

# Python exercises

## Working Files

For some of the exercises, you will require access to **working files**. These are files that we provide so you can complete the exercise. Usually these would appear automatically to you on Coursera's labs. When you work offline, however, you need to download these files yourself.

- You can find these files in: Week 1, Lesson 1, Optional: Working offline

## Solutions

We highly recommend that you attempt to solve the exercises without guidance. However, if you get stuck, we provide solutions to each exercise to help you. These solutions can also help you learn different ways of solving an exercise, so we recommend you look at them.

- You can find the solutions in: Week 1, Lesson 1, Optional: Working offline.
- The solutions are within the file called 'Supporting files and solutions'

## 1: Installing and testing Visual Studio Code

First, you will need to install Python and Visual Studio Code on your computer. The steps vary depending on your specific operating system.

You will need to download and install:

- Python 3.11.5 or newer from here: <https://www.python.org/downloads/>
- Microsoft Visual Studio Code (VS Code) from here: <https://code.visualstudio.com>

If you face difficulties installing the above on your computer, please search online "How to install Python 3 on Windows", "How to install Microsoft Visual Studio Code on macOS" for detailed tutorials.

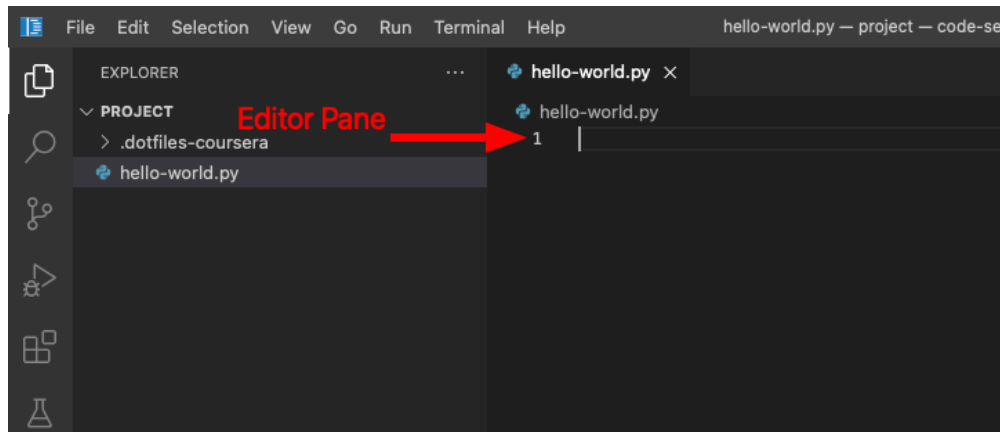
Once you have installed both you will be able to open VS Code and start writing your first Python programs.

### Testing!

When you learn a new programming language, it's common practice to create a simple program in the language to test that everything works. It has become a convention that this

simple program outputs the words "hello, world", and is referred to as a "Hello, world" program.

1. Open VS Code on your computer.
2. Click File->New File
3. Save the file by pressing **CTRL + S (or CMD + S on a Mac)** and name it **hello-world.py**
4. You can write your code in the Editor Pane (see below):



5. Write the following line in the editor pane:  
`print('Hello, world!')`

Now it's time to run the code!

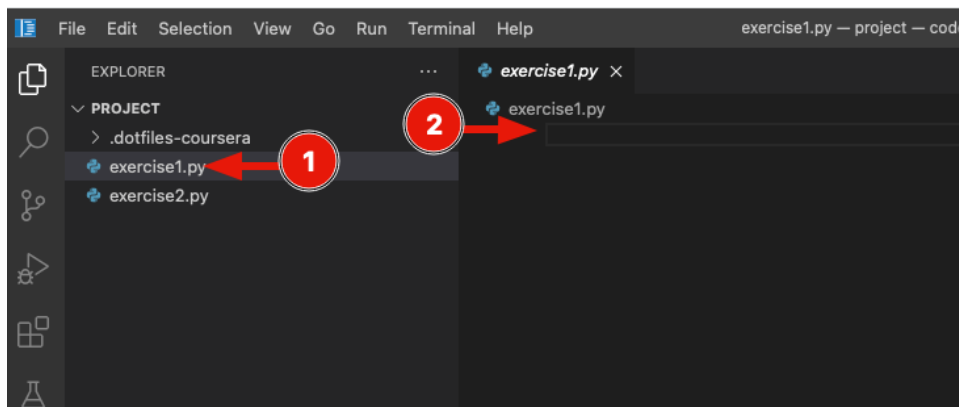
1. From the menu bar, click **Terminal → New Terminal**
2. Run your program by typing `python hello-world.py` (and then hit the ENTER key) in the terminal pane.
3. Observe the message **hello, world!** printed in the terminal!

## 2: Define your own variables

In this exercise, there are two exercises to enable you to practice defining variables within a Python program. Again, we provide the solutions to these exercises.

(Please refer to the first page of this document for instructions on how to access the files you need for this exercise).

1. Download and open the two files: exercise1.py and exercise2.py in VS Code.
2. For each exercise outlined below, open the corresponding file (1) and write the code in the editor (2), as shown in the example figure below.
3. Write your code in the space indicated by the arrow numbered 2.



### Exercise 1

Open exercise1.py and copy-paste the following line into the editor:

```
print(f'Hello, my name is {my_name} and I am {age} years old.')
```

The above print statement will print your name and your age. However, the print uses variables, and the print statement will change depending on the values of the variables. The variables used are enclosed within the curly braces ({variable\_name}).

Define the two variables needed to make this program work. Define those variables in lines 1 and 2, above the print statement. This is because the computer executes the program line by line, so you need to define the variables before they first appear in the print statement; therefore, define the necessary variables with the appropriate values (your name and age) before the print statement (that is, define two variables, one in line 1 and one in line 2).

Remember that a variable that holds text is called a string variable (for example, your name in this exercise). When defining a string variable, you will need to surround the value by

quotes `""`. For example, if your name is Anna, then a variable called `name = "Anna"` (note the quotes around Anna). You do not need the quotes for numbers like your age (for example, `age = 23`).

**Running the code:** Open a terminal (select 'Terminal' → 'New Terminal' from the menu bar on the top) and type `python exercise1.py` in the terminal pane (and hit the ENTER key) to run the program.

**Expected result:** Hello, my name is YOUR NAME and I am YOUR AGE years old.

## Exercise 2

Open `exercise2.py` and define the following string variables at the top: `full_name`, `address_line`, `post_code`, `country` and `telephone_number` (each in a separate line).

Fill in your details for each of these variables. Then create an f-string called `label` (similar to the one we provided to you in `exercise1.py`) that combines all the information together in a way as if it appeared on a parcel that arrived to you, and use the `print` function to print it out, like in the previous exercise. Run and verify your program.

Remember that strings (i.e. text) will need to be surrounded by quotes `""`. Even the telephone number, which in this case is a number, you could surround it with quotes because we are not planning to do mathematical operations on it, so even the telephone number could be defined as a string!

Finally, to define an f-string, you need to place the letter `f` before the quotes `f'my text'` or `f"my text"` (in Python, strings can be defined either with single or double quotes).

Hint: You can print to a different line within the print function by using the new line character, `\n`, so for example, `print(f"Hey\nthere")` will print the hey and there on different lines. `\n` is a special character used for computers to indicate "new line". Use `\n` to create new lines in your label.

**Running the code:** Open a terminal, if not already opened, (click 'Terminal' → 'New Terminal' from the menu bar on the top) and type `python exercise2.py` in the terminal pane (and hit the ENTER key) to run the program.

```
> python exercise2.py
```

Your Full Name,  
Your Address Line,  
Your Post Code,  
Your Country,  
Your Tel Number

### 3: Input and output in Python

These three short exercises will help you practice your Python programming skills on reading input from a user, declaring variables, and doing basic operations on numbers (addition, subtraction, multiplication etc). Don't worry if you get stuck – the solutions are at the end.

