CSC 171 - Introduction to Computer Programming

Lecture #5 - Algorithms and Program Development

What is an Algorithm?

• Process or a set of rules to be followed in calculations or other problem-solving operations more informally a recipe for solving a problem.

Example : Square Root Algorithm

- 1. Guess the square root of the number
- 2. Divide the working number by the guess
- 3. Average the quotient (from 2) and the guess
- 4. Make the new guess the average from step 3
- 5. If the new guess is "sufficiently different" from the old guess, go back to step 2, else halt.

Algorithm vs. Program

- An *algorithm* is a description of how to solve a problem
- A *program* is an implementation of an algorithm in a particular language to run on a computer (usually a particular kind of computer)
- Difference between what we want to do and what we actually did

What's the Difference Really?

- We can analyze the algorithm independent of its implementation. This is the science in computer science.
- We can examine how easily, or with what difficulty, a language allows us to realize an algorithm.
- We can examine how different computers impact the realization of an algorithm.

Aspects of an Algorithm

- <u>**Detailed</u>** Provide enough detail to be implementable. Can be tricky to define completely, relies on "common sense"</u>
- <u>Effective</u> the algorithm should eventually halt, and halt in a "reasonable" amount of time. "reasonable" might change under different circumstances (faster computer, more computers, etc.)

Aspects of an Algorithm (continued)

- <u>Specify Behavior</u> the algorithm should be specific about the information that goes in (quantity, type, etc.) and the information that comes out.
- <u>General Purpose</u> algorithms should be idealized and therefore general purpose. A sorting algorithm should be able to sort anything (numbers, letters, patient records, etc.)

A Lot To Do!

- That is a lot to do for the burgeoning programmer.
- Get better as we go along, but good to know what the standards are!

Aspects of a Program: Readability

- We will emphasize, over and over, that a program is an essay on problem solving intended to be read by other people, even if "other people" is you in the future!
- Write a program so that you can read it, because it is likely that sometime in the future **you will** have to read it!

Readability - Naming

- The easiest thing to do that affects readability is good naming
 - Use names for the items you create that reflect their purpose
 - To help keep straight the types used, include that as part of the name. Python does not care about the type stored, but you do!
 - Remember "lower with under"

Bad Code vs. Good Code – Bad Code

```
a = input("Give a number? ")
b, c = 1, 0
while b<=1:
    c = c + b
    b = b + 1
print(a, b, c)
print("Result is ", c/b-1)</pre>
```

Bad Code vs. Good Code – Good Code

```
limit_str \
                                 = input("Range is from 1 to your input: ")
limit_int = int(limit_str)
count_int = 1
sum_int = 0
while count_int <= limit_int:
        sum_int = sum_int + count_int
        count_int = count_int + 1
average_float = sum_int/(count_int - 1)
print("Average of sum of integers from 1 to ",
limit_int, " is ", average_float)</pre>
```

Readability - Comments

- Info at the top, the goal of the code
- Purpose of variables (if not obvious by the name)
- Purpose of other functions being used
- Anything **tricky**. If it took you time to write, it probably is hard to read and needs a comment

Rule 6

- If it was hard to write, it is probably hard to read.
- Add a comment!

Bad Commenting Style

```
# Calculate the average of a sum of consecutive
integers in a given range
# Input the value
limit_str \
                                 = input("Range is from 1 to your input:")
# Convert the input string to an input
limit_int = int(limit_str)
# Assign 1 to the counting variable
count_int = 1
# Assign 0 to the sum
sum int = 0
```

```
# While loop runs while the counting variable is
smaller tha nth input value
while count_int <= limit_int :
    # Add the count and the sum, reassign to sum
    sum_int = sum_int + count_int
    # Add one to the count
    count_int = count_int + 1
# Calculate the average
average_float = sum_int/(count_int - 1)
# Print the result
print("Average of sum of integers from 1 to ",\
    limit_int, " is", average_float)
```

Better Commenting Style

```
# Add up the integers from 1 to the upper limit
while count_int <= limit_int :
    sum_int = sum_int + count_int
    count_int = count_int + 1
# Calculate and the average
average_float = sum_int/(count_int - 1)
print("Average of sum of integers from 1 to ",</pre>
```

```
limit int, " is", average float)
```

Readability - Indenting

- Indenting is a visual cue to say what code is "part of" other code.
- This is not always required as it is in Python, but Python forces you to indent.
- This aids readability greatly.

