Exceptions and Errors

Exceptions and Errors

- When writing a program, we, more often than not, will encounter errors. Error caused by not following the proper structure (syntax) of the language is called syntax error or parsing error.

```
>>> if a < 3
  File "<interactive input>", line 1
    if a < 3
           ^
SyntaxError: invalid syntax
```

Exceptions

- Errors can also occur at runtime and these are called exceptions. They occur, for example, when a file we try to open does not exist (FileNotFoundError), dividing a number by zero (ZeroDivisionError), module we try to import is not found (ImportError) etc. Whenever these type of runtime error occur, Python creates an exception object. If not handled properly, it prints a traceback to that error along with some details about why that error occurred.

Exceptions

divide.py - C:\Documents and Settings\admin\Desktop\intro-python\examples\exception\divide.py (3.4.4)

File Edit Format Run Options Window Help

```
A = int(input("Enter first number: "))
B = int(input("Enter second number: "))
C=A/B
print("the result is: ",C)

#Output

Enter first number: 10
Enter second number: 0
Traceback (most recent call last):
  File "C:\Documents and Settings\admin\Desktop
        \intro-python\examples\exception\divide.py",
        line 3, in <module>
    C=A/B
ZeroDivisionError: division by zero
```

Python Built-in Exceptions

- Illegal operations can raise exceptions. There are plenty of built-in exceptions in Python that are raised when corresponding errors occur.

P ns

Exception	Cause of Error
AssertionError	Raised when **assert** statement fails.
AttributeError	Raised when attribute assignment or reference fails.
EOFError	Raised when the **input()** functions hits end-of-file condition.
FloatingPointError	Raised when a floating point operation fails.
GeneratorExit	Raise when a generator's **close()** method is called.
ImportError	Raised when the imported module is not found.
IndexError	Raised when index of a sequence is out of range.
KeyError	Raised when a key is not found in a dictionary.
KeyboardInterrupt	Raised when the user hits interrupt key (Ctrl+c or delete).

P ns

MemoryError	Raised when an operation runs out of memory.
NameError	Raised when a variable is not found in local or global scope.
NotImplementedError	Raised by abstract methods.
OSError	Raised when system operation causes system related error.
OverflowError	Raised when result of an arithmetic operation is too large to be represented.
ReferenceError	Raised when a weak reference proxy is used to access a garbage collected referent.
RuntimeError	Raised when an error does not fall under any other category.
StopIteration	Raised by **next()** function to indicate that there is no further item to be returned by iterator.

P...ns

SyntaxError	Raised by parser when syntax error is encountered.
IndentationError	Raised when there is incorrect indentation.
TabError	Raised when indentation consists of inconsistent tabs and spaces.
SystemError	Raised when interpreter detects internal error.
SystemExit	Raised by **sys.exit( )** function.
TypeError	Raised when a function or operation is applied to an object of incorrect type.
UnboundLocalError	Raised when a reference is made to a local variable in a function or method, but no value has been bound to that variable.
UnicodeError	Raised when a Unicode-related encoding or decoding error occurs.

Python Built-in Exceptions

UnicodeEncodeError	Raised when a Unicode-related error occurs during encoding.
UnicodeDecodeError	Raised when a Unicode-related error occurs during decoding.
UnicodeTranslateError	Raised when a Unicode-related error occurs during translating.
ValueError	Raised when a function gets argument of correct type but improper value.
ZeroDivisionError	Raised when second operand of division or modulo operation is zero.

Exception Handling

- When an exception occurs in Python, it causes the current process to stop and passes it to the calling process until it is handled. If not handled, our program will crash. For example, if function A calls function B which in turn calls function C and an exception occurs in function C. If it is not handled in C, the exception passes to B and then to A. If never handled, an error message is spit out and our program come to a sudden, unexpected halt.

