

**Computer science
Higher level
Paper 2****Mock examination — Set 1**2 hours

Instructions to candidates

- Do not open this examination paper until instructed to do so.
- Answer all of the questions from **one** of the options.
- The maximum mark for this examination paper is **[80 marks]**.

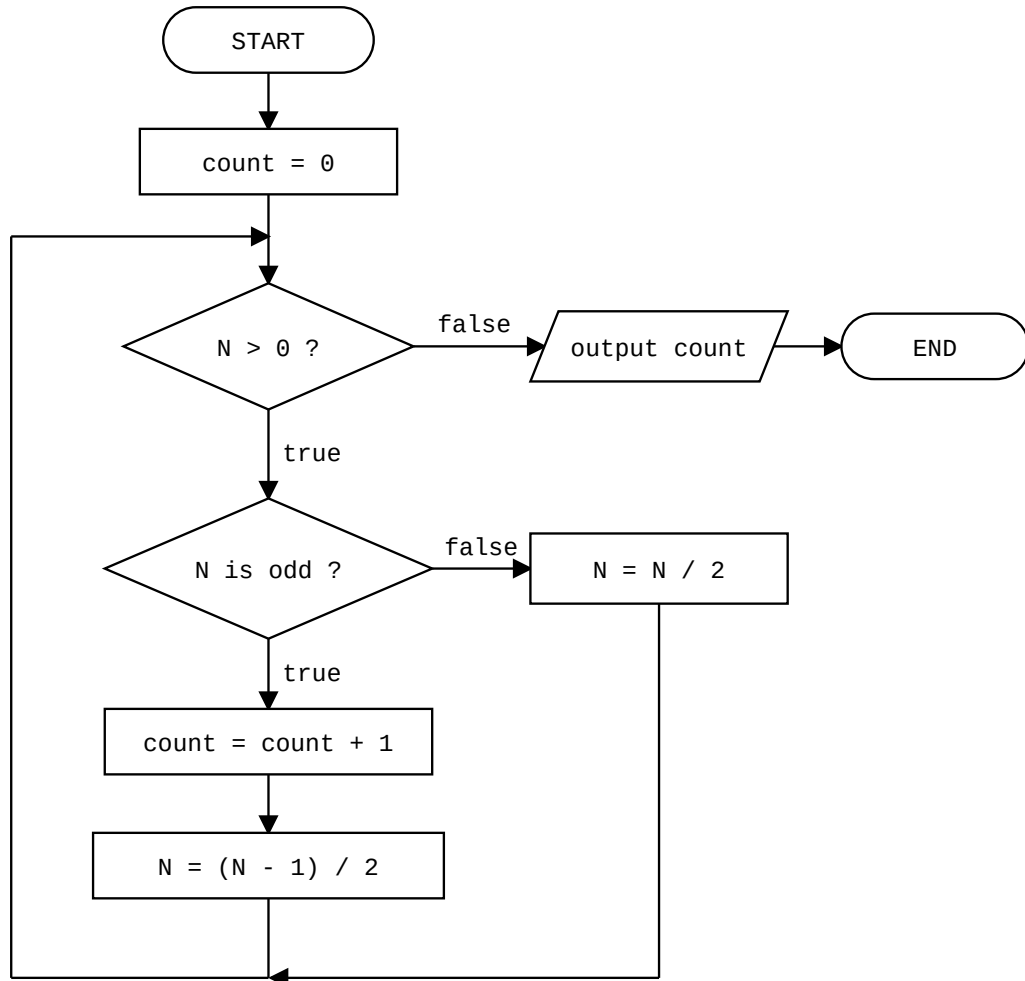
Option	Questions
Option A — Java	1 – 6
Option B — Python	7 – 12

Option A — Java

Answer **all** questions. All code that forms part of your responses must be written in Java.

- The flowchart in **Figure 1** represents an algorithm that processes a value, N , which is supplied when the algorithm starts.

Figure 1: Flowchart for an algorithm that processes N



- State **one** requirement for N that should be included in the input specification for this algorithm. [1]

- Copy and complete the trace table for this flowchart, where $N = 13$. [3]

N	$N > 0 ?$	N is odd ?	count	output
13			0	

The efficiency of an algorithm can be expressed using Big O notation.

- Describe the time complexity of the algorithm in **Figure 1**, using Big O notation. [2]

(Option A continues on the following page)

(Option A, question 1 continued)

Table 1 shows a list of values that has been sorted into ascending order.

Table 1: A sorted list of values

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Value	5	11	16	23	29	34	42	47	53	58	64	70	77	81	96

- (d) Describe the steps that a binary search would follow to find the value 29 in this list. [4]

Queues are used in computer science to process items in the order in which they arrive.

Table 2 shows the fundamental operations that operate on a queue.

Table 2: Fundamental operations that operate on a queue

Operation	Description
enqueue(x)	Adds the element x to the back of the queue.
dequeue()	Removes the element at the front of the queue and returns it.
front()	Returns the element at the front of the queue without removing it from the queue.
isEmpty()	Returns true if the queue is empty, else it returns false.

The following operations are carried out, in order, on an empty queue:

enqueue(7)
enqueue(3)
enqueue(12)
dequeue()
front()
enqueue(5)
dequeue()
enqueue(9)

- (e) Sketch the queue after these operations have been carried out.

You should clearly indicate the front and the back of the queue. [2]

A different queue contains whole numbers. Some of these numbers are negative.

- (f) Using only the queue operations shown in **Table 2**, describe the steps that will remove all the negative numbers from the queue, so that the remaining numbers stay in their original order. [3]

(Option A continues on the following page)

(Option A continued)

2. A ferry company sells tickets for crossings between island ports. Each ticket shows a sailing code made of the departure port, the arrival port and the departure time, separated by hyphens. For example, the sailing code ARD-BRE-0915 is for the 09:15 crossing from the port ARD to the port BRE. Each port code has exactly three letters and each departure time has exactly four digits.

Each ferry can carry a maximum of 150 passengers. For each sailing, the booking system stores the names of the passengers.

- (a) Outline **one** advantage of using a static data structure, rather than a dynamic data structure, to store the names of the passengers for a sailing. [2]

The numbers of passengers on six sailings are stored in the array passengers:

```
int[] passengers = {48, 112, 35, 90, 61, 27};
```

The array is sorted into ascending order using a bubble sort. In each pass, adjacent elements are compared, starting at the beginning of the array.

- (b) State the contents of passengers after each of the **first two** passes of the bubble sort. [2]

A ticket clerk types the number of tickets that a customer wants to buy. The method readTickets() converts the text typed by the clerk into an integer:

```
public static int readTickets(String text)
{
    int tickets = 0;
    try
    {
        tickets = Integer.parseInt(text);
    }
    catch (NumberFormatException e)
    {
        System.out.println ("Enter a whole number");
    }
    finally
    {
        System.out.println ("Input checked");
    }
    return tickets;
}
```

- (c) Describe what happens when readTickets("3x") is called. [2]

(Option A continues on the following page)

(Option A, question 2 continued)

The fares paid for five tickets on one sailing are stored in the array `fares`. The following code is intended to output the lowest of these fares:

```
double[] fares = {14.50, 9.75, 22.00, 9.75, 18.25};
double lowest = 0;
for (int i = 0; i < 5; i++)
{
    if (fares[i] < lowest)
        lowest = fares[i];
}
System.out.println ("Lowest fare: " + lowest);
```

- (d) Identify the logic error in this code and outline how it should be corrected. [2]

The method `departureTime()` is passed a sailing code and returns its departure time in the format hours:minutes. For example, `departureTime("ARD-BRE-0915")` returns "09:15".

- (e) Construct the method `departureTime()`. [3]

At the end of each day, the sailing code of every ticket sold that day is written to the text file `tickets.txt`, one sailing code per line. The first three lines of the file are shown:

```
ARD-BRE-0915
CAL-ARD-1030
ARD-CAL-1400
...
```

The method `countFrom()` is passed a three-letter port code. It reads `tickets.txt` line by line until the end of the file is reached and then outputs the number of tickets sold for sailings that depart from that port. For example, `countFrom("ARD")` outputs the number of tickets sold for sailings that depart from ARD. The file is currently closed.

- (f) Construct the method `countFrom()`. [4]

(Option A continues on the following page)

(Option A continued)

3. A museum uses object-oriented programming (OOP) to manage the exhibits in its collection. Every exhibit is either a painting or a fossil.

