



UNIVERSIDAD DE GUADALAJARA
Centro Universitario de los



Manejo de Excepciones

Programación avanzada

Ingeniería en electrónica y computación

Tarea #1

Entrega: 2021-03-19

Fernando Sánchez Plascencia

219341143

Luis David Olvera Aguilera

219341178

Juan Carlos Duran Zepeda

219893731

Luis Alberto Ruelas Barbosa

215537205

¿Qué son las excepciones?

Las Excepciones son una estructura de control que implementan ciertos lenguajes de programación con el objetivo de manejar condiciones “anormales” o no esperadas dentro del programa que normalmente impedirían el continuo desarrollo del programa, pero que pueden ser tratados en el mismo curso del programa.

Por ejemplo, un programa puede admitir cierto número de errores en el formato de los datos y continuar su proceso para producir el mejor resultado posible en lugar de producir una salida aparatosa llena de mensajes de error probablemente incomprensibles para el usuario. Muchas veces, la acción asociada a una excepción es simplemente producir un mensaje informativo y terminar; otras veces, es sólo indicación de la necesidad de un cambio en la estrategia de resolución del problema.

Manejo de excepciones

Para el manejo de excepciones los lenguajes proveen ciertas palabras reservadas, que nos permiten manejar las excepciones que puedan surgir y tomar acciones de recuperación para evitar la interrupción del programa o, al menos, para realizar algunas acciones adicionales antes de interrumpir el programa.

En el caso de Python, el manejo de excepciones se hace mediante los bloques que utilizan las sentencias ***try***, ***except*** y ***finally***.

*Dentro del bloque **try** se ubica todo el código que pueda llegar a levantar una excepción, se utiliza el término levantar para referirse a la acción de generar una excepción.*

exception LookupError

La clase base para las excepciones que se generan cuando una clave o índice utilizado en un mapa o secuencia que no es válido: IndexError, KeyError.

exception IndexError

Se genera cuando un subíndice de secuencia está fuera del rango. (Los índices de la rebanada son truncados silenciosamente para caer en el intervalo permitido; si un índice no es un entero, se genera TypeError.)

```
import sys
try:
    foo = [a, s, d, f, g]
    print foo[5]
except IndexError as e:
    print e
print sys.exc_type
```

Output

```
C:/Users/TutorialsPoint1~.py
list index out of range
<type 'exceptions.IndexError'>
```

exception TypeError

Se genera cuando una operación o función se aplica a un objeto de tipo inapropiado. El valor asociado es una cadena que proporciona detalles sobre la falta de coincidencia de tipos.

El código de usuario puede generar esta excepción para indicar que un intento de operación en un objeto no es compatible y no debe serlo. Si un objeto está destinado a soportar una operación dada pero aún no ha proporcionado una implementación, NotImplementedError es la excepción adecuada para generar.

Pasar argumentos del tipo incorrecto (por ejemplo, pasar a list cuando se espera un int) debería dar como resultado TypeError, pero pasar argumentos con el valor incorrecto (por ejemplo, un número fuera límites esperados) debería dar como resultado ValueError.

```
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't convert 'int' object to str implicitly
```

exception KeyError

Se genera cuando no se encuentra una clave de asignación (diccionario) en el conjunto de claves existentes (mapa).

```
>>> ages = {'Jim': 30, 'Pam': 28, 'Kevin': 33}
>>> ages['Michael']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Michael'
```


exception OSError([arg])/exception OSError(errno, strerror[, filename[, winerror[, filename2]]])

Esta excepción se produce cuando una función del sistema retorna un error relacionado con el sistema, que incluye fallas de E/S como file not found o disk full (no para tipos de argumentos ilegales u otros errores incidentales).

La segunda forma del constructor establece los atributos correspondientes, que se describen a continuación. Los atributos predeterminados son None si no se especifican. Para compatibilidad con versiones anteriores, si se pasan tres

argumentos, el atributo `args` contiene solo una tupla de 2 de los dos primeros argumentos del constructor.

