

Programmazione avanzata a.a. 2021-22  
A. De Bonis

**Introduzione a Python**  
**VII lezione**

66

## Iteratori

- Se una classe supporta l'iteratore possiamo ottenere un riferimento ad esso tramite la funzione `iter()`
  - Si invoca `iter` passando come argomento un'istanza della classe
- Per ottenere il prossimo elemento nella classe invochiamo `next()` sull'iteratore ottenuto
- Viene lanciata un'eccezione quando non ci sono più elementi nell'istanza della classe

Programmazione Avanzata a.a. 2021-22  
A. De Bonis

67

67

```

lista = [85,23,59]
print(lista)
it = iter(lista)
print(it)
print(next(it))
print(next(it))
print(next(it))

```

→

```

[89, 23, 59]
<list_iterator object at 0x10217aa58>
89
23
59

```

```

lista = [59, 42, 90]
print(lista)
it = iter(lista)
print(it)
print(next(it))
print(next(it))
print(next(it))
print(next(it))

```

→

```

[59, 42, 90]
<list_iterator object at 0x10ad62908>
59
42
90
Traceback (most recent call last):
  File "/Users/adb/Documents/r.py", line 8, in
<module>
    print(next(it))
StopIteration

```

← **eccezione**

Programmazione Avanzata a.a. 2021-22  
A. De Bonis

68

68

## Gestire l'eccezione

```

lista=[59, 42, 90]
print(lista)
it = iter(lista)
print(it)

while True:
  try:
    print(next(it))
  except Exception as e:
    break

```

Programmazione Avanzata a.a. 2021-22  
A. De Bonis

69

69

## Gestire l'eccezione

- Ad una clausola try possono essere associate piu` clausole except.
- Una clausola except puo` includere piu` eccezioni (all'interno di una tupla)
  - esempio: ... except (RuntimeError, TypeError, NameError):  
... pass
- Una clausola except che include un'eccezione di un certo tipo A puo` gestire solo eccezioni di tipo A o di sottoclassi di A.
- L'ultima clausola except puo` omettere i nomi delle eccezioni. Questo uso di except deve essere fatto con molta cautela perche' potrebbe nascondere un vero errore di programmazione. Puo` essere usato per stampare un messaggio di errore e lanciare nuovamente l'eccezione .
- Lo statement `try ... except` ha una clausola `else` opzionale che, quando presente, deve seguire tutte le clausole `except`. È utile per inserire codice che deve essere eseguito quando la clausola `try` non lancia un' eccezione.
- Nello statement `try` puo` contenere una clausola `finally` che viene eseguita immediatamente prima che venga completato lo statement `try`. Questa clausola viene eseguita sia nel caso in cui il `try` produca un'eccezione sia nel caso non venga prodotta un'eccezione. (vedere esempi nella documentazione)

Programmazione Avanzata a.a. 2021-22  
A. De Bonis

70

70

## Gestire l'eccezione

file `exampleExceptions1.py`

```
import sys

try:
    f = open(sys.argv[2], sys.argv[1])
    print(sys.argv[2], 'has', len(f.readlines()), 'lines')
    f.close()
except OSError:
    print('cannot open', sys.argv[2])
except:
    print('errore inatteso')
    raise #viene rilanciata l'eccezione prodotta nel try
```

Se non viene lanciata l'eccezione `OSError`, ogni altro tipo di eccezione viene catturare dalla seconda clausola `except`

eseguiamo lo script con `python exampleExceptions1.py r filename`

se il file `filename` non esiste

cannot open filename

se il file `filename` esiste e contiene tre linee

filename has 3 lines

Programmazione Avanzata a.a. 2021-22  
A. De Bonis

71

71

## Gestire l'eccezione

file exampleExceptions1.py

```
import sys

try:
    f = open(sys.argv[2], sys.argv[1])
    print(sys.argv[2], 'has', len(f.readlines()), 'lines')
    f.close()
except OSError:
    print('cannot open', sys.argv[2])
except:
    print('errore inatteso')
raise #viene rilanciata l'eccezione prodotta nel try
```

Se non viene lanciata l'eccezione OSError, ogni altro tipo di eccezione viene catturare dalla seconda clausola except

eseguiamo lo script con **python exampleExceptions1.py y filename**

```
errore inatteso
Traceback (most recent call last):
  File "exampleExceptions1.py", line 5, in <module>
    f = open(arg, sys.argv[1])
ValueError: invalid mode: 'y'
```

