

Chapter 3

Algorithms & 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

Detailed: Provide enough detail to be implementable. Can be tricky to define completely, relies on “common sense”

Effective: 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 (2)

Specify Behavior: the algorithm should be specific about the information that goes in (quantity, type, etc.) and the information that comes out.

General Purpose: 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(2): 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!

What Does this Do?

```
a = input("give a number: ")
b,c = 1,0
while b <= a :
    c = c + b
    b = b + 1
print (a,b,c)
print ("Result: ", float(c)/b - 1)
CL03-1.py
```

```
# average of a sum of integers in a given range
limit_str = input("range is 1 to 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)
```

CL03-2.py

Readability(3): 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

Readability(4): 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.

Aspects of Programming (2)

Robust: 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

Correct: Our programs should produce correct results. Much harder to ensure than it looks!

More on Problem Solving

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

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

From “Problem Solving”,

DeFranco & Vinsonhaler

Be Proactive

See it

Simplify it

Stir it up

Pause and Reflect

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 it simpler (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.

Babylonian Square Root Example

Algorithm

1. Guess the square root of the number.
2. Divide the number by the guess.
3. Average the quotient (from step 2) and the guess
4. Make the new guess the average from step 3.
5. If the new guess differs from the old guess by more than the specified tolerance, go back to step 2; otherwise stop

Program Algorithm Skeleton

User provides 3 inputs: an integer number to find the square root of, an integer initial guess, a floating-point tolerance.

Apply the algorithm

Output the original conditions, its square root and the number of iterations.

CL03-5.py