Cronograma

- Manejo de Excepciones.
- Clausulas try-except-finally
- Sentencia raise
- Excepciones definidas por el usuario
- Ejercicios.

Excepciones

- Las excepciones son errores detectados durante la ejecución del programa.
- Por ejemplo:

  Dividir un número entre 0

  Intentar acceder a un archivo para lectura y que no exista.

  Errores de manipulación de diferentes tipos de objetos.
- Estos ejemplos generan una excepción informando que hay un problema.

Excepciones (2)

- Si la excepción no se captura, el flujo de ejecución se interrumpe y se muestra la información asociada a la excepción

```
def division(a, b):
        return a / b

def calcular():
        division(1, 0)

calcular()
```

```
<terminated> C:\Users\GustavoSignorele\workspace\TestCurso\clase12-misc.py
Traceback (most recent call last):
  File "C:\Users\GustavoSignorele\workspace\TestCurso\clase12-misc.py", line 2, in <module>
    print (1/0)
ZeroDivisionError: division by zero
```

Captura de excepciones

- Las sentencias ***try - except - finally*** permiten capturar las posibles excepciones del código y así evitar que aborte el programa y tratar el error adecuadamente.

  try:

  f = file("archivo.txt")

  except:

  print ("El archivo no existe")

- La sentencia try engloba el código que pensamos que puede producir una excepción.
- En caso de que la excepción se produzca, se corta la ejecución del try y se pasa a ejecutar el código de la sentencia ***except***
- Si no se produce ninguna excepción, se ejecuta todo el código que está en el try y se omite el except

Diferentes tipos de excepciones

- Cuando se produce la división por cero, Python informa del error: **ZeroDivisionError**. Hay muchos tipos de excepciones

4 + var*3

Traceback (most recent call last):

File "<stdin>", line 1, in ?

NameError: name 'var' is not defined

Esto significa que cuando se usa una variable no definida, tira una excepción NameError

'6' + 2

Traceback (most recent call last):

File "<stdin>", line 1, in ?

TypeError: cannot concatenate 'str' and 'int' objects

La excepción *TypeError* se tira cuando se opera con datos que no pueden mezclarse

Diferentes tipos de errores (2)

Si conocemos el tipo error que podría producirse, es posible manejar diferentes situaciones según el tipo:

```
try:
        num = int("233b")
        print "Aca ya no llegamos"
        print numero
except NameError:
        print "La variable no existe"
except ValueError:
        print "El valor no es un numero"
```

Si al producirse la excepción, el tipo no coincide con ninguno de los manejadores de excepción, Python lanza su manejador (e interrumpe la ejecución).

Diferentes tipos de errores (3)

- Podemos definir una excepción para el caso en que no se cumpla ninguna de las anteriores (debe ser siempre la última de las sentencias ***except***).
- Es importante destacar que si bien luego de un bloque **try** puede haber varios bloques **except**, se ejecutará a lo sumo, uno de ellos.

```
try:
        # aquí ponemos el código que puede lanzar excepciones
except IOError:
        # entrará aquí en caso que se haya producido
        # una excepción IOError
except ZeroDivisionError:
        # entrará aquí en caso que se haya producido
        # una excepción ZeroDivisionError
except:
        # entrará aquí en caso que se haya producido
        # una excepción que no corresponda a ninguno
        # de los tipos especificados en los except previos
```

Ejemplos

- En este ejemplo, el try - except que se ejecuta es el que está dentro de la función llamada f().
- Una aclaración: en el except usamos una variable, en este caso llamada e (podría llamarse de otra manera). En esa variable, se guarda la excepción, y en ella, quedan cargados ciertos datos relativos al error, que pueden servirnos para comprender qué pasó.

```
def f():
    try:
        x = int("four")
    except ValueError as e:
        print(type(e))
        print("got it in the function :-) ", e)
try:
    f()
except ValueError as e:
    print("got it :-) ", e)
print("Let's get on")
```

Salida

```
<type 'exceptions.ValueError'>
('got it in the function :-) ', ValueError("invalid literal for int() with base 10: 'four'",))
Let's get on
```

Ejemplos (2)

En este ejemplo, el try - except que se ejecuta está fuera de la función, en la invocación

```
def f():
    try:
        x = int("four")
    except IOError as e:
        print(type(e))
        print("got it in the function :-) ", e)
try:
    f()
except ValueError as e:
    print("got it :-) ", e)
print("Let's get on")
```

```
('got it :-) ', ValueError("invalid literal for int() with base 10: 'four'",))
Let's get on
```

