

Excepcions

Tecnologia de la Programació

Sebastià Vila-Marta

Enginyeria de Sistemes TIC
Universitat Politècnica de Catalunya
<http://itic.cat>

17 de març de 2013

- 1 En el tema anterior...
- 2 Errors en temps d'execució
- 3 Excepcions
- 4 Excepcions com a objectes
- 5 Transmissió d'excepcions
- 6 Per a la setmana vinent ...

- Hi ha objectes iguals i idèntics. Podem definir l'igualtat emprant mètodes especials.
- L'herència permet modelar relacions «és un». Les subclasses hereten atributs i mètodes de forma automàtica.
- Els mètodes heretats es poden redefinir.
- La delegació és una estratègia que permet a una classe cedir part dels càlculs a les subclasses.

Els programes... peten!

Fixem-nos en el següents programes:

```
x = 0
print 45/x

f = open('fitxer-inexistent.txt', 'r')
print f.read()
f.close()
```

Ambdós programes possiblement avortaran en executar-se atès que contenen errors. La seva sortida per pantalla seria del tipus:

```
Traceback (most recent call last):
  File "p.py", line 2, in <module>
    print 45/x
```

ZeroDivisionError: integer division **or** modulo by zero

És important aprendre a llegir els volcats de pila (**traceback**).
Observeu:

- Quina és la causa de l'error?
- On s'ha produït?
- En quin moment de l'execució?

Observem un programa i el resultat d'executar-lo:

```
def f1(x,y):  
    return x/y
```

```
def f2(b):  
    v = f1(b,0)  
    return v
```

```
if __name__ == "__main__":  
    print f2(33)
```

```
Traceback (most recent call last):  
  File "p.py", line 10, in <module>
```

```
    print f2(33)
```

```
  File "p.py", line 6, in f2
```

```
    v = f1(b,0)
```

```
  File "p.py", line 2, in f1
```

```
    return x/y
```

```
ZeroDivisionError: integer division or modulo by zero
```

- Quina és la causa de l'error?
- On s'ha produït?
- En quin moment de l'execució?

No tots els errors són iguals

Hi ha diversos tipus d'errors. Entre aquests:

Errors d'execució

Els que es produeixen durant l'execució d'un programa.

- Es produeixen a l'executar-se un programa i poden manifestar-se o no segons les condicions en què s'executa: valors de les dades, disponibilitat de recursos, etc.
- Habitualment acaben avortant l'execució i presentant un volcat de pila.
- Poden ser:
 - Recuperables: per exemple si es produeixen per que un usuari ha escrit malament una dada en un formulari.
 - Irrecuperables: per exemple si la causa és un malfuncionament del disc.

Tractament d'excepcions

Els errors en temps d'execució són inacceptables per a moltes aplicacions. Particularment per als **sistemes encastrats**.

Alguns llenguatges ofereixen eines de **tractament d'excepcions** per reconduir les coses quan es produeix un error d'aquest tipus. Python és un d'ells.

Excepció

- Circumstància singular que es produeix durant l'execució del programa.
- Instància de la classe **Exception** que representa una excepció.

Cada vegada que es produeix una excepció durant l'execució:

- Es crea una instància d'**Exception**.
- Es modifica el flux d'execució usual del programa amb l'objectiu de buscar un «recollidor» (**catcher**)

El recollidor d' excepcions

La sentència usada per capturar aquesta excepció és **try**. Té la següent sintaxi:

```
try:
    # sentencies a supervisar
except:
    # sentencies que s'executen si les supervisades aixequen una excepcio
else:
    # sentencies a executar si les supervisades no aixequen cap excepcio
finally:
    # sentencies a executar despres passi el que passi
```

Per exemple, el següent programa:

```
try:
    x = 0
    y = 45/0
except:
    print "Error al canto!"
else:
    print y
finally:
    print "S'imprimeix igualment"
```

Escriuria la frase "Error al canto!".

Les excepcions són objectes

- Les excepcions són objectes.
- Existeix una jerarquia de classes d'excepcions. Això permet especialitzar les excepcions i tenir un tipus d'excepció per a cada situació rellevant.

```
BaseException
```

```
+-- SystemExit
```

```
+-- KeyboardInterrupt
```

```
+-- Exception
```

```
  +-- StandardError
```

```
    |   +-- BufferError
```

```
    |   +-- ArithmeticError
```

```
    |   |   +-- ZeroDivisionError
```

```
    |   +-- AssertionError
```

```
    |   +-- AttributeError
```

```
    |   +-- EnvironmentError
```

```
    |   |   +-- IOError
```

```
    |   +-- LookupError
```

```
    |   |   +-- KeyError
```

```
    |   +-- MemoryError
```

```
    |   +-- RuntimeError
```

```
    |   |   +-- NotImplementedError
```

