COVER PAGE FOR A WRITTEN EXAMINATION/TEST

Name of subject	: Data Structures & Algorithms
Subject code	: 822188-B-6
Date of examination	: December 11, 2020, 8.30 AM
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. 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	: December 11, 2020, 8.30 AM
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. 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!

Exam 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: Coin sequence

File: `E01_coin_sequence.py`

One-line summary

In this exercise, you calculate the probability of a sequence of identical coin tosses in a larger series of coin tosses.

Context

Suppose you flip a coin *n* times. What is the probability that there will be a sequence of *k* identical coin flips in a row in your series of *n* coin flips? Assume that you do not know how to calculate the exact answer to this question. In that case, you can approximate the probability through a Monte Carlo simulation. A Monte Carlo simulation of this problem consists of a large number *nsim* of simulations of *n* coin flips. For each simulation, you determine whether or not a sequence of *k* identical coin flips in a row occurred. The probability is the number of simulations in which such a sequence did occur divided by the total number of simulations.

Exercise description

Write a function `simulate()` that estimates the above-mentioned probability on the basis of its three arguments: *n* (the number of coin flips), *k* (the length of the sequence of identical coin flips), and *nsim* (the number of simulations). You may assume that *n*, *k*, and *nsim* are positive integers and that $k \leq n$. The function `simulate()` should return the probability of a sequence of *k* identical coin flips in a row in a series of *n* coin flips. Note that the results of identical calls to `simulate()` will (and should!) differ slightly each time you run the code, due to the fact that the series of coin flips in the simulations are randomly generated.

Examples

```
print( simulate( 3, 2, 100000 ) ) # approximately 0.750

print( simulate( 5, 3, 100000 ) ) # approximately 0.500

print( simulate( 4, 3, 100000 ) ) # approximately 0.375

print( simulate( 10, 4, 100000 ) ) # approximately 0.465
```

Exercise 2: Weighted 11-proof

File: `E02_weighted11proof.py`

One-line summary

In this exercise you calculate the control digit for a number using a weighted 11-proof calculation.

Context

Many systems which use personalized numbers have an 11-proof check to determine whether a number is legal. For instance, bank account numbers are using it as a quick check to see if a number is correctly typed. The 11-proof check calculates a control digit for a number, which is usually added to the start or the end of the number for which the calculation is done. The weighted 11-proof check diverges from the common 11-proof check in that it uses a list of weights in the calculation.

Exercise description

The function `weighted11proof()` gets two parameters: an integer `number`, and a list of integers called `weights`. It calculates the control digit for the weighted 11-proof for `number`, and returns it. The calculation is as follows:

You multiply each digit of the number by the "appropriate" weight (see below for an explanation of what "appropriate" is in this case) in the `weights` list. You take the sum of all these calculations. Then you take the remainder R when dividing this sum by 11. If R is zero then the control digit is zero, if R is 1 then the control digit is 1, otherwise the control digit is 11 minus R.

What is the "appropriate" weight in the `weights` list? For the last digit of the number, you take the first weight in the `weights` list; for the next-to-last digit in the number, you take the second weight in the `weights` list; etc. If the `weights` list has at least as many weights as there are digits in the number, this suffices. Otherwise, when you run out of weights in the `weights` list you simply restart at the beginning again.

You may assume that the function is called with a positive integer as `number`, and a list of `weights` which are all positive integers (none are zero). The list of `weights` will contain at least one weight.

Examples

Suppose the list of weights is [2, 4, 8, 7]. For clarity, in the calculations below I have underlined the weights.

If the number is 619, then the calculation is 2*9 + 4*1 + 8*6 = 70. Note that the digits of the number are processed from right to left, while the weights are processed from left to right. 70 divided by 11 gives remainder 4. 11 – 4 = 7. Thus the control digit is 7.

`weighted11proof( 619, [2, 4, 8, 7] )` returns 7

If the number is 30000, then the calculation is 2*0 + 4*0 + 8*0 + 7*0 + *(here we have to restart at the beginning of the weights list)* 2*3 = 6. 6 divided by 11 gives remainder 6. 11 – 6 = 5. Thus the control digit is 5.

`weighted11proof( 30000, [2, 4, 8, 7] )` returns `5`

If the number is 612053 then the calculation is 2*3 + 4*5 + 8*0 + 7*2 + 2*1 + 4*6 = 66. 66 divided by 11 gives remainder 0. Remainder zero is returned as is.

`weighted11proof( 612053, [2, 4, 8, 7] )` returns `0`

Hint

To start, turn the number in a list of its digits (you may have to turn the number into a string in between). During processing, it might be helpful to reverse the list of digits (it just simplifies things).

Exercise 3: Spell Checker

File: `E03_spell_check.py`

One-line summary

In this exercise your implement a function that corrects spelling errors.

Exercise description

One method for removing spelling errors from a text is to use a dictionary of common spelling errors. The file `wikipedia.txt` is a list of misspellings by authors of Wikipedia articles. Each line in `wikipedia.txt` consists of a misspelling and the correct spelling of a word, separated by a space.

Your task is to write a function `correct_spelling()`. Given a dictionary of common spelling errors, the `correct_spelling()` function should correct the spelling in a sentence. This sentence is provided as a string. The function `correct_spelling()` thus takes two arguments: `sentence` (string) and `dictfile` (string; the name of a file that contains a spelling error dictionary, in this case `wikipedia.txt`). The function should return a corrected version of the sentence, in which misspellings that are in `wikipedia.txt` are replaced by the correct spelling of the word. If the file with the dictionary of spelling errors does not exist, the function should return -1. You may assume each misspelling occurs in the dictionary only once.

You may assume that `sentence` contains only letters and spaces, so no punctuation. You do not need to take capitals into account.

Examples

The function `correct_spelling()` should correct all spelling errors in the sentence "I recomend the aplication of a progrom that corrects each and eveyr mispelling". The function call

```
correct_spelling("I recomend the aplication of a progrom that corrects each and eveyr mispelling", "wikipedia.txt")
```

should therefore return:

```
I recommend the application of a program that corrects each and every misspelling
```

The problem with this type of approach to a spelling checker is that only spelling errors that exist in the dictionary are corrected. This is not the case for all spelling errors in the sentence "I am disatisfied with tghe perfromance of tihs algoritm", because the spelling error "perfromance" is not in the spelling error dictionary in `wikipedia.txt`. The function call:

```
correct_spelling("I am disatisfied with tghe perfromance of tihs algoritm", "wikipedia.txt")
```

therefore returns

```
I am dissatisfied with the perfromance of this algorithm
```

Exercise 4: Backpack

File: `E04_backpack.py`

One-line summary

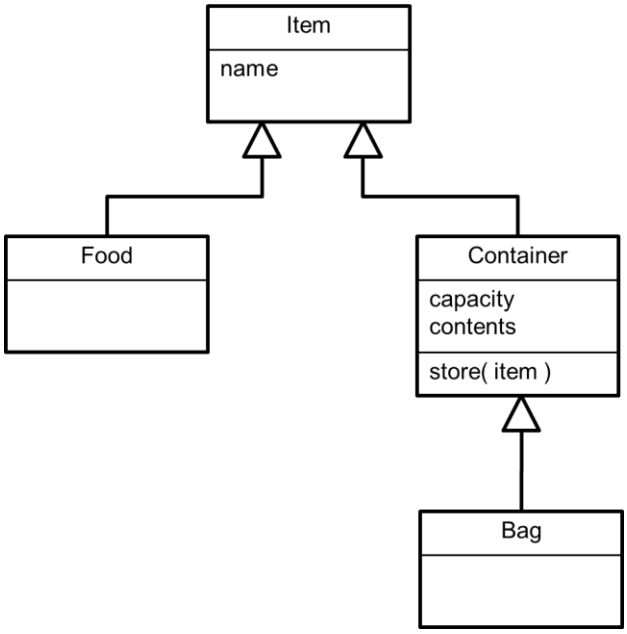
In this exercise, you implement a rudimentary inventory system that can be used in a computer game.

Context

Many computer games have an inventory system. The player collects items which they can store in, for instance, a backpack. The backpack has limited capacity, but the player can store other containers in the backpack, to expand the storage space. For instance, in a game a player may have a backpack in which they can store a maximum of 12 items. However, among these items there may be bags, which can hold 4 items. So, by storing 12 bags in their backpack, the player has expanded their capacity to 48 items. Each of these 48 items can again be a bag, which would expand the player's inventory space to 192 items (as long as the player is able to find enough bags in the game).

