COVER PAGE FOR DIGITAL ON CAMPUS EXAMINATION IN CANVAS

Name of the course: Data Structures & Algorithms
Course code: 822188-B-6
Date of the examination: 30-04-2022
Duration of the examination: 3 hours
Lecturer: Pieter Spronck **ANR**: 447823
Lecturer's telephone number available during examinations: 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.

All mobile devices (telephone, tablet, etc.) must be switched off in the exam room and should be stored in your jacket or bag. You are not allowed to carry these during the exam. It is also not allowed to wear a watch (both analog and digital) in the exam room.

Exam instructions

1. Only the following examination aids are permitted during the exam:
   a. Books, lecture notes (either in printed form, book form, or digitally supplied on the university computers):
      - Python documentation as supplied with Python
      - 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
   b. Blank scrap paper and pens. Scrap paper and other hard copy materials may be taken home by the students at the end of the exam.
   c. Student's own calculator (any).
   d. Internet sources: https://jupyterlab.uvt.nl (this is the Jupyterlab server of the university).
   e. Computer programs: Any of the programs in the default installation, including Python, Anaconda, Spyder, Jupyter, and IDLE.
2. In case a student does not comply with instructions paragraph 1, this can be qualified as (a suspicion of) fraud. The EER, Rules and Guidelines of the Examination Board of the School and the General Guidelines apply.
3. The exam has 5 questions.
4. The expected passing score (the score to pass the exam) is 6. The score is calculated by taking 40% of the midterm score and 60% of the exam score, unless the exam score is higher than the midterm score (or the student has no midterm

score), in which case the exam score counts for 100%. Standard rounding to half points applies, except that 5.5 or higher will be rounded to 6, while 5.4 or lower will be rounded to 5.

5. Each question is worth 2 points.
6. All questions require the student to write code, or write comments in a code file. For writing code, the main requirement for getting all the points for the answer is that the code processes the tests given by the instructors correctly. In degenerative cases (such as code being unintelligible, code being exceptionally slow, code only being able to handle the test cases supplied in the template files but not other test cases, or a student replacing a loop with a long list of conditional statements) points will be subtracted. Code that crashes may not get points at all.
7. Each question has its own template code file, in which the students have to fill in a function, or add comments to functions. These code files are to be submitted as answers. Please meet the following requirements:
   a. Students should add their name and student number at the top of each file as a comment.
   b. Only the standard Python modules may be used to develop code (as discussed in the first 32 notebooks), so no numpy or pandas.
   c. Students may add extra tests in the main() function that is found in each of the code files.
   d. 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.
   e. Functions never need to ask the user for input. All inputs for the functions are supplied via parameters.
   f. 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.
   g. Code files should be tested before submitting, especially if notebooks were used to develop the code (to catch copying mistakes).
   h. If a student wants to add remarks to submitted code, please do so as comments in the code files that are submitted.
   i. Five separate code files must be submitted. 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!
8. Not all questions are equally hard, and it tends to vary between students what they consider to be hard. If you are stuck on a question, turn to the next one, and return to the question you got stuck on later. Think before you start coding – finding a good approach tends to save a lot of time.
9. If a student has questions during the exam, they should ask those to an instructor present, or 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 on the contents of the exam with anyone but the instructors is not allowed!

10. The instructors will supply a discussion of each of the questions on Canvas after the grading has been completed. If any questions remain after you have viewed these discussions, please contact the instructors via the Canvas Inbox to arrange an exam inspection. This may be arranged via Zoom or Teams.

You can start the examination now, good luck!

Resit Data Structures & Algorithms 2021/2022

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, and find the exam (it will be either under Assignments or Quizzes). 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.

We wish you good luck.

Exercise 1: Same letters

File: `E01_same_letters.py`

One-line summary

In this exercise you extract from a list of words all the words that have the same letters as a given word.

Exercise description

The function `same_letters()` gets two parameters: a string `word`, and a list of strings `wordlist`. The function creates an output list of those words in `wordlist` which contain the same letters as `word`, and returns that output list. Words in the output list should remain in the same order as they were in `wordlist`.

"Containing the same letters" means that the output list contains those words which contain each letter in `word` at least once, and no letters that are not in `word`.

The letters do not have to occur the same number of times. For instance, if `word` is "barber" and the word "bear" is in `wordlist`, then "bear" goes in the output list, even though it only contains one "b" and one "r" while "barber" contains two of each. The same also holds in the reverse order: if `word` is "bear" and "barber" is in `wordlist`, then "barber" goes in the output list, even though it contains the "b" and "r" twice, while "bear" only contains them once.

You may assume that all the words consist of lower-case letters of the alphabet only.

Examples

```
same_letters( "bear",   [ "bare", "beer", "barber", "bar", "bears" ] )
```
returns `['bare', 'barber']`

```
same_letters( "barber", [ "bare", "beer", "bear", "bar", "bears" ] )
```
returns `['bare', 'bear']`

```
same_letters( "grape",  [ "pear", "banana", "raspberry" ] )
```
returns `[]`

Exercise 2: Happy numbers

File: `E02_happy.py`

One-line summary

In this exercise, you determine whether a number is a happy number.

Context

Take an integer, and add up the squares of its digits. This gives a new integer. Repeat the procedure with the new integer, and continue repeating the procedure until one of two things happens: (1) you end up at 1, or (2) you end up at an integer that you already had before. If you end up at 1, the integer you started with is "happy." Otherwise, it is "sad." Integers 0 and lower are "sad" by definition.

Here are two examples:

13 is happy, because $1^2+3^2=10$, and $1^2+0^2=1$, so you end up at 1.

