EXCEPTION HANDLING, DEBUGGING, AND INDEFINITE LOOPS

CSSE 120—Rose Hulman Institute of Technology

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
- Add to end of module:

  -
    ```
    if __name__ == '__main__'
        main()
    ```

Two underscores

Great for testing!

What is Exception Handling?

- Mechanism to deal with special or "exceptional" cases in a program, like error conditions
- Developers can write code that
  - Detects exceptions while program is running
  - Deals with the exceptions

The Need for Exception Handling

- Separates main code from code for special cases
  - Keeps the main code clean
  - Makes the expected special cases clear
  - Says “do these steps and if an exception occurs, handle it this way”
- Example:
  - Download from Lessons → Modules to Download … → Session 10 → slope.py
  - Try with:

    (5, 6) and **(10, 4)** **(3, 6)** and **(10, 12)** **(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 "The slope of the line is %0.3f." % (slope)
except ZeroDivisionError:
    print “The line has an infinite slope.”
```

Multiple "except" Clauses

- Like multi-way decisions with if-elif-else statements

```
try:
    <body>
except <errorType1>:
    <handler1>
...
except <errorTypeN>:
    <handlerN>
except:
    <defaultHandler>
```

Multiple "except" Clauses Example

- Try **(3, 300)** and **(a, 500)**
- Modify slope.py to include an except clause for a syntax error exception
- Try **(2, 100)** and **(4 100)** missing comma!
- Try
  - **>>> dir(__builtins__)**
    # shows names of exceptions

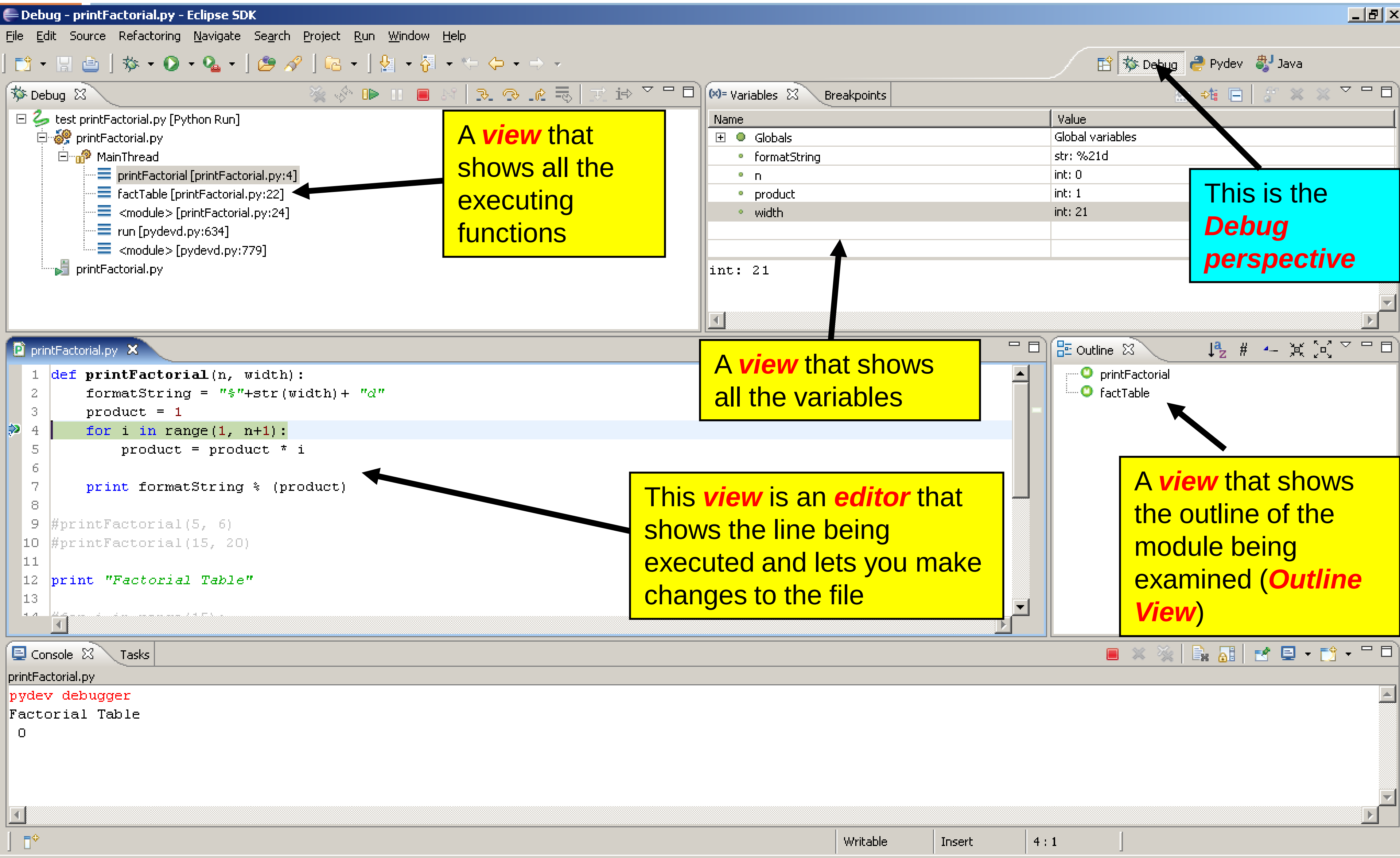
Debugging

- Debugging includes:
  - Discovering errors
  - Coming up with a hypothesis about the cause
  - Testing your hypothesis
  - Fixing the error
- Ways to debug
  - Insert print statements to show program flow and data
  - Use a debugger:
    - A program that executes another program and displays its runtime behavior, step by step
    - Part of every modern IDE

Using a Debugger

- Typical debugger commands:
  - Set a breakpoint—place where the debugger will pause the program
  - Single step—execute one line at a time
  - Inspect a variable—look at its changing value over time
- Debugging Example
  - Download printFactorial.py from Modules to Download → Session 10

Sample Debugging Session: Eclipse


Debug - printFactorial.py - Eclipse SDK
File Edit Source Refactoring Navigate Search Project Run Window Help
Debug
Pydev
Java
Debug
Variables
Breakpoints
printFactorial.py
Outline
Console
Tasks
Writable
Insert
4 : 1

```
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]
  printFactorial.py
```

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

```
int: 21
```

```
def printFactorial(n, width):
    formatString = "%"+str(width)+ "d"
    product = 1
    for i in range(1, n+1):
        product = product * i

    print formatString % (product)

#printFactorial(5, 6)
#printFactorial(15, 20)

print "Factorial Table"
```

```
printFactorial
factTable
```

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

A ***view*** that shows all the executing functions

This is the ***Debug perspective***

A ***view*** that shows all the variables

This ***view*** is an ***editor*** that shows the line being executed and lets you make changes to the file

A ***view*** that shows the outline of the module being examined (***Outline View***)

Tips 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!

Use the scientific method:
•hypothesize,
•experiment,
•fix bug,
•repeat experiment

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
- Typically 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?

Combining While Loops and Exception Handling

- Download getFile.py from Modules to Download → Session 10
  - Put file in same project that we've been using
  - Run the module with various inputs, like:
    - getFile.py
    - slope.py
    - notARealFileName
- Let's fix it!

Hint: You'll need code like the fixed version of this for your homework!

Speed Reading

- See Homework 10 instructions on Angel: Lessons → Homework → Homework 10
- Reading and ANGEL quiz due session 11
- Problem 3:
  - Due session 12, but you'll have more homework assigned session 11 so start now!
  - Pair Programming