2. Excepciones

Programación II

Grado en Inteligencia Artificial

M. Alonso, M. Cabrero y E. Hernández

fic facultade de informática da coruña

UNIVERSIDADE DA CORUÑA

Contenidos

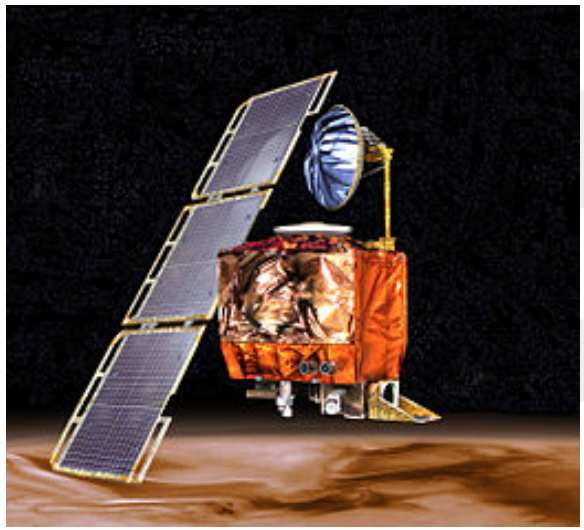
Introducción

¿Qué es un error?

- "Se ha debido a un error informático" es una frase muy usada, a veces para encubrir verdaderos errores humanos.
- El ordenador cumple el cometido para el que se le ha programado.
- La Ingeniería del Software no es una ciencia exacta:
  - El programador más experimentado comete errores (*bugs*).
  - No se respetan las especificaciones iniciales.
  - No se consideran ciertas situaciones improbables pero no imposibles.
  - No se hacen pruebas exhaustivas durante el ciclo de desarrollo.
- Es necesario que el programa se anticipe y sea capaz de manejar situaciones excepcionales.

Errores informáticos famosos

- Mars Climate Orbiter iba a fotografiar Marte pero no envió ni una sola foto porque el sistema de control de la nave esperaba medidas en el Sistema Métrico Decimal (kms.) pero desde la Tierra se enviaron en Sistema Métrico Anglosajón (millas).



Errores informáticos famosos

- En la Guerra del Golfo de 1991, un misil iraquí destruye la base de Dhahran en Arabia Saudí porque el software del sistema antimisiles redondeaba los valores del reloj digital, lo cual produjo un retraso acumulado al cabo del tiempo.
- La consecuencia es que al responder unas milésimas de segundo más tarde el misil ya no estaba en el punto calculado y no fue posible destruirlo.

Errores informáticos famosos

- El famoso baile del caballo del rapero PSY rompió el contador de YouTube cuando alcanzó un total de $2,147,483,647$ reproducciones.
- La causa estuvo en que el contador usaba arquitectura de 32 bit.

Errores informáticos famosos

- En el año 2012, la firma de inversión *Knight Capital Group* perdió 440 millones de dólares (10 millones por minuto) por un error en su software de inversiones financieras.
- El problema radicó en que realizó transacciones sin controlar que estaban generando pérdidas en lugar de beneficios.

Errores informáticos famosos

- La empresa Boeing detectó un grave error de overflow de enteros que podría apagar todos los generadores eléctricos si el avión hubiera estado encendido durante más de 248 días.

Errores informáticos famosos

- `https://en.wikipedia.org/wiki/List_of_software_bugs`
- ...y los que no se conocen.