The abstract class Exhibit is shown:

```
public abstract class Exhibit{

    private String code;        // unique exhibit code, e.g. "PA-0193"
    private String title;
    protected double value;    // estimated value in euros

    public Exhibit(String code, String title, double value)
    {
        this.code = code;
        this.title = title;
        this.value = value;
    }

    public String getCode()
    {
        return code;
    }

    public String getTitle()
    {
        return title;
    }

    public abstract double insuranceCost();
}
```

(Option A continues on the following page)

(Option A, question 3 continued)

The classes Fossil and Painting inherit from Exhibit. The Fossil class is shown:

```
public class Fossil extends Exhibit{

    private String period;    // geological period, e.g. "Jurassic"
    private boolean fragile;  // true if the fossil is fragile

    public Fossil(String code, String title, double value,
                  String period, boolean fragile)
    {
        super(code, title, value);
        this.period = period;
        this.fragile = fragile;
    }

    public double insuranceCost()
    {
        if (fragile)
            return value * 0.03;
        else
            return value * 0.01;
    }
}
```

The Painting class is partially shown:

```
public class Painting extends Exhibit{

    private String artist;
    private boolean onTour;    // true while the painting is lent to
                               // another museum

    public Painting(String code, String title, double value,
                   String artist)
    {
        // code missing
    }

    public void setOnTour(boolean status)
    {
        this.onTour = status;
    }

    public double insuranceCost()
    {
        // code missing
    }
}
```

- (a) Outline **one** advantage of using inheritance in the design of the Painting and Fossil classes.

[2]

(Option A continues on the following page)

Turn over

(Option A, question 3 continued)

- (b) Construct a UML class diagram for the classes `Exhibit`, `Painting` and `Fossil`, showing the relationship between them.

Constructors do not need to be included. [3]

- (c) Outline why `Exhibit` and its method `insuranceCost()` have been declared as abstract. [3]

When a new `Painting` object is created, it is not on tour.

The annual insurance cost of a painting is calculated as follows:

- The basic cost is 1.5 % of the painting's value.
- If the painting is on tour, the basic cost is doubled.
- If the result is less than 250 euros, the cost is 250 euros.

- (d) Construct the constructor and the method `insuranceCost()` for the `Painting` class. [5]

The details of the exhibits are stored in a text file. Each record starts with the letter P (painting) or F (fossil), followed by the data for that exhibit. The programmers decide to use the factory design pattern to create the `Exhibit` objects from these records.

- (e) Describe how the factory design pattern could be applied in this situation. [3]

The driver class stores all of the museum's exhibits in the array `collection`, declared as `Exhibit[] collection`. Every element of `collection` refers to either a `Painting` object or a `Fossil` object.

The method `insuranceReport(Exhibit[] collection, double limit)` outputs the code and title of every exhibit whose annual insurance cost is greater than `limit`. It then outputs the total annual insurance cost of all the exhibits in `collection`.

- (f) Construct the method `insuranceReport()`.

Use any of the methods shown in the `Exhibit` class. [4]

(Option A continues on the following page)

(Option A continued)

4. A language-learning app keeps a record of the Spanish words that each learner has mastered. The words mastered by each learner are stored in a HashSet.

A HashSet is implemented using a hash table. When the load factor of the hash table becomes greater than a fixed limit, the hash table is resized so that it has more slots.

- (a) Outline why the hash table is resized when its load factor becomes too high. [2]

Two HashSets, wordsA and wordsB, hold the words mastered by two learners:

```
wordsA: [gato, libro, agua, verde, mesa]
wordsB: [agua, rojo, gato, playa]
```

The following code is run:

```
wordsB.removeAll(wordsA);
System.out.println(wordsB);
wordsA.addAll(wordsB);
System.out.println(wordsA.size());
```

- (b) State the output of this code. [2]

At the end of each lesson, the words that a learner typed correctly are stored in the array typed. The same word may appear more than once in typed.

The method `countNew(String[] typed, HashSet<String> mastered)` adds every word in typed to the set mastered. It returns the number of different words in typed that were not already in mastered.

For example, if typed contains rojo, sol, gato, sol and mastered contains agua, rojo, gato, playa, the method adds sol to mastered and returns 1.

