

# Assignment 1

**Due** Monday by 4pm      **Points** 5

## CSC108H Assignment 1: Where's That Word?

**Deadline:** October 1, 2018 by 4:00pm

**Initial results:** October 3, 2018 by 4:00pm

**Resubmission with 20% deduction (optional):** October 5, 2018 by 4:00pm (no lates accepted)


**Final results:** October 9, 2018 by 4:00pm

**What is resubmission?** The A1 test results will typically be released within 48 hours of the deadline. You may choose to resubmit, fixing any errors detected by our tests, or to submit a late submission without the benefit of tests. These “resubmissions” will be accepted up until the deadline above with a 20% deduction. No late assignment resubmissions will be accepted (the assignment late penalty scheme does not apply to resubmissions). Since a 20% penalty is applied and the highest mark a resubmission can receive is 80%, the resubmission is most commonly used by students whose code had a minor error that resulted in many test cases failing. The majority of students do not and should not resubmit.

## Where's That Word?

In this assignment, you will implement a two-player word search game. There are several small parts to this task; all of them use concepts that you are learning about during the first three weeks of the course.

## Goals of this Assignment

- Use the Function Design Recipe ([design\\_recipe.pdf](#) ) to plan, implement, and test functions.
- Write function bodies using variables, numeric types, strings, and conditional statements. (You can do this whole assignment with only the concepts from Weeks 1, 2, and 3 of the course.)
- Learn to use Python 3, Wing 101, provided starter code, a checker module, and other tools.


We have prepared

[a video to show examples of how the game will work](#) 

(<https://www.youtube.com/watch?v=zX0fBYDNYfw>)

## How to not be interviewed for an academic offence

*"The truth is that on every campus, a large proportion of the reported cases of academic dishonesty come from introductory computer science courses, and the reason is totally obvious: we use automated tools to detect plagiarism"*

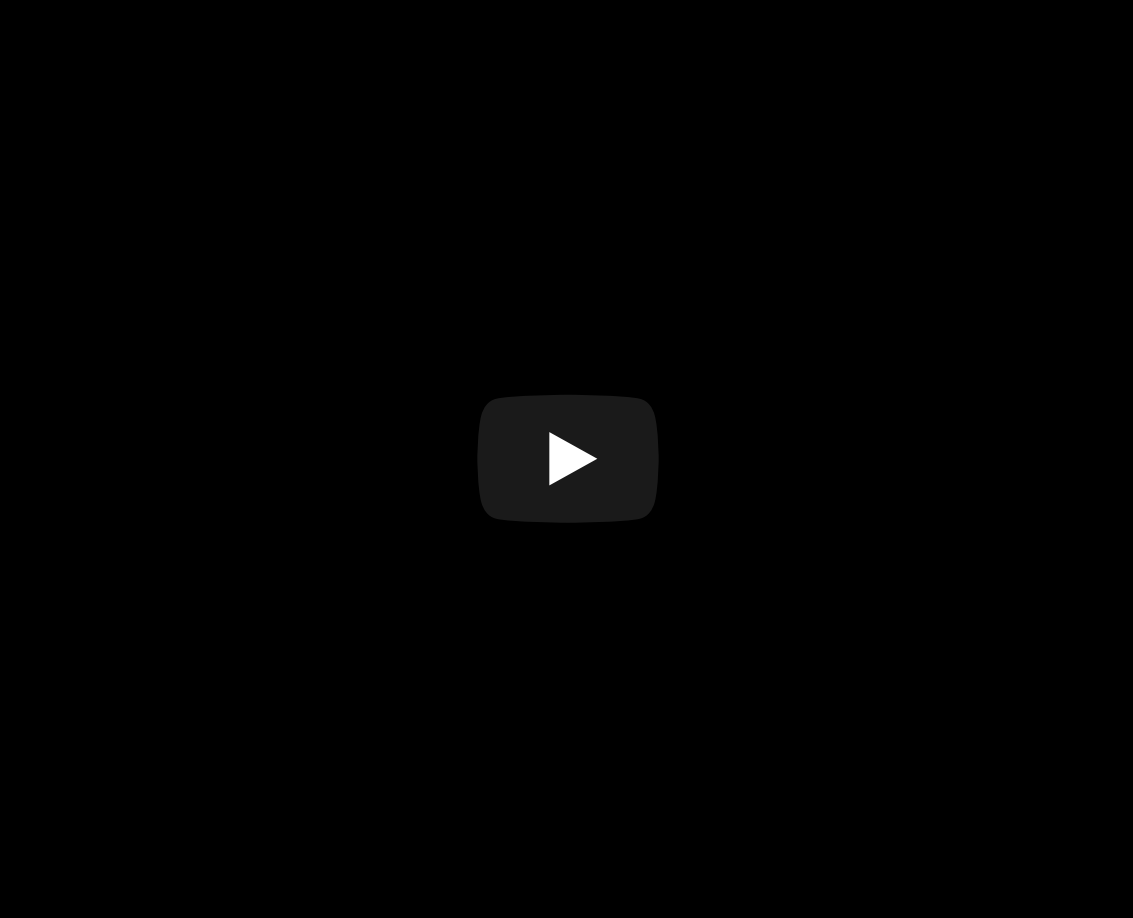
Professor Ed Lazowska, chair of computer science and engineering at the University of Washington, Why Computer Science Students Cheat ([Link](#)  ([https://www.pcworld.com/article/194486/Computer\\_Science\\_Students\\_C](https://www.pcworld.com/article/194486/Computer_Science_Students_C))).

