PROBLEM SOLVING

A computer system consists of two essential parts: its hardware and software. “Hardware” generally refers to the tangible pieces that you can touch: the actual physical components such as the CPU (central processing unit), memory, I/O devices, and the network. “Software” on the other hand consists of the programs stored on the computer that we want to run. The software is what gives the machine its behavior, and allows it to do useful work (or play). Software includes the operating system, which is responsible for maintaining the hardware in good working order. Besides the operating system is the application software that we use on a regular basis like Web browsers and Microsoft Office.

To illustrate the difference between hardware and software, think of a restaurant. In one sense, when we “go to a restaurant,” we are thinking of the building location that houses the restaurant. In another sense, when we are “inside the restaurant,” we are enjoying the food and its atmosphere, which are a product of the restaurant as a business. Someday the restaurant (business) might move to a new location (building). It’s the same food, employees, clientele and atmosphere, but in a different location. The old location has been taken over by a different business, though its exterior still looks the same.

In this course and in several other courses offered by the computer science department, you learn how to write your own software, so that you can have the computer do exactly what you want and solve your specific problems. We are generally not interested in designing new computer hardware – that is closer to the subject of computer engineering, rather than computer science.

The heart of any computer program is its *algorithm*. The algorithm tells us in English how we go about solving the problem. As mentioned before, there are three important features of any successful algorithm:

- Unambiguous: This means that anybody should be able to follow your directions.
- Detailed: The algorithm must be precise in its description of how the input, output, variables, and operators are to be used.
- Deterministic: The order in which the steps are taken must be absolutely clear. When one step is completed, it should be obvious what the next step in the recipe should be.

Problem-solving strategy

Whenever we wish to solve a problem in computer science, it is important to adhere to a general problem-solving strategy. The following table shows how I like to remember it, with these five steps. Beside each step, I have indicated a common obstacle that could occur at that point in the process.

#	Problem-solving steps	Common obstacle
1	Understand the requirements of the problem, including its inputs and outputs.	The requirements are vague or ambiguous, or otherwise we don't understand what the problem is asking us to do. In this case, we seek clarification.
2	Write the solution in English "pseudo-code". This step is the most important, and this is where we need to invest most of our time.	We understand the problem, but we don't know how to solve it!
3	Write code in a programming language, such as Python.	We know how to solve the problem by hand, but we don't know enough of the programming language to adequately convey our solution.
4	Compile the program. Note that the word "compile" means for the computer to translate your program from whatever language you typed it in, into machine (binary) language, which is the native language of the computer.	After we typed in the program, it does not compile because there are syntax errors. We need to re-edit the program.
5	Run and test the program.	The program compiles, but when we run it we (sometimes) get a run-time error or the output is incorrect. This probably means we made a mistake back in step 2 because our algorithm is flawed or incomplete.

Beginners often have trouble with this procedure because they try to do steps 2 and 3 at the same time. It is easier to type the computer code once you already have a good idea on how to solve the problem. Otherwise, you will spend lots of time editing your code and making mistakes.

After a program has been tested and it seems to work for all conceivable inputs, then we can decide to refine it further. For example, we may want to make the output more attractive, or make the input process more user-friendly. Or we may want to generalize the algorithm to solve a larger variety of input situations.

Steps 1-5 above comprise the "problem-solving procedure", in other words the steps you need to follow to solve a problem from scratch. But what if someone has told you how to solve the problem in English, and your job is to transcribe the pseudocode into Python? In this case, you are simply following steps 3-5, because 1 and 2 have been done for you already.

A FIRST LOOK AT COMPUTER PROGRAMS

Objectives:

- Define a computer program, and describe what it looks like
- Identify certain things that we expect to find in computer programs, such as comments, statements that perform input, calculations, and output
- How to run a program

Why are we interested in creating our own software? The main reason is that we want to harness the computational power of the computer, to have direct control of the machine. Sometimes, existing software that is installed on the computer is not sufficient for us. It doesn't do exactly what we want. Writing our own programs can be extremely useful and even fun for people to use, such as a game, or converting data to an image.

In order for us to create software, we need to learn a computer programming language. There are thousands of programming languages, but only a few have become especially famous in computer science, such as Java, PHP, Python and C++. These languages are machine independent. This means that if you create a program in such a language, it should be able to run on any type of machine. Programming languages have been around for about 60 years, and the modern ones used today have many built-in features that simplify coding. Many common tasks and calculations are pre-defined as part of the language, such as sorting data, opening files, surfing the Web, creating a graphical user interface, etc.

A computer program is a list of instructions written in a computer language that solves some problem. In this course, we will use the Python language to write our programs. Python is a computer programming language. The text that makes up a computer program is called *source code*, or simply "code" for short. Note that this use of the word code is a mass noun, similar to the word money. In other words, when talking about source code, we always say code in the singular.

When we get into lab, a certain routine is going to emerge. Here are some of the steps.

1. A program is submitted to the computer.
2. We run the program.
3. As the program is running, it may ask for input from you.
4. When the program is finished, we look at its output.

To gain confidence at writing short computer programs, we need to get into the habit of *reading* some computer code. Study carefully the code examples in this short book and ones we do in class. Eventually, you should be able to read several lines of code, and have a precise idea in your mind of exactly what it is doing.

There are a few reasons why learning a programming language like Python will be much easier than learning a natural language like French:

- There are very few words to learn.
- Those words are already in English.
- The grammatical rules are very simple.

So, the challenge of computer problem solving is not learning the language. It is that writing a computer program forces us to think in a way that is logical, organized, and precise – more so than in everyday English.

