COVER PAGE FOR A WRITTEN EXAMINATION/TEST

Name of subject	: Data Structures & Algorithms
Subject code	: 822188-B-6
Date of examination	: January 25, 2021
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 32 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. They may also use a calculator and may have something to drink during the exam, as long as nothing is written on the bottle or cup.
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	: January 25, 2021
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 32 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. They may also use a calculator and may have something to drink during the exam, as long as nothing is written on the bottle or cup.
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.

Regulations for online proctored exams

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!

Resit 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 Quizzes, and select the Exam. You find all the files there, and you can also submit your answers there.

Make sure that you place any `.txt` file which is 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. In particular, if you use the Jupyter server for coding, upload them too.

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: Dice game

File: `E01_dice_game.py`

One-line summary

In this exercise, you simulate the outcome of a simple dice game in which players roll different numbers of dice. The player with the highest total wins the game.

Context

Suppose that you play a dice game in which the players roll different numbers of dice (for instance, you roll 1 die and your opponent rolls 3), and the player who rolls the highest total wins. Of course, such a game can only be fair if the player who rolls more dice gets a lower payout than if the player who rolls less dice wins. What a "fair" payout would be depends on the probability that a player wins this game.

You can calculate the probability of winning such a game exactly, but you can also approximate it using a Monte Carlo simulation. A Monte Carlo simulation of the proposed dice game consists of a large number of simulations of games. For each simulation, you compare the total of your dice to the total of your opponent's dice. The probability of winning the game is the number of simulations in which your total is higher than your opponent's total.

Exercise description

Write a function `simulate_dice_game()` that estimates the probability of winning the above-mentioned dice game on the basis of its three arguments: *ndice* (the number of dice you roll), *ndice_opp* (the number of dice your opponent rolls), and *nsim* (the number of simulations). You win a game if the total of your dice is higher than the total of your opponents' dice. If the totals of your dice and your opponent's dice are identical, both players roll again. You may assume that *ndice*, *ndice_opp*, and *nsim* are positive integers, higher than zero. The function `simulate_dice_game()` should return the probability of winning the game for the player who rolls *ndice* dice, with the given values of *ndice* and *ndice_opp*. Note that the results of identical calls to `simulate_dice_game()` will (and should!) differ slightly each time you run the code, due to the fact that the dice rolls are randomly generated.

Examples

```
print( simulate_dice_game( 2, 2, 10000 ) ) # approximately .500

print( simulate_dice_game( 3, 2, 10000 ) ) # approximately .837

print( simulate_dice_game( 2, 3, 10000 ) ) # approximately .163

print( simulate_dice_game( 5, 4, 10000 ) ) # approximately .765
```

Exercise 2: Vigenère cipher

File: `E02_vigenere.py`

One-line summary

In this exercise, you encode a string with a Vigenère cipher.

Context

With a Caesar cipher, letters are encoded by replacing them with a letter a specific distance from them in the alphabet; for instance, if the distance is 3, then "ABC" would be encoded as "DEF". The alphabet circles around after the last letter, so with distance 3, "XYZ" would be encoded as "ABC".

A Vigenère cipher is similar to a Caesar cipher, except that the distances used may differ for each position in the string to be encoded, and are based on a code word. The distances used are the indices of the letters of the code word in the alphabet. For instance, if the code word is "CAT", then the distances are 2, 0, and 19, derived from respectively the C, the A, and the T.

Each letter of the string will be encoded using the distance associated with the letter at the same position in the code word. As the code word tends to be shorter than the string to be encoded, it is assumed to be repeated as often as needed to encode the whole string. For instance, if the code word is "CAT", and the string to be encoded is 9 letters long, the code word used for the string as a whole is "CATCATCAT".

Exercise description

Write a function `vigenere()` which encodes a string using a Vigenère cipher, as described in the context, and returns the encoded string. The function gets two parameters: the string to be encoded, and the code word. You may assume that all letters in the string to be encoded and the code word are capitals. There may be different characters in the string to be encoded (such as punctuation and spaces), but these do not need to be encoded. The code word will only contain letters from the alphabet.

Examples

`vigenere( "CATCATCAT", "DOG" )` returns `"FOZFOZFOZ"`

`vigenere( "ATTACKATDAWN", "LEMON" )` returns `"LXFOPVEFRNHR"`

`vigenere( "ATTACK AT DAWN", "LEMON" )` returns `"LXFOPV MH OEIB"`

Hint

Use the `ord()` and `chr()` to turn characters into their ASCII code and back, and remember that in the ASCII code, the letters of the alphabet have consecutive numbers in alphabetical order (A=65, B=66, C=67, etc.). You can use the modulo (%) operator to make the alphabet circle around, so that after the Z comes the A again.

Exercise 3: Morse code

File: `E03_morse_code.py`
Data file: `morsecode.txt`

One-line summary

In this exercise your implement a function that decodes messages sent in Morse code.

Context

Morse code is a system for transmitting messages through an alternation of signals with a long and short duration. Short signals are known as dots, whereas long signals are referred to as dashes. Originally, the signals were electrical pulses along a telegraph wire. Morse code, however, can be used to transmit messages through audio tones or light flashes as well. In this exercise, you will decode messages in Morse code.

Exercise description

Your task is to write a function `decode()`. Given a text file that contains the Morse code alphabet, the `decode()` function should decode a message sent in Morse code. This message is provided as a string. The function `decode()` thus takes two arguments: `message` (string) and `keyfile` (string; the name of the file that contains the Morse code alphabet, in this case `morsecode.txt`). The function should return the decoded message. Note that the function *must* read the Morse code alphabet from the file provided!

The contents of the file `morsecode.txt` are as follows: each line contains a character, followed by a tab, followed by a series of periods and dashes. The periods and dashes are the Morse code for the character with which the line starts. If this file does not exist, the function should return an empty string.

The string to be decoded contains only periods (`.`), dashes (`-`), spaces, and forward slashes (`/`). The spaces separate the different encoded characters. A forward slash ("/") indicates a space between words, and will always have a space to the left and right of it. The forward slash is typically not part of the Morse code alphabet, and is therefore not in the file `morsecode.txt`. You need to make sure that your function processes this character appropriately nonetheless.

Example

`decode( ".... . .-.. .-.. --- --..-- / .-- --- .-. .-.. -..", "morsecode.txt" )` should return `"HELLO, WORLD"`

Hint

Store the Morse code alphabet in a dictionary after reading it from the file.

Exercise 4: Astronomy

File: `E04_astronomy.py`

One-line summary

In this exercise, you complete several classes which describe astronomical objects.

Context

A star system consists of a star and zero or more planets. A planet may have zero or more moons circling around it. This exercise describes a star system in an object oriented way.

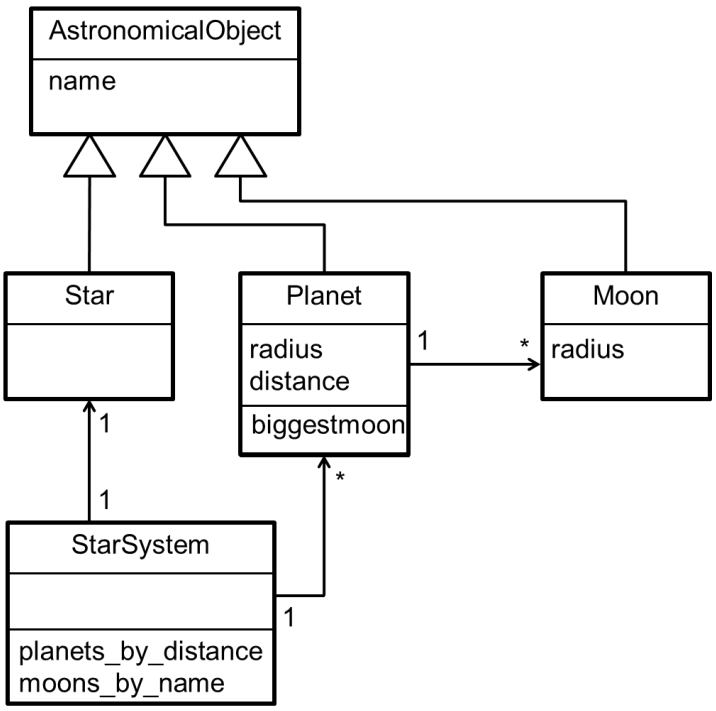
Exercise description

The code for this exercise consists of several classes, with relationships between them, which describe a simple star system. Some of the classes have methods which provide information on the star system or the objects in it. You have to complete the classes.

