# CS118 Problem Solving

**Definition of Problem Solving**

***Problem solving is the act of specifying a sequence of tasks which use existing tools and information, and which are executed to produce a desired end result.*** [i.e. Problem solving is create a solution from steps we know how to do.]

**Applying the act of Problem Solving through the use of computers:**

Problems can be solved by numerous methods, therefore each approach can have a different number and/or sequence of **tasks**.

***Tasks*** are efforts which have their own purpose in achieving the goal of the main task:

- End Result: all or part of the end goal of the parent task, or
- Intermediate values: the generation or organization of information for other tasks to utilize, or
- Flow Control: control the execution of other tasks.

***Tools***

Tools are the fundamental instruments which can accomplish tasks in computational problem solutions. Every problem-solving mechanism relies on a set of tools that are fundamental. These are the things which "we know how to do". More about these on the next page.

***Information***

The sequence of tasks is constructed such that the information available flows through the tasks – if information is needed by a task, it must be:

- a fact that is constant for every instance of the problem (*Source*: **known**)

  We generally are concerned only with information unique to the problem being solved. Some items we will not consider when breaking down our problem:

  Formulas: To humans, a formula is information. But that is because we use that information to determine a set of steps so we can determine an answer. When we are determining a set of steps that (when applied) will give us a numeric answer, we see that a formula is just a concise way of expressing some of those steps. Therefore: **Formulas are methods, they are not data** – we will not treat them like information.

  Numeric constants like $\pi$, or $e$ or $\hbar$ are not information we worry about manipulating since they are known, constant, and easily reproduced.

  But we *would* be concerned with values which limit the scope of the problem – such as how many times a simulation will run, or how many terms a series should use in its computation.

- information that is provided to the problem from outside (*Source*: **input**)

- information that is computed by previous actions of the problem solving (*Source*: **previous step**)

If the solution's tasks do not use all of the information available to the problem then either that information is not needed or a task is missing. You have no doubt seen this when a math/physics/chemistry word/story problem appears to provide more information than is needed – was it extra information or did you miss something in your solution? One of the tasks of the programmer is to create a solution which uses information as effectively as possible. Sometimes it may be more effective to bring in information (from the user or a file) while other times it may be more effective to get it from previous steps (i.e. to have the solution compute it).

**Be sure to refer to the Problem Solving Standard Format document**

**INPUT, OUTPUT, DEFINE**

There are five commonly-used tools (and one rarely used tool) for planning a solution to a computational problem. Three of these are simple, intuitive tools. They are like a hammer or a screwdriver – obvious in their purpose and simple in their use. These are listed below with an explanation for each.

**INPUT**
Bring information into the problem solution from outside – for example: from other processes, secondary storage (e.g. disk file), sensors, or the user. Without saving this would be useless, so there is an implied "DEFINE" included with the INPUT tool and you do not need a separate DEFINE tool.

**OUTPUT**
Send information out of the problem solution – for example, to other processes, secondary storage (e.g. disk file), or the user.

**DEFINE**
Associate information with an identifier (i.e. a "variable") for later use in the problem solution. Also used as a tool for making changes to data.

## For each of the exercises in this assignment, provide a <u>separate</u> simple ASCII .TXT file – use Notepad on Windows and TextEdit on Mac.

Name the files as `EX09_01.txt`, `EX09_02.txt`, `EX09_03.txt`, etc

**1.** Regarding the microwave analogy from an earlier exercise:

*Analogy*
When you press buttons on a microwave oven, you are providing instructions to the device ("Turn on for 60 seconds") and then you execute those instructions by pressing another button ("Start"). Some microwave ovens permit you to provide longer "programs" ("Cook for 5 minutes at 100% power, then cook for 10 minutes at 50% power"). Those instructions are not saved, so you get to enter them every time you want to cook something. The analogy applies to computer programs except that we'll save our "cooking instructions" for re-use. And they'll be much more complex in what they can accomplish.

a. What are possible INPUTS to that "program"?

b. What are possible OUTPUTS from that "program"?
(Remember: Outputs do NOT have to go to the user.)

For each of the following exercises, prepare a Standard Format solution that uses any or all of the four tools specified above – INPUT, OUTPUT, DEFINE. Use only one tool per task. Follow the format of the example.

*Example:*
*Have a computer program determine the time it will take to drive a car for a distance and speed provided by the user, then store the distance, speed, and time on disk.*

Solution:

```
INPUT from user: "distance"
INPUT from user: "speed"
DEFINE "time" using distance, speed
OUTPUT to disk: distance, speed, time
```

Demo program – This is an implementation of the solution shown above:
From Canvas Files in the Demos folder, download this file: `demo_car_time.pyc`

Double-click on the file you downloaded and answer the questions as they appear. The steps of the solution have been implemented in the program so you can see how a program is an *implementation* of a *solution*.

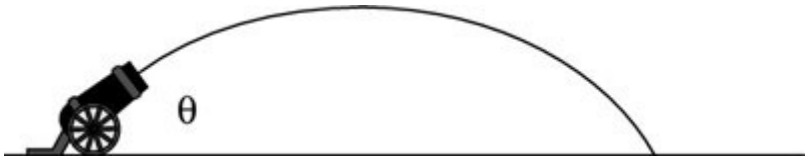
Understanding:
· If you don't understand how the problem statement gets to the solution provided, ask.
· If you don't understand how the solution provided relates to the demo program, ask.