We are very good at detecting cheating. We hate finding it, though, so we have written this short guide on how to avoid being interviewed by us for an academic offence.

A typical penalty for a first offence is a **zero on the assignment**. The case will also be entered into the **UofT academic offence database**. (If you get caught a second time ever as an undergrad, the penalties are much, much more severe.)

Our tips:

- Remember that figuring out *what* code needs to be written is the most important part of the assignment. Typing out code based on steps that someone else gave you is considered cheating, as you are submitting someone else's ideas as your own.
- Don't search the web for solutions. We've already done that and will be comparing what we found with what you submit. It



[Minimize Video](#)

when you complete the required functions.

## Starter code

For this assignment, we are giving you some files, including some Python starter code files. See the Files to Download section below for more information.

## Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Constants are special variables whose values do not change once assigned. A different naming convention (uppercase pothole) is used for constants, so that programmers know to not change their values. For example, in the starter code, the constant `FORWARD_FACTOR` is assigned the value 1 at the beginning of the module and the value of `FORWARD_FACTOR` should never change in your code. When writing your code, if you need to refer to the multiplicative factor of a forwards word (see Scoring the Game below), you should use `FORWARD_FACTOR`, rather than 1. The same goes for the other constant values.

Using constants simplifies code modifications and improves readability. If we later decide to use a different forward factor, we would only have to change the factor in one place (the `FORWARD_FACTOR` assignment statement), rather than throughout the program.

We have provided the following constants in the starter code for you to use in your solutions. Read on to see how they should be used in your code.

Constants provided in the starter code

To represent directions		To represent factors/scoring		To represent players/winners	
Constant	Value	Constant	Value	Constant	Value
<code>UP</code>	'up'	<code>UP_FACTOR</code>	4	<code>P1</code>	'player one'
<code>DOWN</code>	'down'	<code>DOWN_FACTOR</code>	2	<code>P2</code>	'player two'
<code>FORWARD</code>	'forward'	<code>FORWARD_FACTOR</code>	1	<code>P1_WINS</code>	'player one wins'
<code>BACKWARD</code>	'backward'	<code>BACKWARD_FACTOR</code>	3	<code>P2_WINS</code>	'player two wins'
		<code>THRESHOLD</code>	5	<code>TIE</code>	'tie game'
		<code>BONUS</code>	12		

doesn't help to change variable names and move functions around. Our software will still find those.

- Don't ask your friend for their solution, even if you just want it to "see how to solve it". Don't show your friends your solution either.
- Don't use a solution from a previous year. We have them all, and will be comparing your submissions to them.
- Don't get solutions, or partial solutions, from a tutor. They frequently provide the same code or ideas to more than one person. We know because we frequently catch people who do it.
- Only ask for detailed help on your code from official CSC108 course staff, including the Help Centre.
- If you can't figure out how to write a function, it's often because you haven't fully understood an underlying concept. Review the materials on that topic, experiment in the Python shell, and ask us for help!
- Remember that it is better to submit an assignment with a few missing functions than to cheat! You can still earn partial credit.

# Scoring in the Game

The number of points earned for finding a word depends on the word's direction and on the number of words left to be found. Each direction has a multiplicative factor shown in the table below. The table above shows the corresponding constants we provided in the starter code. These values are based on how difficult it is (for a human player!) to notice a word in each direction.