Tipos de errores

Errores de sintaxis

- Detectados por el intérprete de Python con un mensaje `SintaxError`.

```
>>> while True print('Hola mundo!')
  File "<stdin>", line 1
    while True print('Hola mundo!')
                    ^
SyntaxError: invalid syntax
```

Errores de semántica

- Detectados a posteriori (no por el intérprete).
- El programa no produce el resultado esperado (p.ej. en la línea 7 se escribe `i` en lugar de `j` con lo que se suma 3 veces el primer elemento de la lista).

```
1  # error semántico
2  lista = [1, 2, 3]
3  i = 0
4  suma = 0
5
6  for j in range(len(lista)):
7      suma += lista[i]
8  print(suma)
```

Errores de ejecución

- El programa para bruscamente su ejecución debido a:
  - Uso incorrecto del programa por parte del usuario (p.ej. ingresa una cadena y se espera un número).
  - Bugs del programador (p.ej. acceder a la quinta posición de una lista de tres elementos).
  - Fallan los recursos externos al programa (p.e. se intenta leer un archivo y se encuentra dañado).

```
>>> lista = [1, 2, 3]
>>> print(lista[4])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>>
```

Excepciones

Definición

- Al producirse un error de ejecución, el intérprete de Python lanza una *excepción*.
  - Es un evento que interrumpe el flujo de ejecución del programa.
  - Notifica el tipo de error ocurrido.
  - Puede ser detectada y "atrapada" a través del propio programa para manejarla de forma elegante.
  - Si no se atrapa el programa termina.
- En ocasiones, es el propio programa quien explícitamente lanza una excepción.

Papel de las excepciones

- *Manejo de errores*: Un error lanza la excepción y, se ignora y el programa para, o se captura.
- *Notificación de eventos*: Señalizar condiciones inválidas sin tener que devolver *flags* de resultado.
- *Manejo de casos especiales*: En lugar de incluir código para testear situaciones improbables comprobar si se cumplen la condiciones (sentencia `assert`).
- *Acciones de terminación*: En caso de producirse un error, ¿qué acciones es necesario ejecutar? P.e. cerrar un fichero abierto, no actualizar la base de datos.

Capturando excepciones

“Cazando” excepciones

```
>>> valor = int(input('Introducir un valor entero: '))
Introducir un valor entero: 4x
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '4x'
```

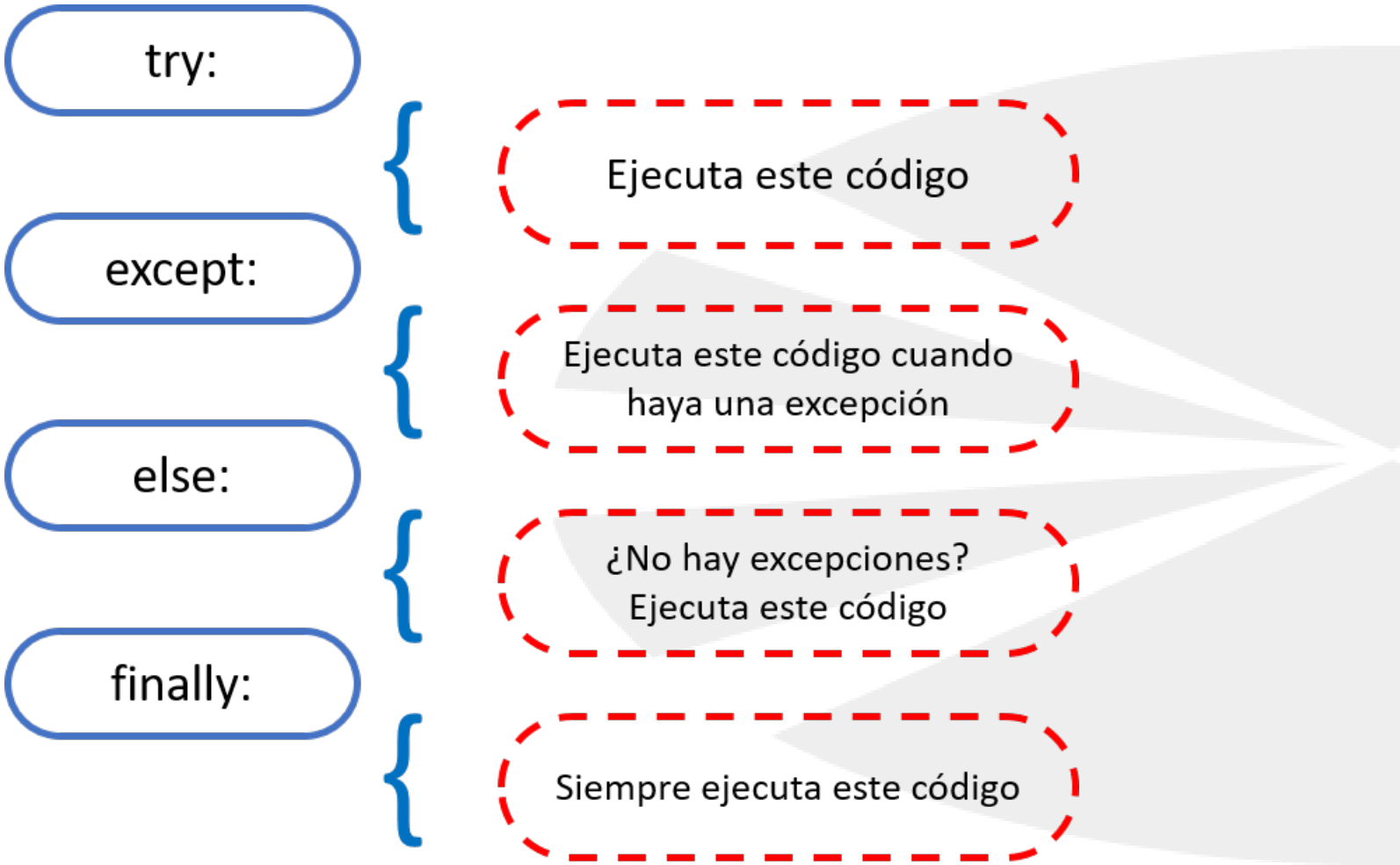
- El usuario introduce 4x en lugar de un entero.
- No es posible convertir la cadena a un entero.
- Se lanza la excepción, se imprime el mensaje de error y fin del programa.