4 is sad, because $4^2=16$, $1^2+6^2=37$, $3^2+7^2=58$, $5^2+8^2=89$, $8^2+9^2=145$, $1^2+4^2+5^2=42$, $4^2+2^2=20$, $2^2+0^2=4$, and so we end up at 4 again.

Note: the procedure is guaranteed to end (this is easy to prove mathematically, but we won't bother you with that), so if you implement it correctly you do not have to be afraid of endless loops.

Exercise description

Write a function `happy()` which takes one argument, namely an integer. The function determines whether the argument is "happy." If it is, the function returns `True`, otherwise `False`.

Examples

`happy( 0 )` returns `False`
`happy( 4 )` returns `False`
`happy( 13 )` returns `True`

Exercise 3: Removing comments from a file

Files: `E03_remove_comments.py, infile1.txt` to `infile 5.txt`

One-line summary

In this exercise you remove comments from a file which are marked by "/*" and "*/".

Exercise description

Write a function `remove_comments()` which gets two string parameters: the name of an input file and the name of an output file. The function reads the contents of the input file, removes all the comments from the contents, and writes the adapted contents to the output file. It returns the number of characters written to the output file. If the input file does not exist, the function returns `-1`. You do not need to check for any other file errors.

The start of a comment is marked with the two-character combination "/*" (forward slash followed by asterisk). The end of the comment is marked by the first occurrence of the two-character combination "*/" after the start of the comment. The start and end of the comment are considered to be part of the comment, so when you remove a comment, they have to be removed as well.

A few remarks:

- Comments may stretch over multiple lines of a file.
- There may be more than one comment in a file.
- There is no overlap between the start and end of a comment, so the combination "/*/" is *not* the start *and* end of a comment, but the combination "/**/" is the start and end of a comment.
- If you find the start of a comment which is never followed by the end of a comment, or the end of a comment which is not preceded by the start of a comment, then those are *not* comments.
- Comments are never "nested," so you only need to search for the first occurrence of the end of a comment after the start of a comment and not take into account what is actually in the comment.

Example

If the input file contains the following three lines (without a newline at the end of the last line):

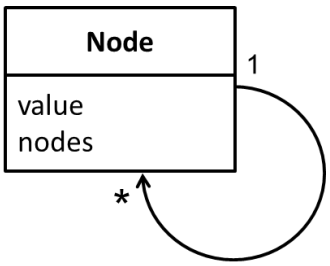
```
XX/* comment */XX/* Another
comment */XX
XXX/**/XXX
```

then the output file will contain the following two lines:

```
XXXXXX
XXXXXX
```

and the function returns `13` (12 `X`s and one newline).

Exercise 4: Tree



File: `E04_tree.py`

One-line summary

In this exercise, you examine a tree in which each node can have an arbitrary number of branches; you count the number of nodes in the tree and search for a value in the tree.

Context

When trees are used in software, they are often binary trees, but not necessarily so. A node in a tree can have any number of branches, if it stores references to the branches in, for instance, a list.

Exercise description

The exercise file contains a class `Node`, which has a `value` and a list of `nodes`. The nodes in the list are the roots of the subtrees which form the branches under the node. If the list is empty, then the node is a leaf node.

Two methods in the class `Node` must be implemented.

The method `numnodes()` gets no arguments, and returns the number of nodes in the (sub)tree of which the node is the root (this includes the node itself).

The method `find()` gets one argument, namely a value, and returns `True` if the value is found somewhere in the tree, and `False` otherwise. As there is no ordering in the implementation of the tree, you will have to search the entire tree to find the value. You can do this using a depth-first search or a breadth-first search (your choice).

Example

The following code creates a tree with three levels, with value 1 at the top level, values 2 to 4 at the second level, and values 5 and 6 at the third level below 2. This tree has 6 nodes, and the method `find()` returns `True` for values 1 to 6, and `False` for any other value.

```
tree = Node(1,[Node(2,[Node(5,[]),Node(6,[])]),Node(3,[]),Node(4,[])])
print( tree.numnodes() )
for i in range( 10 ):
    print( i, tree.find( i ) )
```

Note

Your code must actually traverse the tree. Trying to use a string representation of the tree created with the `__repr__()` method is considered wrong (we state this explicitly as some students tried to do this on a previous exam).

Exercise 5: Reverse

File: `E05_reverse.py`

One-line summary

In this exercise you have to determine the time complexity of five functions which reverse a string.

Exercise description

Each of the functions, which are all named `reverse_`*x*`()` (*x* being a number), gets one parameter: `sentence` which is a string. The function returns the reverse of this 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:

- Concatenating two strings, one of length `m` and one of length `n`, is `O(m+n)`
- Taking a substring of length `n` from a string is `O(n)`
- The iterator `range()` is `O(1)`
- The function `len()` is `O(1)`
- Using the `list()` function on a string of length `n` is `O(n)`
- The string method `join()` is `O(n)`, where `n` is the length of the list argument
- The list method `reverse()` is `O(n)`, where `n` is the length of the list

Here are descriptions of the algorithms:

- `reverse_1()` takes each letter of the string one by one and adds it at the start of a new string.
- `reverse_2()` processes the letters of the string by index from last to first, and adds them to a new string.
- `reverse_3()` removes the last letter from the string and adds it to a new string, continuing to do this until the original string is empty.
- `reverse_4()` turns the string into a list, reverses the list, and then joins the list to become a new string.
- `reverse_5()` recursively splits the string into the first letter and the rest of the string, and then adds the first letter to the end of what the recursive call returns.