Multiplicative factors based on direction

Direction	Forward	Down	Backward	Up
Factor	1	2	3	4

If there are `THRESHOLD` or more words left to be found, the number of points for a correct guess is simply `THRESHOLD` times the factor for the appropriate direction. If the number of words left to be found is less than `THRESHOLD`, then the total points earned for a correct guess is two times `THRESHOLD` minus the number of left to be found before the guess, all multiplied by the factor for the appropriate direction. On top of that there is an extra bonus of `BONUS` points for finding the last word.

For example, suppose there are five words left to be found, and we correctly guess a backward word. Then the number of points for the correct guess is:

$$\text{THRESHOLD} * \text{BACKWARD\_FACTOR} = 15.$$

If there were only two words left to be found, and we correctly guess a backward word, then the number of points for the correct guess is:

$$(2 * \text{THRESHOLD} - 2) * \text{BACKWARD\_FACTOR} = 24 \text{ points.}$$

If there was only one word left to be found, and we correctly guess a backward word, then the number of points for the correct guess is:

$$(2 * \text{THRESHOLD} - 1) * \text{BACKWARD\_FACTOR} + \text{BONUS} = 29 \text{ points.}$$

When you are calculating the number of points for a correct guess, you should use the constants, since during grading we could change them to have different values (e.g., `BACKWARD_FACTOR` could be changed to `5` when we grade your code).

## Files to Download

Please download [a1.zip](#) and extract the zip archive. The following paragraphs explain the files you have been given.

Starter code: `puzzle_functions.py`

This file contains some constants, the header and the complete docstring (but not body!) for the first function you are to write, and some helper functions that we have written for you to use in your solutions. You will update this file to include the complete functions that you write. When you have written all of the functions, you may try playing the game with a different puzzle by changing the value of the constant `PUZZLE_FILE` to `'puzzle2.txt'`.

Starter code: `puzzle_program.py`

This file contains the main program and when it is run, the functions that you wrote and put in the `puzzle_functions.py` file will be called. **Do not** make any changes to the `puzzle_program.py` file. You will be able to understand most of this code by week six.

Data: `puzzle1.txt` and `puzzle2.txt`

These files contain *Where's That Word?* puzzles. You do not need to edit these files. To view the puzzles, in Wing101 go to `File`, `Open`, change the drop-down menu selection to `All files (*)`, select the puzzle file, and click the `Open` button.

Checker: `a1_checker.py`

We have provided a checker program that you should use to check your code. See below for more information about `a1_checker.py`

# What to do

In the starter code file `puzzle_functions.py`, complete the following function definitions. Use the function design recipe that you have been learning in class, and write complete docstrings for each function.

We have included the type contracts in the table; please read through the starter code in `puzzle_program.py` to see how these functions will be used.

List of functions to implement for A1

Function name (Parameter types) -> Return type	Description
<code>get_current_player(bool) -&gt; str</code>	The parameter represents a boolean that is <code>True</code> if and only if the current player is player one. Return a string representing the current player (the value of either constant <code>P1</code> or constant <code>P2</code> ). (Please note: we have provided the complete docstring for this function in the starter code as an example.)
<code>get_winner(int, int) -&gt; str</code>	The first parameter represents the score of player one, the second represents the score of player two. Return the value of constants <code>P1_WINS</code> , <code>P2_WINS</code> , or <code>TIE</code> as appropriate based on the score. In this game, the highest score wins.
<code>reverse(str) -&gt; str</code>	The parameter represents a string that we want to reverse. Return a reversed copy of that string. Note: you must not use any of Python's functions or methods to complete this function.
<code>get_row(str, int) -&gt; str</code>	<p>The first parameter represents a puzzle and the second represents a row number. Return the letters in the row corresponding to the row number, excluding the newline character. The first row is row number 0.</p> <p>For example, if our puzzle is the string <code>'abcd\nefgh\nijkl\n'</code>, and we wanted to find row 1 (which contains 4 letters), we would expect the string <code>'efgh'</code> to be returned.</p> <p>Hint: you will need to call on one of the helper functions provided in the starter code.</p>
<code>get_factor(str) -&gt; int</code>	The parameter represents a direction (the value of one of the constants <code>UP</code> , <code>DOWN</code> , <code>FORWARD</code> and <code>BACKWARD</code> ). Return the multiplicative factor associated with this direction.
<code>get_points(str, int) -&gt; int</code>	<p>The first parameter represents a direction (the value of one of the constants <code>UP</code>, <code>DOWN</code>, <code>FORWARD</code> and <code>BACKWARD</code>) and the second represents the number of words left to be found before this guess. Return the points that would be earned if we were to now find some word in this direction. Follow the approach given in the Scoring in the Game section of this handout.</p> <p>Hint: you will need to call on one of the functions above as a helper function.</p>
<code>check_guess(str, str, str, int, int) -&gt; int</code>	The first parameter represents the puzzle, the second represents a direction (value of one of the constants <code>UP</code> , <code>DOWN</code> , <code>FORWARD</code> and <code>BACKWARD</code> ), the third represents the guessed word, the fourth represents the row or column number, and the fifth represents the number of words left to be found before this guess. If this guessed word is found in this puzzle at this location (row or column) and in this direction, return the number of points earned for this guess. Otherwise, return 0.