“Cazando” excepciones

```
while True:
    try:
        valor = int(input('Introducir un valor: '))
        break
    except ValueError:
        print('No es un valor entero!')
```

- Bloque `try`: Zona donde atrapar cualquier excepción que se produzca.
- Bloque `except`: Acción a ejecutar si se lanza una excepción del tipo dado dentro del bloque `try`.
- Si se lanza la excepción, el flujo de ejecución cambia: salta a la primera sentencia del bloque `except`.

Sintaxis de `try...except`



Sintaxis de try...except

```
try:
    # Operaciones van aquí
    ......................
except Exception1:
    # Si hay Exception1, ejecutar este bloque.
except Exception2:
    # Si hay Exception2, ejecutar este bloque.
    ......................
except (Exception3[, Exception4[,...Exception5]]]):
    # Si hay cualquier excepción de la lista
    # de excepciones ejecutar este bloque.
    ......................
else:
    # Si no hay excepción, ejecutar este bloque.
finally:
    # De cualquier manera ejecutar este bloque.
    # Sentencias de finalización (p.ej. cierre ficheros)
```

Ejemplo de try...except

```
try:
    fh = open("testfile", "a")
    fh.write("Fichero test para manejo de excepciones")
except IOError:
    print ("""Error: no encuentro el fichero
                  o no puedo leer los datos""")
else:
    print ("Escritura del fichero correcta")
    fh.close()
```

Si el fichero no tiene permisos de escritura:

```
Error: no encuentro el fichero o no puedo leer los datos
```

