Class Overview

- Syllabus and Course Information
- Nested Statements and Functions
- Procedures

Class syllabus

- Copy available on-line
- Grading policies, quiz schedules, cheating penalties, etc. explained – read it!

Class web page

- www.cs.drexel.edu/cs123/spring2009
- bbVista page
  - learning.drexel.edu

Class schedule

- Lab every other week during the term
- Weeks 2, 4, 6, 8
  - Attendance required
  - 4 Labs
  - Verification sheet (one per lab team)
    - Do the work in lab
    - Get sheet signed by lab staff
    - Turn in verification sheet at end of lab period
    - Some labs may allow completion of work outside lab (see lab directions for these opportunities)
Other class requirements

• Take on-line quiz in weeks 3,5,7,9
  – Based on lab from week before.
  – Typically due 4:30pm Friday
  – Take quiz from any internet/browser-enabled computer.
  – Use CLC in UCROSS 147 for face-to-face help
    • Nearly constantly staffed in odd weeks
• Proficiency exam in week 10.

What counts towards class grade, what doesn’t count

• Doing lab work during lab.
  – Make up labs for those with valid excuse
  – Talk to your instructor to get into a make-up lab. Can get credit
    for a make-up lab only with instructor’s permission, since only
    they record your lab grades.
• Take quiz in week after the lab.
  – Some questions may involve dynamically generated components
    so you won’t get the exact same questions all the time.
  – Some questions on the quiz will have “how do I do” which allows
    you to find out if you got it right or wrong and try again.
  – May retake quiz as many times as you wish, up to deadline.
  – System can hold at most 150 people at once so avoid taking
    exams at the last minute.
• Proficiency exam is “traditional” – take it proctored in
  class in week 10.

What’s new in CS 123?

• Online consulting hours
  – Chat room where you can get course help
  – Check the course website for details
  – Send any suggestions for additional features
    to your instructor via e-mail

Nested Statements

• Nested expressions are statements which
  are placed inside one another
• We have been doing simple nesting all
  term
  – Every time we use list of lists
  – Every time we use parenthesis in expressions
  – Every time we call a function inside another

\[
3 \left( s + \frac{x}{(1 + x)^2} \right) + x^2 \left[ [1, 5], [2, 6], [3, 7] \right]
\]

\[
\sin(\cos(\pi + 5))
\]

Nested Conditionals

• It’s often useful to nest conditional (if)
  statements
• Say you want to check for a bunch of
  conditions and each condition has another
  set of conditions to check
  – Example: Converting GPA to +/- letter grades
    and counting how many of a certain grade

Nested If Example

```python
if grade>=3.6
    numAs := numAs+1;
    if grade >=3.8
        then results[i] := "A-";
        else results[i] := "A";
    end if;
    elif grade >=2.6
    then numBs := numBs +1;
        if grade >=3.3
            then results[i] := "B+";
            elif grade >=2.8
                then results[i] := "B";
                else results[i] := "B-";
                end if;
    end if;
```
Nested Loops

- Another useful operation is to nest loops within one another
  - Allows us to concisely perform operations on a list, table or list of lists
- The inner loop(s) run for every iteration of the outer loop

Nested Loop Syntax

```cpp
for i from 1 to n do
    for j from 1 to m do
        # some action
        end do;
    end do;
```

- The inner loop iterates m times
- The outer loop iterates n times
- Inner loop is run for each iteration of the outer loop
- Thus the action is performed a total of n*m times

Nested Loop Example

```cpp
for i from 1 to 3 do
    for j from 2 to i+1 do
        print("i=",i," j=",j);
        end do;
    end do;
```

```
1, 2
   "end inner loop"
2, 2, 3
   "end inner loop"
3, 3
```

```
Total = 30
roll1 = random()
roll2 = random()
endGame = false
while !endGame do
    while roll1 + roll2 != 2 do
        count = count + 1
    end do;
    if count = "plural " then
        plural = "a"
    else
        plural = "s"
    end if;
    print("You rolled for snake eyes", count, plural);
    total = total + count;
end do;
print("Average number of rolls required to get snake eyes = ", total/output);
```

Perform the following simulation:

- Roll two dice
- Repeat until snakes eyes
- Print out number of rolls

Print the average number of rolls required to get snake eyes

Advantages and Pitfalls

- Typically you want to perform certain operations in a certain order and the only way is to nest statements
- This flexibility allows you to do many things, but the disadvantage is that things can become overly complex
- Should not be necessary to go more than 2-3 levels deep
Scripts Recall

