

EXCEPTION HANDLING, DEBUGGING, AND INDEFINITE LOOPS

Conditional Program Execution

- Programs (scripts)
 - ▣ Modules designed to run directly (not imported)
- Libraries
 - ▣ Modules imported and designed not to run directly
- Hybrid
 - ▣ Can do both
- `__name__`
- `>>> if __name__ == '__main__':`
`main()`

What is Exception Handling?

- Mechanism to deal with special or exceptional cases in a program
 - Developers can write code that
 - Catches exceptions while program is running
 - Deals with the exceptions

The Need for Exception Handling

- A way to separate code that addresses special cases from algorithm code
- Says “do these steps and if an exception occurs , handle it this way”
- Example problem
 - ▣ Download `slope.py` from Angel: Lessons → Modules to Download in Class → Session 9 → `slope.py`
 - ▣ Run with points (5, 6) and (10, 4)
 - ▣ Run with points (3, 6) and (10, 12)
 - ▣ Run with points (3, 300) and (3, 500)

Exception Handling: try statement

- Can use **if** statement to take care of this special case.
- We can instead use exception handling: **try** statement

try:

<body>

except <errorType>

<handler>

Algorithm code goes in body of try clause

Type of exception that could be generated

Exceptions are caught and handled in except clause.

Using Exception Handling

- Use exception handling code to fix slope.py

try:

```
deltaY = y2 - y1
```

```
deltaX = x2 - x1
```

```
slope = deltaY / float(deltaX)
```

```
print "\nThe slope of the line is %0.3f." % (slope)
```

except **ZeroDivisionError**:

```
print "Found division by zero error."
```

```
print "The line has an infinite slope."
```

Multiple “except” Clauses

- Like multi-way decisions with **if-elif-else** statements
- Becomes **try-except-except ...** statement
- Modify **slope.py** to include an except clause for a name error exception
 - ▣ Run with points (3, 300) and (a, 500)
 - ▣ `>>>dir(__builtins__)` # shows names of exceptions
- Modify **slope.py** to include an except clause for a syntax error exception
 - ▣ Run with input (3, 300) and (a 500) **missing comma!**

Debugging

- Testing for the presence of errors and correcting them
- Why is this important?
- Ways to debug
 - ▣ Insert logging messages to show program flow
 - ▣ Use a debugger
 - A program that you can use to execute another program and analyze its runtime behavior

Using a Debugger

- Debugger: A program that you can use to execute another program and analyze its runtime behavior
- Debugging commands
 - ▣ Set breakpoints
 - ▣ Single step
 - ▣ Inspect variables
- Debugging Example
 - ▣ Download [printFactorial.py](#)

Sample Debugging Session: Eclipse

Debug - printFactorial.py - Eclipse SDK

File Edit Source Refactoring Navigate Search Project Run Window Help

Debug Pydev Java

Debug

- test printFactorial.py [Python Run]
- printFactorial.py
 - MainThread
 - printFactorial [printFactorial.py:4]
 - factTable [printFactorial.py:22]
 - <module> [printFactorial.py:24]
 - run [pydevd.py:634]
 - <module> [pydevd.py:779]

Variables Breakpoints

Name	Value
Globals	Global variables
formatString	str: %21d
n	int: 0
product	int: 1
width	int: 21

int: 21

printFactorial.py

```
1 def printFactorial(n, width):
2     formatString = "%"+str(width)+ "d"
3     product = 1
4     for i in range(1, n+1):
5         product = product * i
6
7     print formatString % (product)
8
9 #printFactorial(5, 6)
10 #printFactorial(15, 20)
11
12 print "Factorial Table"
13
14
```

Outline

- printFactorial
- factTable

Console Tasks

```
printFactorial.py
pydev debugger
Factorial Table
0
```

Annotations:

- A **view** that shows debugging progress
- This is the **Debug perspective**
- A **view** that shows variable inspection
- This **view** is controlled by an **editor** that lets you make changes to the file
- A **view** that shows the outline of the module being examined (**Outline View**)
- Tabbed **views** (**Console, tasks**)

Writable Insert 4 : 1

How to Debug Effectively

- Reproduce the error
- Simplify the error
- Divide and conquer
- Know what your program should do
- Look at the details
- Understand each bug before you fix it
- Practice!

The min-of-n Problem

- Consider a program to return the minimum of three values
- Create a Python program called `minimum.py`
- Solve the min-of-n problem together by exploring the following approach: comparing each to all
 - ▣ if $x1 \leq x2 \leq x3$:
`min_val = x1`
 - ▣ Debug: What is wrong with this approach?

Comparing Each to All

- One solution is comparing each to all
- Complex expression
- Test multiple conditions using and
- It works, but what happens if we need to handle more than 3 values?

Sequential Processing

- Computer keeps track of the min value so far
- Code is simple
 - ▣ Only two decisions
 - ▣ Elegant and easy to understand

Pair Programming Exercise

- Write a function called `minOfN()` that solves the general case for the min-of-n values problem
- `minOfN()` Should
 - ▣ Prompt for the number of values
 - ▣ Prompt the user to enter each value, one at a time
- Submit to `Minimum Value drop Box` on Angel in Lessons → In-Class Exercise Drop Boxes folder

Review: Definite Loops

- Review: For loop
 - ▣ Definite loop: knows *a priori* the number of iterations of loop body
 - ▣ Counted loop: sequence can be generated by `range()`
 - ▣ Example for loop in `slideshow.py`
- Syntax:
 - ▣ `for <var> in <sequence>:`
 <body>

Is This Loop a Definite Loop?

```
#Open the file
inputFile = open(inputFileName, 'r')

# process each line of file
for line in inputFile:
    image = Image(imageCenter, line.rstrip())
    image.draw(win)
    time.sleep(delay)

win.getMouse()
inputFile.close()
win.close()
```

Indefinite Loops

- Number of iterations is not known when loop starts
- Is a conditional loop
 - ▣ Keeps iterating as long as a certain condition remains true
 - ▣ Conditions are Boolean expressions
- Implemented using while statement
- Syntax:

```
while<condition> :  
    <body>
```

While Loop

- A *pre-test loop*
 - ▣ Condition is tested at the top of the loop
- Example use of while loops

Nadia deposits \$100 in a savings account each month. Each month the account earns 0.25% interest on the previous balance. How many months will it take her to accumulate \$10,000?

While Loops & Exception Handling

```
while True:
```

```
    try:
```

```
        fileName = raw_input("Enter input file name: ")
```

```
        myfile = open(fileName.strip())
```

```
        print "myfile =", myfile
```

```
        break
```

```
    except IOError:
```

```
        print "could not find file", fileName + ". Try again!"
```

```
# do something with open file...
```

```
myfile.close()
```

Hint: You'll need code like this for your homework!

Homework

- See Homework 9 instructions on Angel:
Lessons → Homework → Homework 9
- Homework is due at start of next class session