# The Practice of Computing Using PYTHON

William Punch Richard Enbody

## Chapter 3

# Algorithms & Program Development

Addison-Wesley is an imprint of PEARSON

## 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: Summing Numbers

- Sum the numbers between 1 and 100 (inclusive)
- Think about this; we’ll come back to it later

## 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 do**

## What’s the Difference, Really?

- we can analyze the algorithm independent of its implementation.
- 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.)

## 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 = raw_input("give a number: ")
a = int(a)
b,c = 1,0
while b <= a :
    c = c + b
    b = b + 1
print "Result:", float(c)/(b - 1)
```

```
# average of a sum of integers in a given range
limitStr = raw_input("range is 1 to input: ")
limitInt = int(limitStr)
countInt = 1
sumInt = 0
while countInt <= limitInt:
    sumInt = sumInt + countInt
    countInt = countInt + 1
average = float(sumInt)/(countInt - 1)
print "Average:", average
```

# 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!

## Example: Summing Numbers

- Sum the numbers between 1 and 100 (inclusive)

# Example: Summing Numbers

- Sum the numbers between 1 and 100 (inclusive)
- Can write a loop:

```
sum = 0
for i in range(1, 101):
    sum = sum + i
print sum
```

- Is there a better way?

## 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

## 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.

# Multiplication Algorithm

- How to the find the product of two numbers using only addition (no multiplication)?

## Multiplication Algorithm

1. Get input integers a and b
2. Set sum to 0
3. Add a to the sum b times

## Prime-Verification Algorithm

- How to determine if a positive integer is prime?

## Prime-Verification Algorithm

```
1. Get input integer n
2. For each number i from 2 through sqrt(n)
    1. If n % i == 0, n is not prime, exit loop
3. If loop completed normally, n is prime
```

## Babylonian Square Root Example

## Square Root Algorithm

- How to find the square root of an integer?
- This is difficult!

## Square Root Algorithm

1. Get input integer n
2. Guess the square root of n
3. Divide the n by the guess
4. Average the quotient (from 3) and the guess
5. Make the new guess the average from step 3
6. If the new guess is “sufficiently different” from the old guess, go back to step 3, else halt.