NATIONAL JUNIOR COLLEGE
SENIOR HIGH 2 PRELIMINARY EXAMINATION
HIGHER 2

COMPUTING 9569/02

Paper 2 (Lab-based) **16 August 2022**

3 hours

Additional Materials: Electronic version of `TEN.TXT` data file
Electronic version of `HUNDRED.TXT` data file
Electronic version of `THOUSAND.TXT` data file
Electronic version of `Task2_timing.py` data file
Electronic version of `TASK3DATA1.CSV` data file
Electronic version of `TASK3DATA2.CSV` data file
Electronic version of `TIDES.TXT` data file
Electronic version of `Task2_timing.py` 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 number of marks is given in the brackets [ ] at the end of each question or part question. The total number of marks for this paper is 100.

This document consists of **13** printed pages.

[Turn Over

1 In mathematics, a polynomial in x is an expression $p(x)$ consisting of a single variable x and co-efficients, that involves only the operations of addition, subtraction, multiplication, and non-negative integer exponentiation of x.

For example, $x^2 - 4x + 7$, $8x^3 - 5x + 6$ are examples of polynomials.

Task 1.1

The Python function `eval()` can be used to evaluate a mathematical expression in the string format to give a numerical value. For example, `eval('5**2+3*4-1')` will return the value `36`.

Write a function `fun(expression,num)` that takes in a string expression of a polynomial and evaluate the value of the polynomial when the variable x in the polynomial is set to `num`. [2]

Polynomials can be plotted on the xy-axes and the area of the region R bounded by the polynomial $p(x)$, the lines $x = a$, $x = b$, and the x-axes is a quantity of interest to a lot of people. For this question, we assume $a \leq b$.

Task 1.2

One way to obtain the approximation of the area of the region R is to do the following procedure:

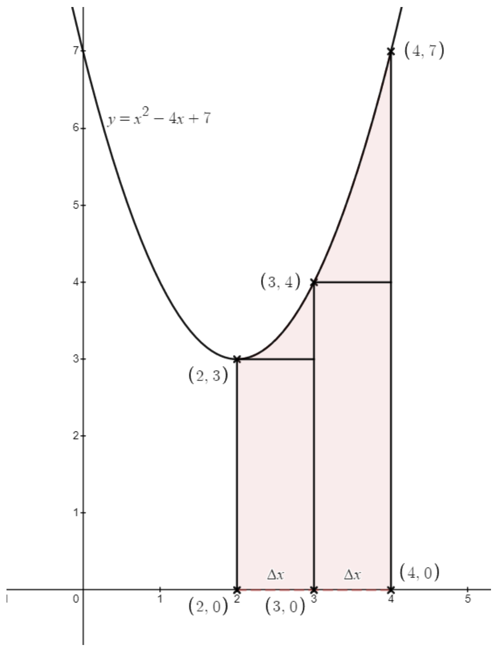
(a) divide the line segment between the vertical lines $x = a$ and $x = b$ into n segment of equal lengths Δx,

(b) for each line segment produced, we generate a rectangle with width Δx and the height is the value of the polynomial at the **left**most x-value of the line segment,

(c) compute the area of the rectangles generated this way and sum them,

(d) The sum is the approximated area of the region R.

The following is an example of the approximation given by the procedure above, when $a = 2$,$b = 4$, $n = 2$ and polynomial $p(x) = x^2 - 4x + 7$.



Write a function `left_sum(expr, a, b, num)` to implement the process above. The function in a polynomial expression `expr`, and numerics `a`, `b` and `num`, where `a` and `b` are the x-values of the region R and `num` is the number of line segments we are using to do the approximation. [3]

Test your function with the following call:

`left_sum('8*x**3-5*x+6', 2, 4, 100)` [1]

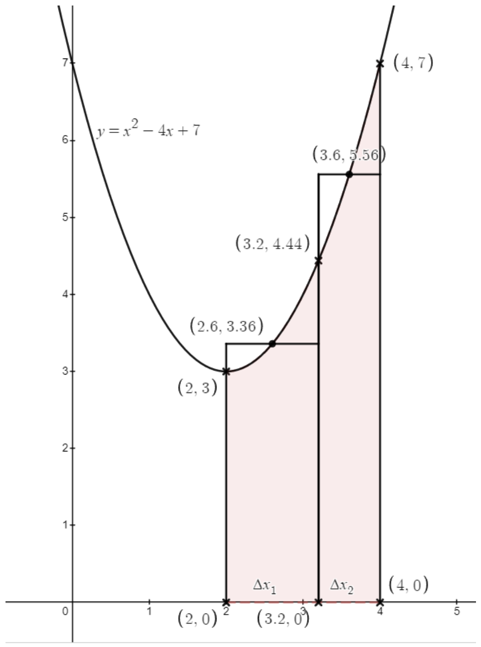
Task 1.3

Yet another way to obtain the approximation of the area of the region R is to use a similar procedure as of Task 1.2, with the following modification to part (a) and (b):

(a") divide the line segment between the vertical lines $x = a$ and $x = b$ into n line segments l_i of random lengths. The `random.uniform(a,b)` method generates a random value between a and b.