Jerarquía de excepciones

- https://docs.python.org/3.5/library/exceptions.html#exception-hierarchy
- *Exception* se llama la excepción más genérica. Si capturamos esa dentro de un except, entonces, capturamos cualquiera. Todas las excepciones son *Exception*

```
+-- Exception
     +-- StopIteration
     +-- StopAsyncIteration
     +-- ArithmeticError
     |    +-- FloatingPointError
     |    +-- OverflowError
     |    +-- ZeroDivisionError
     +-- AssertionError
     +-- AttributeError
     +-- BufferError
     +-- EOFError
     +-- ImportError
     +-- LookupError
     |    +-- IndexError
     |    +-- KeyError
     +-- MemoryError
     +-- NameError
     |    +-- UnboundLocalError
     +-- OSError
     |    +-- BlockingIOError
     |    +-- ChildProcessError
     |    +-- ConnectionError
     |    |    +-- BrokenPipeError
     |    |    +-- ConnectionAbortedError
     |    |    +-- ConnectionRefusedError
```

Excepción genérica

```
import sys
def test():
    try:
        f = open("test.txt", "w")
        s = f.readline()
        i = int(s.strip())
    except ValueError as e:
        print (e)
    except Exception as e:
        print (e.args)
        print (e.__doc__)
        print ("{}".format(sys.exc_info()[0]))
        print (e.__class__.__name__)
        print (type(e).__name__)
        print('An exception occurred: {}'.format(e))
        print (e)
test()
```

```
('File not open for reading',)
I/O operation failed.
<type 'exceptions.IOError'>
IOError
IOError
An exception occurred: File not open for reading
File not open for reading
```

En este ejemplo, en el except genérico, se muestran diferentes formas para mostrar los datos relativos a la excepción.

Sentencia *finally*

- Luego de las sentencias ***except***, puede ubicarse un bloque ***finally*** donde se escriben las sentencias de finalización. El bloque ***finally*** se ejecuta siempre, sin importar si surgió una excepción o no.

```
try:
        archivo = open("miarchivo.txt")
        # procesar el archivo
except IOError:
        print "Error de entrada/salida."
        # realizar procesamiento adicional
except:
        # procesar la excepción
finally:
        # si el archivo no está cerrado hay que cerrarlo
        if not(archivo.closed):
                archivo.close()
```

Raise

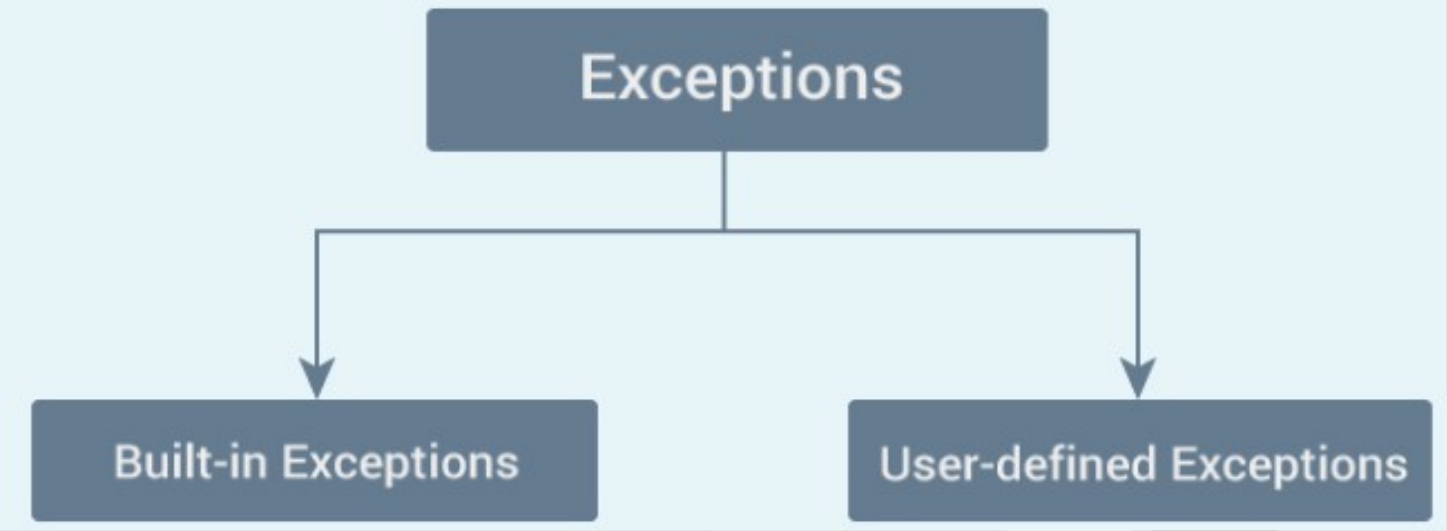
- La sentencia **raise** nos permite forzar (“levantar”) una excepción específica:

```
try:
    raise NameError('HiThere')
except NameError:
    print('An exception flew by!')
    raise
```