Once you understand the example, try these exercises:

**Remember: You are writing *solutions* which will use the problem solving tools to describe what a computer program will do – you are not writing computer programs!**

**2.** Write a Standard Format solution for this problem:

*Credit to: https://www.dummies.com/education/science/physics/calculate-the-range-of-a-projectile-fired-at-an-angle/*




Have a computer program compute and display the horizontal distance a ballistic projectile travels for a user-provided initial velocity and launch angle. According to the reference site, the formula to compute this distance is:

$$s = \frac{2v_0^2 \ \sin\theta \ \ \cos\theta}{g}$$

where $v_0$ is initial speed, $\theta$ is the angle with the horizontal, and $g$ is the acceleration due to gravity on Earth. The formula assumes the projectile launches from the ground, stops at the ground, and that there is no air friction.

Your solution for this will be a set of steps using the five Standard Format tools: INPUT, OUTPUT, DEFINE, TEST, and REPEAT. For this problem there is no need to test, nor repeat so use only INPUT, OUTPUT, and DEFINE. You may need to use any or all of these more than once, and the order of the steps matters – for example, we cannot calculate `s` until we have all of the needed values.

Special note: Even though we humans have the above formula, the solution will not use it! It is only used so that we may determine what data will be needed and perhaps decide on how we wish to breakdown the problem. **We will never INPUT, DEFINE, OUTPUT, TEST, REPEAT or EXECUTE a formula.**

**3.** Write a Standard Format solution for this problem:
Have a computer program determine the GPA of the user for the most recent completed semester and save that information on disk. Assume the user took exactly four classes (all equal credits) and received a grade of 4, 3, 2, 1, or 0 in each class.

**TEST, REPEAT, EXECUTE**

These three tools are used to control flow within the solution. Two of them – TEST and REPEAT – are used in virtually all solutions of non-trivial size. The EXECUTE task is rarely used and simply provides the means to handle one specific job of the solution - you will likely not need it.

**TEST**
Compare information to decide whether specific tasks should be executed.

How it works:
1. The task numbers controlled by a TEST are known as the "action tasks". Action tasks will only execute if the TEST condition is true.

2. If a TEST condition is true, the action tasks are executed. When those are complete, the next task after the action tasks is executed and normal task processing continues. If the TEST condition is false, the action tasks are skipped and normal task processing continues with the first task after the action tasks.

**Testing**
Identify which tasks to perform when a condition is true. (Note that the tasks being executed need to be sequentially listed – you can't execute a task that isn't in sequence with the other tasks to be executed.)

For example:

```
TEST: if choice is "S", T9
TEST: if str is "Fred Flintstone", T12-T15
```

**No concept of ELSE**
In many programming languages (as in the English language) we have the concept of "else" – an alternative to perform when the testing results in a false condition. There is an ambiguity that arises with these languages and to avoid that ambiguity the concept of "else" does not exist in our problem solving plan – you must create an alternative testing task. In the example below, T10 provides what would normally be an "else":

```
T08: TEST: if choice is "S" or "T", T09
   T09: OUTPUT to user: message for a good choice
T10: TEST: if choice is not "S" and not "T", T11
   T11: OUTPUT to user: message that it's a bad choice
```

For each of the following exercises, prepare a simple Standard Format solution that uses any or all of the tools specified above, except for `REPEAT` and `EXECUTE`. Use only one tool per task. Follow the format of the example – note that the `TEST` and `REPEAT` tools require you to specify which tasks they are controlling, so you will need to number all of your tasks.

Example: Have a computer program determine the time it will take to drive a car for one of two routes each of known distance where the choice of route and average speed are provided by the user; then store the distance, speed, and time on disk.

Solution: Note that it is not possible for both T03 and T05 to have true conditions – thus, either T04 or T06 will execute, but not both.

```
T01: INPUT from user: "route_chosen"
T02: INPUT from user: "avg_speed"
T03: TEST if route_chosen = 1, T04
   T04: DEFINE "distance" as 120
T05: TEST if route_chosen = 2, T06
   T06: DEFINE "distainces" as 145
T07: DEFINE "time" using distance, speed
T08: OUTPUT to disk: distance, speed, time
```

**Remember: You are writing *solutions* which will use the problem solving tools to describe what a computer program will do – you are not writing computer programs!**

**4.** Write a Standard Format solution for this problem:
Have a computer program compute the horizontal distance a projectile travels and display it to the user but only if the angle with the horizontal is greater than 0 – otherwise, display an error message to the user and stop execution. [HINT: There is no tool to "stop execution". To stop executing means to have no more tasks to process.]

**REPEAT**
Have one or more tasks repeat when a condition is satisfied. This has its own comparison since repeating should eventually end.

**EXECUTE**
A rarely-used tool, this is for one specific circumstance – we will not be using it.

For each of the following exercises, prepare a Standard Format solution that uses any or all of the tools specified above, except for `EXECUTE`. Use only one tool per task. Follow the format of the example – note that the `TEST` and `REPEAT` tools require you to specify which tasks they are controlling, so you will need to number all of your tasks.

Example:
Have a computer program ask the user for a number between 1 and 5, inclusive. Continually ask the user to enter a number between 1 and 5 (inclusive) until s/he does so. Report to the user how many tries it took. Do not use more than five tasks to do this.

```
T01: DEFINE "count" as 0

   T02: INPUT "user_number"
   T03: DEFINE: add 1 to "count"
T04: REPEAT until user_number is between 1 and 5, T02-T03

T05: OUTPUT using count
```

**Remember: You are writing *solutions* which will use the problem solving tools to describe what a computer program will do – you are not writing computer programs!**

**5.** Write a Standard Format solution for this problem:
Have a computer program determine and display to the user the number of days the user ate breakfast over a year. Have it ask the user every day for 365 days if s/he ate breakfast and keep count of the number of days that answer was "Yes". Assume the user will answer the question only once per day. Do not use more than ten tasks to do this.