# General Statement

You must write this examination starting with **Template for Functions (chapter 5+).py** or **Basic function.py** (same file, different name).Refer to the **Style Requirements.pdf** document posted in Blackboard for more requirements. These instructions will not specifically state requirements that you are expected to follow from that document.

# The Problem

We’re building a simple role-playing game.

# Understand the Problem

## The Task

In this third-level simulation, a warrior is now defined as a set of 6 characteristics: strength, energy, speed, direction, location, and name. These characteristics are read from a file at the beginning of the simulation. **I will run your program with my own test file(s)**.

There are still two field locations, one place for each warrior. This means that the warrior’s position is being tracked with two variables, and they must be kept in sync. Any time you change a warrior’s location (including when it is read the first time), you must change the field location for that warrior as well.

The locations are numbers that indicate where the warrior is in a line moving left and right only; (field\_warrior1== 7) or (warrior1\_location == 7) means that warrior1 is in location 7. If a warrior’s direction is “right”, when it runs its location will go up; if its direction is “left” the location goes down when it runs.

You will define two warriors and move them around. After any action you will display the game state (all the current values of each warrior’s characteristics including the locations). The warriors keep moving until at least one warrior runs out of energy and dies.

## Specifics

Change all the placeholders in the program header to be your name, program name, what the program does, and the date. Set up the template with two functions and fill in the function headers with all appropriate values. Make sure that the header description and values match the code at all times.

Declare a Boolean variable that will be True as long as both warriors have energy but turns False if either one’s energy drops to 0 or below. **You must use this variable** to determine whether one or both warriors has died and it’s time for the simulation to end.

Declare variables for strength, energy, speed, direction, location, and name for each warrior. The variable names begin with warrior1\_ or warrior2\_ (we aren’t using “red” or “blue” anymore because each warrior now has a name characteristic (so, warrior1\_strength, etc.).

## Setup

The warrior characteristics are stored in a file named **warriors.txt**. There are two records in the file, six lines each. In order, the lines are [pay attention!] the warrior’s name, energy, strength, speed, location, and direction.

## Play (main program)

The steps are: (1) build the warriors, (2) display the starting state, (3) then while both warriors are alive (a, b) move each warrior in turn and (c) then display the current game state. After the loop finishes, (4) display who died.

### Build the warriors

Open **warriors.txt**, read all the characteristics for the first warrior into the appropriate variables, read all the characteristics for the second warrior, then close the file. Set time to 0.

### Display the game state (function display\_warrior\_status)

Display the time, call **display\_warrior\_status**() with the warrior1 variables as parameters, in the order specified in the function header. After this, call the function using the warrior2 variables.

### Do these things while both warriors are alive:

### Move the warriors

Move warrior1 by setting its location to the value returned by calling **move\_warrior**() using these parameters: warrior1’s speed, direction, location, and warrior2’s location.

if the value returned is negative, reverse warrior1’s direction and set its location to the positive (absolute value) of the returned variable.

Move warrior2 in the same way, passing it’s speed, direction, location, and warrior1’s location and processing the return value for a new location and/or direction.

Be sure to sync the field location for each warrior with its new value.

### Update the warriors’ conditions

Increase time by 1.

Subtract the square of each warrior’s speed from its energy.

Subtract twice the warrior’s speed from its strength.

### Display the new game state

Print the game state as described above

### Test the warrior’s conditions

If either warrior’s energy is 0 or less, that warrior dies (this might happen to both).

### Final report

Print the **name** of each warrior who is dead. If neither warrior is dead, print “ERROR” because you shouldn’t get to this point in the program unless one of them died.

## Function display\_warrior\_status()

Function **display\_warrior\_status**() has 6 parameters: name, energy, strength, speed, direction, location. Output is **one line** in this format:

<name>: (energy <val>, strength <val>, speed <val>, direction <val>) at location <val>.

The function does not return a value.

Be sure that the function header is complete: function name, description, the list of functions called, the list of parameters and their types, and the return value name and type.

## Function move\_warrior()

Function **move\_warrior**() has four parameters: speed, direction, location, and other\_location.

The return variable will be a new value (a new location) calculated from the location parameter. Set this variable to the value of location. Then do this as many times as the value for speed:

1. depending on the direction, either (right) add 1 to the new location or (left) subtract one from the new location
2. if the new value is the same as other\_location, then undo the addition or subtraction, reverse direction, and add or subtract 1 based on the new direction

Here is now a problem: if the direction has changed, then there are now two values to return: a new location and a new direction. There are many ways to address this problem.

**Do not return multiple values**. That is a problem that is solved with Lists (chapter 7) or similar techniques like structures or reference parameters in other languages. Python’s “multiple return” capability is really using Lists by seeming not to. When you use Lists I want you to be obvious and open about it. **That’s in the next assignment**.

Instead use the old time way this kind of problem was dealt with: it’s got its problems (biggest one: it misinterprets when a warrior legally moves to a location below 0), which is why newer methods developed. But I want you to see why quick solutions are not always best.

1. if the direction has changed, make the return value negative. In the main program, test the returned value and if it is negative, change the warrior’s direction and set the warrior’s location to the positive version of the new location.

Return the value of the new location.

Be sure that the function header is complete: function name, description, the list of functions called, the list of parameters and their types, and the return variable name and type.

### Extra Credit:

[5 points] Put both functions in a library named **warrior.py** and import warrior into the main program file.

[5 points] Use exceptions to handle errors in the file operations

## Special Notes:

1. **Do not use** lists *even if you know how to do those things* (that’s the next assignment). The purpose of this assignment is to see if you have the skills to use **only** the Python tools described in chapters 1-6 to solve this problem. Using more advanced features demonstrates that you don’t know the simpler tools well enough, so using them will **lower** your score.
2. You earn up to 60% by correct operation and up to 40% with good style, readability, and documentation.

Submit (in **Blackboard**) the file **combat.py**. If you do the extra credit, also submit **warrior.py**.

*Name the file(s)* ***exactly*** *as given*. **Do not add anything to the program name**. **Not** your name, **not** your ID number, **not** words like “assignment 3” or “revised”, or **any** additional characters as part of the file name.

The **Python** program is scored with a maximum value of **100** **points**.