- La última sentencia del ejemplo (**raise** sin especificar excepción), hace que se relance la excepción en curso

```
An exception flew by!
Traceback (most recent call last):
  File "<stdin>", line 2, in ?
NameError: HiThere
```

Excepciones definidas por el usuario



- Todas las clases propias para el manejo de Excepciones deben derivarse (heredar) de la clase **Exception** de Python

Definidas por el usuario (2)

- En este ejemplo, se define una excepción **Error**, que deriva o hereda de la excepción genérica de Python (**Exception**)
- A su vez, se definen 2 excepciones más, **ValueTooSmallError** y **ValueTooLargeError**, ambas heredan de **Error**
- La instrucción pass es para indicar que no hay nada dentro de la excepción. Luego veremos que se pueden incluir funciones (métodos)

```
# define Python user-defined exceptions
class Error(Exception):
    """Base class for other exceptions"""
    pass

class ValueTooSmallError(Error):
    """Raised when the input value is too small"""
    pass
class ValueTooLargeError(Error):
    """Raised when the input value is too large"""
    pass
```

Definidas por el usuario (3)

Una forma medio simple de usar las excepciones que definimos en la PPT anterior. Recordar que la instrucción *raise* tira (o levanta), la excepción.

```
# you need to guess this number
number = 10
iguales = False
while True and not iguales:
    try:
        i_num = int(input("Enter a number: "))
        if i_num < number:
            raise ValueTooSmallError
        elif i_num > number:
            raise ValueTooLargeError
        iguales = True
    except ValueTooSmallError:
        print("This value is too small, try again!")
    except ValueTooLargeError:
        print("This value is too large, try again!")

print("Congratulations! You guessed it correctly.")
```

Definidas por el usuario (4)

Vamos a crear una excepción propia (*MiError*), sin *pass*. En la excepción, vamos a guardar dos datos: *valor* y *mensaje* (podrían llamarse de cualquier otra forma y podríamos guardar todo lo que quisiéramos).

Esta excepción se podría usar para capturar errores generados al operar con números. En *valor*, guardaríamos el número que causó el error, y en *mensaje*, un string alusivo.

Para poder guardar datos dentro de la excepción, debemos crear una función (o método) llamado *__init__* (A este método se le llama *constructor*). Esta función, que debe llamarse así obligatoriamente, recibe como primer parámetro, SIEMPRE, *self*. Luego, recibe los datos a guardar en la excepción (en este caso, *valor* y *mensaje*, aunque podrían llamarse de cualquier otra forma, no así *self*).

Además se define otra función llamada *__str__*. En ella se devuelve un string legible y agradable, generalmente integrado con los datos que se guardaron en la excepción (valor y mensaje). Este string se usa cada vez que se necesite una impresión de la excepción (por ejemplo, cuando se hace un print)

Definidas por el usuario (5)

```
class MiError(Exception):
    def __init__(self, valor, mensaje):
        self.valor = valor
        self.mensaje = mensaje
    def __str__(self):
        return str("valor: {} - mensaje: {} " .format (self.valor, self.mensaje) )

try:
    raise MiError(2*2, "este es un test")
except MiError as e:
    print('Ocurrio mi excepcion, valor:', e.valor)
    print(e)
```

Dentro del try, se levanta la excepción MiError, y se pasan los parámetros a guardar, según definimos en el constructor, primero el número y luego el mensaje.
Dentro del except, se captura la excepción. Con el primer print imprimimos específicamos valor. Con el segundo, se llama a __str__

```
Ocurrio mi excepcion, valor: 4
valor: 4 - mensaje: este es un test
```

Ejercicio

Escribir una función que reciba dos números (ingresados por el usuario) y a partir de las operaciones especificadas en un archivo (cuyo nombre deberá ser provisto como parámetro), realice los cálculos y devuelva el resultado final de cada línea (resultado que se obtiene operando los dos números recibidos). El archivo contendrá por línea una operación aritmética sencilla; por ej:

+

/

**

Nota: investigar la función de Python llamada eval.

En caso de producirse un error durante la lectura de una línea del archivo, se debe lanzar una excepción propia que informe número de línea, operación problemática y motivo del error.