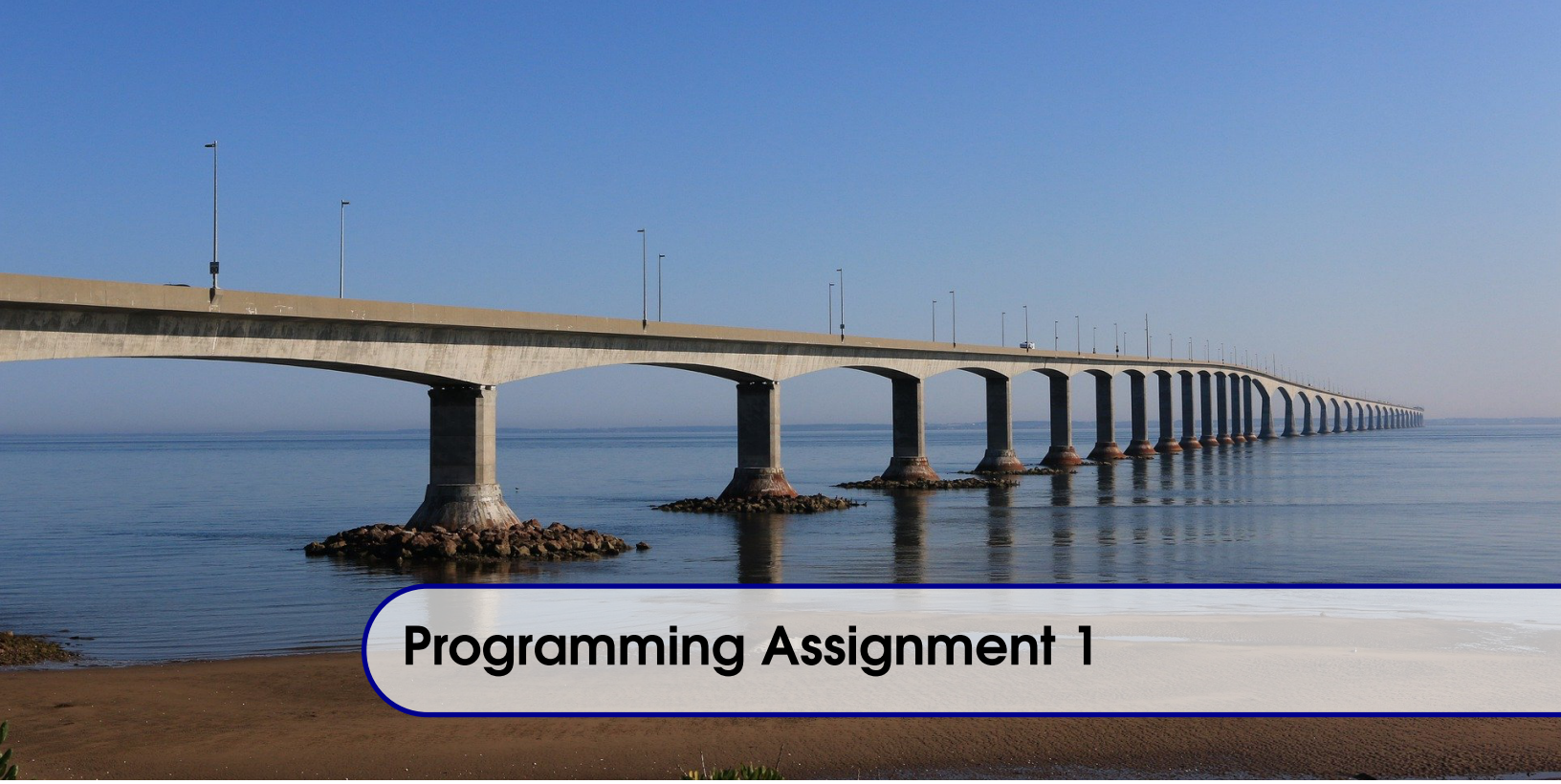

Programming Assignment 1

Assignment Objectives

By completing this assignment you are demonstrating your ability to:

- Decompose problems to create new solutions
- Use variables and expressions to store and manipulate data
- Use console input to take in data from a user
- Create console output to report data to users
- Call functions
- Comment your code appropriately

Scenario

In this course you will have 4 assignments, all of which are going to be related to the same topic: Maze!

Maze Introduction

A $n \times n$ maze is a collection of paths among interconnected cells in a grid, where n is the number of rows and/or the number of columns. The classic maze path-finding problem tries to find a feasible path from a source cell to a destination cell. A white cell allows a path to pass through it whereas a black cell acts as a dead-end and blocks a path. In this assignment, we are making an exception that the black cell can also act as a source cell or a destination cell or both, but it can't appear on a path from the source to the destination.

In this assignment, we are dealing with a naive maze-generator algorithm that creates a not-so-random 8×8 maze. Assume that it always creates a maze with white and black cells appearing next to each other as shown in figure 1. Note that the first cell represented with coordinates $(1,1)$ is always white.