There is a class `AstronomicalObject`. The only attribute this class has is `name` (string). `Star`, `Planet`, and `Moon` are all derived from `AstronomicalObject`. `Star` has no extra attributes. `Planet` and `Moon` have a `radius` (float, in kilometers). `Planet` has a `distance` (float, in millions of kilometers) from the star which it belongs to. `Planet` also has a number of moons associated with it. `Planet` has a method `biggestmoon()`, which returns the moon with the largest radius belonging to the planet (or `None` if the planet has no moons). A `StarSystem` consists of a single `Star` and a list of `Planet` objects. `StarSystem` has two methods: `planets_by_distance()` which returns a list of all the planets belonging to the star system, ordered by their distance from the star from smallest distance to largest distance, and `moons_by_name()` which returns a list of all the moons in the star system, alphabetically ordered by name. All of this is schematically represented by the UML diagram on the next page.

You have to complete the classes, in the following way (it works best if you take these steps in the order given, as you can test after each of them; also, they are ordered from easy to hard):

- You have to make sure that both `Planet` and `Moon` are derived from `AstronomicalObject`.
- You have to create suitable `__init__()` and `__repr__()` methods for both `Planet` and `Moon`; the required parameters for the `__init__()` methods are given in the code.
- You have to implement the `biggestmoon()` method for `Planet`.
- You have to implement the `planets_by_distance()` and `moons_by_name()` methods for `StarSystem`.



Test

The code given constructs a simple star system consisting of our sun, planets Mercury, Venus, Earth, Mars, and Pluto (no discussion about whether Pluto is a planet or not), and the moons of Earth, Mars, and Pluto (Mercury and Venus have no moons).

Example outputs are:

`print( earth )` will use the `__repr__()` method of `Planet` to produce the output
`Earth(6371, 149.6, [Luna(1737)])`
i.e., the name “Earth”, the radius 6371, the distance from the sun 149.6, and a list of moons, of which there is only one, namely Luna with a radius of 1737.

A call to `planets_by_distance()` of the Sol star system will produce a list containing Mercury, Venus, Earth, Mars, and Pluto in that order. Printing this list will also print the details of each of these planets, if `__repr__()` of `Planet` has been implemented correctly.

A call to `moons_by_name()` of the Sol star system will produce a list containing Charon, Deimos, Hydra, Kerberos, Luna, Nix, Phobos, and Styx, in that order.

Outputs for all tests can be seen in the code file.

Exercise 5: Smallest difference

File: `E05_smallest_difference.py`

One-line summary

In this exercise you have to determine the time complexity of five functions which determine the smallest difference between any of two numbers on a list of numbers.

Exercise description

Each of the functions, which are all named `smallestdiff`*x*`()` (*x* being a number), gets one parameter: `numlist` which is a list of integers. The function returns the smallest difference between any of two numbers on the list. E.g., if the list is `[ 3, 0, -5, 5 ]` then the answer is `2`, as the difference between 3 and 5 is 2 and no smaller difference can be found.

For each of the functions, write as a comment in the file what the time complexity is (immediately below the function), and give a brief explanation on how you determined this. If you do not add an explanation, the answer is automatically considered to be wrong. In your big-O notation of the time complexity, you may use `n` to refer to the length of the list.

Background information

You may assume the following:

- The function `abs()` is `O(1)`
- The function `len()` is `O(1)`
- The function `max()` is `O(n)` , where `n` is the length of the sequence from which you take the maximum
- The function `min()` is `O(n)` , where `n` is the length of the sequence from which you take the minimum
- The iterator `range()` is `O(1)`
- The list method `append()` is `O(1)`
- The list method `remove()` is `O(n)`, where `n` is the length of the list
- The list method `sort()` is `O(n*log(n))`, where `n` is the length of the list
- Taking a sublist from a list is `O(n)`, where `n` is the length of the sublist

Here are descriptions of the algorithms:

- `smallestdiff1()` processes each number on the list and determines the difference between that number and all of the remaining numbers on the list, keeping track of the smallest difference.
- `smallestdiff2()` first sorts the list, then calculates for each number on the list the difference between that number and the next number, keeping track of the smallest difference.

- `smallestdiff3()` uses a copy of the list. In a loop it determines the largest number on the list, removes it, and then determines the difference between that largest number and the number which is now the largest on the list. It continues doing that until the list is reduced to one number. It keeps track of the smallest difference.
- `smallestdiff4()` builds a new list consisting of all the differences between all pairs of numbers on the original list. It then sorts the new list and returns the first number in it.
- `smallestdiff5()` recursively returns the smallest of two numbers, namely: (1) the difference between the first two numbers of a sorted version of the list, and (2) the smallest difference found on the remainder of the list. The base case is that the list only has two numbers on it, of which the difference is returned.