Exception Handling

- `try/except`
  Catch and recover from exceptions raised by Python, or by you.
- `try/finally`
  Perform cleanup actions, whether exceptions occur or not.
- `raise`
  Trigger an exception manually in your code.
- `assert`
  Conditionally trigger an exception in your code.
- `with/as`
  Implement context managers

Exception Handling

-

```
try:
    statements              # Run this main action first
except name1:
    statements              # Run if name1 is raised during try block
except (name2, name3):
    statements              # Run if any of these exceptions occur
except name4 as var:
    statements              # Run if name4 is raised, assign instance raised to var
except:
    statements              # Run for all other exceptions raised
else:
    statements              # Run if no exception was raised during try block
```

```
import sys
while True:
    try:
        A = int(input("Enter first number: "))
        B = int(input("Enter second number: "))
        C=A/B
        break
    except:
       print("Oops!",sys.exc_info()[0],"occured.")
       print("Please try again.")
       print()

print("The result is: ", C)


#Output

Enter first number: 10
Enter second number: 0
Oops! <class 'ZeroDivisionError'> occured.
Please try again.

Enter first number: 10
Enter second number: a
Oops! <class 'ValueError'> occured.
Please try again.

Enter first number: 10
Enter second number: 3
The result is:  3.3333333333333335
```

```
import sys
while True:
    try:
        A = int(input("Enter first number: "))
        B = int(input("Enter second number: "))
        C=A/B
        break
    except ValueError:
       print("Oops!",sys.exc info()[1],"occured.")
       print("Please try again, and enter an integer.")
       print()
    except (TypeError, ZeroDivisionError):
       print("Oops!",sys.exc info()[1],"occured.")
       print("Please try again, and do not enter zero.")
       print()
    except:
       print("Oops!",sys.exc info()[1],"occured.")
       print("Please try again.")
       print()

print("The result is: ", C)
```

```
Enter first number: 10
Enter second number: a
Oops! invalid literal for int() with base 10: 'a' occured.
Please try again, and enter an integer.

Enter first number: 10
Enter second number: 0
Oops! division by zero occured.
Please try again, and do not enter zero.

Enter first number: 10
Enter second number: 2
The result is:  5.0
```

As-keyword

- **We can name a variable within an except statement. We use the as-keyword for this. Here we name the IOError "err" and can use it within the clause.**

```
import sys
while True:
    try:
        A = int(input("Enter first number: "))
        B = int(input("Enter second number: "))
        C=A/B
        break
    except ValueError as err:
       print("Oops!",err)
       print("Please try again, and enter an integer.")
    except (TypeError, ZeroDivisionError) as err:
       print("Oops!", err)
       print("Please try again, and do not enter zero.")
    except :
       print("Oops!",sys.exc_info()[0],"occured.")
       print("Please try again.")

print("The result is: ", C)

#Output
Enter first number: 10
Enter second number: a
Oops! invalid literal for int() with base 10: 'a'
Please try again, and enter an integer.
Enter first number: 10
Enter second number: 0
Oops! division by zero
Please try again, and do not enter zero.
Enter first number: 10
Enter second number: 2
The result is:  5.0
```

Raising Exceptions

- In Python programming, exceptions are raised when corresponding errors occur at run time, but we can forcefully raise it using the keyword raise. We can also optionally pass in value to the exception to clarify why that exception was raised.

Raising Exceptions

```
import sys
while True:
    try:
        A = int(input("Enter first number: "))
        B = int(input("Enter second number: "))
        if B == 0 :
            raise  ValueError("That is zero number!")
        C=A/B
        break
    except ValueError as ve:
        print(ve)
        print("Please try again, and do not enter zero.")
    except:
        print("Oops!",sys.exc_info()[0],"occured.")
        print("Please try again.")

print("The result is: ", C)

#Output
Enter first number: 10
Enter second number: 0
That is zero number!
Please try again, and do not enter zero.
Enter first number: 10
Enter second number: 4
The result is:  2.5
```

Else statement

- The else-statement can be used after a try-except block. If no exception is thrown, the else-statements are executed. The else must come after the excepts.

```
import sys
while True:
    try:
        A = int(input("Enter first number: "))
        B = int(input("Enter second number: "))
        C=A/B
    except ValueError:
       print("Oops!",sys.exc_info()[0],"occured.")
       print("Please try again, and enter an integer.")
    except (TypeError, ZeroDivisionError):
       print("Oops!",sys.exc_info()[0],"occured.")
       print("Please try again, and do not enter zero.")
    except:
       print("Oops!",sys.exc_info()[0],"occured.")
       print("Please try again.")
    else:
        print("The result is: ", C)
        break
print('Good Bye')

#Output
Enter first number: 10
Enter second number: e
Oops! <class 'ValueError'> occured.
Please try again, and enter an integer.
Enter first number: 10
Enter second number: 3
The result is:  3.3333333333333335
Good Bye
```

