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: 17-10-2022
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) and https://docs.python.org.
   e. Computer programs: Any of the programs in the default installation, including Python, Anaconda, Spyder, Jupyter, PyCharm, Visual Code, 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. For the course, the expected passing score (the score to pass the exam) is 6. This midterm counts for 40% of the final grade for the course. 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. 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 create classes. A question which needs input files, will have these supplied as well, and these input files should be placed in the same folder as where the template code files are placed. The 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! Renaming a `.ipynb` file to a `.py` file will not work, as it will not magically turn a `.ipynb` file into a Python file.
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!

Exercise 1: Sum and product of three integers

File: `E01_sum_and_product.py`

One-line summary

You are given two integers, and you have to find three integers of which the sum makes the first of the two given integers, and the product makes the second of the two given integers.

Exercise description

The function `sum_and_product()` gets two parameters, which are both integers valued 0 or higher. You should find three integers (also 0 or higher), which, when added up, produce the first of the two parameters, and when multiplied with each other produce the second of the two parameters.

For instance, if the first parameter is 15 and the second is 105, then three integers which produce these are 3, 5, and 7, because 3+5+7=15 and 3*5*7=105.

You return the three integers as a tuple, where the three integers are ordered from low to high. In the example above, you return (3, 5, 7).

If no solution can be found, you return (-1, -1, -1).

Examples

`sum_and_product( 100, 7038 )` returns `(3, 46, 51)`

`sum_and_product( 100, 98 )` returns `(1, 1, 98)`

`sum_and_product( 2, 1 )` returns `(-1, -1, -1)`

Exercise 2: Comparing integers in files

File: `E02_compare_files.py`
Input files: `numbers1.txt` to `numbers6.txt`

One-line summary

In this exercise, you determine the fraction of numbers in one file which are higher than all the numbers in another file.

Exercise description

The function `compare_files()` gets two parameters, both of which are names of files. These files contain integers, one integer per line. They contain nothing else. You may assume that each file contains at least one number. The numbers in the files can appear in any order.

The function determines the fraction of the integers in the second file which are higher than all the integers in the first file. I.e., suppose that the highest integer in the first file is 50, and there are 100 integers in the second file, of which 75 are 51 or higher, then the requested fraction is 0.75. The function returns this fraction.

In case one or both of the file names are of files which do not exist, then the function returns -1. You do not need to check for any other file errors.

Example

`numbers1.txt` contains the numbers `0, 1, 1, 0, 2`
`numbers3.txt` contains the numbers `0, 1, 2, 3, 4, 5, 6, 7, 8, 9`
The highest number in `numbers1.txt` is 2, and there are 7 out of 10 numbers in `numbers3.txt` which are higher than 2. Thus the function returns `0.7`.

Exercise 3: Most frequent N-gram

Files: `E03_most_frequent_ngram.py`

One-line summary

In this exercise you find the N-gram of a particular length which occurs the most in a string.

Context

An N-gram is a sequence of N letters. For instance, “ABA” is a 3-gram, while “ABABC” is a 5-gram. You can investigate N-grams which occur in a string. For instance, in the string “ABABC” there are three different 2-grams, namely: “AB”, “BA”, and “BC”. The 2-gram “AB” occurs twice, as you can see. The string “AAAAA” contains only one 2-gram, namely “AA”, which occurs four times.

Exercise description

Write a function `most_frequent_ngram()` which gets two parameters: a string which is a sequence of letters, and a positive integer which indicates a length. The function should return the N-gram of the specified length which occurs the most in the string. If there are multiple candidates which occur the most, you may return any of them (though in the test examples provided there is always only one answer). You may assume that the length specified is at least 1 and at most the length of the string.

Hint

A possible approach (though not the only approach) for this exercise is to build a dictionary where the N-grams are the keys and the values are the corresponding counts.

Examples

`most_frequent_ngram( "AABBB", 1 )` returns `"B"`, as the string contains 3 Bs and only 2 As.

`most_frequent_ngram( "AABBB", 2 )` returns `"BB"`, as the 2-gram BB occurs twice, and the other 2-grams (AA and AB) each only occur once.