El constructor a menudo retorna una subclase de `OSError`, como se describe en `OS exceptions` a continuación. La subclase particular depende del valor final `errno`. Este comportamiento solo ocurre cuando se construye `OSError` directamente o mediante un alias, y no se hereda al derivar.



```
# Importing os module
import os

# os.ttyname() method in Python is used to get the terminal
# device associated with the specified file descriptor.
# and raises an exception if the specified file descriptor
# is not associated with any terminal device.
print(os.ttyname(1))
```

Output :

```
OSError: [Errno 25] Inappropriate ioctl for device
```

- **Args**

La tupla de argumentos dada al constructor de excepción. Algunas excepciones predefinidas (como `OSError`) esperan un cierto número de argumentos y asignan un significado especial a los elementos de esta tupla, mientras que otras normalmente se llaman solo con una sola cadena que da un mensaje de error.

- **Errno**

Un código de error numérico de la variable C, `errno`.

- **Winerror**

En Windows, esto le proporciona el código de error nativo de Windows. El atributo `errno` es entonces una traducción aproximada, en términos POSIX, de ese código de error nativo.

En Windows, si el argumento del constructor `winerror` es un entero, el atributo `errno` se determina a partir del código de error de Windows y el argumento `errno` se ignora. En otras plataformas, el argumento `winerror` se ignora y el atributo `winerror` no existe.

- **Strerror**

El mensaje de error correspondiente, tal como lo proporciona el sistema operativo. Está formateado por las funciones de C, `perror()` en POSIX, y `FormatMessage()` en Windows.

- **filename/Filename2**

Para las excepciones que involucran una ruta del sistema de archivos (como `open()` o `os.unlink()`), `filename` es el nombre del archivo pasado a la función. Para las funciones que involucran dos rutas del sistema de archivos (como `os.rename()`), `filename2` corresponde al segundo nombre de archivo pasado a la función.

exception FileNotFoundError

Se genera cuando se solicita un archivo o directorio pero no existe. Corresponde a `errno ENOENT`.

```
try:
    with open(file, 'r') as file:
        file = file.read()
        return file.encode('UTF-8')
except OSError as e:
    print(e.errno)
```

```
FileNotFoundError: [Errno 2] No such file or directory: 'test.txt'
```

exception InterruptedError

Se genera cuando una llamada entrante interrumpe una llamada del sistema. Corresponde a errno EINTR.

exception PermissionError

Se genera cuando se intenta ejecutar una operación sin los derechos de acceso adecuados, por ejemplo, permisos del sistema de archivos. Corresponde a errno EACCES y EPERM.

```
Exception PermissionError: PermissionError(13, 'Permission denied') in  
<bound method workbook.__del__ of <xlsxwriter.workbook.workbook object at  
0x00000000032c3400>> ignored
```

exception TimeoutError

Se genera cuando se agota el tiempo de espera de una función del sistema a nivel del sistema. Corresponde a errno ETIMEDOUT.

```
def requests_get(url, **kwargs):  
    USER_AGENTS = (  
        'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.7; rv:11.0) Gecko/20100101 Firefox/11.0',  
        'Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:22.0) Gecko/20100101 Firefox/22.0',  
        'Mozilla/5.0 (Windows NT 6.1; rv:11.0) Gecko/20100101 Firefox/11.0',  
        ('Mozilla/5.0 (Macintosh; Intel Mac OS X 10_7_4) AppleWebKit/536.5 (KHTML, like Gecko) '  
        'Chrome/19.0.1084.46 Safari/536.5'),  
        ('Mozilla/5.0 (Windows; Windows NT 6.1) AppleWebKit/536.5 (KHTML, like Gecko)  
        'Chrome/19.0.1084.46'  
        'Safari/536.5')  
    )  
    try:  
        r = requests.get(  
            url,  
            timeout=12,  
            headers={'User-Agent': random.choice(USER_AGENTS)}, **kwargs  
        )  
    except ConnectionError:  
        exit_after_echo('Network connection failed.')  
    except Timeout:  
        exit_after_echo('timeout.')  
    return r
```


