

Decision Structures and Debugging

Rose-Hulman Institute of Technology
Computer Science and Software Engineering

Check out 08-DecisionStructures from SVN

Grading Status

- Programs through HW6 are graded
- See ANGEL for grades
- Update your projects and look for **CONSIDER**: comments

Two ways to loop through a list

- Directly

```
nums = [3.2, 8.1, 0.9]
```

```
total = 0
```

```
for num in nums:  
    total += num
```

```
for point in points:  
    cir = Circle(point, center)
```

- Using index

```
nums = [3.2, 8.1, 0.9]
```

```
    0     1     2
```

```
total = 0
```

```
for i in range(len(nums)):  
    total += nums[i]
```

```
for i in range(len(points)):  
    cir = Circle(points[i], center)
```

Mail box number on your quiz

Q1

CSSE120 Concept Map

Operations

+

Data

=

Objects

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

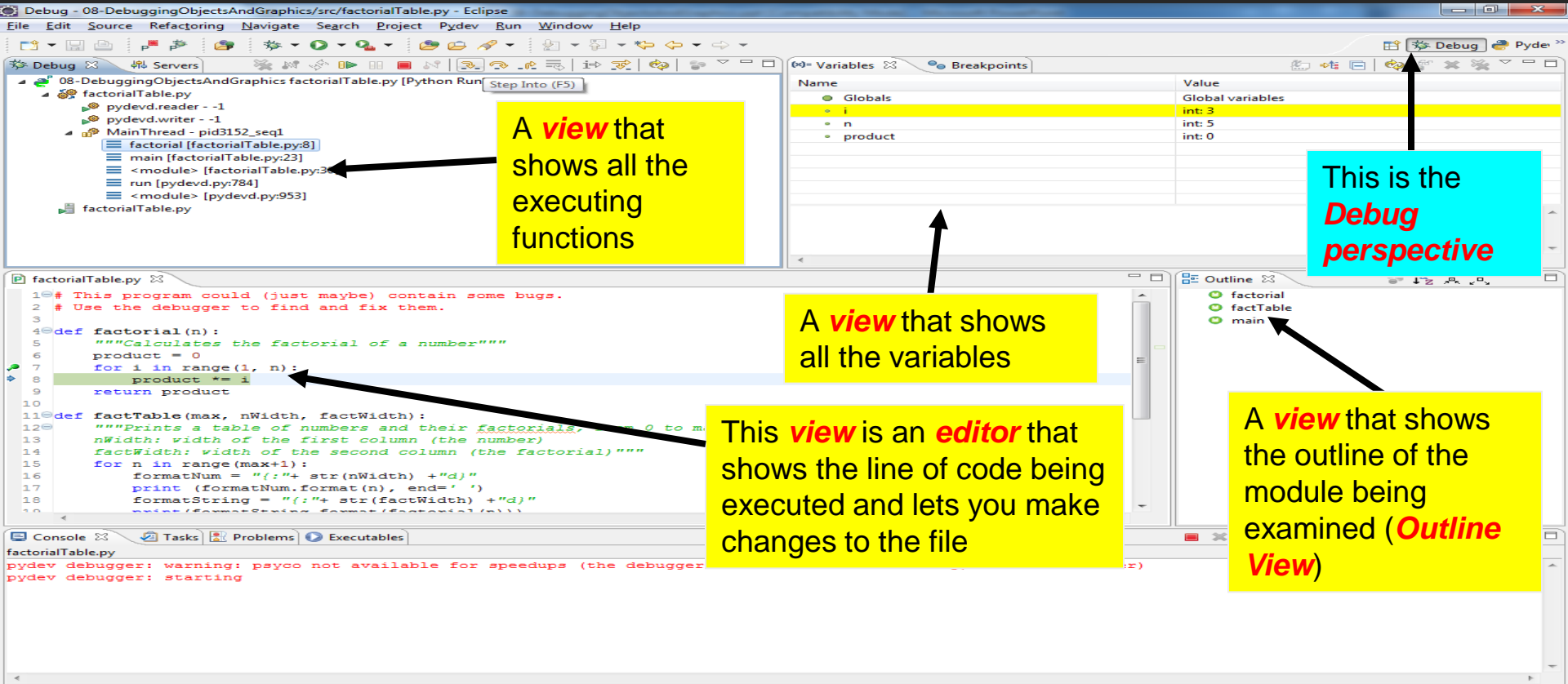
Using a Debugger

- Typical debugger commands:
 - *Set a breakpoint*—place where you want the debugger to pause the program
 - *Single step*—execute one line at a time
 - *Inspect a variable*—look at its changing value over time
- Debugging Example
 - In the **factorialTable.py** file
 - Start a debugging session:
 - Window → Open Perspective → other → Debug → Okay
 - Click debug icon (top-left side of work bench)

Using the debugger in Eclipse

- Set a breakpoint
 - Double click in left margin of editor view
- Step over (when you know a function works)
 - Click step-over icon or use F6 key
- Variable inspection
 - Look at the new value of `i`, `cir` after each time through the loop

Sample Debugging Session: Eclipse



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 of code 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!