(b") for each line segment l_i produced, we generate a rectangle with width Δx_i and the height is the value of the polynomial at middle x-value of the line segment,

The following is an example of the approximation given by the procedure above, when $a = 2$,$b = 4$, $n = 2$, polynomial $p(x) = x^2 - 4x + 7$. The two intervals l_1 and l_2 in the example has lengths $\Delta x_1 = 1.2$ units and $\Delta x_2 = 0.8$ units.



Write a function `random_sum(expr, a, b, num)` to implement the process above. The function in a polynomial expression `expr`, and numerics `a`, `b` and `num`, where `a` and `b` are the x-values of the region R and `num` is the number of line segments we are using to do the approximation. [4]

Test your function with the following call:

`random_sum('8*x**3-5*x+6', 2, 4, 100)` [1]

2 For this question you are provided with three text files, each contains a valid list of positive integers, one per line:

- `TEN.txt` has 10 lines
- `HUNDRED.txt` has 100 lines
- `THOUSAND.txt` has 1000 lines.

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)` that:

- takes a string filename which represents the name of a text file
- reads in the contents of the text file
- returns the content as a list of integers. [3]

Task 2.2

One method of sorting is the bubble sort.

Write a function `task2_2(list_of_integers)` that:

- takes a list of integers
- implements a bubble sort algorithm
- returns the sorted list of integers in ascending order. [5]

Task 2.3

Another method of sorting is the insertion sort.

Write a function `task2_3(list_of_integers)` that:

- takes a list of integers
- implements a insertion sort algorithm
- returns the sorted list of integers in ascending order. [5]

Task 2.4

Yet another method of sorting is the quicksort.

Write a function `task2_4(list_of_integers)` that:

- takes a list of integers
- implements a quicksort algorithm
- returns the sorted list of integers in ascending order. [7]

Task 2.5

Even another method of sorting is the merge sort.

Write a function `task2_5(list_of_integers)` that:

- takes a list of integers
- implements a merge sort algorithm
- returns the sorted list of integers in ascending order. [7]

Task 2.6

Call your functions `task2_2 task2_3, task2_4, task2_5` with the contents of the file `TEN.txt`, printing the returned lists. For example, using the following statement:

`print(task2_2(task2_1('TEN.txt'))).` [1]

Task 2.7

The `timeit` library is built into Python and can be used to time simple function calls. Example code is shown in `Task2_timing.py`. (The sample code assumes that it has access `task2_2` function.)

Using the `timeit` module, or other evidence, and the three text files provided with this question, compare and contrast, including mention of orders of growth, the time complexity of the different sorting algorithms.

Save your Jupyter notebook for Task 2. [5]

3 A hobbyist programmer is trying to create a web browser. A feature he wants the web browser to have is the ability to revisit previously accessed web pages. To achieve this, he designed a class, `Stack`, to represent a stack of the user's browsing history, which includes the web pages with its associated page title and time of visit.

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

To facilitate the ease of retrieval of the pages, the programmer decide to create a class `Page` to store the addresses, its title and the timestamp when it was first accessed. The class will store the following data:

- `page_address` - stored as a string
- `page_title` - also stored as a string
- `time_stamp` - a string with the format `YYYY-MM-DDTHH:MM:SSZ`, where `YYYY-MM-DD` and `HH:MM:SS` are the dates and hours at which the addresses were visited.

The class has the following methods defined on it:

- `get_address()`: returns the address of the visited page
- `get_title()`: returns the title of the visited page
- `get_date()`: returns `YYYY-MM-DD` part of the time stamp
- `get_hour()`: returns `HH:MM:SS` part of the time stamp
- `display()`: returns a string of the form `Page(<page_address>, <page_title>, <time_stamp>)` where `<page_address>`, `<page_title>`, `<time_stamp>` are the corresponding attributes of the `Page` object.

Write the class `Page` with its associated methods in Python. [5]

Task 3.2

Write the `Stack` class in Python. Use of a simple Python list is not sufficient. Include the following methods:

- `push(page_object)` inserts the `page_object` at the top of the stack
- `pop()` attempts to pop the `page_object` at the top of the stack; if the item was not present, return `None`
- `search(page_title)` returns a Boolean value: `True` if a page with title `page_title` is in the stack, `False` if not in the stack
- `count()` should return the number of elements in the stack, or zero if empty
- `to_String()` should return a string containing a suitably formatted list with the elements being page objects in the stack, separated by a comma and a space, with square brackets at either end, eg. in the form:

```
[
Page(https://ovh.net/diam/erat/fermentum/justo.html, Monitored radical
synergy, 2021-09-17T09:33:05Z),
Page(http://bloglovin.com/felis/donec/semper.xml, De-engineered
asymmetric structure, 2021-09-17T09:33:35Z)
]
```

[9]

Test `Stack` by using the data in the file `Task3data1.csv`, where line 1 is the first site visited. Use the `to_String()` method to print the contents of the stack. [3]

Task 3.3

To manage the number of browsing history, the programmer decided to limit the number of elements in the stack to 12.

