IMPERIAL COLLEGE LONDON

EXERCISES

MSc Artificial Intelligence
MRes Artificial Intelligence and Machine Learning
for Internal Students of the Imperial College of Science, Technology and Medicine

PAPER COMP70053=COMP97123

PYTHON PROGRAMMING

Instructions

- There are 17 questions in this exercise sheet.
- You may use only the Python Standard Library for these exercises. External libraries such as `numpy`, `matplotlib`, `scikit-learn` and `pandas` are not allowed.

Provided files

- The skeleton files are provided in `code/`.
- The skeleton files for each question are provided under its own subdirectory `code/qN/`.
- Please keep the directory structure as given.
- You are provided with files to aid you with testing (these are named `test_qN.py`). You may freely edit these for your own testing purposes. These will not be assessed during your programming exam.
- You may create as many new files as you wish in each subdirectory `qN/`.

Tips

- Please make sure that your scripts run without any syntax errors. The examiner may penalise you otherwise during your programming exam.
- Think before you code!

Question 1

[*Josiah's rating: easy*]

Note: This is Q1 from the 2022/23 August resit exam.

In `q1/q1.py`, please complete the function `count_vowels(string)` to compute how many characters in a given `string` are vowels (*a*, *e*, *i*, *o*, or *u*). Your function should return an `int`.

You may assume that `string` will always be a `str` made up of lowercase letters and/or blank spaces. This includes the empty string or a string with only blank spaces (both of which should return `0`).

Sample input and output

`string`	**returns**
lymph	`0`
python	`1`
snake	`2`
neutral	`3`
binary number	`4`
very good exam	`5`

Question 2

[*Josiah's rating: easy*]

In `q2/q2.py`, complete the function `longest_word(words)` that returns the longest word in `words` and also the length of this longest word.

`words` will always be a `list` of `str`.

The function should return a `tuple`, where the first item in the tuple is the longest word, and the second item is the length of this word.

If more than one word have the same highest length, return the one that occurs first in the list.

You may assume that there is always at least one item in the list.

Sample input and output

`words`	**returns**
`["I", "love", "Python", "programming", "the", "most"]`	`("programming", 11)`
`["Caught", "in", "a", "landslide", "no", "escape", "from", "reality"]`	`("landslide", 9)`
`["Llanfairpwllgwyngyll"]`	`("Llanfairpwllgwyngyll", 20)`
`["laugh", "reach", "go", "climb", "jump"]`	`("laugh", 5)`
`["", "", "", "", ""]`	`("", 0)`

Question 3

[*Josiah's rating: easy*]

In `q3/q3.py`, complete the function `filter_e(string)` that takes a `string`, removes all words containing an `"e"` from the `string`, and returns the new string.

Assume that words in the `string` are separated by a whitespace.

Sample input and output

`string`: `"I swear by the moon and the stars in the sky"` **returns**: `"I by moon and stars in sky"`
`string`: `"She sells sea shells by the sea shore"` **returns**: `"by"`
`string`: `"1 22 333 4444 55555"` **returns**: `"1 22 333 4444 55555"`
`string`: `""` **returns**: `""`

Question 4

[*Josiah's rating: easy*]

Note: This is Q2 from the 2021/22 exam. Everybody received full marks for this question in that edition!

In `q4/q4.py`, complete the function `count_repeats(word)` to check whether every letter in the given `word` occurs the same number of times. For example, `"byebye"` has all three characters occurring exactly twice. Your function should return an `int` indicating the number of repeats if all letters occur the same number of times, or `0` if not all characters repeat the same number of times.

Your function should be **case-sensitive**, so `"A"` and `"a"` are treated as different characters.

You can assume that `word` will be a `str` made up of digits and lowercase letters.

Sample input and output

`word`	**returns**
byebye	2 (`b`, `y` and `e` all appear twice)
abba	2
aabbcacbabcc	4
123456789	1 (all characters appear exactly once)
cake	1
yaba1yaba1	0 (because `a` appears four times while the others appear twice)
dingdong	0 (because `i` and `o` appear once while the others appear twice)

Question 5

[*Josiah's rating: easy to medium*]

In `q5/q5.py`, complete the function `convert(seconds)` that converts `seconds` (an `int`) to an hour/minute/second format.

The function should return a `tuple (hours, minutes, seconds).`

You can assume that `seconds` will always be a non-negative integer.