Select the Open Visual Studio Code button above and use the exercise1.py, exercise2.py and exercise3.py files provided to you to write the code for each of the exercise below.

#### Exercise 1

The Central Bank of Python gives you a loan with an interest rate of 4%. Write a Python program to read the loan amount from the user and calculate and print the total interest the customer will pay in a year. (interest = amount \* (interest rate / 100)).

#### Call to action

In Visual Studio Code, open the exercise1.py file. You will see a skeleton code with multiple TODO comments to help you complete this exercise. Each TODO block you will see (they appear as Python comments with # and the keyword TODO) is a small sub-task for you. Have a look at all the TODO comments and complete the code required for each.

**Running the code:** Open a terminal (click "Terminal" → "New Terminal" from the menu bar on the top) and type **python exercise1.py** in the terminal pane (and hit the ENTER key) to run the program.

#### Expected output:

```
> python exercise1.py
What is the amount you want to borrow? 1000
The interest you will pay in a year for £1000.0 with an interest rate of 4% is £40.0
```

#### Exercise 2

You are asked to create a Python program that reads a temperature in Celsius and converts and print it to Fahrenheit. The formula is  $F = (C * 1.8) + 32$  (where F is Fahrenheit and C is Celsius).

### Call to action

In Visual Studio Code, open the exercise2.py file. You will see a skeleton code with multiple TODO comments to help you complete this exercise. Each TODO block you will see (they appear as Python comments with # and the keyword TODO) is a small sub-task for you. Have a look at all the TODO comments and complete the code required for each.

Running the code: Open a terminal, if not already opened, (click "Terminal" → "New Terminal" from the menu bar on the top) and type `python exercise2.py` in the terminal pane (and hit the ENTER key) to run the program.

### Expected output:

```
> python exercise2.py
Please provide temperature in Celsius: 32
Temperature in Fahrenheit: 89.6
```

## Exercise 3

Write a Python program that reads two numbers and calculates and prints the result of addition, subtraction, multiplication and division. The input value can be an integer or a float (hint: if integer or float, always convert input to float).

### Call to action

In Visual Studio Code, open the exercise3.py file. You will see a skeleton code with multiple TODO comments to help you complete this exercise. Each TODO block you will see (they appear as Python comments with # and the keyword TODO) is a small sub-task for you. Have a look at all the TODO comments and complete the code required for each.

Running the code: Open a terminal, if not already opened, (click "Terminal" → "New Terminal" from the menu bar on the top) and type `python exercise3.py` in the terminal pane (and hit the ENTER key) to run the program.

### Expected output:

```
> python exercise3.py
Please provide the first number: 6
Please provide the second number: 2
Addition: 8.0
```

Subtraction: 4.0  
Multiplication: 12.0  
Division: 3.0



## 4: If statements

In this exercise, you will practice your skill of writing if statements in Python.

Imagine that a teacher in a high school asked you to create a computer program that will help her to classify student marks. Your program will read a mark (integer) from the user and will print the classification of the mark according to the following rules:

The program will print:

- 'A' if the mark is between 8...10 (inclusive).
- 'B' if the mark is between 5...7 (inclusive).
- 'C' if the mark is between 4...6 (inclusive).
- 'F' if the mark is between 1...3 (inclusive).
- 'Wrong input' if anything else is provided.

You need to write a Python program that will read an input from the user, representing a mark, and it will print the classification according to the rules above.

### Call to action

Open Visual Studio Code and then open the `if_statements.py` file. You will see a skeleton code with multiple TODO comments to help you complete this exercise. Each TODO block you will see (they appear as Python comments with leading `#` and the keyword `TODO`) is a small sub-task for you. Have a look at all the TODO comments and complete the code required for each.

Once you complete the code for each TODO, try to run the solution. Open a terminal: from the menu bar, Terminal → New Terminal. Then run `python if_statements.py`.

### Expected outcome:

```
> python if_statements.py
Please provide the mark of the student: 10
Student grade: A

> python if_statements.py
Please provide the mark of the student: 5
Student grade: B

> python if_statements.py
Please provide the mark of the student: 4
Student grade: C

> python if_statements.py
Please provide the mark of the student: -1
Wrong input: value must be between 1 and 10
```



## 5: Loops

In these exercises, you will be able to practice writing loops in Python. We will provide you with some real-world problems that you will need to solve using loops. This will be a great opportunity for you to see how useful for and while loops are.

### Exercise 1: For loops

Think back to the previous exercise of creating a program to classify students' grades. Recall that the classification is as follows:

6. It will print 'A' if the mark is between 8..10 (inclusive).
7. It will print 'B' if the mark is between 5..7 (inclusive).
8. It will print 'C' if the mark is between 4..6 (inclusive).
9. It will print 'F' if the mark is between 1..3 (inclusive).
10. It will print 'Wrong input' if anything else is provided.

The teacher wants you to extend this program now to allow them to read all the numbers of the class and print the mean of the class as well. So your program should print the grade for each student, but it should use a loop to do this for 10 students and also print the mean of the class at the end.

- Assume there are 10 students in the class.
- At the end, the program should also print the mean of the class. The mean is the total marks of the class divided by the number of students. That is, if the total of all marks from all the students is 75 and there are 10 students, the mean will be 7.5 (75 divided by 10).

#### Call for action

Open Visual Studio Code and then open the **for\_loops.py** file. You will see a skeleton code with multiple TODO comments to help you complete the exercise. Each TODO block you will see (they appear as Python comments with # and the keyword TODO) is a small sub-task. Have a look at all the TODO comments and complete the code required for each.

Running the code: Open a terminal (select Terminal → New Terminal from the menu bar on the top) and type **python for\_loops.py** in the terminal pane (and hit the ENTER key) to run the program.

#### Example output:

```
> python for_loops.py
Please provide the mark of student 1: 10
Student 1 grade: A
Please provide the mark of student 2: 5
Student 2 grade: B
Please provide the mark of student 3: 3
Student 3 grade: F
Please provide the mark of student 4: 8
Student 4 grade: A
Please provide the mark of student 5: 7
```

Student 5 grade: B  
Please provide the mark of student 6: 6  
Student 6 grade: B  
Please provide the mark of student 7: 7  
Student 7 grade: B  
Please provide the mark of student 8: 8  
Student 8 grade: A  
Please provide the mark of student 9: 9  
Student 9 grade: A  
Please provide the mark of student 10: 9  
Student 10 grade: A  
The mean of the class is 7.2

## Exercise 2: While loops

You are asked to implement a text-based 'guess the number' game in Python. This will be an interactive game between the computer and the user. There will be a secret number and the user will need to guess that number.

Your program will enclose a secret number of your choice. Then, the user will be asked to guess the secret number, ideally with the least number of guesses. Your program will need to keep a counter to count the number of guesses until the user hits the secret. At the end, if the user finds the secret, it should print a congrats message to the user but also the number of guesses it took for the user to find the secret. Your program should keep asking the user for a guess until the secret is found.

### Call to action

Open Visual Studio Code and then open the **while\_loops.py** file. You will see a skeleton code with multiple TODO comments to help you complete the exercise. Each TODO block you will see (they appear as Python comments with # and the keyword TODO) is a small sub-task. Have a look at all the TODO comments and complete the code required for each. Running the code: Open a terminal, if not already opened, (select Terminal → New Terminal from the menu bar on the top) and type **python while\_loops.py** in the terminal pane (and hit the ENTER key) to run the program.

### Example output:

```
> python while_loops.py  
Provide your guess: 3  
Wrong input. Try again.  
Provide your guess: 10  
Wrong input. Try again.  
Provide your guess: 9  
Wrong input. Try again.  
Provide your guess: 100  
Wrong input. Try again.  
Provide your guess: 7
```

Congrats! The secret number is 7. You found the secret after 5 attempts.