What does a computer program look like? In essence it works like a recipe: it must list all of the necessary ingredients (data) and instructions (steps) that the computer needs to obey. Cooking may be a good analogy, because it solves an important problem: "I'm hungry!" ☺ What do we generally see in recipes? Well, here is one:

- Brown the beef for 15 minutes. Drain the grease.
- Dice carrot, celery, and onion (otherwise known as mirepoix)
- Cut up and boil 6 potatoes until soft.
- Mash the potatoes.
- Add flour, spices, sauce, and mirepoix to the beef.
- Put the meat mixture into a casserole. Top with potatoes.
- Bake in the oven at 400 degrees for 30 minutes.

Discussion:

1. What dish is this recipe preparing? (In other words, what is the output?) How can you tell?
2. List all the ingredients (input) of the recipe. Why is it important to know all of the ingredients before you start the recipe?
3. Which steps in the recipe imply that we need to continue or wait for something, or do some process repeatedly?
4. At what point(s) in the recipe would it be necessary for a cook to check the progress of a step or to make a decision? What are the decisions or judgments that need to be made?

A simple program will have these elements.

- Comment(s)
- Input statement(s)
- Calculation statement(s)
- Output statement(s)

Here is an example program. Programs are usually contained in files, and this one may be called hello.py. By convention, the name of a file containing Python source code will end in ".py".

```
# hello.py
# This program will ask the user to enter his/her name,
# and then say hello to that person.
# This program is very simple.  There are no calculations.

# Input
name = input("Please tell me your name:  ")

# Output
print("Hello, ", name, ", nice to meet you!")
```

Here is a second example program called animals.py.

```
# animals.py
# Let’s illustrate simple I/O and calculations.
# Add up how many animals on a farm.

horses = int(input("How many horses?  "))
pigs = int(input("How many pigs?  "))
chickens = int(input("How many chickens?  "))

total = horses + pigs + chickens

print("You have a total of ", total, " animals.")
```

Computer programs share one thing with poetry: they are arranged in lines. Lines that begin with a # symbol are called comments. Comments are included in a computer program for the benefit of the human reader. The computer actually ignores comments. It is extremely important for us to put comments in our programs to help us remember the purpose of a program. Also, notice that some of the lines are blank. Python allows us to format our code to make the program more readable to us. The remaining lines are the actual Python statements. In the first example above, there are just two Python statements. One performs input, and the other does output.

Let’s look at a more interesting program, one that computes the area of a circle. It can be stored in a file called circle.py:

```
# circle.py
# This program will find the area of the circle whose
# radius is supplied by the user.
# Later on, we could modify the program by having it
# also compute the circumference.

import math

# Input
radius = float(input("What is the radius of the circle?  "))

# Calculations
area = math.pi * radius * radius

# Output
print("The area of the circle is ", area, ".")
```

This program contains four statements.

1. The first statement "import math" tells Python that we intend to make use of some special mathematical constants or functions. In this case, we want to use Python's built-in constant value for pi rather than typing it in ourselves.
2. An input statement for the circle's radius. We use Python's built-in input() function to get what the user types. We then ask Python to convert this input into a real number by using the float() function.
3. A calculation statement where we specify the formula for the area of a circle.
4. The output statement.

Many of the programs we will write in this course will have this simple structure: input statements come first, then calculations, and then finally output. The Python system will execute the statements in the order that they are specified.

One important thing to note is that *the computer has no common sense*. It may be an impressive machine, but we do have to tell the computer how to calculate the area of a circle. In a computer program, we must spell out details like this, because the computer is very good at following our literal instructions. If we enter the area formula incorrectly, the computer will never warn us we made that kind of mistake.

Well, now you have seen some computer programs. Let's examine them closely. What do they contain?

- Statements – A statement performs one step of the program. Each "line of code" may have one statement. There are several types of statements in Python that we will study. Sometimes, we will use the words instruction or command interchangeably with the word statement. However, in later courses, you will see that there is a slight difference in meaning between these words.

- Variables – A variable is a place in memory to hold information, such as a person's name or the area of a circle. Every variable has a name, and the name of a variable is called an identifier. Variables have an implicit type, such as integer, float (i.e. real number), and string (i.e. text).
- Constants
- Expressions – often the stuff you see in parentheses or on the right side of an = sign.

The way that we type out programs should make them easy to read.

- Whitespace – This includes spaces, tabs and newlines. They are generally ignored by the Python system. But they are necessary for human readability. For example, please put a space on either side of an operator like + or =.
- Indentation – Sometimes, statements need to be grouped together, and we convey this by indenting. We won't need to do this yet.
- Continuation – Lines of code should not exceed 80 characters. If for some reason you do need a long statement, put a backslash (\) at the end of the line, and continue typing the statement on the next line. I think we will rarely need to have such long lines in this course.

At this point, you may notice that some words we use are also found in ordinary mathematics. For example, in algebra, one talks about constants, variables, and expressions. One major difference between computing and algebra terminology has to do with the = sign. In algebra, the = sign signifies an equation. An equation is a fact stating that two numbers are the same. But in computer programming, we don't use the word equation, and we don't work out equations as you would in algebra. A computer program is a list of instructions for the computer to perform, rather than simply a list of facts about numbers.

Discussion:

1. What is a computer program?

2. What does a computer program look like?

3. Does a program resemble anything you have seen before?

4. What little things do we see in a computer program?

5. How can we tell if it works correctly or not?

6. If incorrect, what could have gone wrong?

7. What different kinds of statements have we seen so far?