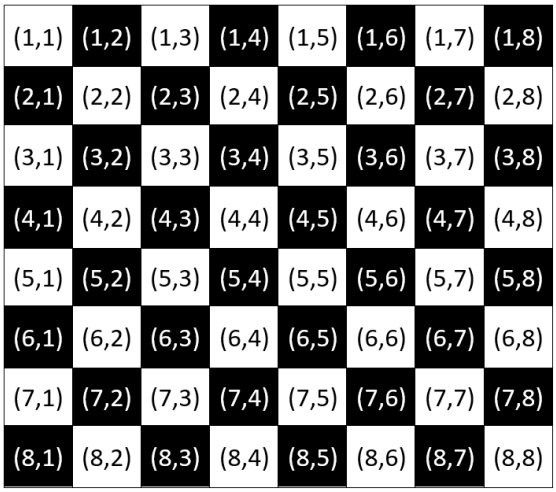

Figure 1: A not-so-random 8×8 maze

Assignment 1 requirements

- You are not allowed to use conditional statements (which we have not studied yet).
- You must solve this problem using arithmetic expressions.
- You can make use of built-in Python functions.
- The code must be properly commented, and it will carry marks.

Problem 1

Write a program in Python that performs the following tasks in order:

1. Takes the coordinates of the source cell from the user.
2. Takes the coordinates of the destination cell from the user.
3. Prints 0 if both source and destination cells are of the same colour, prints 1 otherwise.

```
Enter the x (first) coordinate for the source cell:1
Enter the y (second) coordinate for the source cell:1
Enter the x (first) coordinate for the destination cell:7
Enter the y (second) coordinate for the destination cell:5
Evaluated expression resulted in: 0
```

Figure 2: Problem 1 - sample run 1.

```
Enter the x (first) coordinate for the source cell:1
Enter the y (second) coordinate for the source cell:1
Enter the x (first) coordinate for the destination cell:6
Enter the y (second) coordinate for the destination cell:5
Evaluated expression resulted in: 1
```

Figure 3: Problem 1 - sample run 2.

Problem 2

Write a program in Python that performs the following tasks in order:

1. Takes the coordinates of the source cell from the user.
2. Takes the coordinates of the destination cell from the user.
3. Computes and prints the minimum number of moves (the length of a diagonal path) required to go from source to destination if both source and destination cells exist on the same (white) diagonal; print 0 otherwise. Please note that a valid path can only pass through the white cells, so in this problem, both source and destination cells will have to be white for a valid path to exist.

```
Enter the x (first) coordinate for the source cell:1
Enter the y (second) coordinate for the source cell:1
Enter the x (first) coordinate for the destination cell:5
Enter the y (second) coordinate for the destination cell:5
The length of a diagonal path from ( 1 , 1 ) to ( 5 , 5 ) is : 4
```

Figure 4: Problem 2 - sample run 1.

```
Enter the x (first) coordinate for the source cell:1
Enter the y (second) coordinate for the source cell:1
Enter the x (first) coordinate for the destination cell:5
Enter the y (second) coordinate for the destination cell:6
The length of a diagonal path from ( 1 , 1 ) to ( 5 , 6 ) is : 0
```

Figure 5: Problem 2 - sample run 2.

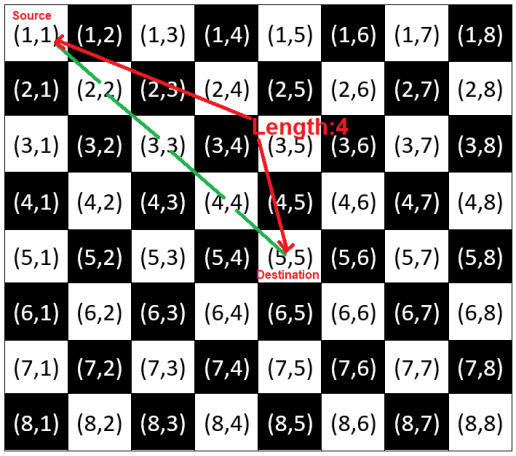

Figure 6: An example of a valid diagonal path of length 4 from (1,1) to (5,5).

A few examples:

- source(1,1) -> destination(5,5) => steps (1,1)->(2,2)->(3,3)->(4,4)->(5,5), length = 4
- source(5,7) -> destination(8,4) => steps (5,7)->(6,6)->(7,5)->(8,4), length = 3
- source (1,1) -> destination (8,5) => not on the same diagonal path, so print 0
- source (1,2) -> destination (4,6) => not on the same diagonal path, so print 0
- source (2,4) -> destination (5,8) => not on the same diagonal path, so print 0
- source (1,2) -> destination (5,6) => not a valid diagonal path, so print 0

Submitting your assignment

Your submission will consist of one zip file containing solutions to all problems (see below). This file will be submitted via a link provided on our Moodle page just below the assignment description. This file must be uploaded by 5pm (Charlottetown time) on the due date in order to be accepted. You can submit your solution any number of times prior to the cutoff time (each upload overwrites the previous one). Therefore, you can (and should) practice uploading your solution and verifying that it has arrived correctly.

What's in your zip file?

A zip file is a compressed file that can contain any number of folders and files. Name your zip file using this format.

Example:
Submission file: `asnX_studentnum.zip`
Where `X` is the assignment number between 1 and 4, and `studentnum` is your student ID number.

Your zip file should contain a PyCharm project for each problem in the assignment. Each project folder and Python file should be named as `problem#.py` where the # is the problem number.