More Aspects of Programming

- <u>*Robust*</u>: As much as possible, the program should account for inputs that are not what is expected. More on this with error handling in Chapter 14
- <u>Correct</u>: Our programs should produce correct results. Much harder to ensure than it looks!

The Problem is "Problem-Solving"

- Remember, two parts to our goal:
 - Understand the problems to be solved
 - Encode the solution in a programming language, e.g. Python

Mix of Both

- The goal in each class is to do a little of both: problem solving and Python
- It is terribly important that we impress on you to try and understand how to solve the problem **first** before you try and code it.

Steps to Problem Solving

- Engage/Commit
- Visualize/See
- Try it/Experiment
- Simplify
- Analyze/Think
- Relax

Engage

- You need to commit yourself to addressing the problem.
 - Don't give up easily
 - Try different approaches
 - Set the "mood"
- Just putting in time does not mean you put in a real effort!!!

Visualize/See the Problem

- Find a way that works for you, some way to make the problem tangible.
 - Draw pictures
 - Layout tables
 - Literally "see" the problem somehow
- Everyone has a different way, find yours!

Try it/Experiment

- For some reason, people are afraid to just **try** some solution. Perhaps they fear failure, but experiments, done just for you, are the best way to figure out problems.
- Be willing to try, and fail, to solve a problem. Get started, don't wait for enlightenment!

Simplify

- Simplifying the problem so you can get a handle on it is one of the **most powerful** problem solving tools.
- Given a hard problem, make is **simplier** (smaller, clearer, easier), figure that out, then ramp up to the harder problem.

Think It Over/Analyze

- If your solution isn't working:
 - -Stop
 - Evaluate how you are doing
 - Analyze and keep going, or start over.
- People can be amazingly "stiff", banging their heads against the same wall over and over again. Loosen up, find another way!

One More Thing, Relax

- Take your time. Not getting an answer right away is not the end of the world. Put it away and come back to it.
- You'd be surprised how easy it is to solve if you let it go for awhile. That's why **starting early** is a luxury you should afford yourself.

Square Root Algorithm

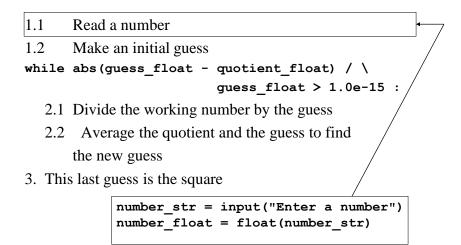
- 1. Get a number and make an initial guess of its square root
- 2. As long as the guess isn't close enough:
 - 2.1 Divide the working number by the guess
 - 2.2 Average the quotient and the guess to find the new guess
- 3. This last guess is the square

- 1. Get a number and make an initial guess of its square root
- 2. As long as the guess isn't close enough:
 2.1 Divide the working number by the guess
 2.2 Average the quotient and the guess to find the new guess
 3. This last guess is the square
 while abs(guess_float quotient_float) / \

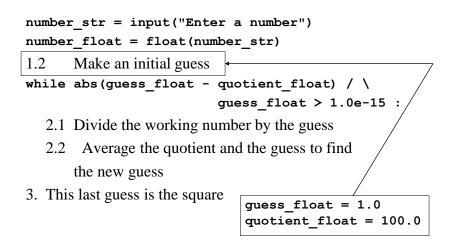
guess float > 1.0e-15 :

Refining the Square Root Algorithm

 Get a number and make an initial guess of its square root while abs(guess_float - quotient_float) / \ guess_float > 1.0e-15 :
 2.1 Divide the working number by the guess
 2.2 Average the quotient and the guess to find the new guess
 3. This last guess is the square
 1.1 Read a number
 1.2 Make an initial guess



Refining the Square Root Algorithm



Refining the Square Root Algorithm

print("The square root of ", number_str, " is ",\
 guess_float)

SquareRoot.py

Find the square root of a number

Input the number and make an initial guess number_str = input("Enter a number") number_float = float(number_str) guess_float = 1.0

You need an initial value of the quotient for the loop's condition quotient_float = 100.0

```
print("The square root of ", number_str, " is ",\
guess_float)
```

Reminder, Rules so Far

- 1. Think before you program!
- 2. A program is a human-readable essay on problem solving that also happens to execute on a computer.
- 3. The best way to improve your programming and problem solving skills is to practice!
- 4. A foolish consistency is the hobgoblin of little minds
- 5. Test your code, often and thoroughly
- 6. If it was hard to write, it is probably hard to read. Add a comment.