## 6: Functions

The aim of this exercise is to improve the previous 'guess the number' game to make it better and more intelligent! You will implement functions to make the game better by generating a random secret, providing hints (how far/close the guess is) to the user. In this lab, you will have the opportunity to practice writing functions in Python.

For this exercise, you will need to do the following:

1. Call the get\_random\_secret function that we implemented for you to get a random secret (instead of having a hard-coded one).
2. Write a function to check the validity of the user input. It should return True if the input is between 0 and 100 (inclusive) or False otherwise (negative number or greater than 100).
3. Write a function that will estimate how far the guess is from the secret and provide nice messages to the user to guide them.

### Call to action

Open Visual Studio Code by clicking the button above. Then open the functions.py file. You will see a skeleton code with multiple TODO comments to help you complete this exercise.

Each TODO block (they appear as Python comments with # and the keyword TODO) is a small sub-task for you. Have a look at all the TODO comments and complete the code required for each. If you get stuck, please have a look at the solution for a hint.

**Running the code:** Open a terminal (select Terminal → New Terminal from the menu bar on the top) and type python functions.py in the terminal pane (and hit the ENTER key) to run the program.

**Example output (your output may differ since the secret and user input will be different):**

```
> python functions.py
Please provide a guess: -10
Please provide a number within the range [0, 100].
Please provide a guess: 10
Cold
Please provide a guess: 90
Cold
Please provide a guess: 30
Hot
Please provide a guess: 50
Very hot!
Please provide a guess: 60
Hot
Please provide a guess: 55
Very cold
```

Please provide a guess: 45

Congrats! The secret was 45. You found the secret after 7 attempts.

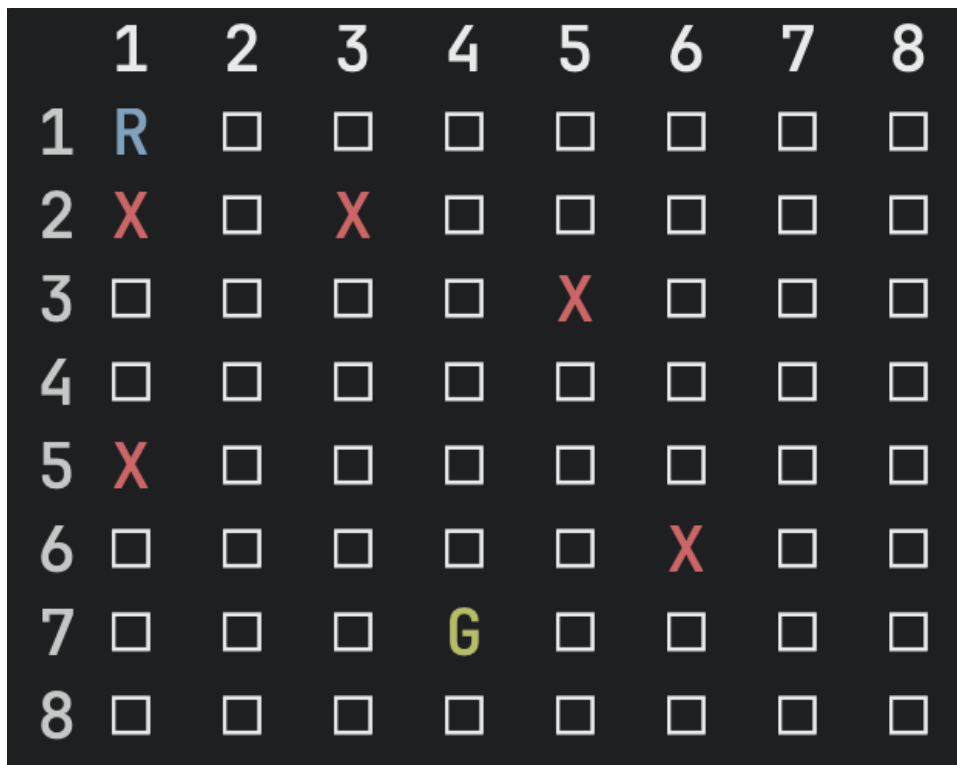
## 7: Implementing a basic version of the robot navigator

In the previous step, we visited the problem of a robot moving in space while avoiding obstacles. Let's implement a first basic version of a program to solve the problem.

Please read this all sections carefully before attempting the exercise.

### Start here

- Open Visual Studio Code by selecting the button above.
- Select Terminal → New Terminal from the file menu.
- Run **python main.py env1** in the terminal pane. You should see the following output:



	1	2	3	4	5	6	7	8
1	R	□	□	□	□	□	□	□
2	X	□	X	□	□	□	□	□
3	□	□	□	□	X	□	□	□
4	□	□	□	□	□	□	□	□
5	X	□	□	□	□	□	□	□
6	□	□	□	□	□	X	□	□
7	□	□	□	G	□	□	□	□
8	□	□	□	□	□	□	□	□

- The robot is currently in cell 1,1 (row=1, column=1) indicated by the letter R.
- Cells that are marked with X are obstacles, if the robot attempts to go into such cells the program will terminate.
- The cell with the letter G is the goal cell the robot needs to get to.

(Note: You will come across some code provided in separate Python files in different directories. You do not need to understand this code or attempt to modify the code. This code creates the text-based and graphical/visual-based interface for the project.)

### The functions available to you

This section will describe how you can use the functions provided for you (and what each does) so you can use this interface to solve the problem.

You will need to modify the main.py file only and write your code in that file. Please do not change any of the other files.

You can run the program in the terminal using: **python main.py env1**. This will run the program using env1.

In main.py you will find the following line:

```
robot = utils.get_environment(sys.argv[1])
```

This creates a robot in the specified environment (env1, env2 etc). With the robot object above, you can use the following commands:

Function	Explanation
<u>robot.move(Direction.Up)</u> <u>robot.move(Direction.Down)</u> <u>robot.move(Direction.Left)</u> <u>robot.move(Direction.Right)</u>	Moves the robot to the next cell towards the direction provided. For example, if the robot is currently in cell (1, 1) and you command <u>robot.move(Direction.Right)</u> then the robot will be in cell (1, 2).
<u>robot.position</u>	Returns the position of the robot (as a cell). You can use <u>robot.position.row</u> to access the row and <u>robot.position.column</u> to access the column.
<u>robot.goal_cell</u>	Returns the goal cell. You can use <u>robot.goal_cell.row</u> to access the row and <u>robot.goal_cell.column</u> to access the column.
<u>robot.num_motions</u>	Returns the number of times the robot moved from initial position. If the robot performed 10 motions, this will return 10.

This is what you need to know for now!

### The problem you need to solve

Let's start with the simplest solution possible: hard-code the move steps for the robot to move to the goal.

Look at the board above and decide what the first step is for the robot to move (left, right, up or down)? What is the second step? What is the third step? Find the correct steps the robot needs to execute sequentially to reach the goal and write them into main.py.

However, here are some rules:

1. If you need to move the robot in the same direction for more than one step, use a loop instead of typing the same command over and over again. Use loops to group moves in the same direction and that need to happen consecutively for more than one step. If the robot needs to move Up, Up, Up, Left, Right, Right you can group Up, Up, Up in a loop, execute a single Left and group Right, Right in another loop.



2. Your program should print, 'Hooray! Robot reached the goal!!!'. But this should only be printed if the robot has indeed reached the goal (i.e., you should write code to check and verify):
3. Finally, after you print the success message, your program should also print the number of steps the robot took to reach the goal. Have a look at the table above to find out a useful command for this sub-task.

### Call to action

Open Visual Studio Code and then open the main.py file. You will see a skeleton code with multiple TODO comments to help you complete this exercise. Each TODO block you will see (they appear as Python comments with # and the keyword TODO) is a small sub-task for you. Have a look at all the TODO comments and complete the code required for each.