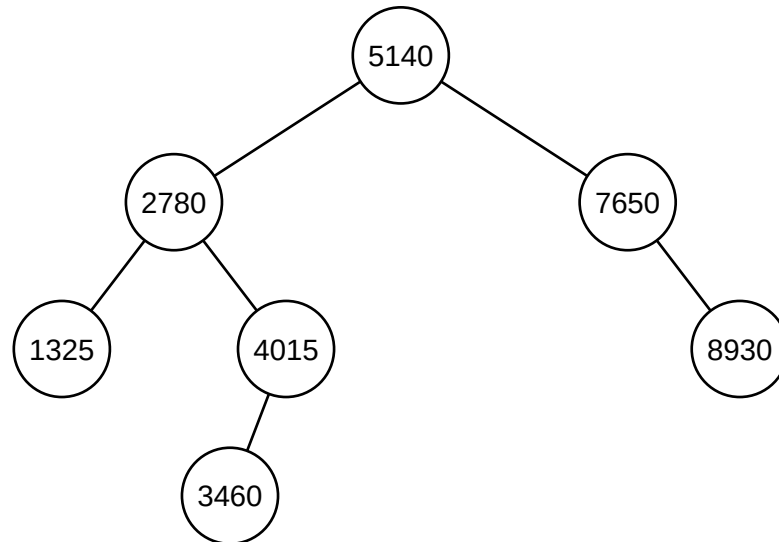
- (c) Construct the method `countNew()`. [3]

(Option A continues on the following page)

(Option A continued)

5. A hardware store stores the product codes of the items it sells in a binary search tree (BST). Each product code is a four-digit integer. The BST after the first seven product codes have been inserted is shown in **Figure 2**.

Figure 2: The binary search tree after the first seven product codes have been inserted



- (a) Sketch the binary search tree after the product codes 6285 and 4790 have been inserted into it. [2]

In a post-order traversal, the left subtree is traversed, then the right subtree, then the node is visited.

- (b) State the order in which the product codes in **Figure 2** are visited in a post-order traversal. [2]
- (c) Explain why the order in which product codes are inserted affects the time taken to find a product code in the binary search tree. [2]

Each product code has a check value, which is calculated by the recursive method `check(int code)`:

```
public int check(int code)
{
    if (code == 0)
        return 0;
    else
        return code % 10 + check(code / 10);
}
```

- (d) (i) State the value returned by `check(4015)`. [1]
- (ii) Outline why `check()` uses more memory than an iterative method that performs the same task. [2]

(Option A continues on the following page)

(Option A, question 5 continued)

The store's labelling system needs to know how many times a given character appears in a product description. For example, the call `countChar("STEEL SHELF BRACKET", 'E')` returns 4.

- (e) Construct the recursive method `countChar(String desc, char c)`. [4]

(Option A continues on the following page)

(Option A continued)

6. A digital photo frame shows a slideshow of photos. The photos are stored in the nodes of a singly linked list, in the order in which they are shown.

The linked list makes use of the Node class, in which the data for each node consists of the file name of the photo and the number of seconds for which it is displayed. This class is shown:

```
class Node{

    public String fileName;    // e.g. "harbour.jpg"
    public int seconds;        // time for which the photo is shown
    public Node next;

    public Node(String f, int s)
    {
        this.fileName = f;
        this.seconds = s;
        this.next = null;      // initialize next pointer as null
    }
}
```

The Node object head points to the first node in the linked list.

The driver class contains the method `removePhoto(String name)`. This method removes the first node whose file name is equal to name. If no node matches, it outputs a suitable message and the linked list is not changed.

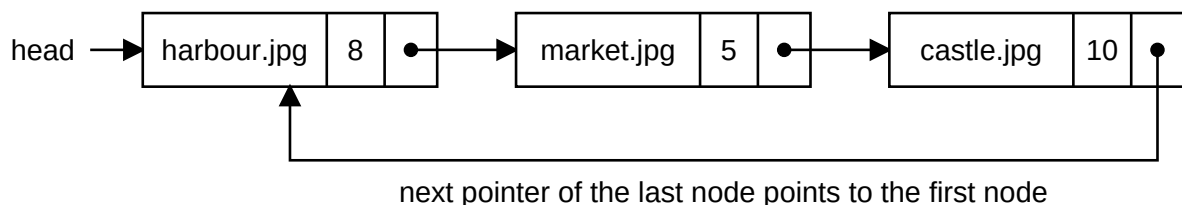
- (a) Construct the method `removePhoto()`.

Assume the list is not empty.

[5]

The photo frame is updated so that the slideshow repeats continuously. The linked list is changed to a circular linked list, in which the next pointer of the last node points to the first node, as shown in **Figure 3**.

Figure 3: The photos stored in a circular linked list



The method `addPhoto(Node newPhoto)` adds the node `newPhoto` to the end of the circular linked list, so that the new photo is shown immediately before the slideshow returns to the first photo. If the list is empty, `newPhoto` becomes the only node.

