MINISTRY OF EDUCATION, SINGAPORE
in collaboration with
CAMBRIDGE ASSESSMENT INTERNATIONAL EDUCATION
General Certificate of Education Advanced Level
Higher 2

COMPUTING

9569/02

Paper 2 (Lab-based)

October/November 2020

3 hours

Additional Materials: Electronic version of `people.txt` data file
Insert Quick Reference Guide

READ THESE INSTRUCTIONS FIRST

Answer **all** questions.

All tasks must be done in the computer laboratory. You are not allowed to bring in or take out any pieces of work or materials on paper or electronic media or in any other form.

Approved calculators are allowed.

Save each task as it is completed.

The use of built-in functions, where appropriate, is allowed for this paper unless stated otherwise.

Note that up to **6** marks out of 100 will be awarded for the use of common coding standards for programming style.

The numbers of marks is given in brackets [ ] at the end of each question or part question.
The total number of marks for this paper is 100.

This document consists of **9** printed pages, **3** blank pages and **1** Insert.

Singapore Examinations and Assessment Board

Cambridge Assessment International Education

Instruction to candidates:

Your program code and output for each of Task 1 to 3 should be saved in a single `.ipynb` file. For example, your program code and output for Task 1 should be saved as

`TASK1_<your name>_<centre number>_<index number>.ipynb`

1 Name your Jupyter Notebook as

`TASK1_<your name>_<centre number>_<index number>.ipynb`

The task is to implement a hashing function using the modulus function and ASCII codes.

The hash is implemented with the following pseudocode, acting on a string `string_value`, which returns an integer, `h`, representing the hash.

```
h ← 0
FOR i ← 0 TO length(string_value) - 1
    val ← 33 * (ASCII value of string_value[i])
    h ← (h + val) % 1024
NEXT i
RETURN h
```

For each of the sub-tasks, add a comment statement at the beginning of the code, using the hash symbol '#' to indicate the sub-task the program code belongs to, for example:

```
In [1] :  # Task 1.1
          Program code

          Output:
```

Task 1.1

Write a function `task1_1(string_value)` that:

- takes a string value `string_value`
- implements the hash algorithm to produce an integer value
- returns that integer value. [5]

Test your function using the following **three** calls:

- `task1_1("Hello")`
- `task1_1("Hallo")`
- `task1_1("Hullo")` [3]

Task 1.2

Strings are often combined with an original value (known as the seed) before their hash is calculated. This makes it harder to use reverse engineering to retrieve the original string.

Write a function `task1_2(seed, string_value)` that:

- takes two string values `seed` and `string_value`
- concatenates these two string values together
- uses the function in Task 1.1 to return the hash generated for the concatenated value. [2]

Test your function with the following **three** calls:

- `task1_2("seed-one", "Hello")`
- `task1_2("seed-two", "Hello")`
- `task1_2("seed-three", "Hello")` [3]

Save your Jupyter Notebook for Task 1.

2 Name your Jupyter Notebook as

`TASK2_<your name>_<centre number>_<index number>.ipynb`

The task is to:

- generate a list of random integers
- write this list to a file
- read the list from the file
- sort the list using a merge sort
- write the sorted list to a second file.

For each of the sub-tasks, add a comment statement, at the beginning of the code using the hash symbol '#' to indicate the sub-task the program code belongs to, for example:

```
In [1] :  # Task 2.1
          Program code

          Output:
```

Task 2.1

Write a function `task2_1(filename, quantity, maximum)` that:

- accepts three parameters:
  - `filename`, a string representing the name of a file
  - `quantity`, an integer representing the number of random integers to generate
  - `maximum`, representing the largest value that a random integer can take
- generates `quantity` random numbers between 0 and `maximum` (inclusive)
- writes those values, one per line, to a file named `filename`. [4]

Generate 1000 random numbers between 0 and 5000 (inclusive) and save them to a file called `randomnumbers_<your name>_<centre number>_<index number>.txt`

[2]

Task 2.2

Write a function `task2_2(list_of_integers)` that:

- takes a list of integers, `list_of_integers`
- sorts them into ascending order using merge sort
- returns the sorted list. [7]

Use the list `[56,25,4,98,0,18,4,5,7,0]` to test your function.

For example, the condition

```
task2_2([56,25,4,98,0,18,4,5,7,0]) == [0,0,4,4,5,7,18,25,56,98]
```

should return `True`. [2]

Task 2.3

Write a function `task2_3(filename_in, filename_out)` that:

- accepts two parameters:
  - `filename_in` represents the input file name
  - `filename_out` represents the output file name
- reads the integers from the input file
- uses your `task2_2` function to sort the integers
- writes the integers to the output file. [5]

The function should read the random numbers from

`randomnumbers_<your name>_<centre number>_<index number>.txt`

and then write the sorted integers to

`sortednumbers_<your name>_<centre number>_<index number>.txt` [3]

Save your Jupyter Notebook for Task 2.

3 Name your Jupyter Notebook as

`TASK3_<your name>_<centre number>_<index number>.ipynb`

The task is to write a function that takes a sequence of characters that represents a quantity of data and unit, and translates this quantity to a different unit.

The basic unit of data is the byte (B):

- A kilobyte (KB) is 10^3 bytes
- A megabyte (MB) is 10^6 bytes
- A gigabyte (GB) is 10^9 bytes
- A terabyte (TB) is 10^{12} bytes.

For example, 8KB has 8000 bytes.