Finally

- This clause is always executed, even if an error is raised. We can use "finally" statements as a way to clean up, or ensure completion of tasks.

```
import sys
while True:
    try:
        A = int(input("Enter first number: "))
        B = int(input("Enter second number: "))
        C=A/B
    except ValueError:
       print("Oops!",sys.exc_info()[0],"occured.")
       print("Please try again, and enter an integer.")
    except (TypeError, ZeroDivisionError):
       print("Oops!",sys.exc_info()[0],"occured.")
       print("Please try again, and do not enter zero.")
    except:
       print("Oops!",sys.exc_info()[0],"occured.")
       print("Please try again.")
    else:
        print("The result is: ", C)
        break
    finally:
        print("If you got the result, it's ok,")
        print("otherwise please be careful!")

#Output
Enter first number: 10
Enter second number: 0
Oops! <class 'ZeroDivisionError'> occured.
Please try again, and do not enter zero.
If you got the result, it's ok,
otherwise please be careful!
Enter first number: 10
Enter second number: 2
The result is:  5.0
If you got the result, it's ok,
otherwise please be careful!
```

Assert

- This statement causes an AssertionError when an expression is false. We pass an expression (or value) as the first argument. Python stops and signals the assert call.

```
assert test, data          # The data part is optional
```

Assert

```
def f(x):
    assert x > 0, 'x must be negative'
    return x ** 2


print(f(5))
print(f(-5))

#Output
25
Traceback (most recent call last):
  File "C:/Documents and Settings/admin/Desk
    print(f(-5))
  File "C:/Documents and Settings/admin/Desk
    assert x > 0, 'x must be negative'
AssertionError: x must be negative
```

Assert

```
def f(x):
    assert x > 0, 'x must be negative'
    return x ** 2

while True:
    try:
        A = int(input("Enter a number: "))
        B=f(A)
    except AssertionError as err:
       print("An error ", err, " occured, try again")
    else:
        print(B)
        Y = input("Do you want continue(Y/N): ")
        if Y!= 'y' :
            break

#output
Enter a number: -5
An error  x must be negative  occured, try again
Enter a number: 5
25
Do you want continue(Y/N): n
```

Nested try

- Some try and except can be used nested

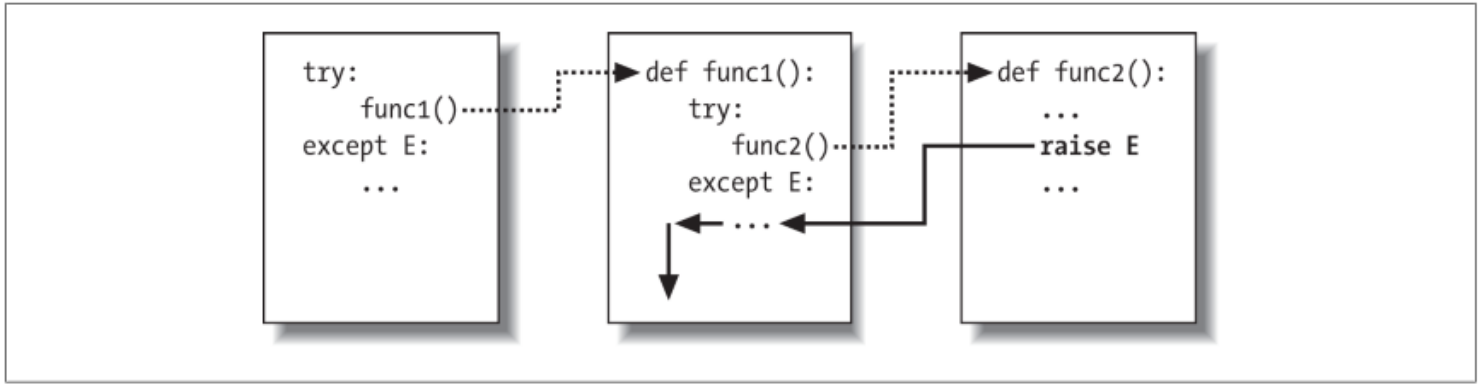
```
try:
    foo()
except FooError:
    handle_foo()
else:
    try:
        bar()
    except BarError:
        handle_bar()
    else:
        try:
            baz()
        except BazError:
            handle_baz()
        else:
            qux()
finally:
    cleanup()
```