- (b) Without writing code, describe the steps in the method `addPhoto()`.

[5]

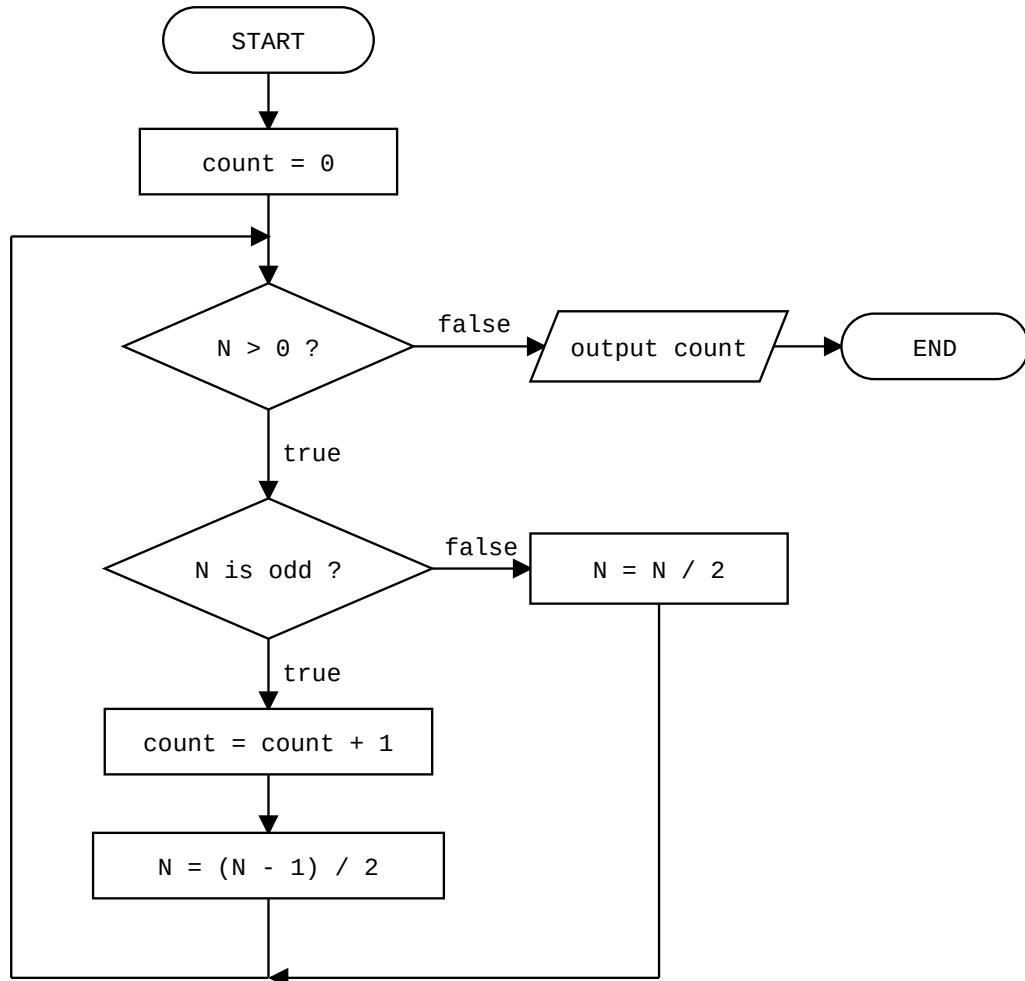
End of Option A

Option B — Python

Answer **all** questions. All code that forms part of your responses must be written in Python.

7. The flowchart in **Figure 4** represents an algorithm that processes a value, N , which is supplied when the algorithm starts.

Figure 4: Flowchart for an algorithm that processes N



- (a) State **one** requirement for N that should be included in the input specification for this algorithm. [1]
- (b) Copy and complete the trace table for this flowchart, where $N = 13$. [3]

N	$N > 0 ?$	N is odd ?	count	output
13			0	

The efficiency of an algorithm can be expressed using Big O notation.

- (c) Describe the time complexity of the algorithm in **Figure 4**, using Big O notation. [2]

(Option B continues on the following page)

(Option B, question 7 continued)

Table 3 shows a list of values that has been sorted into ascending order.

