ECE12: Introduction to Programming
Lecture 20

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 20: Overview

- Exception Handling
  - Introduction
  - Raising exceptions
    - **`raise`** statement
    - **`assert`** statement
    - Python exception classes
  - Catching and handling exceptions
    - **`try`** - **`except`** statement
  - Creating user-defined exceptions

Exception Handling

- Introduction
  - Exception handling is required in situations where some code detects an error condition and is unable to handle it
    - an exception is *raised* or *thrown*
  - Exceptions may be caught and handled by exception handlers to take appropriate actions
    - an exception is *caught* and *handled*
  - Any exception that is not caught and handled will terminate the program execution with an error message
    - Abort with error message is the default exception handler
- Examples
  - Invalid arguments to arithmetic operations
    - Division by zero, square root of negative numbers, ...
  - Input/Output errors, file read/write error, disk full, ...
  - etc. etc.

Exception Handling

- Raising Exceptions
  - Implicit exceptions raised by built-in functions
    - Built-in operators and functions can raise exceptions
  - Explicit exceptions raised by programmer
    - **`raise`** statement
      - Syntax:
        - » **`raise`** *ExceptionClass*, *ExceptionArgument*
        - » *ExceptionClass* is predefined or user-defined
        - » *ExceptionArgument* is optional
      - Example:

```
def SetMinute(self, minute=0):
    """sets the minute of a time object"""
    if (0 <= minute <= 59):
        self.minute = minute
    else:
        raise ValueError, "Minute value out of range 0-59"
```

Exception Handling

- Raising Exceptions
  - Assertions
    - **`assert`** statement
      - Syntax:
        - » **`assert`** *Test*, *ErrorMessage*
        - » *Test* is a condition that is expected to evaluate to true; if it evaluates to false, an assertion is raised
        - » *ErrorMessage* is optional
      - Example:

```
def CircleArea(radius):
    """computes the area of a circle"""
    assert radius >= 0, "Expected non-negative radius!"
    return 3.14159 * radius * radius
```

      - Assertions are debugging statements
        - » executed only if **__debug__** is true
        - » ignored if optimization (-o) is turned on

Exception Handling

- Python Exception Classes (part 1/2)

– Exception	Exception base class
• StandardError	Error base class
– ArithmeticError	Arithmetic exceptions
» FloatingPointError	Failure of floating-point operation
» OverflowError	Arithmetic overflow
» ZeroDivisionError	Division or modulus by zero
– AssertionError	**`assert`** statement failure
– AttributeError	Invalid class/object attribute
– EnvironmentError	External errors
» IOError	Input/output error
» OSError	Operating system error
– EOFError	End-of-file reached
– ImportError	**`import`** statement failure
– KeyboardInterrupt	CTRL-C!
– LookupError	Indexing and key errors
» IndexError	Index out of range
» KeyError	Non-existing dictionary key
– ...	

Exception Handling

- Python Exception Classes (part 2/2)
  - Exception
    - StandardError — Error base class
      - ...
      - MemoryError — Out of memory
      - NameError — Unknown identifier
      - RuntimeError — Generic error
      - SyntaxError — Parsing error
      - SystemError — Error in the interpreter
      - SystemExit — generated by **`sys.exit()`**
      - TypeError — Invalid type for operation
      - ValueError — Invalid value
    - Warning — Warning base class
      - UserWarning — Default warning
      - DeprecationWarning — Deprecated feature
      - SyntaxWarning — Dubious syntactic feature
      - FutureWarning — Language feature change
      - RuntimeWarning — Dubious runtime feature

Exception Handling

- Catching and Handling Exceptions
  - **`try`** - **`except`** statement
    - Syntax:
      - **`try:`**
        `    # do something`
        `except` *ExceptionClass*, *ExceptionArgument:*
        `    # handle the exception`
        `except ...`
        `    # handle more exceptions`
        `else:`
        `    # do this if no exception occurred`
      - multiple **`except`** clauses are allowed
      - one optional **`else`** clause is allowed
      - *ExceptionClass* is predefined or user-defined
      - *ExceptionArgument* is optional

Exception Handling

- Catching and Handling Exceptions
  - **`try`** - **`except`** statement
    - Example:

```
try:
    # input
    radius = int(raw_input("Enter the radius: "))
    # compute
    area = CircleArea(radius)
    # output
    print "The circle area is", area
except AssertionError, message:
    print message
except ValueError, message:
    print "Invalid value:", message
except KeyboardInterrupt:
    print "CTRL-C!"
except EOFError:
    print "End-of-file error! No input given!"
```

Exception Handling

- Creating User-defined Exceptions
  - Class Exception can be extended by inheritance
    - Example:

```
from exceptions import Exception

class RadiusError(Exception):
    def __init__(self, message):
        self.message = message

def CircleArea(radius):
    """computes the area of a circle"""
    if radius < 0:
         raise RadiusError, "Expected non-negative radius!"
    return 3.14159 * radius * radius

try:
    radius = int(raw_input("Please enter the radius: "))
    area = CircleArea(radius)
    print "The circle area is", area
except RadiusError, ErrorObject:
    print "Radius error:", ErrorObject.message
```