Ejemplo de try...except

```
while True:
    try:
        dividendo = int(input('Introducir dividendo: '))
        divisor = int(input('Introducir divisor: '))
        cociente = dividendo / divisor
    except (ValueError, ZeroDivisionError):
        print('Error: valor incorrecto')
    except:                         # Resto excepciones
        print('Error desconocido')
    else:
        print('El resultado es ', cociente)
        break
```

```
Introducir dividendo: t
Error: valor incorrecto
Introducir dividendo: 3
Introducir divisor: 0
Error: valor incorrecto
```

```
Introducir dividendo: 3
Introducir divisor: 'a'
Error: valor incorrecto
Introducir dividendo: 3
Introducir divisor: 4
El resultado es  0.75
```

Ejemplo de try...except

```
try:
    f = open("prueba.txt", "wt")
    print("Fichero abierto")
    f.write("prueba\n")
except PermissionError:
    print("Error: no hay permiso de escritura")
finally:
    if 'f' in globals() :
        f.close()
    print("Fichero cerrado")
```

Si no hay ningún error que impida la apertura del fichero, se crea el objeto `f` (manejador del fichero) y se cerrará después de ejecutar `print` y `write`.

Uso de claves en try...except

```
def convertir_temp(var):
    try:
        return int(var)
    except ValueError as argumento:
        print ('El argumento no es un número\n', argumento)

# Llamada a la función.
convertir_temp("xyz")
```

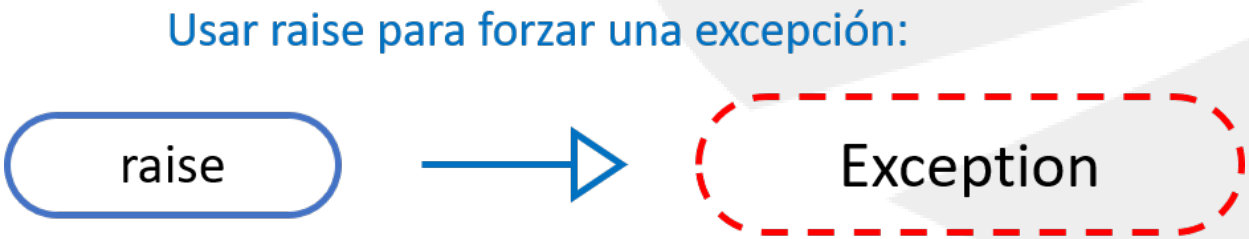
El *argumento* de la excepción da información adicional sobre el problema.

```
El argumento no es un número
 invalid literal for int() with base 10: 'xyz'
```

Lanzando excepciones

Sentencia `raise`

- Lanza una excepción si el programa detecta un error.
- Evita la finalización prematura del programa.
- `raise ClaseExcepcion(mensaje)`.
- Crea una instancia de la clase `ClaseExcepcion` con el mensaje como parámetro.



Ejemplo de raise

```
def minimo (arg1, arg2):
    if arg1 is None or arg2 is None:
        raise TypeError('Args. de minimo() no pueden ser None')
    if arg1 < arg2:
        return arg1
    else:
        return arg2
```

```
>>> y = minimo (15, None)
Traceback (most recent call last):
  File "prueba.py", line 9, in <module>
    y = minimo (x, None)
  File "prueba.py", line 3, in minimo
    raise TypeError('Argumentos no pueden ser None')
TypeError: Args. de minimo() no pueden ser None
```

Aserciones

Sentencia assert

- `assert expresion,[argumentos]`
- Se testea la expresión, y si es falsa, se lanza una excepción `AssertionError` a la cual se le pasan los argumentos.
- La excepción podrá entonces ser capturada.
- Su propósito es establecer restricciones definidas por el programador, no atrapar errores de programación (usar `raise`).
- Sirve para depurar el código. Si Python se ejecuta en modo optimizado (`__debug__` es `False`), se ignorarán los assert. Para ello `python -O script.py`

