MINDFUL LEARNING

python

A language is not worth knowing unless it teaches you to think differently

- Larry Wall, Randal Schwartz

Professional

python

Errors, Exceptions and Debugging

Error Handling in Python

- Python provides two very important features to handle any unexpected error in your programs and to add debugging capabilities in them
  - **Exception Handling**
  - **Assertions**

MINDFUL LEARNING

python

Error Types

EXCEPTION NAME	DESCRIPTION
Exception	Base class for all exceptions
StopIteration	Raised when the next() method of an iterator does not point to any object.
SystemExit	Raised by the sys.exit() function.
StandardError	Base class for all built-in exceptions except StopIteration and SystemExit.
ArithmeticError	Base class for all errors that occur for numeric calculation.
OverflowError	Raised when a calculation exceeds maximum limit for a numeric type.
FloatingPointError	Raised when a floating point calculation fails.
ZeroDivisonError	Raised when division or modulo by zero takes place for all numeric types.
AssertionError	Raised in case of failure of the Assert statement.
AttributeError	Raised in case of failure of attribute reference or assignment.
EOFError	Raised when there is no input from either the raw_input() or input() function and the end of file is reached.
ImportError	Raised when an import statement fails.
KeyboardInterrupt	Raised when the user interrupts program execution, usually by pressing Ctrl+c.
LookupError	Base class for all lookup errors.
IndexError	Raised when an index is not found in a sequence.

MINDFUL LEARNING

python

Error Types

EXCEPTION NAME	DESCRIPTION
EnvironmentError	Base class for all exceptions that occur outside the Python environment.
IOError	Raised when an input/ output operation fails, such as the print statement or the open() function when trying to open a file that does not exist.
IOError	Raised for operating system-related errors.
SyntaxError	Raised when there is an error in Python syntax.
IndentationError	Raised when indentation is not specified properly.
SystemError	Raised when the interpreter finds an internal problem, but when this error is encountered the Python interpreter does not exit.
SystemExit	Raised when Python interpreter is quit by using the sys.exit() function. If not handled in the code, causes the interpreter to exit.
ValueError	Raised when the built-in function for a data type has the valid type of arguments, but the arguments have invalid values specified.
RuntimeError	Raised when a generated error does not fall into any category.
NotImplementedError	Raised when an abstract method that needs to be implemented in an inherited class is not actually implemented.

Exception

- An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.
- In general, when a Python script encounters a situation that it cannot cope with, it raises an exception.
- An exception is a Python object that represents an error.
- When a Python script raises an exception, it must either handle the exception immediately otherwise it terminates and quits.

MINDFUL LEARNING

python

The try/except/else Statement

```
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
```

The assert Statement

- When it encounters an assert statement, Python evaluates the accompanying expression, which is hopefully true.
- If the expression is false, Python raises an *AssertionError* exception.

```
def KelvinToFahrenheit(Temperature):
   assert (Temperature >= 0),"Colder than absolute zero!"
   return ((Temperature-273)*1.8)+32
print KelvinToFahrenheit(273)
print int(KelvinToFahrenheit(505.78))
print KelvinToFahrenheit(-5)
```

```
32.0
451
Traceback (most recent call last):
File "test.py", line 9, in
print KelvinToFahrenheit(-5)
File "test.py", line 4, in KelvinToFahrenheit
assert (Temperature >= 0),"Colder than absolute zero!"
AssertionError: Colder than absolute zero!
```

MINDFUL LEARNING

python

Example

```
try:
   fh = open("testfile", "w")
   fh.write("This is my test file for exception handling!!")
except IOError:
   print "Error: can\'t find file or read data"
else:
   print "Written content in the file successfully"
   fh.close()
```

MINDFUL LEARNING

python

The try/finally Statement

- The finally block is a place to put any code that must execute, whether the try-block raised an exception or not.

```
try:
        statements                      # Run this action first
finally:
        statements                      # Always run this code on the way out
```

```
try:
   fh = open("testfile", "w")
   fh.write("This is my test file for exception handling!!")
finally:
   print "Error: can\'t find file or read data"
```

MINDFUL LEARNING

python

Unified try/except/finally

```
try:                                    # Merged form
        main-action
except Exception1:
        handler1
except Exception2:                      # Catch exceptions
        handler2
...
else:                                    # No-exception handler
        else-block
finally:                                # The finally encloses all else
        finally-block
```

Mindful Learning

python

Unified try/except/finally

- The net effect is that the finally is always run, regardless of whether:
  - An exception occurred in the main action and was handled.
  - An exception occurred in the main action and was not handled.
  - No exceptions occurred in the main action.
  - A new exception was triggered in one of the handlers.

MINDFUL LEARNING

python

The raise Statement

- To trigger exceptions explicitly, you can code raise statements.
- Their general form is a raise statement consists of the word raise, optionally followed by the class to be raised or an instance of it:

```
raise instance  # Raise instance of class
raise class     # Make and raise instance of class: makes an instance
raise           # Reraise the most recent exception
```

Example

```
def functionName( level ):
   if level < 1:
      raise "Invalid level!", level
      # The code below to this would not be executed
      # if we raise the exception
```

- In order to catch an exception, an "except" clause must refer to the same exception thrown either class object or simple string.

```
try:
   Business Logic here...
except "Invalid level!":
   Exception handling here...
else:
   Rest of the code here...
```

Standard errors such as IOError, etc. can also be raised

User Defined Exceptions

- In Python, users can define such exceptions by creating a new class.
- 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
- In the following example:
  - We create a user-defined exception called **CustomError** which is derived from the **Exception** class.
  - This new exception can be raised, like other exceptions, using the raise statement with an optional error message.

Mindful Learning

python

Example

```
# 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
```

MINDFUL LEARNING

python

Example continued..

```
# our main program
# user guesses a number until he/she gets it right

# you need to guess this number
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!")
       print()
   except ValueTooLargeError:
       print("This value is too large, try again!")
       print()

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

Run this program and study it carefully.

The **with…as** Construct

- Python's **with** statement was first introduced in Python 2.5.
- It's handy when you have two related operations which you'd like to execute as a pair, with a block of code in between.
- The classic example is opening a file, manipulating the file, then closing it:

```
with open('output.txt', 'w') as f:
    f.write('Hi there!')
```

  - The above with statement will automatically close the file after the nested block of code.
  - The advantage of using a with statement is that it is guaranteed to close the file no matter *how* the nested block exits.
  - If an exception occurs before the end of the block, it will close the file before the exception is caught by an outer exception handler.

The with..as Statement

- The with/as statement is designed to automate startup and termination activities that must occur around a block of code.
- It is roughly like a try/finally statement in that its exit actions run whether an exception occurred or not, but it allows a richer object-based protocol for specifying entry *and exit actions, and may reduce* code size.
- Still, it's not quite as general, as it applies only to objects that support its protocol; try statement can handle many more use cases.

```
with open(r'C:\misc\data') as myfile:
          for line in myfile:
          print(line)
```

Eq

```
myfile = open(r'C:\misc\data')
try:
          for line in myfile:
          print(line)
          ...more code here...
finally:
          myfile.close()
```