You should see the following result:

```
  1  2  3  4  5  6  7  8
1  □  □  □  □  □  □  □  □
2  X  □  X  □  □  □  □  □
3  □  □  □  □  X  □  □  □
4  □  □  □  □  □  □  □  □
5  X  □  □  □  □  □  □  □
6  □  □  □  □  □  X  □  □
7  □  □  □  R  □  □  □  □
8  □  □  □  □  □  □  □  □

Hooray the robot is at the goal position!!!
Number of motions: 9
```

Of course, this is not a great solution, and it will easily break if the robot starts at a different position or if the environment changes. We will work on it and improve our solution! But this was the first step! Well done!

If you get stuck, please have a look at the solution for a hint.

## 8: Implementing the improved algorithm using a path finding algorithm

In the previous exercise, you implemented a working solution that moved the robot from its current position to the goal. Congratulations!

The only problem with this solution is that it will not work in different environments (i.e. if the robot starts at a different position or if the goal is a different cell). That's because the code is hard-coded to work only in the specific environment.

A better solution is needed that will hopefully solve all instances of the problem. An algorithm is complete if it always produces a correct solution where one exists. Therefore, we need to implement a complete algorithm.

We need to implement an algorithm that finds the optimal next action for the robot. This problem is called path finding. There are known algorithms out there that solve the path finding problem. One such algorithm is called A\* (pronounced 'A-star').

### Using the A\* algorithm implemented for you

For this course, you don't need to implement A\* or understand how the algorithm works. We will provide you with an implementation instead. Implementing this algorithm requires knowledge of advanced programming concepts, which we did not introduce to you. What you need to know is that this A\* algorithm will return a path/solution to a path finding problem: given a start and a goal cell, it will give us a sequence of cells that we need to travel through to move from start to goal.

Open [main.py](#) in this lab, you will see certain new lines compared to the basic version. As you will see, we are now using a [path = utils.path\\_finding\(robot\)](#) function to solve the problem. This function returns a path of cells that makes up a solution (if one exists). The A\* algorithm therefore will return a sequence of cells in the variable [path](#). Recall that with the [robot](#) object, you can use the following commands:

Function	Explanation
<u><a href="#">robot.move(Direction.Up)</a></u> <u><a href="#">robot.move(Direction.Down)</a></u> <u><a href="#">robot.move(Direction.Left)</a></u> <u><a href="#">robot.move(Direction.Right)</a></u>	Moves the robot to the next cell towards the direction provided. For example, if the robot is currently in cell (1, 1) and you command <u><a href="#">robot.move(Direction.Right)</a></u> then the robot will be in cell (1, 2).
<u><a href="#">robot.position</a></u>	Returns the position of the robot (as a cell). You can use <u><a href="#">robot.position.row</a></u> to access the row and <u><a href="#">robot.position.column</a></u> to access the column.
<u><a href="#">robot.goal_cell</a></u>	Returns the goal cell. You can use <u><a href="#">robot.goal_cell.row</a></u> to access the row and

	<u>robot.goal</u> <u>cell.column</u> to access the column.
<u>robot.num_motions</u>	Returns the number of times the robot moved from initial position. If the robot performed 10 motions, this will return 10.

### The problem you need to solve

The problem that you need to focus on in this lab is the following. The A\* algorithm returns a sequence of cells that the robot needs to travel through. But we need to find the right action (move right, move left, move up or move down) for the robot at each step. If we are currently at cell (1, 1) and the next cell in the path is to move to the cell (1, 2), in which direction should the robot move? Once the robot moves to (1, 2), the next step in the path is to move to cell (2, 2), again now which direction should the robot move?

### Call to action

Open Visual Studio Code and then open the main.py file. You will see a skeleton code with multiple TODO comments to help you complete this exercise. Each TODO block you will see (they appear as Python comments with leading # and the keyword TODO) is a small sub-task for you. Have a look at all the TODO comments and complete the code required for each. Once you complete the code for each TODO, try to run the solution using different environments and check that the robot reaches the goal every time.

Open a terminal: from the menu bar, Terminal → New Terminal. Then run python main.py env1 . Then try running python main.py env2 and finally python main.py env3. If you get stuck, please check the solutions for hints!