## No Input or Output!

Your `puzzle_functions.py` file should contain the starter code, plus the function definitions specified above. `puzzle_functions.py` must *not* include any calls to the `print` and `input` functions. Do *not* add any `import` statements. Also, do *not* include any function calls or other code outside of the function definitions.

## How should you test whether your code works?

First, run the checker and review ALL output — you may need to scroll. You should also test each function individually by writing code to verify your functions in the Python shell. For example, after defining function `get_winner`, you might call it from the shell (e.g., `get_winner(10, 4)`) to check whether it returns the right value (`'player one wins'`). One call usually isn't enough to thoroughly test the function — for example, we should also test `get_winner(5, 5)` where it should return `'tie game'` and `get_winner(2, 10)` where it should return `'player two wins'`.

Once you've checked each function individually, play the game by running the `puzzle_program.py` starter code to see whether it works as expected. If not, go back to testing the functions individually. It is part of the assignment for you to decide which tests to use.

## CSC108 A1 Checker

We are providing a checker module (`a1_checker.py`) that tests two things:

- whether your functions have the correct parameter and return types, and
- whether your code follows the Python and CSC108 [Python Style Guidelines](#).

To run the checker, open `a1_checker.py` and run it. Note: the checker file should be in the **same** directory as your `puzzle_functions.py`, as provided in the starter code zip file. We have posted

[a demo of the checker being run](https://www.youtube.com/watch?v=SXREdJ1GrzU) ↗ (<https://www.youtube.com/watch?v=SXREdJ1GrzU>)



(<https://www.youtube.com/watch?v=SXREdJ1GrzU>)

and included it in the Week 3 Prepare exercises on PCRS. Be sure to scroll through and read all messages.

**If the checker passes for both style and types:**

- Your function parameters and return types match the assignment specification. **This does not mean that your code works correctly in all situations.** We will run additional tests on your code once you hand it in, so be sure to thoroughly test your code yourself before submitting.
- Your code follows the [Python Style Guidelines](#).

**If the checker fails, carefully read the message provided:**

- It may have failed because one or more of your parameter or return types does not match the assignment specification, or because a function is misnamed. Read the error message to identify the problematic function, review the function specification in the handout, and fix your code.
- It may have failed because your code did not follow the style guidelines. Review the error description(s) and fix the code style. Please see the PyTA documentation ([Link](http://www.cs.toronto.edu/~david/pyta/) ↗ (<http://www.cs.toronto.edu/~david/pyta/>)) for more information about errors.

Make sure the checker passes before submitting.

# Marking


These are the aspects of your work that may be marked for A1:

- **Correctness (80%):** Your functions should perform as specified. Correctness, as measured by our tests, will count for the largest single portion of your marks. Once your assignment is submitted, we will run additional tests not provided in the checker. Passing the checker **does not** mean that your code will earn full marks for correctness.
- **Coding style (20%):** Make sure that you follow that we have introduced and the Python coding conventions that we have been using throughout the semester. Although we don't provide an exhaustive list of style rules, the checker tests for style are complete, so if your code passes the checker, then it will earn full marks for coding style with one exception: docstrings may be evaluated separately. For each occurrence of a PyTA error, one mark (out of 20) deduction will be applied. For example, if a C0301 (line-too-long) error occurs 3 times, then 3 marks will be deducted.

## What to Hand In

**The very last thing you do before submitting should be to run the checker program one last time.**

Otherwise, you could make a small error in your final changes before submitting that causes your code to receive zero for correctness.

Submit `puzzle_functions.py` on [MarkUs](http://markus108.teach.cs.toronto.edu/csc108-2018-09)  [\\_\(http://markus108.teach.cs.toronto.edu/csc108-2018-09\)](http://markus108.teach.cs.toronto.edu/csc108-2018-09) by following the instructions on the syllabus. Remember that spelling of filenames, including case, counts: your file must be named exactly as above.

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder