COVER PAGE FOR A WRITTEN EXAMINATION/TEST

Name of subject	: Data Structures & Algorithms
Subject code	: 822188-B-6
Date of examination	: October 12, 2020, 14.30-17.00
Length of examination	: 2.5 hours
Lecturer	: Pieter Spronck ANR: 447823

Telephone number of departmental secretariat: 8118

Students are expected to conduct themselves properly during examinations and to obey any instructions given to them by examiners and invigilators. Firm action will be taken in the event that academic fraud is discovered.

1. This exam is a programming exam. Students submit their answers in the form of Python code files. Students may use the following editors to write their code:
   - Spyder
   - Vim
   - PyCharm
   - Visual Studio Code
   - Atom
   - Notebooks (either local notebooks or on the JupyterLab server)
   - IDLE

   You may use Anaconda to start an editor or local notebooks.
2. Students should only use modules that come standard with Python, as discussed in the first 20 notebooks. Thus, `numpy` is not allowed.
3. Students may consult the following written materials on their computer:
   - Python documentation as supplied with Python, or via the `docs.python.org` website
   - The book “The Coder’s Apprentice”
   - The book “Think Python”
   - Notebooks, either local notebooks or on the JupyterLab server (`jupyterlab.uvt.nl`)
   - A Python cheat sheet as supplied on Canvas
   - Lecture sheets as supplied on Canvas
   - Anything else that is supplied on Canvas for this course
4. The students may use note paper during the exam.
5. If a student has questions during the exam, they should ask those in the Discussion forum on Canvas. The instructors will keep track of those questions and answer them. Students should NOT post code in the forum! Students may see each other’s questions, but should NOT answer each other’s questions! Communicating with anyone but the instructors is not allowed!
6. This exam contains 5 questions. Each question has its own code file. Students fill in their code in these code files, and submit them as their answers. Please submit five separate code files, and make sure that they are Python (.py) files! Do not change the names of the files! Do not pack them in a ZIP file! Do not submit .ipynb files! Do not submit screenshots! If a student wants to add remarks to submitted code, please do so as comments in the code files that are submitted.
7. Students should add their name and student number at the top of each file as a comment.
8. Students may add extra tests in the `main()` function that is found in each of the code files.
9. If a student needs to add `print()` statements in a function that they are developing for debugging purposes, these `print()` statements should be removed before submitting the final code.

10. Functions never need to ask the user for input. All inputs for the functions are supplied via parameters.
11. If a student wants to use the notebooks as editor, then they should copy the code from the Python files to the notebook in which they want to do the editing, and after having finished their code, copy the code back to the original file. They then submit the original file. Test the file before submitting it, to make sure that nothing went wrong after copying!
12. Make sure that code is easy to understand. When in doubt, add comments.
13. Each question is worth 2 points. The points are added up to form the exam grade. Standardized rounding applies.
14. Not all exercises are equally difficult. We tried to order them from easy to hard, but this may vary between students.
15. Naturally, students may not consult other people during the exam, nor are they allowed to use digital devices to consult anything but the materials listed above.

You can start the exam now, good luck!

COVER PAGE FOR AN ONLINE PROCTORED EXAM

Name of subject	: Data Structures & Algorithms
Subject code	: 822188-B-6
Date of examination	: October 12, 2020, 14.30-17.00
Length of examination	: 2.5 hours
Lecturer	: Pieter Spronck ANR: 447823

Telephone number of departmental secretariat: 8118

Students are expected to conduct themselves properly during examinations and to obey any instructions given to them by examiners and procotors.

You're about to take an online exam. Please read the following information carefully. These regulations apply to all online proctored exams.

Code of honor

Students participating in this exam adhere to the following :

I will take this exam to the best of my abilities, without seeking or accepting the help of any source not explicitly allowed by the conditions of the exam.

I have neither given nor received, nor have I tolerated others' use of unauthorized aid.

Not complying with the statement invalidates the exam for summative use, that is, a grade will not be assigned to your completed exam.

Firm action will be taken in the event academic fraud is discovered.

1. This is an online proctored programming exam. By default, the student's desk must be empty, apart from empty note paper and writing materials.
2. Students submit their answers in the form of Python code files. Students may use the following editors to write their code:
   - Spyder
   - Vim
   - PyCharm
   - Visual Studio Code
   - Atom
   - Notebooks (either local notebooks or on the JupyterLab server)
   - IDLE

   You may use Anaconda to start an editor or local notebooks.

3. Students should only use modules that come standard with Python, as discussed in the first 20 notebooks. Thus, `numpy` is not allowed.
4. Students may consult the following written materials on their computer:
    - Python documentation as supplied with Python, or via the `docs.python.org` website.
    - The book “The Coder’s Apprentice”
    - The book “Think Python”
    - Notebooks, either local notebooks or on the JupyterLab server (`jupyterlab.uvt.nl`)
    - A Python cheat sheet as supplied on Canvas
    - Lecture sheets as supplied on Canvas
    - Anything else that is supplied on Canvas for this course
5. The students may use note paper during the exam.
6. If a student has questions during the exam, they should ask those in the Discussion forum on Canvas. The instructors will keep track of those questions and answer them. Students should NOT post code in the forum! Students may see each other’s questions, but should NOT answer each other’s questions! Communicating with anyone but the instructors is not allowed!
7. This exam contains 5 questions. Each question has its own code file. Students fill in their code in these code files, and submit them as their answers. Please submit five separate code files, and make sure that they are Python (.py) files! Do not change the names of the files! Do not pack them in a ZIP file! Do not submit .ipynb files! Do not submit screenshots! If a student wants to add remarks to submitted code, please do so as comments in the code files that are submitted.
8. Students should add their name and student number at the top of each file as a comment.
9. Students may add extra tests in the `main()` function that is found in each of the code files.
10. If a student needs to add `print()` statements in a function that they are developing for debugging purposes, these `print()` statements should be removed before submitting the final code.
11. Functions never need to ask the user for input. All inputs for the functions are supplied via parameters.
12. If a student wants to use the notebooks as editor, then they should copy the code from the Python files to the notebook in which they want to do the editing, and after having finished their code, copy the code back to the original file. They then submit the original file. Test the file before submitting it, to make sure that nothing went wrong after copying!
13. Make sure that code is easy to understand. When in doubt, add comments.
14. Each question is worth 2 points. The points are added up to form the exam grade. Standardized rounding applies.
15. Not all exercises are equally difficult. We tried to order them from easy to hard, but this may vary between students.
16. Naturally, students may not consult other people during the exam, nor are they allowed to use digital devices to consult anything but the materials listed above.

<u>Regulations for online proctored exams</u>

Exam location requirements (proctoring setup):

1. The lighting in the room must be bright enough to be considered “daylight” quality. Overhead lighting is preferred. If overhead lighting is not available, the source of light must not be behind the student.
2. The student must sit at a desk or table cleared of all objects unless specifically stipulated otherwise on the cover sheet.

3. The area (surfaces) around the student must not have any writing or cheat sheets.
4. The student must be alone in the room.
5. The student should not have any wearables, such as smart watches and/or health checkers other than explicitly permitted.
6. The room must be as quiet as possible. Sounds such as music or television are not permitted.
7. Only items that are specifically permitted in the exam instructions are allowed on the student's desk.
8. If during the exam the use of paper is allowed, the student must show the empty sheets at the start of the exam.
9. Completion of the questionnaire at the end of the exam is compulsory! The student must use this questionnaire to mention all events during the exam that could be relevant in assessment of possible fraud.

Support:
If you encounter technical problems that prevent you from completing the exam, you must report this through the chat functionality. The chat functionality is available during the proctoring set-up and throughout the exam.

Important:
- Examinees are only permitted to visit the toilets if there is a built-in break or individual allowance is given in advance.
- The instructions of examiners and (if applicable) the proctoring agency must be followed.
- No pencil cases/lunchboxes are permitted on desks. Your desk should be clean.
- No use of headphones, earbuds, or any other type of listening equipment is permitted unless permission is given. Disposable earplugs are only allowed when shown to the webcam prior to the examination.
- Do not communicate with any other person by any means, except with the helpdesk through the helpdesk functionality of the Proctoring Agency or Tilburg University.
- Answer the questionnaire at the end of the exam to indicate whether unwanted disturbances occurred that might be registered as an irregularity.

<u>Fraud</u>

1) In the event of a reported (suspicion of) fraud, the Examination Board Rules and Guidelines with regard to fraud apply.
2) In addition to the existing rules regarding fraud laid down in the Examination Board Rules and Guidelines, fraud (or an attempt to fraud) on the part of the student is taken to mean in any case, and among others, the following:
   a. using someone else's proof of identity;
   b. having someone else complete or participate in the exam;
   c. having someone help in completing the exam;
   d. using or attempting to use unpermitted (digital) sources, resources, or devices for communication during the exam;
   e. using or attempting to use unpermitted printed or handwritten documentation;
   f. the student is no longer in sight of the webcam and/or has switched off the microphone and other necessary devices needed for online proctoring, while taking the exam, insofar this

takes place outside the (possible) authorized breaks;

g. (attempted) technical modifications that undermine the proctor system.

Intellectual property rights

1) The intellectual property rights of (online) examination materials are owned by Tilburg University.

2) The production, making available and/or distribution of (parts of) (online) exams to third parties by students of Tilburg University by means of photo, video, film or sound recordings or any other digital form is not permitted. Students who nevertheless make (online) examination materials available to third parties are acting in violation of Tilburg University's House and Conduct Rules and may be excluded by the Examination Board from this or any other exam, and may be sued in court.

You can start the exam now, good luck!

Midterm Data Structures & Algorithms 2020/2021

Please read the exam instructions on the cover page carefully!

Note that academic fraud amounts to handing in code that you did not write yourself, either because you got someone to help you, or because you copied someone else's answers, or because you consulted internet sites which you are not allowed to consult during the exam!

However, you may use code which you find in the books or notebooks to base your answers on.

What academic fraud is and the fact that you are not allowed to commit it, has been pointed out to you now multiple times during the class, via a lecture, via an announcement, via a video, via the cover page of this exam, and right here on this page.

When academic fraud is detected, the exam board will be notified immediately. Consequences may be severe. Denial of knowledge of academic fraud will not be accepted as an excuse.

If you need to download the exam because you received this page on paper, then please go to the Canvas course, to the Assignment entry, and select the "Midterm, October 12, 2020" assignment. You find all the files there, and you can also submit your answers there.

Make sure that you place the `.txt` files which are provided in the same folder as where you edit the code for the third exercise. If you do not, your code may not be able to see them.

If you have questions during the exam, you can ask them via the Discussion forum for the Canvas course. You may read each other's questions, but you may not answer them – this would actually be a violation of the rule that you cannot communicate with others during an exam. Leave the answering to the instructors.

It is possible that within a week after the exam the instructors will make an appointment with you to discuss some of the answers you have given in an online call. The results of this call may weigh in your grade.

We wish you good luck.

Exercise 1: Contains Dividers

File: `E01_contains_dividers.py`

The function `contains_dividers()` gets a single parameter, which is a list of positive integers. The function returns a boolean: it returns `True` if there are two integers in the list of which one is a divider of the other (i.e., one divided by the other results in an integer); otherwise it returns `False`.

Implement the function `contains_dividers()`.

You may assume that the parameter that the function is called with is indeed a list of positive integers. You do not need to check for that. Note that zero is not a positive integer, so you do not need to take into account zeroes.

Examples:

`contains_dividers([7,6,5,4,3,29])` returns `True`, because 6 is divisible by 3

`contains_dividers([7,6,5,4,29])` returns `False`, as no numbers on the list have a divider on the list

Exercise 2: Word Pair Frequencies

File: `E02_word_pair_frequencies.py`

In a string, a word pair is two consecutive words. A word pair occurrence is the number of times that a word pair occurs in a text. For instance, in the string "*The more that you read, the more things you will know. The more that you learn, the more places you'll go*" contains the word pair "more that" twice, and the word pair "the more" four times (case-insensitively).

The function `word_pair_frequencies()` gets a string as parameter. It then creates a dictionary, which has word pairs in the form of tuples as keys, and the number of occurrences of word pairs (integers) as values. It returns this dictionary. Note that, for instance, the word pair "more that" should be stored as the key `("more", "that")`.

Implement the function `word_pair_frequencies()`.

You should convert the string that is given as argument to the function to lower case, and should remove everything that is not a letter before you form the word pairs. Everything that is not a letter should be considered a word divider. A function `clean()` is supplied to do this for you.

The test that is provided in the code file uses a string `eggs_and_ham`, which contains the start of the story *Green Eggs and Ham* by Dr. Seuss. It calls the function `word_pair_frequencies()` with this string, and stores the generated dictionary in the variable `wp_frequencies`. The statement `wp_frequencies.get(("<word1>", "<word2>"), 0)` returns the number of occurrences of the word pair `"<word1> <word2>"` in `eggs_and_ham`.

Examples:

`wp_frequencies.get(("green", "eggs"), 0)` returns 2, because the word pair "green eggs" occurs twice in `eggs_and_ham`

`wp_frequencies.get(("red", "eggs"), 0)` returns 0, because the word pair "red eggs" does not occur in `eggs_and_ham`

`wp_frequencies.get(("and", "ham"), 0)` returns 2, because the word pair "and ham" occurs twice in `eggs_and_ham`

`wp_frequencies.get("green eggs", 0)` returns 0, because the word pair "green eggs" is represented by the key `("green", "eggs")` and not by the string `"green eggs"`.

Exercise 3: Population Count

File: `E03_population_count.py`
Input files: `netherlands.txt`, `germany.txt`, `unitedstates.txt`, `china.txt`

The text file `netherlands.txt` is a tab-separated text file that contains an overview of the ten largest cities in The Netherlands and their population. Each line consists of a city and its population, which are separated by a tab (`"\t"`). The files `germany.txt`, `unitedstates.txt`, and `china.txt` contain similar data for Germany, the United States, and China, respectively.

Write a function `population_count()` that reads in a text file that is given as argument. You may assume that the file contains a list of cities as specified above. The function returns the sum of the population of the ten largest cities in a country (integer). If the file does not exist, you should return -1. You do not need to check for any other file errors.

Note that the numbers in the text files are stored with thousands separators, whereby commas are used to separate groups of three digits (i.e., one million is written as `1,000,000`). You will need to make sure that your function handles these numbers properly.

Example input file `netherlands.txt`:

```
Amsterdam	741,636
Rotterdam	598,199
The Hague	474,292
Utrecht		290,529
Eindhoven	209,620
Tilburg		199,613
Groningen	181,194
Almere		176,432
Breda		167,673
Nijmegen	158,732
```

Be aware that between the city name and the population number there is a tab (\t), and not spaces!

Examples:

`population_count( "netherlands.txt" )` returns 3197920

`population_count( "china.txt" )` returns 106376083

Exercise 4: Is sorted?

File: `E04_is_sorted.py`

A recursive procedure to check whether a list is sorted is the following:

- If the list has one element or less, it is sorted
- If the list has more than one element, it is sorted if the following two statements hold: (1) the last element of the list is greater than or equal to the next-to-last element of the list; and (2) the sublist, consisting of the list without the last element, is sorted
- Otherwise the list is not sorted

Implement a recursive function `is_sorted()` that checks whether a list is sorted. If the list that it gets as argument is sorted, it returns `True`, otherwise it returns `False`.

You may use the recursive algorithm given above, or another recursive algorithm (for instance, one that does not compare the last element of the list to the next-to-last, but the first with the second).

You may assume that the list contains elements which can be compared with each other using comparison operators ==, !=, >, >=, <, and <=.

Note that the algorithm you implement must be recursive; a straightforward iterative algorithm is not acceptable (even though in practice that would probably be preferable). Also note that your function is not allowed to change the original list that is given as argument.

Examples:

`is_sorted([1,2,3,4,5,8,12,14,17,23,25,25,27])` returns `True`.

`is_sorted(["apple","banana","cherry","orange","grape"])` returns `False`, as “grape” should come before “orange”.

Exercise 5: Shape

File: `E05_shape.py`

The file `E05_shape.py` contains two classes: an abstract class `Shape`, and a class `Point`.

The class `Point` describes a two-dimensional point, consisting of an x-coordinate and a y-coordinate. It has a method `distance()`, which you give another point as parameter, and which returns the distance between the two points as a float.

The class `Shape` represents a 2-dimensional shape. Since it is an abstract class, it is not supposed to be created; rather, classes which inherit from `Shape` should create actual shapes. The class `Shape` contains an `__init__()` method which gives the `Shape` a name. This name is filled automatically with the name of the class. It also contains a method `circumference()`, which is supposed to calculate the circumference of the shape, but as this is an abstract class, `circumference()` is not implemented. Finally, `Shape` contains a `__repr__()` method which just displays the name.

Implement classes `Polygon`, `Rectangle`, and `Square`.

A Polygon is a `Shape` (and thus inherits from `Shape`) which gets upon creation a list of `Point`s. These `Point`s are the so-called "vertices" of the `Polygon`, i.e., the corner points. Between two consecutive corner points is a line which is one side of the `Polygon`; such a line is called an "edge." Note that there is also an edge between the first and the last vertex in the list. All shapes shown in the image below are polygons. You may assume that every `Polygon` is created with at least three `Point`s in the vertices list, and that they describe a legal `Polygon` (e.g., they are not on a straight line).

Besides the `__init__()` method, which should include a call to the `__init__()` method of `Shape` to set the name of the object (no argument needs to be given to this call, then this name will automatically be "Polygon"), the `Polygon` gets two more methods. The first is the `__repr__()` method, which displays the name of the `Polygon`, with the list of vertices next to it between parentheses (see the example below). The second is an implementation of the `circumference()` method, which calculates the circumference of the Polygon as the sum of the length of all the edges (you may calculate the length of an edge by making use of the `distance()` method from `Point`).

A `Rectangle` is a `Polygon` which is specified by a single `Point` (the top-left corner), a width, and a height. The `Rectangle`'s sides are parallel to the x-axis and y-axis, so it is not tilted. Rather than creating the `Rectangle` as an independent class, it should inherit from `Polygon`, and create the `Rec-tangle` by initializing it with a call to the `__init__()` method of `Polygon`, with the appropriate list of vertices. Only the `__init__()` method of the `Rectangle` should be implemented, all other methods it inherits from `Polygon`. You may assume that the `__init__()` method of `Rectangle` indeed gets called with a `Point`, and a legal width and height.

A `Square` is a `Rectangle` whereby the width and height are equal. A `Square` simply inherits from `Rectangle`, and only the `__init__()` method of `Square` should be implemented. You may assume that the `__init__()` method of `Square` gets called with a `Point` and a legal side.

The example calls in the `E05_shape.py` file create four different shapes, which are all displayed in the image below. Two of them, the big shape and the triangle, are created as `Polygon`s. The rectangle is created as a `Rectangle`, and the square is created as a `Square`.