exception ValueError

Se genera cuando una operación o función recibe un argumento que tiene el tipo correcto pero un valor inapropiado, y la situación no se describe con una excepción más precisa como IndexError.

```
n = int(input("Please enter a number: "))

Please enter a number: 23.5

-----
ValueError                                Traceback (most recent call last)
<ipython-input-9-02fbe8840e1a> in <module>
----> 1 n = int(input("Please enter a number: "))

ValueError: invalid literal for int() with base 10: '23.5'
```

exception BaseException

La clase base para todas las excepciones integradas. No está destinado a ser heredado directamente por clases definidas por el usuario (para eso, use Exception). Si str() se invoca en una instancia de esta clase, se devuelve la representación de los argumentos de la instancia o la cadena vacía cuando no hay argumentos.

Ejemplo:

Exception

Todas las excepciones integradas que no existen en el sistema se derivan de esta clase. Todas las excepciones definidas por el usuario también deben derivarse de esta clase.

Ejemplo:

```
try:
    x = 5 / 0
except ZeroDivisionError as e:
    # `e` is the exception object
```

```
print("Got a divide by zero! The exception was:", e)
# handle exceptional case
x = 0
finally:
    print "The END"
# it runs no matter what execute.
```

exception ArithmeticError

La clase base para aquellos incorporados excepciones que se levantó para diversos errores aritméticos: OverflowError, ZeroDivisionError, FloatingPointError.

Ejemplo:

```
import sys

try:

    7/0

except ArithmeticError as e:

    print e

    print sys.exc_type

    print 'This is an example of catching ArithmeticError'
```

exception ZeroDivisionError

Se genera cuando el segundo argumento de una operación de división o módulo es cero. El valor asociado es una cadena que indica el tipo de operandos y la operación.

Ejemplo:

```
try:
```

```

x = 5 / 0
except ZeroDivisionError as e:
    # `e` is the exception object
    print("Got a divide by zero! The exception was:", e)
    # handle exceptional case
    x = 0
finally:
    print "The END"
    # it runs no matter what execute.

```

excpetion OverflowError

Se genera cuando el resultado de una operación aritmética es demasiado grande para ser representado. Esto no puede ocurrir para los números enteros (que preferirían aumentar MemoryError que darse por vencidos). Sin embargo, por razones históricas, OverflowError a veces se genera para enteros que están fuera de un rango requerido. Debido a la falta de estandarización del manejo de excepciones de punto flotante en C, la mayoría de las operaciones de punto flotante no están marcadas.

Ejemplo:

```

def get_pi(precision, lib: PiLibType = PiLibType.INTEGER):
    """Get value of pi with the specified level of precision, using passed numeric or
    library.

    :param precision: Precision to retrieve.
    :param lib: Type of numeric value or library to use for calculation.
    :return: Pi value with specified precision.
    """
    try:
        if lib == PiLibType.INTEGER:
            return pi_using_integer(precision)
        elif lib == PiLibType.FLOAT:
            return pi_using_float(precision)
        elif lib == PiLibType.DECIMAL:

```

```

        return pi_using_decimal_lib(precision)
    elif lib == PiLibType.MPMATH:
        return pi_using_mpmath_lib(precision)
except OverflowError as error:
    # Output expected OverflowErrors.
    Logging.log_exception(error)
except Exception as exception:
    # Output expected Exceptions.
    Logging.log_exception(exception, False)

```

exception FloatingPointError

No se utiliza actualmente.

exception AttributeError

Se genera cuando falla una referencia de atributo (consulte Referencias de atributo) o una asignación. (Cuando un objeto no admite referencias de atributos o asignaciones de atributos en absoluto, TypeErrorse genera).

