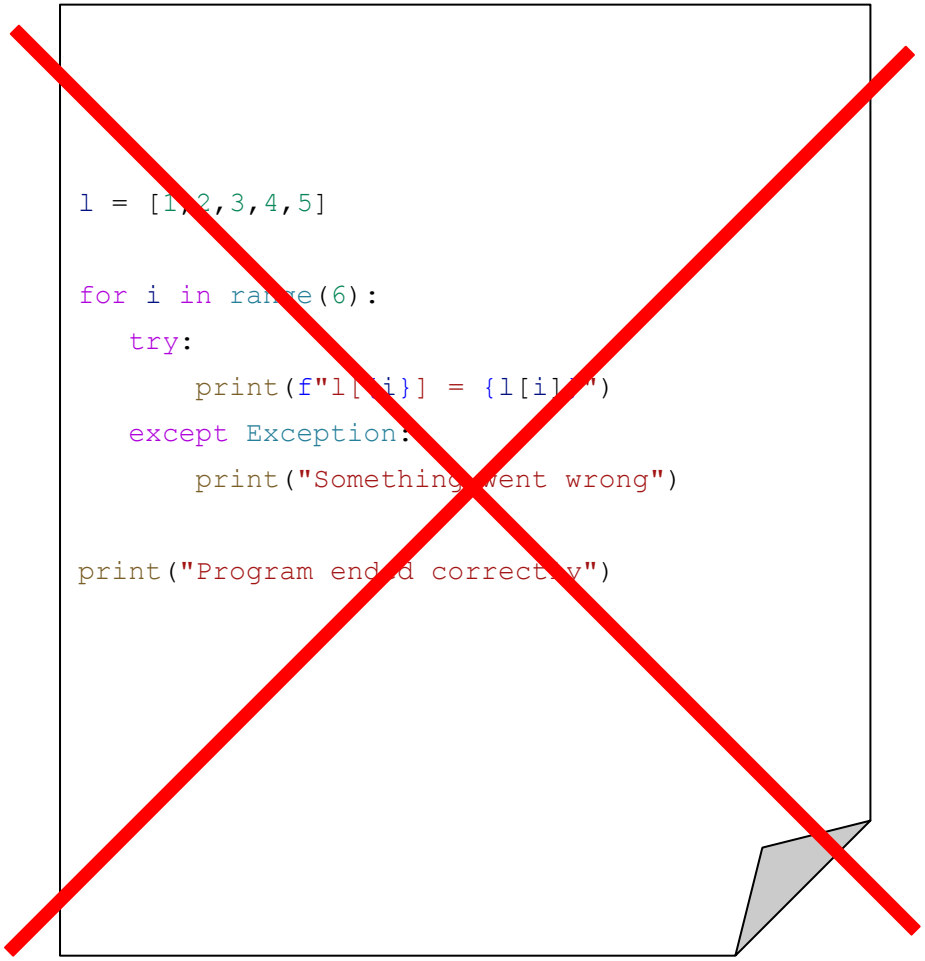
```
l = [1,2,3,4,5]

for i in range(6):
    try:
        print(f"l[{i}] = {l[i]}")
    except Exception:
        print("Something went wrong")

print("Program ended correctly")
```

Gestire le Eccezioni

- Cosa c'è di sbagliato in questo programma ?
- Non bisogna gestire le eccezioni che derivano da errori nel codice (bug), ma piuttosto correggere gli errori
- Bisogna gestire le eccezioni che potrebbero causare l'arresto anomalo del programma a causa di un imprevisto, ad esempio per la gestione dell'input, un file non trovato, errore di connessione, etc



```
l = [1,2,3,4,5]

for i in range(6):
    try:
        print(f"l[{i}] = {l[i]}")
    except Exception:
        print("Something went wrong")

print("Program ended correctly")
```

Lanciare Eccezioni

- Le eccezioni non derivano necessariamente da eventi che si manifestano durante l'esecuzione del codice (ad esempio per accesso a file inesistente o divisione per zero)
- Ma possono essere lanciate utilizzando l'istruzione raise
- Per esempio posso lanciare un'eccezione se una funzione di cui sono l'autore viene chiamata con i parametri sbagliati

```
def foo(l:List):
    if type(l) != list:
        raise ValueError(\
            "function foo takes a list argument, not {}"\
            .format(type(l)))
    for e in l:
        print(e)

foo([1,2,3])
foo(2)
```

Lanciare Eccezioni

- Le eccezioni non derivano necessariamente da eventi che si manifestano durante l'esecuzione del codice (ad esempio per accesso a file inesistente o divisione per zero)
- Ma possono essere lanciate utilizzando l'istruzione raise
- Per esempio posso lanciare un'eccezione se una funzione di cui sono l'autore viene chiamata con i parametri sbagliati

```
def foo(l:List):
    if type(l) != list:
        raise ValueError(\
            "function foo takes a list argument, not {}"\
            .format(type(l)))
    for e in l:
        print(e)

foo([1,2,3])
foo(2)
```

Exception has occurred: ValueError ×

```
function foo takes a list argument, not <class 'int'>
  File "/home/giorgio/Documents/Repos/FIM/03_exceptions/ex03.py",
line 8, in foo
    raise ValueError(\
  File "/home/giorgio/Documents/Repos/FIM/03_exceptions/ex03.py",
line 15, in <module>
    foo(2)
```