Exercise description

The setup for the code that you have to complete is simple. There is an `Item` class, from which all other classes are derived. `Food` is an `Item`. `Container` is also an `Item`. `Bag` is a `Container`. Below you find an UML diagram which displays this. In this exercise, the only class that you must implement is `Container`.



An `Item` has a `name` attribute (you may ignore the `optional` attribute which you see in the code; it is meant for the extra challenge described below). `Food` has no extra attributes. `Container` you have to implement. `Bag` has no extra attributes beyond what `Container` offers.

`Container` has four methods, which you must all implement:

- `__init__( self, name, capacity )`: This method initializes a new container; it method gets a `capacity` attribute, which indicates the number of items that may be stored in the container. The method should also prepare the container for storing items in it.
- `__contains__( self, item )`: As you (should) know, implementing the `__contains__()` method allows the `in` operator to work on the class (via operator overloading); this method should return `True` if `item` is in the container, or can be found if one explores all the containers that are in the container. Otherwise it returns `False`. (For instance, if a backpack contains a bag, and that bag contains a purse, and that purse contains an orange, then there is an orange `in` the backpack.)
- `store( self, item )`: This method stores the `item` in the container, as long as the container has sufficient capacity left. It returns `True` if it succeeds in storing the item, otherwise `False`. Note: you just need to check if there is room in the container itself, and not in any container which is stored in the container.
- `__repr__( self )`: This method returns a string which contains the name of the container, and, between parentheses, the capacity and all the items stored in the container, including what you see if you look in the containers stored in the container, etc. In principle this whole "unpacking" is done automatically if the contents of the container are held by a list, as the `__repr__()` method of a list does this for you.

Examples

The `main()` function runs some tests on the classes that you create. It first creates a "big bag" container with capacity 4 in which it tries to store 5 food items (the fifth cannot be stored). Next, it creates a new "big bag" with capacity 4, a "medium bag" with capacity 3, a "small bag" with capacity 2, and a "tiny bag" with capacity 1. It stores the medium bag in the big bag, and the small and tiny bags in the medium bag. It also stores some food items in the various bags. It does several tests on the constructed collection of bags, for which the desired output is shown in the Python file.

Hint

Searching through the bags is very much like tree search.

Optional challenge

Unless you take precautions, things will go drastically wrong if a container is allowed to contain itself, or if container A is allowed to store container B which already contains container A. This would lead to "loops" in the inventory system. As an optional challenge, you may write some code in the `store()` method which prevents this from happening (simply returns `False` when you try to do this). If you decide to do this, please set the `optional` attribute in `Item` to `True`. The tests in `main()` will then include tests on this optional feature.

Exercise 5: String count

File: `E05_stringcount.py`

One-line summary

In this exercise you have to determine the time complexity of five functions which determine the count of a particular letter in a string.

Exercise description

Each of the functions, which are all named `countx()` (*x* being a number), gets two parameters: `s` which is a string, and `letter` which is a single letter. The function returns the number of times the letter occurs in the string.

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 string.

Background information

You may assume the following:

- The function `len()` is `O(1)`
- The function `int()` is `O(1)`
- The function `list()` is `O(n)`, where `n` is the length of the object turned into a list
- The list method `remove()` is `O(n)`, where `n` is the length of the list
- The string method `split()` is `O(n)`, where `n` is the length of the string
- The string method `join()` is `O(n)`, where `n` is the length of the list that is joined
- Taking a substring from a string is `O(n)`, where `n` is the length of the substring

Here are descriptions of the algorithms:

- `count1()` processes each letter of the string, adding 1 to a count if it is the sought letter.
- `count2()` initializes a count to the length of the string, then subtracts 1 from the count each time it finds a letter which should not be counted.
- `count3()` turns the string into a list, then counts how often the letter can be removed from the list.
- `count4()` splits the string on the letter, then joins it again. By comparing the length of the string before the operation with the length of the string after the operation, the count can be determined.
- `count5()` recursively divides the string in two, until it needs to deal with a string of a single letter. If that letter is the sought letter, it returns 1, otherwise zero. By adding up the results of each recursive call, the total count is determined.