`most_frequent_ngram( "AABBB", 5 )` returns `"AABBB"`, as there is only one 5-gram in this string, namely the whole string.

Exercise 4: Citizens

File: `E04_citizens.py`

One-line summary

In this exercise, you create citizens which live in a city, where cities are part of a country; you calculate the average age of the citizens in a city, and of all the citizens in a country.

Exercise description

The exercise file contains a class `Citizen`, which has a `name` and an `age`. This class is already complete.

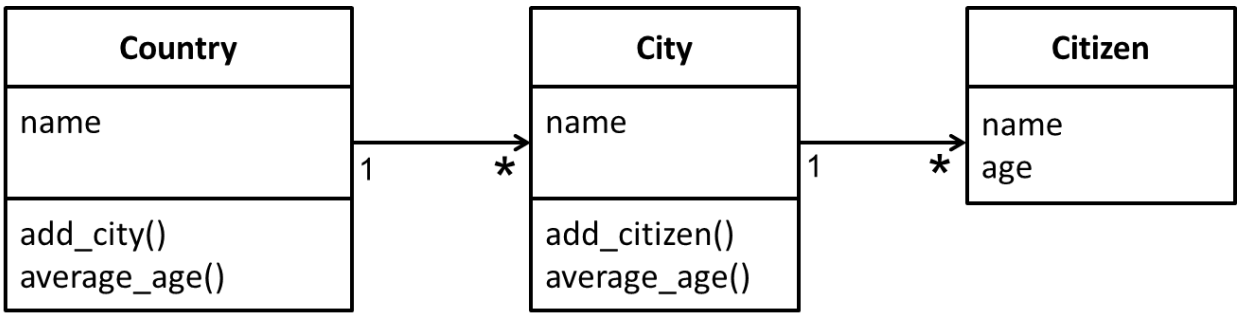
There is also a class `City`, which has a `name`. A city may house multiple citizens, which are added to the city using the method `add_citizen()`. The method `average_age()` calculates the average age over all citizens in the city. It returns -1 if there are no citizens in a city. You should create reasonable implementations for the methods `__init__()`, `add_citizen()`, `average_age()`, and `__repr__()`.

Finally, there is a class `Country`, which has a `name`. A country may have multiple cities, which are added to the country using the method `add_city()`. The method `average_age()` calculates the average age over all citizens in the country. It returns -1 if there are no citizens in a country. You should create reasonable implementations for the methods `__init__()`, `add_city()`, `average_age()`, and `__repr__()`.

Below you find the UML diagram which describes the class structure to be created.

Example

In the template code given, two cities are created, one with four citizens and one with six. These are added to a country. Average ages are printed for the cities and the country, and the correct answers are specified next to the print-statements.



Exercise 5: Housing market

File: `E05_housing_market.py`

One-line summary

In this exercise you calculate the sales prices for different kinds of properties on the housing market, based on location and type of property.

Exercise description

The template file for the exercise contains a class `Location` and a class `Property`. A location has a `name` and a square-meter `price`. A property has a `name`, a `size`, and a `location`. The price of a property is calculated by multiplying the size by the square-meter price of the location. E.g., a property of size 100 with a square-meter price of 1000, costs 100,000. These two classes have already been implemented according to the given specifications. This calculation is done by a method `price()`.

Three classes inherit from `Property`: `Apartment`, `House`, and `Castle`. You must implement these three classes. In particular, you have to ensure that the `price()` method for each of these classes calculates the correct price. Make sure that you implement the correct inheritance relationships!

An `Apartment` is a `Property` and behaves just like the general property.