Table 3: A sorted list of values

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Value	5	11	16	23	29	34	42	47	53	58	64	70	77	81	96

- (d) Describe the steps that a binary search would follow to find the value 29 in this list. [4]

Queues are used in computer science to process items in the order in which they arrive.

Table 4 shows the fundamental operations that operate on a queue.

Table 4: Fundamental operations that operate on a queue

Operation	Description
enqueue(x)	Adds the element x to the back of the queue.
dequeue()	Removes the element at the front of the queue and returns it.
front()	Returns the element at the front of the queue without removing it from the queue.
isEmpty()	Returns true if the queue is empty, else it returns false.

The following operations are carried out, in order, on an empty queue:

enqueue(7)
enqueue(3)
enqueue(12)
dequeue()
front()
enqueue(5)
dequeue()
enqueue(9)

- (e) Sketch the queue after these operations have been carried out.

You should clearly indicate the front and the back of the queue. [2]

A different queue contains whole numbers. Some of these numbers are negative.

- (f) Using only the queue operations shown in **Table 4**, describe the steps that will remove all the negative numbers from the queue, so that the remaining numbers stay in their original order. [3]

(Option B continues on the following page)

(Option B continued)

8. A ferry company sells tickets for crossings between island ports. Each ticket shows a sailing code made of the departure port, the arrival port and the departure time, separated by hyphens. For example, the sailing code `ARD-BRE-0915` is for the 09:15 crossing from the port `ARD` to the port `BRE`. Each port code has exactly three letters and each departure time has exactly four digits.

Each ferry can carry a maximum of 150 passengers. For each sailing, the booking system stores the names of the passengers.

- (a) Outline **one** advantage of using a static data structure, rather than a dynamic data structure, to store the names of the passengers for a sailing. [2]

The numbers of passengers on six sailings are stored in the list `passengers`:

```
passengers = [48, 112, 35, 90, 61, 27]
```

The list is sorted into ascending order using a bubble sort. In each pass, adjacent elements are compared, starting at the beginning of the list.

- (b) State the contents of `passengers` after each of the **first two** passes of the bubble sort. [2]

A ticket clerk types the number of tickets that a customer wants to buy. The function `read_tickets()` converts the text typed by the clerk into an integer:

```
def read_tickets(text):
    tickets = 0
    try:
        tickets = int(text)
    except ValueError:
        print ("Enter a whole number")
    finally:
        print ("Input checked")
    return tickets
```

- (c) Describe what happens when `read_tickets("3x")` is called. [2]

The fares paid for five tickets on one sailing are stored in the list `fares`. The following code is intended to output the lowest of these fares:

```
fares = [14.50, 9.75, 22.00, 9.75, 18.25]
lowest = 0
for i in range (0, 5):
    if fares[i] < lowest:
        lowest = fares[i]
print ("Lowest fare: ", lowest)
```

- (d) Identify the logic error in this code and outline how it should be corrected. [2]

The function `departure_time()` is passed a sailing code and returns its departure time in the format `hours:minutes`. For example, `departure_time("ARD-BRE-0915")` returns `"09:15"`.

- (e) Construct the function `departure_time()`. [3]

(Option B continues on the following page)

Turn over

(Option B, question 8 continued)

At the end of each day, the sailing code of every ticket sold that day is written to the text file `tickets.txt`, one sailing code per line. The first three lines of the file are shown:

```
ARD-BRE-0915
CAL-ARD-1030
ARD-CAL-1400
...
```

The function `count_from()` is passed a three-letter port code. It reads `tickets.txt` line by line until the end of the file is reached and then outputs the number of tickets sold for sailings that depart from that port. For example, `count_from("ARD")` outputs the number of tickets sold for sailings that depart from ARD. The file is currently closed.

(f) Construct the function `count_from()`.

[4]

(Option B continues on the following page)

(Option B continued)

9. A museum uses object-oriented programming (OOP) to manage the exhibits in its collection. Every exhibit is either a painting or a fossil.

