Department of Computer & Information Sciences

CS995 Introduction to Programming Principles

Wednesday 6th December 2023 **10am - 1pm**
Duration: 3 hours

Attempt All Questions (Total 100 Marks)

General instructions

This is an open-book individual programming exam. Students are allowed to access teaching material that is hosted on MyPlace during the exam. External web pages are not available during the exam. Students are not allowed to communicate with other students or anyone else during the exam, following standard exam conditions.

Marking Criteria

- Implementation - each question has an associated number of marks, which correspond to a successful implementation that matches the question. **(90 marks)**
- Commenting - commenting is used appropriately, following examples given in the teaching material. **(5 marks)**
- Style - PEP8 style compliance and naming conventions. **(5 marks)**

Submission

Software source code must be submitted using the MyPlace submission link that is associated with this exam. Source code must be submitted as a single zip file named “solution.zip”. The zip file and the source code must not include personal identifiable data such as the student name or registration number. The source code must be submitted during the time that is associated with the exam. Submissions after the exam has ended will not be accepted.

Q.1 Create a class named `File` in a file named `file_catelogue.py`. Create a constructor for the `File` class that accepts the input parameters:

- `name` - string.
- `size` - integer.
- `executable` - Boolean, with default of `False`.
- `content` - string, with default of an empty string.
- `modified` - string, with default of an empty string.

The constructor should assign input values to data members of the same name, where the `modified` data member should be of type `datetime` rather than a string. The input `modified` string value should be converted into a `datetime` value by assuming that the string uses the ISO `datetime` format. For example, a string that is formatted with the ISO `datetime` format can be formatted using:

```
from datetime import datetime
datetime_value = datetime.fromisoformat(modified)
```

If the `modified` string value is empty, the constructor should use the current `datetime`, by calling the `datetime.now()` function.

(5 marks)

Q.2 Create an `__repr__` function for the `File` class. The `__repr__` function should return a string that can be evaluated to create a duplicate object.

(5 marks)

Q.3 Create an `__eq__` function for the `File` class. The function should return `True` if two objects contain the same data member values and `False` if one or more of the data member values are different.

(5 marks)

Q.4 Create a class named `Directory` in the `file_catelogue.py` file. Create a constructor for the `Directory` class that accepts the input parameters:

- `name` - string.
- `files` - list, with default of an empty list.

The constructor should assign input values to data members of the same name, creating a shallow copy of the input `files`.

(5 marks)

Q.5 Create an `__repr__` function for the `Directory` class. The `__repr__` function should return a string that can be evaluated to create a duplicate object.

(5 marks)

Q.6 Create a member function named `total_size` for the `Directory` class. The function should return the total size of all files in the `Directory` object as an integer.

(10 marks)

Q.7 Create a member function named `to_csv` for the `Directory` class. The function should save each `File` object as a separate row in an output CSV file. The output CSV file should contain the columns `name`, `size`, `executable`, `content` and `modified`. The `modified` date should be saved as an ISO formatted string.

(10 marks)

Q.8 Create a member function named `from_csv` for the `Directory` class. The `from_csv` function should be able to read input data values from the file that is written by calling `to_csv`. The `from_csv` function should use the data values that are read from the CSV file to create new `File` objects, appending them to the list of files within the `Directory` object.

(10 marks)

Q.9 Create a member function named `find_by_name` for the `Directory` class that accepts the input parameter:

- `search_string` - string.

The `search_string` input parameter should use Python regular expression syntax. The value of `search_string` should be compiled and used to match with file names, using:

```
import re
match_alg = re.compile(search_string)
if match_alg.match(file.name):
    # The file.name matches the regular
    # expression search_string.
```

The compiled regular expression (`match_alg`) can be used several times within the function. The `find_by_name` function should catch the `re.error` exception that is thrown by `re.compile` if the input string is not a valid Python regular expression. The `find_by_name` function should return a list of `File` objects that match the input `search_string`.

(10 marks)

Q.10 Create a member function named `find_by_modified` for the `Directory` class that accepts the input parameters:

- `modified` - `datetime`.
- `comparison` - integer, defaulting to 0.

If the value of `comparison` is 0, the `find_by_modified` function should return a list of `File` objects that have a modified `datetime` that is less than or equal to the input `modified` value.

If the value of `comparison` is 1, the `find_by_modified` function should return a list of `File` objects that have a modified `datetime` that is greater than or equal to the input `modified` value.

(10 marks)

Q.11 Create a file named `test_file_catalogue.py` that contains unit tests for the `File` and `Directory` classes. These tests should verify the correct behaviour of:

- The `__repr__` function of the `File` class.
- The `__repr__` function of the `Directory` class.
- The `to_csv` and `from_csv` functions of the `Directory` class.
- The `find_by_name` function of the `Directory` class.
- The `find_by_modified` function of the `Directory` class.

(15 marks)

END OF PAPER

(Dr. W. H. Bell)