- Recall that scripts are useful snippets of code that solve some problem (encouraged in quizzes)
- You copy and paste the script and change the variables to quickly solve a particular class of problems

Procedures

- Procedures are functions which can be described in a series of steps
- Natural extension of script which eliminates the need for 'copy and paste'
  - Any function can be turned into a procedure
  - Any script can be turned into a procedure
- Use a procedure the same way you use a function
  - You have been using procedures all year (map, sum, rand …)

Procedure Syntax / Usage

```maple
exampleproc := proc(param1, param2, ...) local var1, var2, var3;
statement1;
statement2;
statement3;
...
return expression1;
end proc;
```

Simple Procedure Example

```maple
sumproc := proc(numToSumTil)
local a, total;
total := 0;
for i from 1 to numToSumTil do
    total := total + i;
end do;
return total;
end proc;
sumproc(50);
```

Procedure Background Information

- Procedures are isolated from the rest of the maple worksheet
  - Don’t rely on any variables outside the procedure definition
  - Every variable you need should be:
    - Provided as input using a parameter variable
    - Declared local with the ‘local’ keyword, meaning it only exists inside the procedure definition
  - ‘return’ keyword says what the final value of the function should be.

Converting From Script To Procedures

**Step 1: What variables are required by my script?**

- The list L
- The number of rows, numRows
- The number of columns, numCols

These are our parameters to the procedure, everything else depends on them. They are our ‘input’
Converting From Script To Procedures

Step 2: What variables are defined / used only within my script?

- The for loop variable, i
- The for loop variable, j

These are our local variables.

Step 3: Copy/edit the script commands into a procedure.

```plaintext
Local i, j;
for i from 1 to numSides do
    for j from 1 to numSides do
        print("%d * %d = ", i, j);
    end do;
end do;
print("\n"); # End the row of output and do:
return NULL;
end proc;
```

Step 4: Reuse it by calling it like a function. Enjoy.

```plaintext
ask(10); # [1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0, 2.1]
```

In-class exercise

- How do we convert the script for rolling dice?
  - Let’s create two procedures, one that runs one instance of the simulation
  - The second runs the simulation ‘numTrials’ times and prints the average

Recall the script on the next slide:

```plaintext
In-class exercise:

- How do we convert the script for rolling dice?
  - Let’s create two procedures, one that runs one instance of the simulation
  - The second runs the simulation ‘numTrials’ times and prints the average

Recall the script on the next slide:

```plaintext
total := 0;
roll(sides); # [1,2,3,4,5,6]
end;
for i from 1 to numTrials do
    count := 0;
    while roll(sides) <> roll(sides) do
        count := count + 1;
        print("\n"); # End the row of output and do:
    end do;
    print("\nAverage number of rolls to make snake eyes = ", count / i);
end do;
```

Perform the following simulation:

- Roll two dice
- Repeat until snakes eyes
- Print out number of rolls

Print the average number of rolls required to get snake eyes

- Roll two dice
- Repeat until snakes eyes
- Print out number of rolls

```plaintext
Perform the following simulation:

- Roll two dice
- Repeat until snakes eyes
- Print out number of rolls

Print the average number of rolls required to get snake eyes

```plaintext
roll(sides); # [1,2,3,4,5,6]
end;
```
What you should do now?

• Connect to class web page: www.cs.drexel.edu/cs123/spring2009
• Start up Maple 12
• Read Lab 1 directions.
• Do the work with your partner(s). Both should try to do the work, but the grader will need to look at only one answer to give you credit for doing the problem.
• Objective – finish at least through 2.1 in lab; 2.2 through 2.4 may be shown to grader during Lab 2

Finishing up – save files

• Make sure your name/user id/section number/ date, time/instructor name are on the verification sheet.
• Get the verification sheet signed and handed in.
• Save worksheet on desktop if you haven’t done so already.
• Submit a copy to Blackboard site.
• Email a copy to yourself and/or your lab partners as an attachment so you can look at what you did for review purposes later.
• You may finish 2.2-2.4 outside of lab, bring it next time.

Next week – Take quiz 1!

• Take the first quiz, starting next week (monday morning).
• Go to CLC if you need face-to-face help
• Don’t wait until the last minute to discover that you needed more time to complete the quiz!

• Don’t forget about 2.2-2.4 next week.