Sample input and output

`seconds`	**returns**
`38`	`(0, 0, 38)`
`176`	`(0, 2, 56)`
`3823`	`(1, 3, 43)`
`0`	`(0, 0, 0)`

Question 6

[*Josiah's rating: easy to medium*]

In `q6/q6.py`, please complete the function `flip(grid)` that flips the values in `grid` from `0` to `1` and from `1` to `0`.

You can assume that `grid` is always a two-dimensional list of any width or height (at least 1×1). Each row will have the same width, and each column the same height.

You can also assume that the value of each grid cell can either only be `0` or `1`.

Sample input and output

`grid`	**returns**
`[[0]]`	`[[1]]`

`[[0, 1],`
` [1, 0]]`
	`[[1, 0],`
	` [0, 1]]`

`[[1, 1, 0, 0],`
` [0, 1, 0, 1]]`
	`[[0, 0, 1, 1],`
	` [1, 0, 1, 0]]`

`[[0, 0],`
` [0, 0],`
` [0, 0],`
` [1, 1],`
` [1, 1]]`
	`[[1, 1],`
	` [1, 1],`
	` [1, 1],`
	` [0, 0],`
	` [0, 0]]`

`[[1, 1, 1, 1, 1],`
` [1, 0, 0, 0, 1],`
` [1, 0, 0, 0, 1],`
` [1, 0, 0, 0, 1],`
` [1, 1, 1, 1, 1]]`
	`[[0, 0, 0, 0, 0],`
	` [0, 1, 1, 1, 0],`
	` [0, 1, 1, 1, 0],`
	` [0, 1, 1, 1, 0],`
	` [0, 0, 0, 0, 0]]`

Question 7

[*Josiah's rating: easy to medium*]

In `q7/q7.py`, please complete the function `count_palindromes(start, end)` that returns how many palindromic numbers there are between the range `start` and `end` (both inclusive).

Palindromic numbers are integers that have the same value when read from left to right and from right to left. For example, `5`, `88`, `474`, and `1991` are palindrome numbers.

You can assume that `start` and `end` are always non-zero integers, and that the value of `start` will always be smaller than or equal to `end`.

Sample input and output

`start`	`end`	**returns**
0	10	9
0	100	19
100	999	90
1009	9999	90
340	450	90
2002	2002	1
75	75	0

Question 8

[*Josiah's rating: medium*]

In `q8/q8.py`, complete the function `minmax_swap(number)` that swaps two digits in a given `number` (an `int`).

The function should return a `tuple` with two elements:

- The first element is the smallest `int` that can be obtained by swapping any two digits in `number`.
- The second element is the largest `int` that can be obtained by swapping any two digits in `number`.

If `number` is the smallest (or largest) after all possible swaps, then keep the `number` unswapped.

All numbers after swapping must not have leading zeros, e.g. `05432` is invalid.

You can assume that `number` will always be a positive integer and with no leading zeros.

Sample input and output

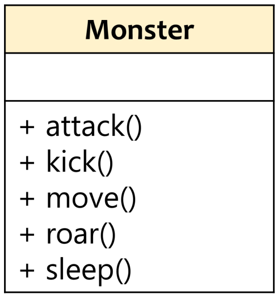
`number`	**returns**
`31572`	`(13572, 71532)`
`91213`	`(11293, 93211)`
`9876`	`(6879, 9876)` (The number itself is already max, so no swapping for max)
`27`	`(27, 72)`
`400`	`(400, 400)` (No swapping can occur)
`5`	`(5, 5)` (No swapping can occur)

Question 9

[*Josiah's rating: medium*]

Your task is to complete the `Python` class in `q9/q9.py`.

`Python` should be a subclass of `Monster` from the `monster` module provided by us. You are provided only with the pre-compiled bytecode for `monster` (for Python 3.10) and not the source code. The class diagram for `monster.Monster` is provided below.



`Python` should override two methods of its parent `Monster` class: `move()` and `roar()`.

The `move()` method should return the string `"sliding"` when invoked.
The `roar()` method should return the string `"hiss"` when invoked.

Sample usage and output

```
>>> python = Python()
Initialising Monster.
>>> python.move()
'sliding'
>>> python.roar()
'hiss'
>>> python.sleep()
'sleeping'
>>> python.attack()
sliding
hiss
kicking
hiss
sleeping
```

Question 10

[*Josiah's rating: medium to hard*]

Note: This is from the 2022/23 exam.

In `q10/q10.py`, please complete the function `most_shared_interest(json_filename)` that loads a JSON database given by `json_filename` and returns information about the pair of students who are members in the same society the most number of times.

The input JSON file is expected to have the following structure:

```
{
    "students": {
        "1": "John",
        "2": "Paul",
        "3": "Ringo",
    },
    "societies": {
        "1": "Badminton",
        "2": "Golf",
        "3": "Hockey"
    },
    "memberships": [
        {"society": "1", "student": "1"},
        {"society": "1", "student": "2"},
        {"society": "1", "student": "3"},
        {"society": "2", "student": "2"},
        {"society": "2", "student": "1"},
        {"society": "2", "student": "3"},
        {"society": "3", "student": "3"},
        {"society": "3", "student": "1"}
    ]
}
```

where

- `"students"` gives the name of each student indexed by an ID.
- `"societies"` gives the name of each society indexed by an ID.
- `"memberships"` gives a list of memberships, where each membership entry indicates that a `"student"` belongs to a `"society"`. In the above example, `"John"`, `"Paul"` and `"Ringo"` belong to the `"Badminton"` society.

The function should return a `dict` with two keys:

- `"pair"`: The value should be a `list` containing the name of the two students who co-occur in the most number of societies.
- `"societies"`: The value should be a `list` containing the list of society names that the pair of students share in common.

In the example above, `"John"` and `"Ringo"` are both in three societies compared to `"John"` and `"Paul"` (two societies) and `"Paul"` and `"Ringo"` (two societies). The function should thus return:

```
{ "pair": ["John", "Ringo"],
  "societies": ["Badminton", "Golf", "Hockey"]
}
```

Both the list of `"pair"` and `"societies"` should be sorted in ascending alphabetical order.

You can assume that all students and societies have different names, and that the given JSON file will always exist, and that the JSON file structure will always be valid.

Sample input and output

Using the file `"students.json"` in `q10/`, `most_shared_interest("students.json")` should return

```
{
 "pair": ["Chia-Hsin", "Ludovico"],
 "societies": [
     "Archery", "Badminton", "Basketball", "Boat", "Boxing",
     "Cheerleading", "Cricket", "Crossfit", "Cycling", "Dodgeball",
     "Fencing", "Football", "Golf", "Lacrosse", "Rugby",
     "Sailing", "Squash", "Synchronized Swimming", "Tennis",
     "Triathlon", "Ultimate Frisbee", "Volleyball"
 ]
}
```

[Disclaimer: the data in `students.json` are artificially generated and bear no resemblance to the students' real world co-curricular activities.]

Question 11

[*Josiah's rating: medium to hard*]

In `q11/q11.py`, please complete the function `most_frequent_common_word(filename1, filename2)` to return the most frequent word in common across two text files `filename1` and `filename2`, and its average frequency.

The most frequent word is the word that:

- occurs in both text files and
- has the highest **average** frequency across the two text files.

The function should return a `tuple` with 2 elements: the first is the most frequent common word (a `str`), the second is the average frequency (a `float`).

For example, assume `filename1` and `filename2` have only two words in common (e.g. `good`, `bad`). The word `good` occurs 6 times in `filename1` and 4 times in `filename2`. The word `bad` occurs 7 times in `filename1` and once in `filename2`. Therefore, the average frequency for `good` is `5` (i.e. $\frac{6+4}{2}$) and for `bad` is `4` (i.e. $\frac{7+1}{2}$). Therefore, the function should return the tuple `(good, 5)`.

You can assume that each word in the file is separated by a space. You can also assume that the texts contain only words consisting of all lowercase alphabetic letters (`a` to `z`).

Assume that `filename1` and `filename2` always points to valid text files.

Sample input and output

Using the text files provided in `q11/`:

`most_frequent_common_word("princeton.txt", "peace_in_a_palace.txt")` should return: `("the", 33.5)`

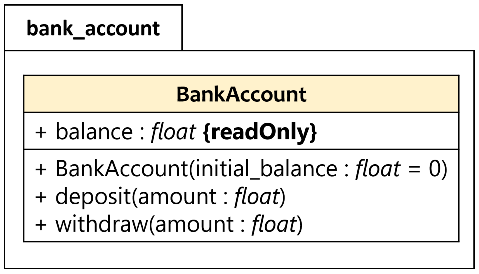
`most_frequent_common_word("copernicus.txt", "michael_oaktree.txt")` should return: `("the", 121.5)`

[Advanced note: In Natural Language Processing, "the" is usually considered a *stopword* that is sometimes removed during pre-processing. We won't be doing this for this question.]

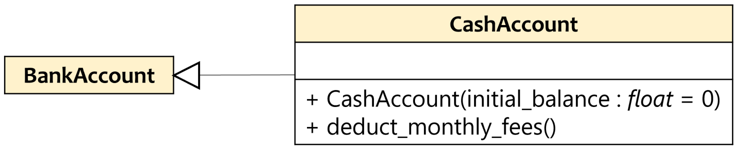
Question 12

[*Josiah's rating: medium to hard*]

In `q12/`, you are given a compiled version of the `bank_account` module. The source code is not provided. The `bank_account` module contains a given `BankAccount` class. The constructor takes an optional `initial_balance` as argument (defaults to `0`). The mutator methods `deposit(amount)` and `withdraw(amount)` will update the `BankAccount` balance. The balance is accessible via the read-only attribute `balance`. See an example usage of this class in `q12/test_q12.py`.


bank_account
BankAccount
+ balance : *float* **{readOnly}**
+ BankAccount(initial_balance : *float* = 0)
+ deposit(amount : *float*)
+ withdraw(amount : *float*)

`CashAccount` is a special type of `BankAccount` that allows users to deposit and withdraw money from a cash machine, but with a service charge for each cash withdrawal. The first four cash withdrawals of the month are free, but all subsequent withdrawals will each incur a £1.00 service fee. The fees will only be deducted by the bank at the end of each month when the bank will compute and deduct the correct amount in one go using the method `deduct_monthly_fees()`. There is no charge for making a deposit.



Your task is to complete the `CashAccount` class in `q12/q12.py`. More specifically:

- Define `CashAccount` as a subclass of `BankAccount`.
- Implement a `deduct_monthly_fees()` method that deducts the fees and resets the withdrawal transaction count for the month. Remember that the first four withdrawals are free.
- You may also need to override the `__init__()` constructor and the `withdraw()` method inherited from the `BankAccount` superclass to keep track of the withdrawal transaction count.

[The bytecode for `bank_account` was compiled on Python 3.10.12. In case you encounter problems on different versions of Python, I have provided the source code in `q12/src/`. But you should assume that the source code will not be provided during the exam when this type of question is asked.]

Sample output

The expected output after running `test_q12.py`:

```
Creating a new cash account in October with an initial balance of £1000.0.
User makes 10 deposits of £10 each in October...
Balance after the 10 deposits:  £1100.0
User makes 10 withdrawals of £100 each in October...
Balance after the 10 withdrawals:  £100.0
The bank deducts its service fee at the end of October
Balance after deduction:  £94.0
Starting balance in November:  £94.0
User makes 4 withdrawals of £10 each in November...
Balance after the 4 withdrawals:  £54.0
The bank deducts its service fee at the end of November
Balance after deduction:  £54.0
```

Question 13

[*Josiah's rating: medium to hard*]

Note: This question is loosely inspired from Part 1 of the 2019/20 exam, although this is a completely different question. That exam had a different format.

Assume that you have a `Store` selling many `Products`. Some `Products` can optionally be attached to at most one offer, for example a *Buy N Get 1 Free Offer* (`BuyNGet1FreeOffer`) or a *Percentage Reduction Offer* (`PercentageReductionOffer)`. These offers are applied per `Product` at checkout.

Task 1

Complete the `init_from_json(json_filename)` method of the `Store` class in `q13/q13.py` to populate `Store`'s `products` attribute (a `dict`) with instances of `Products` read from the JSON file given by `json_filename`, indexed by the product ID.

See `q13/products.json` for the expected format of the JSON file. The JSON gives a list of products, where each product has the keys `"id"`, `"name"` and `"price"`. Some products also have an `"offer"` key, indicating that there is an offer associated with the product. This could either be `"bnf1"` (`BuyNGet1FreeOffer`) or `"percent"` (`PercentageReductionOffer`), with `"n"` giving the values of the offer. For example, `n=3` for `"bnf1"` means it's a *Buy 3 Get 1 Free offer*, and `n=0.25` for `"percent"` indicates it's a 0.25 (25%) discount on the product price. You can assume that the JSON file will always exist and that its structure will always be valid.

The constructor of `Product` takes four arguments: `id`, `name`, `price`, and optionally `offer` (which defaults to a `NoOffer` instance). You should set `offer` to a new instance of the `BuyNGet1FreeOffer` class if the `"type"` is `"bnf1"`, and to the `PercentageReductionOffer` class if it's `"percent"`.

Task 2

Complete the `calculate_total(cost_per_item, quantity)` method of both `PercentageReductionOffer` and `BuyNGet1FreeOffer` classes. These methods should return the total price given the cost of a single product and the quantity, after applying the offer.

For `PercentageReductionOffer`, this should be the full cost reduced by the specified percentage.

For `BuyNGet1FreeOffer`, this should be the cost after deducting the free products. For example, for a product with a `cost_per_item` of £`1.00`, for a *Buy 2 Get 1 Free offer*:

- a `quantity` of 3 should return £`2.00` (price of 2 items since 1 is free);

- a `quantity` of 4 should return £`3.00` (price of 3 items since only 1 item can be free);
- a `quantity` of 5 should return £`4.00` (price of 4 items with 1 free item);
- a `quantity` of 6 should return £`4.00` (price of 4 items with 2 free items)

You can run `q13/main.py` for an example scenario.

Sample usage and output

See `q13/test_q13.py` for an example output for Task 1.

For Task 2:

```
>>> offer = PercentageReductionOffer(0.20)
>>> offer.calculate_total(0.50, 4)
1.6
>>> offer = PercentageReductionOffer(0.10)
>>> offer.calculate_total(2.00, 3)
5.4
>>> offer = BuyNGet1FreeOffer(3)
>>> offer.calculate_total(2.00, 4)
6.0
>>> offer = BuyNGet1FreeOffer(3)
>>> offer.calculate_total(2.00, 8)
12. 0
>>> offer = BuyNGet1FreeOffer(1)
>>> offer.calculate_total(2.00, 8)
8.0
```

Question 14

[*Josiah's rating: hard (requires some thinking, but could be easy)*]

In `q14/q14.py`, complete the function `simplify_fraction(numerator, denominator)` that simplifies a fraction `numerator/denominator` to its most simplified form.

The function should return a `tuple (numerator, denominator).`

You can assume that `numerator` and `denominator` are always positive integers.

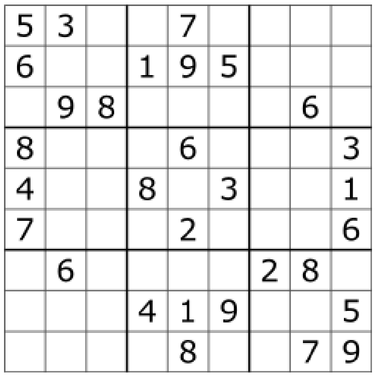
Sample input and output

`simplify_fraction(4, 8)` should return `(1, 2)`
`simplify_fraction(15, 5)` should return `(3, 1)`
`simplify_fraction(200, 300)` should return `(2, 3)`
`simplify_fraction(7, 9)` should return `(7, 9)`

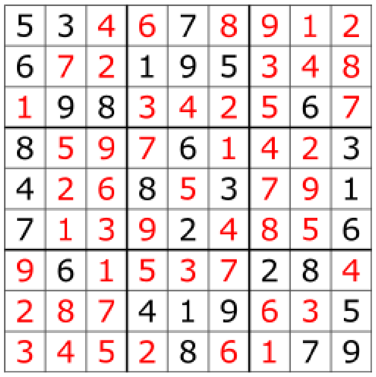
Question 15

[*Josiah's rating: hard*]

Note: This question has been modified from Part 2 of the 2019/20 exam to fit the style of your exam.



5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9



5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

In `q15/q15.py`, please complete the function `is_sudoku_board_valid(board)` that returns a `bool` indicating whether a given Sudoku `board` is valid.

`board` is a two-dimensional `list` of `str` representing a 9×9 grid. Each cell is filled with a digit from `"1"` to `"9"` or a `"."` representing a blank cell.

A Sudoku `board` is valid only if all the following conditions are true:

- Each cell must be a valid string (`"1"` to `"9"` or `"."`)
- Each row does not have any duplicated digits
- Each column does not have any duplicated digits
- Each 3x3 box does not have any duplicated digits

You can assume that the size of `board` will always be 9×9.

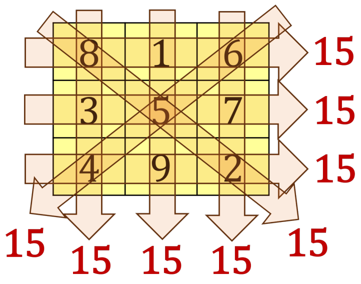
Sample input and output

Please see `q15/test_q15.py` for example valid and invalid boards.