The abstract class Exhibit is shown. The ABC class and the @abstractmethod decorator from the abc module are used to declare Exhibit and its method `insurance_cost()` as abstract.

```
from abc import ABC, abstractmethod

class Exhibit(ABC):

    def __init__(self, code, title, value):
        self._code = code          # unique exhibit code, e.g. "PA-0193"
        self._title = title
        self._value = value        # estimated value in euros

    def get_code(self):
        return self._code

    def get_title(self):
        return self._title

    @abstractmethod
    def insurance_cost(self):
        pass
```

The classes Fossil and Painting inherit from Exhibit. The Fossil class is shown:

```
class Fossil(Exhibit):

    def __init__(self, code, title, value, period, fragile):
        super().__init__(code, title, value)
        self._period = period      # geological period, e.g. "Jurassic"
        self._fragile = fragile    # True if the fossil is fragile

    def insurance_cost(self):
        if self._fragile == True:
            return self._value * 0.03
        else:
            return self._value * 0.01
```

(Option B continues on the following page)

(Option B, question 9 continued)

The Painting class is partially shown:

```
class Painting(Exhibit):  
  
    # attributes:  _artist      name of the artist  
    #              _on_tour     True while the painting is lent to  
    #                      another museum  
  
    def __init__(self, code, title, value, artist):  
        # code missing  
  
    def set_on_tour(self, status):  
        self._on_tour = status  
  
    def insurance_cost(self):  
        # code missing
```

- (a) Outline **one** advantage of using inheritance in the design of the Painting and Fossil classes. [2]

The underscore character in `self._code` identifies the attribute as private.

- (b) Construct a UML class diagram for the classes Exhibit, Painting and Fossil, showing the relationship between them.

Constructors do not need to be included. [3]

- (c) Outline why Exhibit and its method `insurance_cost()` have been declared as abstract. [3]

When a new Painting object is created, it is not on tour.

The annual insurance cost of a painting is calculated as follows:

- The basic cost is 1.5 % of the painting's value.
- If the painting is on tour, the basic cost is doubled.
- If the result is less than 250 euros, the cost is 250 euros.

- (d) Construct the constructor, `__init__()`, and the method `insurance_cost()` for the Painting class. [5]

The details of the exhibits are stored in a text file. Each record starts with the letter P (painting) or F (fossil), followed by the data for that exhibit. The programmers decide to use the factory design pattern to create the Exhibit objects from these records.

- (e) Describe how the factory design pattern could be applied in this situation. [3]

(Option B continues on the following page)

(Option B, question 9 continued)

The driver code stores all of the museum's exhibits in the list `collection`. Every element of `collection` refers to either a `Painting` object or a `Fossil` object.

The function `insurance_report(collection, limit)` outputs the code and title of every exhibit whose annual insurance cost is greater than `limit`. It then outputs the total annual insurance cost of all the exhibits in `collection`.

(f) Construct the function `insurance_report(collection, limit)`.

Use any of the methods shown in the `Exhibit` class.

[4]

(Option B continues on the following page)

(Option B continued)

10. A language-learning app keeps a record of the Spanish words that each learner has mastered. The words mastered by each learner are stored in a set.

A set in Python is implemented using a hash table. When the load factor of the hash table becomes greater than a fixed limit, the hash table is resized so that it has more slots.

- (a) Outline why the hash table is resized when its load factor becomes too high. [2]

Two sets, `words_a` and `words_b`, hold the words mastered by two learners:

```
words_a = {"gato", "libro", "agua", "verde", "mesa"}
words_b = {"agua", "rojo", "gato", "playa"}
```

The following code is run:

```
print(words_b - words_a)
print(len(words_a | words_b))
```

- (b) State the output of this code. [2]

At the end of each lesson, the words that a learner typed correctly are stored in the list `typed`. The same word may appear more than once in `typed`.

The function `count_new(typed, mastered)` adds every word in `typed` to the set `mastered`. It returns the number of different words in `typed` that were not already in `mastered`.

For example, if `typed` contains `rojo`, `sol`, `gato`, `sol` and `mastered` contains `agua`, `rojo`, `gato`, `playa`, the function adds `sol` to `mastered` and returns 1.