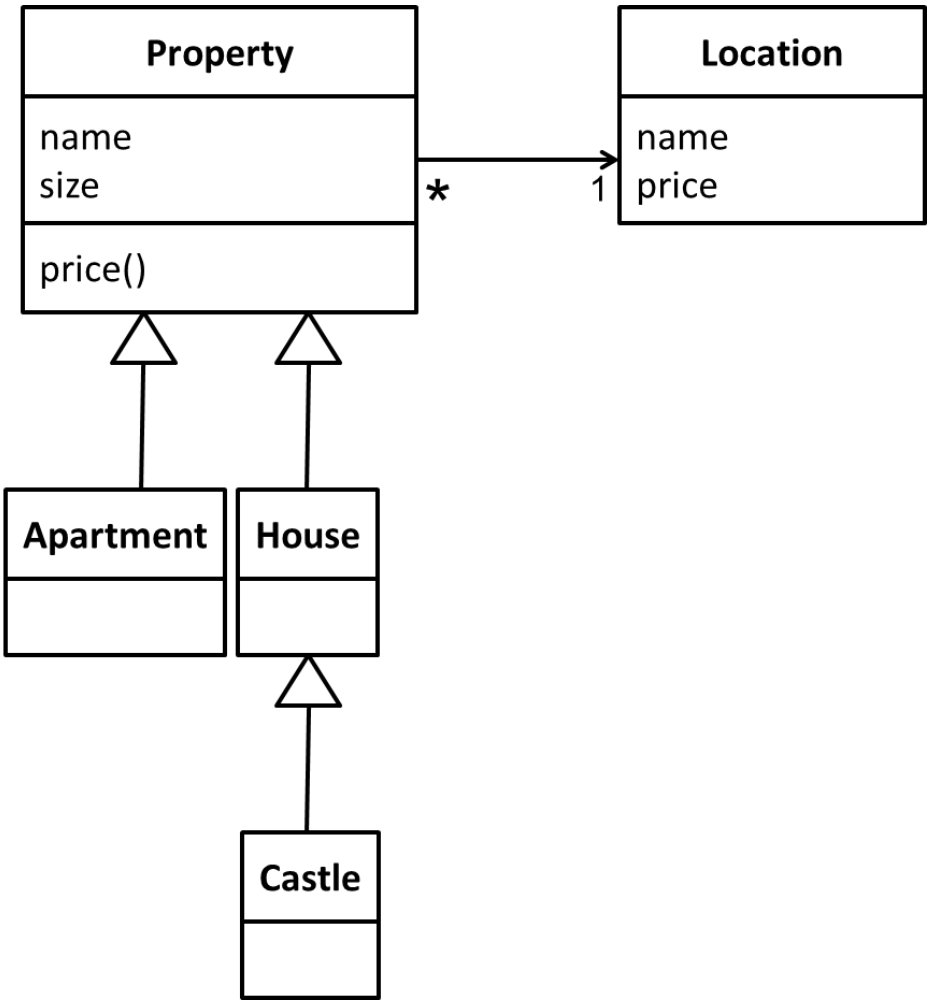
A `House` is a `Property`. However, a house also has a garden, of which the size is specified when the house is created (i.e., the size of the garden is an extra, fourth parameter given to the `__init__()` method). To calculate the price of a house, to the size is added one-fourth of the size of the garden. E.g., if the size of a house is 100 and there is a garden of size 50, with a square-meter price of 1000, the price is 112,500. For the class `House`, you have to implement the methods `__init__()`, `price()`, and `__repr__()`.

A `Castle` is a `House`. As it may be difficult to find a buyer for a castle, the price of a castle is lowered for every day that it is for sale. Therefore, to the `price()` method of a castle an extra parameter is given, namely the number of days-for-sale. Lower the price of the castle by 0.02% for every day that it is for sale, with a maximum of 1000 days for sale (after 1000 days, the price is not lowered further). This entails that you multiply the calculated price by `(1-days_for_sale*0.0002)`. For the class `Castle`, you have to implement the method `price()`.

On the next page you find a UML diagram which describes the class structure to be created.

Example

In the template code given, two apartments, two houses (at two different locations), and a castle are created, and their prices are calculated. For the castle, the calculation is done for different numbers of days-for-sale. The correct answers are given next to the calculations. Most test code is commented out as it would crash without a correct implementation. Uncomment the code for the tests you wish to run.


Property
name
size
price()
Location
name
price
*
1
Apartment
House
Castle