Afirmar (assert) que se cumple una condición

Ejemplo de assert

```
def KelvinToFahrenheit(Temperature):
    assert (Temperature >= 0),"Colder than absolute zero!"
    return ((Temperature-273)*1.8)+32

print(KelvinToFahrenheit(273))
print(KelvinToFahrenheit(-5))
```

```
32.0
Traceback (most recent call last):
  File "assert.py", line 6, in <module>
    print (KelvinToFahrenheit(-5))
  File "assert.py", line 2, in KelvinToFahrenheit
    assert (Temperature >= 0),"Colder than absolute zero!"
AssertionError: Colder than absolute zero!
```

Ejemplo de assert

```
1  def itemsPares(lista):
2      nuevaLista = []
3      for i in range(len(lista)):
4          if i % 2 == 0:
5              nuevaLista.append(lista[i])
6      assert len(nuevaLista) * 2 >= len(lista)
7      return nuevaLista
8
9  print(itemsPares([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]))
10 assert(itemsPares([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]) == [1, 3, 5, 7, 9])
```

```
[1, 3, 5, 7, 9]
```

Si por error la línea 3 se codifica como

```
for i in range(len(lista)-2):
```

```
Traceback (most recent call last):
  File "assert2.py", line 9, in <module>
    print(itemsPares([1, 2, 3, 4, 5, 6, 7, 8, 9, 10]))
  File "assert2.py", line 6, in itemsPares
    assert len(nuevaLista) * 2 >= len(lista)
AssertionError
```

Anexo

Algunos tipos de excepciones estándar

- `AssertionError`: Fallo de sentencia de aserción.
- `EOFError`: Condición de fin de fichero (EOF).
- `FloatingPointError`: Fallo en punto flotante.
- `IOError`: Errores de operaciones de E/S.
- `ImportError`: Fallo de importación de módulo.
- `IndexError`: Índice de secuencia fuera del rango.
- `KeyError`: La clave no se encuentra en el diccionario.
- `KeyboardInterrupt`: Interrupción (i.e. Control-C)
- `NameError`: No se encuentra un identificador.
- `ValueError`: Argumento de valor inapropiado.
- `ZeroDivisionError`: División por cero.

Todas derivan de la clase base `Exception`.

Ejemplos de excepciones estándar

```
>>> lst = [1, 2, 3]
>>> lst[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>> 2 + '3'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
>>> lsst[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'lsst' is not defined
>>> int('12.04')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '12.04'
```

Excepciones creadas por el usuario

```
class Error(Exception):
    """Base class for other exceptions"""
    pass

class ErrorValorNegativo(Error):
   """Raised when the input is negative"""
   pass

class ErrorValorMuyPequeno(Error):
    """Raised when the value is too small"""
    pass

class ErrorValorMuyGrande(Error):
    """Raised when the value is too large"""
    pass
```

Excepciones creadas por el usuario

```
# main program
# Takes input till the user inputs correct value

number = 11

while True:
    try:
        num = int(input("Número: "))
        if num < 0:
            raise ErrorValorNegativo
        elif num < number:
            raise ErrorValorMuyPequeno
        elif num > number:
            raise ErrorValorMuyGrande
        break
    except ErrorValorNegativo:
        print("Negativo")
    except ErrorValorMuyPequeno:
        print("Muy pequeño")
    except ErrorValorMuyGrande:
        print("Muy grande")
print("Valor correcto")
```

```
Número: -15
Valor negativo

Número: 0
Valor demasiado pequeño

Número: 15
Valor demasiado grande

Número: 11
Valor correcto
```

Bibliografía

- Michael T. Goodrich, Roberto Tamassia, and Michael H. Goldwasser. 2013. Data Structures and Algorithms in Python (1st. ed.). Wiley Publishing.
- Mark Lutz. 2003. Learning Python (2nd. ed.). O'Reilly & Associates, Inc., USA.
- `https://realpython.com/python-exceptions/`
- `https://www.tutorialspoint.com/python3/python_exceptions.htm`

2. Excepciones

Programación II

Grado en Inteligencia Artificial

M. Alonso, M. Cabrero y E. Hernández

fic facultade de informática da coruña

UNIVERSIDADE DA CORUÑA