Ejemplo: `def test():`

```

try:
    Logging.line_separator("CREATE BOOK", 50, '+')
    # Create and output book.
    book = Book("The Hobbit", "J.R.R. Tolkien", 366, datetime.date(1937, 9, 15))
    Logging.log(book)

    # Output valid attributes.
    Logging.log(book.title)
    Logging.log(book.author)

```

```

    # Output invalid attribute (publisher).
    Logging.log(book.publisher)
except AttributeError as error:
    # Output expected AttributeErrors.
    Logging.log_exception(error)
except Exception as exception:
    # Output unexpected Exceptions.
    Logging.log_exception(exception, False)

```

exception EOFError

Se genera cuando la input()función alcanza una condición de fin de archivo (EOF) sin leer ningún dato. (NB: los métodos io.IOBase.read()y io.IOBase.readline()devuelven una cadena vacía cuando presionan EOF).

Ejemplo:

```

def main():
    try:
        Logging.log(sys.version)
        title = input("Enter a book title: ")
        author = input("Enter the book's author: ")
        Logging.log(f'The book you entered is \'{title}\' by {author}.')
    except EOFError as error:
        # Output expected EOFErrors.
        Logging.log_exception(error)
    except Exception as exception:
        # Output unexpected Exceptions.
        Logging.log_exception(exception, False)

```

excpetion KeyboardInterrupt

Se genera cuando el usuario pulsa la tecla de interrupción (normalmente Control-Co Delete). Durante la ejecución, se realiza una comprobación periódica de las interrupciones. La excepción hereda de BaseException para no ser atrapado accidentalmente por el código que captura Exception y así evitar que el intérprete salga.

Ejemplo:

Traceback (most recent call last):

File "filename.py", line 119, in <module>

print 'cleaning up...

keyboardinterrupt

exception NameError

Se genera cuando no se encuentra un nombre local o global. Esto se aplica solo a nombres no calificados. El valor asociado es un mensaje de error que incluye el nombre que no se pudo encontrar.

Ejemplo:

line 1, in <module>

input_variable = input("Enter your name: ")

File "<string>", line 1, in <module>

NameError: name 'dude' is not defined

exception SystemExit

Esta excepción la genera la sys.exit()función. Hereda de en BaseException lugar de Exception para que no sea atrapado accidentalmente por el código que captura Exception. Esto permite que la excepción se propague correctamente y haga que el intérprete salga. Cuando no se maneja, el intérprete de Python sale; no se imprime ningún seguimiento de pila. El constructor acepta el mismo argumento opcional que se le pasa sys.exit(). Si el valor es un número entero, especifica el estado de salida del sistema (pasado a la exit()función de C); si es

así None, el estado de salida es cero; si tiene otro tipo (como una cadena), se imprime el valor del objeto y el estado de salida es uno.

Ejemplo:

```
class Quitter(object):

    def __init__(self, name):

        self.name = name

    def __repr__(self):

        return 'Use %s() or %s to exit' % (self.name, eof)

    def __call__(self, code=None):

        # Shells like IDLE catch the SystemExit, but listen when their

        # stdin wrapper is closed.

        try:

            sys.stdin.close()

        except:

            pass

        raise SystemExit(code)

__builtin__.quit = Quitter('quit')

__builtin__.exit = Quitter('exit')
```

Conclusión

Las excepciones son una parte importante de la programación, no solo en Python, en cualquier lenguaje de programación es importante controlar el flujo y continuidad de nuestro código, ya que la manera correcta y optima de un código de programación es que no colapse con ningún tipo de error, ya sea por parte del usuario al introducir un dato erróneo o por parte del programador en el caso de los errores generados al no predecir si el usuario pudiera ingresar datos incorrectos.

La manera en la que podemos controlar estos errores es mediante las excepciones que nos da Python e incluso agregar las propias para en caso de que el lenguaje no cuente con un error predefinido en su código fuente, darle por medio de nuestro propio manejo de excepciones información al usuario respecto a lo que está haciendo mal en el software o cualquier error que se pudiese generar en el código.

Es importante conocer el manejo de esta herramienta que nos da Python, ya que esto nos permite presentar un código mas profesional y que está preparado para cualquier tipo de error que se pudiese generar en la ejecución de nuestro software.