# Exceptions

2 Nov 2010
CMPT140
Dr. Sean Ho
Trinity Western University

## Options for error handling

- Use a combination of these:
  - Ask the user to be nice:
    - User manual, precondition comments, prompts
  - Print an error message to screen
  - Set a result flag:
    - e.g., return False upon error
  - Panic and die: sys.exit()
  - Raise an exception: ZeroDivisionError

## Exceptions

- Exceptions are a way of terminating execution of the current context
- When an exception is raised (thrown),
  - execution of the current procedure stops, and
  - Control jumps to the nearest exception handler (catches the exception)
- The exception handler can cleanup
- Execution then continues after that block
- If the exception reaches outermost level, an error message is automatically generated

# try / except

- If an exception is raised within a try block,
- Execution of the block terminates and control jumps to the except clause:

```
try:
    while True:
        numer = int(input('Numerator: '))
        denom = int(input('Denominator: '))
        print( '%d / %d = %d' % \
          (numer, denom, numer / denom) )
except:
    print 'Oops!'
```

## Catching specific exceptions

- Don't just catch all exceptions!
  - May hide a genuine error, hard to debug
- Catch only specific exceptions we anticipate:

```
try:
    while True:
        numer = int(input('Numerator: '))
        denom = int(input('Denominator: '))
        print( '%d / %d = %d' % \
          (numer, denom, numer / denom) )
except ZeroDivisionError:
    print( 'Oops! Divide by zero!' )
```

- Any other exception falls through to the next exception handler

## Handling exceptions

- The standard math.sqrt() raises ValueError on a negative argument:

```
from math import sqrt
sqrt(-1)            # ValueError
```

- We can handle this:

```
try:
    num = float(input('Find sqrt of: '))
    result = sqrt(num)
    print( 'The square root is', result )
except ValueError:
    print( "Can't take square root of", num )
```

- Can also use a tuple of multiple exception types

## Raising exceptions

- We can force exceptions to be raised:
  (this is not what ZeroDivisionError was intended for!)

```
try:
    while True:
        if input('Guess a number: ') == '5':
            raise ZeroDivisionError
except ZeroDivisionError:
    print( 'You got it!' )
```

- Within a handler, can re-raise the current exception:

```
try:
    x = 5 / 0
except ZeroDivisionError:
    print( 'oops, divided by zero!' )
    raise                         # raises ZeroDivisionError
```

# 'else' clauses for exceptions

- The optional else clause is executed only if the try block completes without throwing any exceptions:

```
try:
    for tries in range(3):
        if input('Guess a number: ') == '5':
            raise ZeroDivisionError
except ZeroDivisionError:
    print( 'You got it!' )
else:
    print( 'Too bad, you ran out of tries!' )
```

# 'finally' clauses for exceptions

- The optional finally clause is always executed before leaving the section, whether an exception happened or not.

```
try:
    for tries in range(3):
        if input('Guess a number: ') == '5':
            raise ZeroDivisionError
except ZeroDivisionError:
    print( 'You got it!' )
else:
    print( 'Too bad, you ran out of tries!' )
finally:
    print( 'Bye!' )
```

# Example: robust input

```
while True:
    try:
        userIn = int(input(“Num of people? ”))
    except (SyntaxError, NameError):
        print( “Please enter a number!” )
    except TypeError:
        print( “Enter just an integer, thanks!” )
    except KeyboardInterrupt:
        print( “OK, you want to quit!” )
        break
    else:
        break
```

# Using exceptions: functions

- Exceptions are an elegant way for functions to indicate errors:
  - Invalid input
    - Parameters don't satisfy pre-conditions
  - Error during execution (runtime error)
    - Computed a bad value, can't continue
- It's good custom to specify in the docstring what exceptions your function might raise
  - In Java, must declare unhandled excepts!
- Programs that call your function may wrap it in a try/except block to handle your errors

# Example: discriminant

```
def discrim(a, b, c):
    """Find discriminant of a x**2 + b x + c = 0.
    Pre: a, b, c are all floats or ints.
    Post: returns sqrt(b**2 – 4 a c), if it exists.
    Exceptions: raises ValueError if discriminant
        doesn't exist."""
    from math import sqrt
    return sqrt( b**2 – 4.0*a*c )


try:
    d = discrim(2, 1, 3)
except ValueError:
    print( "No real roots!" )
```

# Auxiliary data with exceptions

- Create an exception with auxiliary data; raise it

```
try:
    raise Exception('apples', 'oranges')
```

- Catch the exception and assign it to a variable:

```
except Exception as exc:
    print( exc.args )
```

- Here, exc is assigned to the exception object
- Auxiliary data (list of arguments) are passed together with the exception: get it with .args
- Use this to specify additional info about the error: perhaps some explanatory text