Programmazione Avanzata a.a. 2021-22  
A. De Bonis

72

72

## Gestire l'eccezione

file exampleExceptions2.py

```
import sys

for arg in sys.argv[1:]:
    try:
        f = open(arg, 'r')
    except OSError:
        print('cannot open', arg)
    else:
        print(arg, 'has', len(f.readlines()), 'lines')
        f.close()
```

- In questo esempio (dalla documentazione) la modalità di apertura del file è fissata nel codice e sulla linea di comando vengono passati i nomi dei file.
- Se non viene lanciata l'eccezione OSError, vuol dire che il file è aperto in lettura e nella clausola else viene stampato il numero di linee del file

eseguiamo lo script con **python exampleExceptions2.py file1 file2 file3**

file1 e file3 non esistono; file2 esiste e contiene 4 linee

```
cannot open file1
file2 has 4 lines
cannot open file3
```

Programmazione Avanzata a.a. 2021-22  
A. De Bonis

73

73

## Lanciare un'eccezione

- Quando si lancia un'eccezione con `raise`, l'eccezione può essere creata invocando il costruttore della classe con o senza argomenti. Se non si specificano degli argomenti allora si può usare semplicemente il nome della classe (senza le parentesi tonde).
- Gli argomenti vengono memorizzati nella variabile `args` dell'eccezione istanziata
- In una clausola `except` il nome dell'eccezione può essere seguita da una variabile. In questo caso la variabile si lega all'istanza dell'eccezione con gli argomenti dell'eccezione memorizzati in `.args`. L'istanza dell'eccezione definisce `__str__()` in modo che gli argomenti possano essere stampati direttamente senza dover fare riferimento ad `.args`.
- Esempio presente nella documentazione:

```
>>> try:
...     raise Exception('spam', 'eggs')
... except Exception as inst:
...     print(type(inst))    # the exception instance
...     print(inst.args)    # arguments stored in .args
...     print(inst)        # __str__ allows args to be printed directly,
...                         # but may be overridden in exception subclasses
...     x, y = inst.args    # unpack args
...     print('x =', x)
...     print('y =', y)
...
<class 'Exception'>
('spam', 'eggs')
('spam', 'eggs')
x = spam
y = eggs
```

A. De Bonis

74

74

## Generatori

- Una funzione generatore è un modo semplice ed immediato per creare un iteratore (detto generatore)
  - I metodi del generatore `__iter__()` e `__next__()` sono creati automaticamente
  - `__iter__()`: restituisce l'iteratore stesso
  - `__next__()`: viene utilizzato nei cicli per ottenere il prossimo elemento
- La sintassi per definire una funzione generatore è simile a quella usata per definire una funzione, ma al posto di `return` si usa `yield`
- Quando si incontra un `yield` l'esecuzione del generatore è sospesa, viene restituito il valore indicato da `yield`
- `next()` comincia l'esecuzione di un generatore o la riprende dall'ultima espressione `yield` eseguita ripartendo da dove l'esecuzione era stata sospesa

Programmazione Avanzata a.a. 2021-22  
A. De Bonis

75

75

## Generatori

```
def gen_range(n):
    k=0
    while k<n:
        yield k
        k += 1
```

```
ite=gen_range(10)
while(True):
    try:
        i=next(ite)
        print(i, end=' ')
    except Exception as e:
        break
```

```
for i in gen_range(10):
    print(i, end=' ')
```

0 1 2 3 4 5 6 7 8 9

0 1 2 3 4 5 6 7 8 9

Si tratta di una sorta di funzione che genera una sequenza di valori restituiti uno per volta tramite **yield**

Nel generatore non possono coesistere **yield** e **return**


Programmazione Avanzata a.a. 2021-22  
A. De Bonis 76

76

## Generatori

```
def reverse(data):
    for index in range(len(data)-1, -1, -1):
        yield data[index]

for char in reverse('programmazione'):
    print(char, end="")
```



enoizammargorp

Programmazione Avanzata a.a. 2021-22  
A. De Bonis 77

77

## qualche dettaglio in piu` su iter()

- `iter()` restituisce un oggetto iteratore e puo` prendere in input uno o due argomenti
- In assenza del secondo argomento, il primo argomento deve essere un collezione che supporta l'iterazione. Nel caso la collezione non supporti il protocollo dell'iterazione viene lanciato `TypeError`.
- Se viene fornito anche il secondo argomento allora il primo argomento deve essere un callable. In questo caso l'iteratore restituito da `iter()` invoca il callable ogni volta che viene invocato il suo metodo `__next__()`. Il secondo argomento svolge il ruolo di sentinella e se `next()` restituisce un valore uguale a questo argomento viene lanciata l'eccezione `StopIteration`.