Question 16

[*Josiah's rating: hard*]

In `q16/q16.py`, please complete the function `is_magic_square(matrix)` that returns a `bool` indicating whether a given `matrix` is a magic square.



A 2D matrix of size $N \times N$ is a magic square if:

- the matrix is made up of positive integers 1 to N^2 (inclusive);
- each integer in the matrix only occurs once (no repeats); and
- the sum of the numbers in each row, each column and both main diagonals are all the same (they all sum to a common "magic constant").

`matrix` is a two-dimensional `list` of `int` representing a square $N \times N$ grid, where N is any positive integer. You can assume that the `matrix` will always be in a valid format.

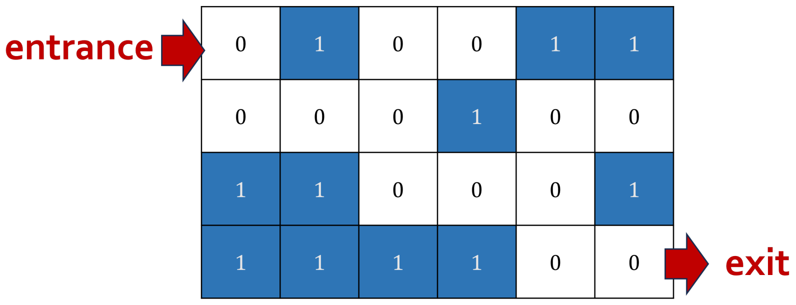
Sample input and output

`matrix`	**returns**
`[[1]]`	`True`
`[[5]]`	`False` (max integer is 1 for a 1×1 matrix)
`[[8, 1, 6], [3, 5, 7], [4, 9, 2]]`	`True`
`[[8, 11, 6], [3, 5, 7], [4, 9, 2]]`	`False` (`11` is outside the range of 1 to N^2)
`[[8, 1, 7], [3, 5, 7], [4, 9, 2]]`	`False` (`7` is repeated)
`[[8, 2, 6], [3, 5, 7], [4, 9, 1]]`	`False` (sum of rows are different)
`[[1, 5, 9], [6, 7, 2], [8, 3, 4]]`	`False` (sum of diagonals are different)
`[[35, 1, 6, 26, 19, 24], [3, 32, 7, 21, 23, 25], [31, 9, 2, 22, 27, 20], [8, 28, 33, 17, 10, 15], [30, 5, 34, 12, 14, 16], [4, 36, 29, 13, 18, 11]]`	`True`

Question 17

[*Josiah's rating: hard*]

In `q17/q17.py`, please complete the function `can_exit_maze(maze)` that returns a `bool` indicating whether there is a valid path in a given `maze` where you can navigate from its entrance to the exit.



A `maze` is an $M \times N$ matrix with elements that can either be `0` or `1`. A `0` indicates a walkable area, and a `1` indicates a wall. The entrance of the `maze` is always at the upper-left corner, and the exit is always at the bottom-right corner. You can only navigate up, down, left, and right (and **NOT** diagonally).

You can assume that `maze` is always a two-dimensional list of any width or height (and will always be at least 2×2). Each row will have the same width, and each column the same height. You may also assume that there is always an entrance (`maze[0][0]` will always be `0`), although there may not always be an exit.

You can also assume that the value of each cell in `maze` will always either only be the integers `0` or `1`.

Sample input and output

`maze`	**returns**
`[[0, 1, 0, 0, 1, 1], [0, 0, 0, 1, 0, 0], [1, 1, 0, 0, 0, 1], [1, 1, 1, 1, 0, 0]]`	True
`[[0, 1, 0, 0, 1, 1], [0, 0, 0, 1, 0, 0], [1, 1, 0, 0, 1, 1], [1, 1, 1, 1, 0, 0]]`	False
`[[0, 1, 1, 0], [0, 1, 1, 1], [0, 1, 1, 0], [0, 0, 0, 1], [1, 1, 0, 0], [1, 1, 1, 0]]`	True
`[[0, 1, 1, 0], [0, 0, 0, 1], [1, 1, 0, 0], [1, 1, 0, 1], [1, 1, 0, 0], [1, 1, 1, 1]]`	False (no exit!)
`[[0, 1], [0, 0]]`	True
`[[0, 1], [1, 0]]`	False