Podem triar les excepcions que recollim

Si només volem recollir les excepcions de la classe **IOError** podem fer-ho així:

try:

```
f = open('fitxer', 'r')
```

except IOError:

```
    print "Fitxer inexistent"
```

- En cas que el fitxer no existís, s'aixecaria una excepcio de tipus **IOError**. La sentència **except** només capturaria aquestes excepcions.
- És preferible indicar, com en aquest exemple, les excepcions que es volen capturar.
- Es pot usar més d'una sentència **except** per poder diferenciar diversos tipus d'excepcions i actuar consegüentment.
- Quan indiquem que volem recollir excepcions de la classe X, es recullen també les de les subclasses d'X.

Es poden aixecar excepcions quan cal

Un programa pot llençar una excepció a través de la sentència **raise**. Això permet gestionar amb el mecanisme d'excepcions situacions que el propi programa detecta. Observeu la següent classe:

```
class Prova(object):  
    def __init__(self):  
        self.l = []  
  
    def afegeix(self, e):  
        self.l.append(e)  
  
    def total(self):  
        if self.l:  
            return sum(self.l)  
        else  
            raise Exception()
```

- Quan se li demana a un objecte de classe Prova que calculi el total i no conté elements ... contesta aixecant una excepció.

Definició de noves excepcions... I

Sovint és convenient definir excepcions específiques pel nostre mòdul, classe o aplicació. Això permet distingir-les de les excepcions pre-definides.

Com les excepcions són classes, simplement cal derivar una nova classe:

```
class SenseDades(Exception):  
    pass
```

Simplement amb això hem definit una nova classe d'excepcions. Ara la podem usar en el nostre exemple fent:

```
class Prova(object):  
    def __init__(self):  
        self.l = []  
  
    def afegeix(self, e):  
        self.l.append(e)  
  
    def total(self):  
        if self.l:  
            return sum(self.l)  
        else  
            raise SenseDades()
```

Definició de noves excepcions... II

Un programa que la volgués capturar podria fer el següent:

```
m = Prova()
```

```
try:
```

```
    s = m.total()
```

```
except SenseDades:
```

```
    print "No hi ha dades per totalitzar"
```

Si ens cal, podem associar dades (atributs) a una excepció. Observeu el següent exemple:

```
class Prova(object):
```

```
    def __init__(self):
```

```
        self.l = []
```

```
    def afageix(self, e):
```

```
        self.l.append(e)
```

```
    def total(self):
```

```
        if self.l:
```

```
            return sum(self.l)
```

```
        else
```

```
            raise SenseDades('Objecte prova buit')
```

```
m = Prova()
```

```
try:
```

```
    s = m.total()
```

```
except SenseDades as e:
```

```
    print e
```

Les excepcions suren I

- Quan es produeix una excepció i no hi ha cap sentència que la reculli, es transmet l'excepció cap el següent nivell de la pila de crides.
- Si finalment l'excepció arriba al programa principal, actua el recollidor per omissió, que avorta l'execució i escriu el volcat de pila que hem après a llegir anteriorment.
- Per exemple, si executem aquest programa:

```
m = Prova()  
print m.total()
```

La sortida seria:

```
Traceback (most recent call last):  
  File "a.py", line 19, in <module>  
    m.total()  
  File "a.py", line 16, in total  
    raise SenseDades('Objecte prova buit')  
__main__.SenseDades: Objecte prova buit
```

Les excepcions suren II

- Vegem un cas més complicat:

```
def f1(x,y):  
    return x/y  
  
def f2(b):  
    try:  
        v = f1(b,0)  
    except ZeroDivisionError:  
        return 0  
    else:  
        return v  
  
if __name__ == "__main__":  
    print f2(33)
```

- 1 `__main__` crida a `f2(33)` que crida a `f1(33,0)`.
- 2 `f1` fa una divisió per zero que aixeca una excepció de classe **ZeroDivisionError**.
- 3 No hi ha recollidor i per tant l'excepció sura. Ara l'excepció la té la crida `v = f1(b,0)`.
- 4 Com hi ha recollidor, es recull. `f2(33)` retorna 0.
- 5 `__main__` escriu 0.

- 1 Estudi de la teoria. A partir de les transparències i les notes que heu pres. Inclou provar els conceptes en el computador.
- 2 Incrementar el xuletari i el glossari del tema.
- 3 Solució del problema especial, que té com objectiu aconseguir la solució més senzilla i entenedora possible.