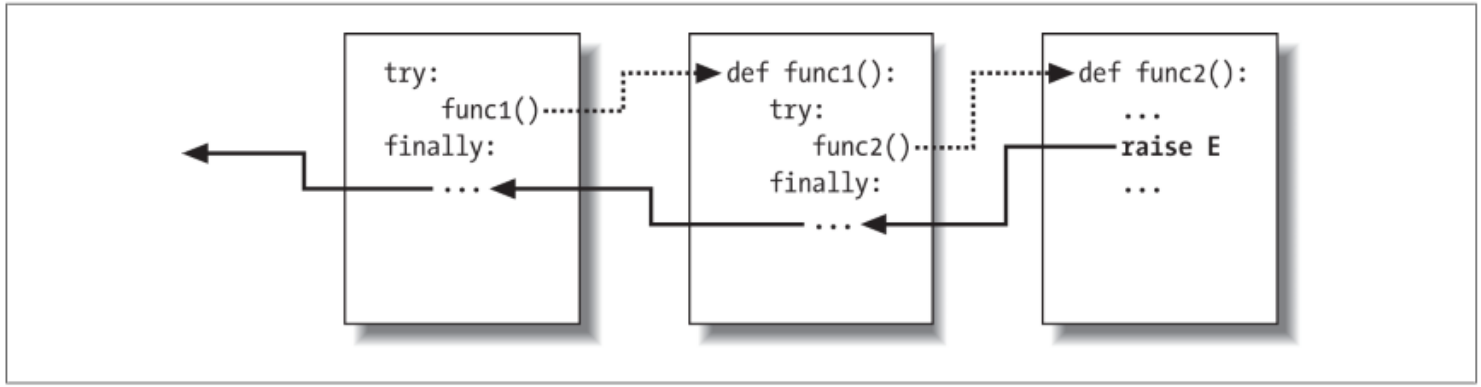
```
def div(x,y):
    try:
        return x / y
    except ZeroDivisionError as err:
       print(err,  " in div is occured.")
       raise ZeroDivisionError('A divid by zero Error')

def methodA(x, y):
    try:
        xi=int(x)
        yi=int(y)
    except ValueError as err:
       print(err,  " in methodA is occured.")
       raise ValueError("A Value Error")
    except TypeError as err:
       print(err,  " in methodA is occured !.")
       raise TypeError('A Type Error')
    else:
        z=div(xi, yi)
        return z

while True:
    try:
        A = input("Enter first number: ")
        B = input("Enter second number: ")
        C=methodA(A, B)
    except ValueError as ve:
       print("An Error is occured. The kind is", ve, ", tray again")
    except Exception as e:
       print("An Error is occured. The kind is", e, ", tray again")
    else:
       print('The result is: ', C)
    finally:
        Y = input("Do you want continue(Y/N): ")
        if Y!= 'y' :
            break
```

Nested try

```
Enter first number: 10
Enter second number: a
invalid literal for int() with base 10: 'a'  in methodA is occured.
An Error is occured. The kind is A Value Error , tray again
Do you want continue(Y/N): y
Enter first number: 10
Enter second number: 0
division by zero  in div is occured.
An Error is occured. The kind is A divid by zero Error , tray again
Do you want continue(Y/N): y
Enter first number: 10
Enter second number: 2
The result is:  5.0
Do you want continue(Y/N): n
```

Nested try


try:
func1()
except E:
...
def func1():
try:
func2()
except E:
...
def func2():
...
raise E
...


try:
func1()
finally:
...
def func1():
try:
func2()
finally:
...
def func2():
...
raise E
...