For each of the sub-tasks, add a comment statement at the beginning of the code using the hash symbol '#' to indicate the sub-task the program code belongs to, for example:

```
In [1] : # Task 3.1
         Program code

         Output:
```

Task 3.1

Write a function called `task3_1(quantity_of_data)` that:

- takes a string, `quantity_of_data`
- tests that the given string is a sequence of digits followed by one of the four approved units shown above (KB, MB, GB, TB).
- returns and displays either:
  - the actual number of bytes represented by the input string

  or
  - the error message, `"invalid data"`. [5]

Test the function fully with suitable test data, including all four approved units.

For example,

```
task3_1("8KB")
```

should return and display `8000` [3]

Task 3.2

Companion units are also defined in terms of powers of 2. These have similar abbreviations, as shown:

- A kibibyte (KiB) is 2^{10} bytes
- A mebibyte (MiB) is 2^{20} bytes
- A gibibyte (GiB) is 2^{30} bytes
- A tebibyte (TiB) is 2^{40} bytes.

Write a second function `task3_2(quantity_of_data)` that:

- takes a string, `quantity_of_data`
- tests that the given string is a sequence of digits followed by one of the eight approved units (KB, KiB, MB, MiB, GB, GiB, TB, TiB)
- returns and displays either:
    - the number of bytes represented by the input string

    or
    - the error message, `"invalid data"` [5]

Test the function fully with suitable test data, including all eight approved units.

For example,

```
task3_2("2MiB")
```

should return and display `2097152` [3]

Task 3.3

Write a third function, `task3_3(quantity_of_data, target_unit)` that:

- takes two strings, `quantity_of_data` and `target_unit`
- tests that `target_unit` is one of the eight approved units from task 3.2
- uses your function `task3_2` to generate the actual number of bytes represented by `quantity_of_data`
- converts the generated number of bytes into `target_unit`
- returns and displays either:
    - the `quantity_of_data` in terms of the `target_unit`

    or
    - the error message, `"invalid data"` [4]

Test the function with **three** suitable sets of values.

For example,

```
task3_3("512MiB", "GiB")
```

should return and display `0.5` [3]

Save your Jupyter Notebook for Task 3.

4 A school has used a text file to store data collected about people who work at the school and students who attend the school. People who have a teaching role at the school are referred to as 'staff'. The school decides to transfer this information into a database.

A web page will then be used to summarise the data. Different information will be visible on the web page, depending on the type of person displayed.

Task 4.1

Create an SQL file called `TASK4_1_<your name>_<centre number>_<index number>.sql` to show the SQL code to create database `school.db` with the single table, `People`.

The table will have the following fields of the given SQLite types:

- `PersonID` – primary key, an auto-incremented integer
- `FullName` – the full name of the person, text
- `DateOfBirth` – the person's date of birth, text
- `ScreenName` – the person's screen name, text
- `IsAdult` – a Boolean using 0 for False and 1 for True, integer.

Save your SQL code as

`TASK4_1_<your name>_<centre number>_<index number>.sql` [4]

Task 4.2

The school wants to use the Python programming language and object-oriented programming to help publish the database content on a web page.

The class `Person` will store the following data:

- `full_name` – stored as a string
- `date_of_birth` – initialised with a string with the format YYYY–MM–DD

The class has two methods defined on it:

- `is_adult()` – returns a Boolean value to indicate whether the person is an adult or not. It:
  - subtracts the year of the `date_of_birth` from the year of today's date
  - returns True if the result is greater than 18, otherwise returns False.

- `screen_name()` – returns a string which creates an identifier to be used as a screen name, which should be constructed as follows:
  - the full name with all spaces and punctuation removed
  - followed by the two-digit month of their birth
  - then the two-digit day of their birth.

  For example, John Tan, born on the 1st of June 2000 (`"2000-06-01"`), would have the screen name `"JohnTan0601"`

Save your program code as

`TASK4_2_<your name>_<centre number>_<index number>.py` [7]

The `Staff` class inherits from `Person`, such that:

- `screen_name()` should be the name followed by `"Staff"`
- `is_adult()` always returns True.

The `Student` class inherits from `Person`, such that the `is_adult()` method always returns False.

Add your program code to

`TASK4_2_<your name>_<centre number>_<index number>.py` [4]

The text file, `people.txt`, contains data items for a number of people. Each data item is separated by a comma, with each person's data on a new line as follows:

- full name
- date of birth in the form YYYY–MM–DD
- a string indicating whether the person is `"Staff"`, `"Student"` or `"Person"`.

Write program code to read in the information from the text file, `people.txt`, creating an instance of the appropriate class for each person (either `Staff`, `Student` or `Person`). [4]

Write program code to insert all information from the file into the `school.db` database.

Run the program.

Add your program code to

`TASK4_2_<your name>_<centre number>_<index number>.py` [8]

Task 4.3

The screen names of the people in the text file, `people.txt`, are to be displayed in a web browser.

Write a Python program and the necessary files to create a web application that enables the list of people to be displayed.

For each record the web page should include the:

- full name
- screen name
- identity as student, staff or person.

Save your program as

`Task4_3_<your name>_<centre number>_<index number>.py`

with any additional files / sub-folders as needed in a folder named

`TASK4_3_<your name>_<centre number>_<index number>` [5]

Run the web application and save the output of the program as

`TASK4_3_<your name>_<centre number>_<index number>.html` [3]