It's the scientific method!

- hypothesize,
- experiment,
- fix bug,
- repeat experiment

Q3

Control Freaks

- Typical: statements execute in order
- Sometimes we want other orders
 - What examples have we seen of this?
- Statements that alter the flow are called *control structures*

Decision, Decisions

- *Decision structures* are control structures that allow programs to "choose" between different sequences of instructions

Simple Decisions

- The if statement
 - if **<condition>**:
 <body>
- Semantics: "if the condition is **True**, run the body, otherwise skip it"

- Simple conditions
 - **<expr>** **<relop>** **<expr>**
 - $6 * 7 \geq 42$

Math	Python
<	<
≤	<=
>	>
≥	>=

Math	Python
=	==
≠	!=

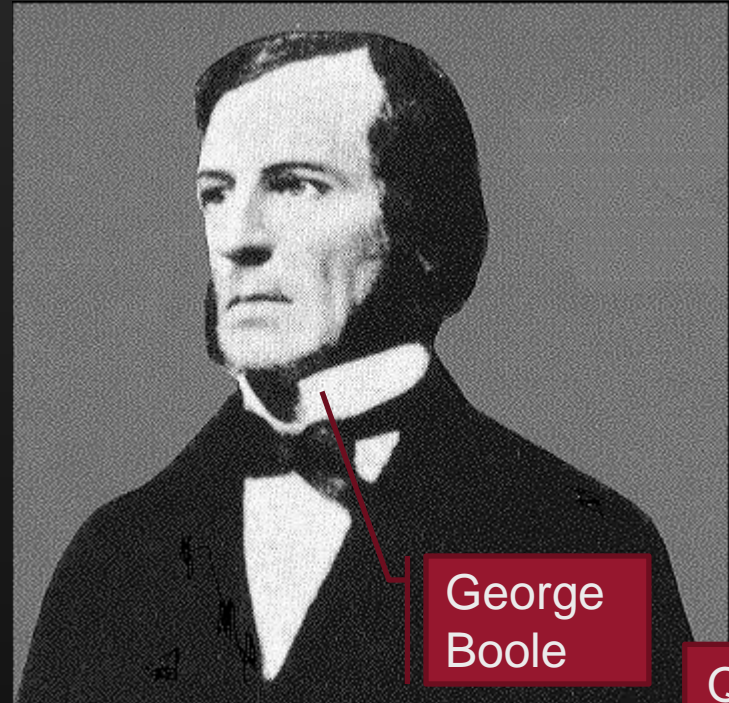
Q4

Exercise

- In module `grade.py`, define a function `grade(score)`
 - where `score` is from 0 to 100
 - and result is "perfect", "passing", or "failing" based on the score

More on Comparisons

- Conditions are Boolean expressions
 - Evaluate to **True** or **False**
- Try these:
 - >>> 3 < 4
 - >>> 42 > 7**2
 - >>> "ni" == "Ni"
 - >>> "A" < "B"
 - >>> "a" < "B"



George
Boole

Q5

Boolean Operators

- and, or, not

P	Q	P and Q
T	T	T
T	F	F
F	T	F
F	F	F

P	Q	P or Q
T	T	T
T	F	T
F	T	T
F	F	F

P	not P
T	F
F	T

Having It Both Ways: if-else

- Syntax:
if <condition>:
 <statementsForTrue>
else:
 <statementsForFalse>
- Semantics:
"If the condition is true, execute the statements for true, otherwise execute the statements for false"

A Mess of Nests

- Can we modify the grade function to return letter grades—A, B, C, D, and F?

Multi-way Decisions

- Syntax:

if <condition1>:

 <case 1 statements>

elif <condition2>:

 <case 2 statements>

elif <condition 3>:

 <case 3 statements>

...

else:

 <default statements>

Reach here if condition1 is false

Reach here if condition1 is false
and condition2 is true

Reach here if both condition1
and condition2 are false

Cleaning the Bird Cage

- Advantages of **if-elif-else** vs. nesting
 - Number of cases is clear
 - Each parallel case is at same level in code
 - Less error-prone
- Implement **gradeFixed** to use **if-elif-else** statement instead of nesting

Wrap up the quiz before starting homework

Finish the quiz

Q9,10

Complete the TODOs in countPassFail.py

Individual Exercise on Decisions

If you finish this, continue with rest of HW08

Reminder: looping through a list

- Directly

```
nums = [3.2, 8.1, 0.9]
```

```
total = 0
```

```
for num in nums:  
    total += num
```

```
for point in points:  
    cir = Circle(point, center)  
    cir.draw(win)
```

- Using index

```
nums = [3.2, 8.1, 0.9]
```

```
    0     1     2
```

```
total = 0
```

```
for i in range(len(nums)):  
    total += nums[i]
```

```
for i in range(len(points)):  
    cir = Circle(points[i], center)  
    cir.draw(win)
```