Exceptions with files

```
import sys

if len(sys.argv) < 3:
    print("Wrong parameter")
    print("./cpy.py filename1 filename2")
    sys.exit(1)


print('Copy  file:')
try:
    fin=open(sys.argv[1])
    fout=open(sys.argv[2],'w')
    try:
         for line in fin:
             fout.write(line.upper())
    except:
       print("An Error is occured during the reading file.")

except:
    print("An Error is occured during the opening file.")
else:
    fin.close()
    fout.close()
```

User-Defined Exception

- Users can define their own exception by creating a new class in Python. This exception class has to be derived, either directly or indirectly, from Exception class. Most of the built-in exceptions are also derived form this class.

```
# 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

# user guesses a number until he/she gets it right

number = 10
while True:
    try:
        i_num = int(input("Enter a number: "))
        if i_num < number:
            raise ValueTooSmallError
        elif i_num > number:
            raise ValueTooLargeError
        break
    except ValueTooSmallError:
        print("This value is too small, try again!")
    except ValueTooLargeError:
        print("This value is too large, try again!")

#output
Enter a number: 5
This value is too small, try again!
Enter a number: 12
This value is too large, try again!
Enter a number: 10
```

```
from random import *
# 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"""
    def __init__(self, value):
        self.value = value

class ValueTooLargeError(Error):
    """Raised when the input value is too large"""
    def __init__(self, value):
        self.value = value

# user guesses a number until he/she gets it right
number = randint(0,10)
while True:
    try:
        i_num = int(input("Enter a number: "))
        if i_num < number:
            raise ValueTooSmallError('This value is too small, try again!')
        elif i_num > number:
            raise ValueTooLargeError('This value is too large, try again!')
    except ValueTooSmallError as exp:
       print(exp)
    except ValueTooLargeError as err:
       print(err)
    else:
        print('O.K., your number is: ', i_num)
        break
```

```
# user guesses a number until he/she gets it right
number = randint(0,10)
while True:
    try:
        i_num = int(input("Enter a number: "))
        if i_num < number:
            raise ValueTooSmallError('This value is too small, try again!')
        elif i_num > number:
            raise ValueTooLargeError('This value is too large, try again!')
    except ValueTooSmallError as exp:
       print(exp)
    except ValueTooLargeError as err:
       print(err)
    else:
        print('O.K., your number is: ', i_num)
        break

#output
Enter a number: 3
This value is too small, try again!
Enter a number: 10
This value is too large, try again!
Enter a number: 5
This value is too small, try again!
Enter a number: 8
This value is too large, try again!
Enter a number: 6
O.K., your number is:  6
```

Exception hierarchy

- **The Exceptions have a hierarchy inheritance.**
- **The class hierarchy for built-in exceptions is as follows.**

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StandardError
      |    +-- BufferError
      |    +-- ArithmeticError
      |    |    +-- FloatingPointError
      |    |    +-- OverflowError
      |    |    +-- ZeroDivisionError
      |    +-- AssertionError
      |    +-- AttributeError
      |    +-- EnvironmentError
      |    |    +-- IOError
      |    |    +-- OSError
      |    |         +-- WindowsError (Windows)
      |    |         +-- VMSError (VMS)
      |    +-- EOFError
      |    +-- ImportError
      |    +-- LookupError
      |    |    +-- IndexError
      |    |    +-- KeyError
      |    +-- MemoryError
      |    +-- NameError
      |    |    +-- UnboundLocalError
      |    +-- ReferenceError
      |    +-- RuntimeError
      |    |    +-- NotImplementedError
      |    +-- SyntaxError
      |    |    +-- IndentationError
      |    |         +-- TabError
      |    +-- SystemError
      |    +-- TypeError
      |    +-- ValueError
      |         +-- UnicodeError
      |              +-- UnicodeDecodeError
      |              +-- UnicodeEncodeError
      |              +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
       +-- ImportWarning
       +-- UnicodeWarning
       +-- BytesWarning
```

Exception hierarchy

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
```

Exception hierarchy

```
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
      +-- RuntimeError
      |    +-- NotImplementedError
      |    +-- RecursionError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
```

Exception hierarchy

```
      +-- SystemError
      +-- TypeError
      +-- ValueError
      |    +-- UnicodeError
      |         +-- UnicodeDecodeError
      |         +-- UnicodeEncodeError
      |         +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
           +-- ResourceWarning
```

End