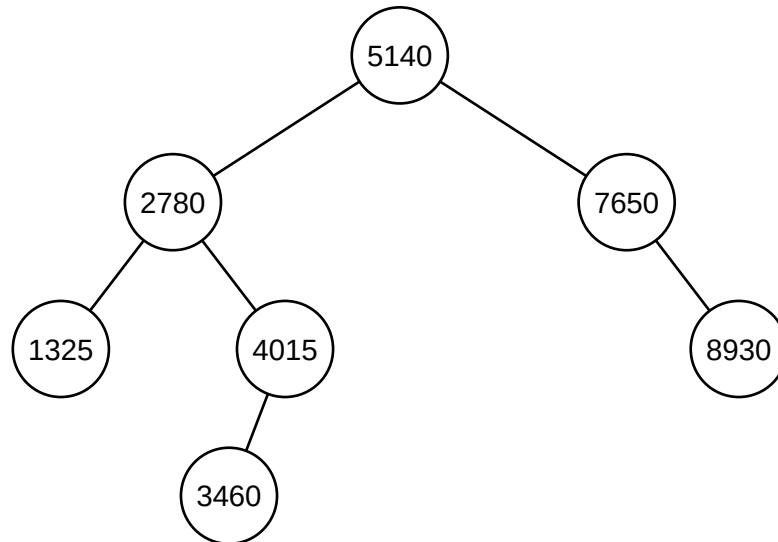
- (c) Construct the function `count_new(typed, mastered)`. [3]

(Option B continues on the following page)

(Option B continued)

11. A hardware store stores the product codes of the items it sells in a binary search tree (BST). Each product code is a four-digit integer. The BST after the first seven product codes have been inserted is shown in **Figure 5**.

Figure 5: The binary search tree after the first seven product codes have been inserted



- (a) Sketch the binary search tree after the product codes 6285 and 4790 have been inserted into it. [2]

In a post-order traversal, the left subtree is traversed, then the right subtree, then the node is visited.

- (b) State the order in which the product codes in **Figure 5** are visited in a post-order traversal. [2]
- (c) Explain why the order in which product codes are inserted affects the time taken to find a product code in the binary search tree. [2]

Each product code has a check value, which is calculated by the recursive function `check(code)`:

```
def check(code):  
    if code == 0:  
        return 0  
    else:  
        return code % 10 + check(code // 10)
```

- (d) (i) State the value returned by `check(4015)`. [1]
- (ii) Outline why `check()` uses more memory than an iterative function that performs the same task. [2]

(Option B continues on the following page)

(Option B, question 11 continued)

The store's labelling system needs to know how many times a given character appears in a product description. For example, the call `count_char("STEEL SHELF BRACKET", "E")` returns 4.

- (e) Construct the recursive function `count_char(desc, ch)`.

The `count()` method must not be used.

[4]

(Option B continues on the following page)

(Option B continued)

12. A digital photo frame shows a slideshow of photos. The photos are stored in the nodes of a singly linked list, in the order in which they are shown.

The linked list makes use of the Node class, in which the data for each node consists of the file name of the photo and the number of seconds for which it is displayed. This class is shown:

```
class Node():  
  
    def __init__(self, file_name, seconds):  
        self.file_name = file_name    # e.g. "harbour.jpg"  
        self.seconds = seconds        # time for which the photo is shown  
        self.next = None              # initialize next pointer as None
```

The Node object `self.head` points to the first node in the linked list.

The driver code contains the method `def remove_photo(self, name)`. This method removes the first node whose file name is equal to `name`. If no node matches, it outputs a suitable message and the linked list is not changed.

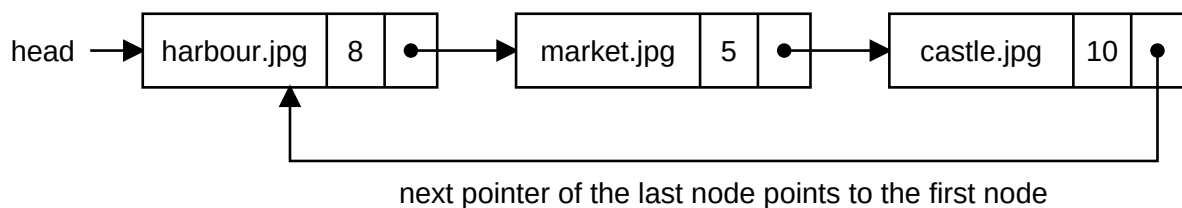
- (a) Construct the method `def remove_photo(self, name)`.

Assume the list is not empty.

[5]

The photo frame is updated so that the slideshow repeats continuously. The linked list is changed to a circular linked list, in which the next pointer of the last node points to the first node, as shown in **Figure 6**.

Figure 6: The photos stored in a circular linked list



The method `def add_photo(self, new_photo)` adds the node `new_photo` to the end of the circular linked list, so that the new photo is shown immediately before the slideshow returns to the first photo. If the list is empty, `new_photo` becomes the only node.

- (b) Without writing code, describe the steps in the method `def add_photo(self, new_photo)`.

[5]

End of Option B