Write a Python subclass `ModStack` using `Stack` as its superclass.

The `push` method in the `ModStack` subclass should ensure if another page is accessed when the number of elements in the new data structure is 12, the earliest visited address will be removed from the history and the latest address is pushed to the top of the new data structure. [4]

Test `ModStack` by using the data in the file `Task3data2.csv`, where line 1 is the first site visited. Print the result of searching the `ModStack` for the page with a title `'Object-based global firmware'`. [2]

Task 3.4

After some experimentation, the programmer realised that using a stack like data structure for a browsing history is rather silly as the user can only move to latest previously visited page before returning back to the desired page. And, once he reached the desired page, the data structure no longer holds the browsing history after that page.

As such, he decided to use a linked list instead.

For this task, ignore the browsing history limit mentioned in Task 3.3.

Write a Python subclass `PageNode` using `Page` as its superclass to be used in the linked list. Include the following attributes for the PageNode object:

- `next` : initializes to `None`. Stores a pointer to the next `PageNode` object in the linked list after the current `PageNode` object.

The subclass also has the following methods defined on it:

- `get_next()`: returns the next `PageNode` object in the linked list. If there's no such object, return `None`.
- `set_next(page)`: set the pointer from the node to the next `PageNode` object in the linked list.

Write the `LinkedList` class in Python. Use of a simple Python list is not sufficient. Include the following methods:

- `insert(page_object)` inserts the `page_object` at the end of the list
- `delete(page_title)` attempts to delete a `PageNode` object containing data `page_title` from the list; if the item was not present, return `None`.
- `to_String()` should return a string containing a suitably formatted list with the elements separated by a comma and a space, with square brackets at either end, eg. in the form:

  ```
  [
  Page(https://ovh.net/diam/erat/fermentum/justo.html, Monitored radical
  synergy, 2021-09-17T09:33:05Z),
  Page(http://bloglovin.com/felis/donec/semper.xml, De-engineered
  asymmetric structure, 2021-09-17T09:33:35Z)
  ]
  ```
- `visit(page_title)` is a procedure that emulates the user visiting an address with a given page title `page_title` in the history. The page title will be searched from the head of the linked list. If the page title is found, the node will first be removed from the linked list and then placed back at the tail of the list. If the page is not found, print `'Page Not Found in History'`.

[9]

Create two `LinkedList` objects by using the data in the file `Task3data2.csv`, where line 1 is the first site visited.

Test the objects with the following methods:

- `visit('Streamlined didactic matrix')`,
- `visit('Advanced locale concept')`.

Use the `to_String()` method to print the resulting contents of the list. [2]

Save your Jupyter notebook for Task 3.

4 A text file, `TIDES.TXT`. contains the low and high tide information for a coastal location tor each day of a month. Each line contains tab-delimited data that shows the date, the time. whether the tide is high or low and the tide height in metres.

Each line is in the format:

```
YYYY-MM-DD\tHH:mm\tTIDE\tHEIGHT\n
```

- The date is in the term YYYY-MM-DD, for example. 2019-08-03 is 3rd August. 2019
- The time is in the form HH:mm, for example. 13:47
- TIDE is either HIGH or LOW
- HEIGHT is a positive number shown to one decimal place
- `\t` represents the tab character
- `\n` represents the newline character

The text file is stored in ascending order of date and time.

Task 4.1

Write program code to:

- read the tide data from the text file `TIDES.TXT`
- insert all information from the file into a SQLite database called `tide.db` with the single table, `Tide`. The table will have the following fields of the given SQLite types:
  - `RecordID` - primary key, an auto-incremented integer
  - `Date` - the date in the format YYYY-MM-DD, text
  - `Time` - the time in the format HH:mm, text
  - `isHigh` - a Boolean using 0 if the field entry for the row is `LOW` and 1 if the entry is `HIGH`, integer
  - `Height` - positive number shown to one decimal place, real [4]

Task 4.2

A researcher decided that he wanted to use the information in `tide.db` to calculate some values from a remote computer. As such, he decided to use socket programming to achieve his goal.

Write a server program that:

- instantiates a server socket,
- binds the socket to localhost and port number 8888,
- listens for incoming request, accepts incoming request and establishes connection with the client,
- presents the client with a menu with the following options:

  1. Highest high tide with one corresponding date and time it happened
  2. Lowest low tide with one corresponding date and time it happened
  3. Largest tidal range
  4. Smallest tidal range

- calculates the values (round to 1 decimal places) requested by the client from the menu by accessing `tide.db`. The *tidal range* in the menu is defined to be the absolute difference between the heights of successive tides; from a high tide to the following low tide or from a low tide to the following high tide,
- send the requested value to the client.

Save your server program in the file `TASK4_SERVER_<your name>.py`. [9]

Task 4.3

Write a corresponding client program that can:

- connect to the server,
- asks the user for an input to request for the values that the server can calculate,
- close the sockets when the service from server is no longer required.

Save your client program in the file `TASK4_CLIENT_<your name>.py`. [3]

END