

Computer science**Standard level****Paper 2****Mock examination — Set 2**1 hour 15 minutes

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 **[50 marks]**.

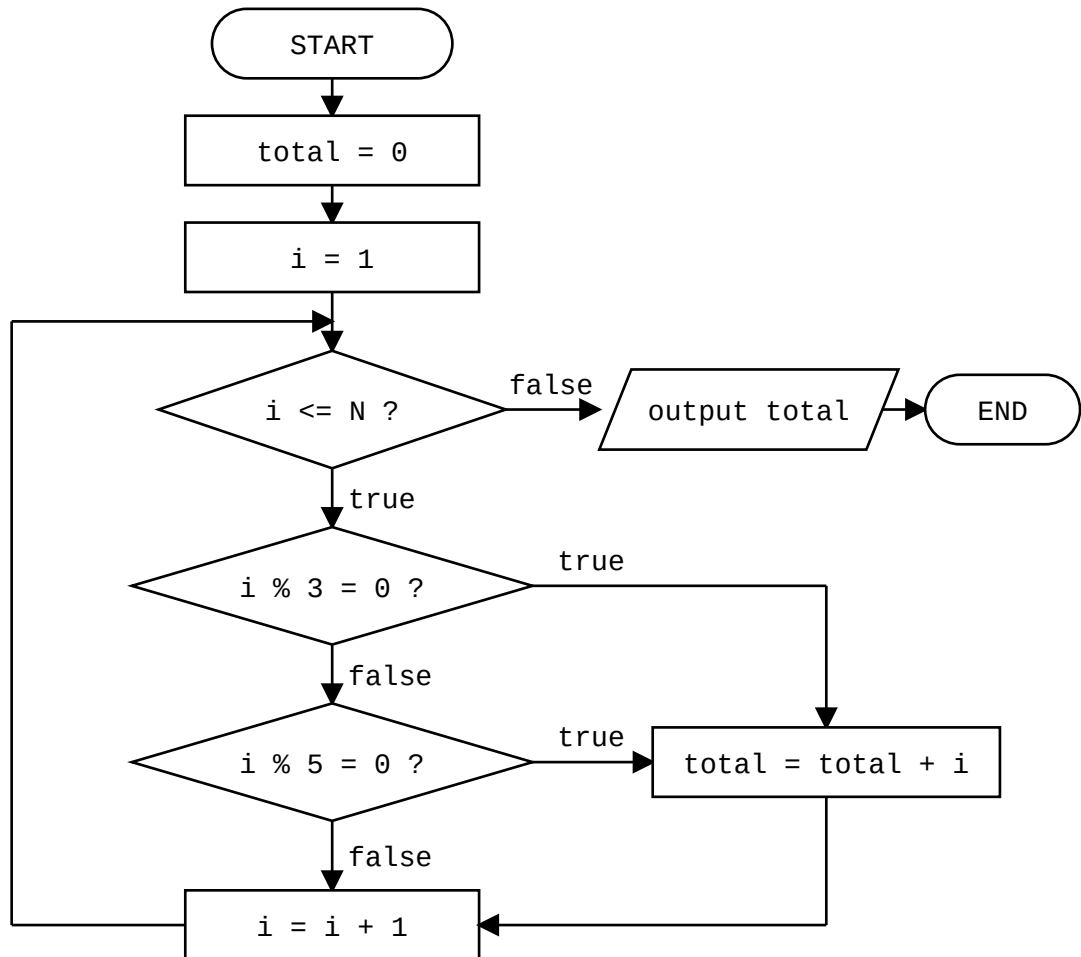
Option	Questions
Option A — Java	1 – 3
Option B — Python	4 – 6

Option A — Java

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

1. **Figure 1** shows a flowchart for an algorithm. A whole number, N , is input when the algorithm starts. In the flowchart, $\%$ gives the remainder after a division. For example, $7 \% 3$ is 1.

Figure 1: Flowchart for a summing algorithm



- (a) Copy and complete the trace table for this flowchart, where $N = 7$.

[3]

i	$i \leq N ?$	$i \% 3 = 0 ?$	$i \% 5 = 0 ?$	total	output
1				0	

Big O notation is used to describe the efficiency of an algorithm.

- (b) Using Big O notation, describe how the time taken by the algorithm in **Figure 1** grows as N increases.

[2]

(Option A continues on the following page)

(Option A, question 1 continued)

Computational thinking can be used to design an algorithm before any code is written.

A bakery wants a program that calculates how much of each ingredient it needs to order every week. The programmer applies computational thinking in the following two steps:

- Step 1: The programmer records only the mass of each ingredient needed to bake one loaf of each type, and ignores details such as the shape of each loaf and the design of the bags in which the loaves are sold.
- Step 2: The programmer notices that the mass of every ingredient is calculated in the same way: the number of loaves to be baked is multiplied by the mass of that ingredient needed for one loaf.

- (c) Identify the computational thinking concept that has been applied in each of these steps.

[2]

Table 1 shows a list of values.

Table 1: A list of values

Index	0	1	2	3	4
Value	29	14	37	8	21

- (d) Describe the steps that a selection sort would follow to sort this list into ascending order.

[3]

(Option A continues on the following page)

(Option A, question 1 continued)

In a school canteen, clean trays are placed in a dispenser, and the last tray put in is the first tray taken out. A program models the dispenser as a stack, `trays`, in which each tray is represented by its number.

Table 2 shows the operations available on `trays`.

Table 2: Operations available on the stack, `trays`

Operation	Description
<code>push(x)</code>	Adds tray <code>x</code> to the top of the stack.
<code>pop()</code>	Removes the tray at the top of the stack and returns it.
<code>peek()</code>	Returns the tray at the top of the stack without removing it from the stack.
<code>isEmpty()</code>	Returns true if the stack is empty, else it returns false.
<code>size()</code>	Returns the number of trays in the stack.

At the start of lunch, `trays` holds three trays: tray 11 at the bottom, then tray 14, then tray 18 at the top. The following operations are then carried out, in order:

```
pop()
push(25)
push(30)
size()
pop()
peek()
push(41)
isEmpty()
```

- (e) Sketch `trays` after these operations have been carried out.

Your sketch should show which tray is at the top of the stack.

[2]

At the end of lunch, `trays` holds the numbers of the trays that are left. A second, empty stack, with the same operations, is also available.

- (f) Using only the operations shown in **Table 2**, describe the steps that will output the largest tray number in `trays`. At the end, `trays` must contain the same trays, in the same order, as at the start.

Assume that `trays` is not empty.

[3]

(Option A continues on the following page)

(Option A continued)

2. A recycling centre collects four materials: glass, paper, plastic and metal. At the end of each day, the mass of each material collected that day is recorded in kilograms, as a whole number. The centre is open every day.

A program stores the masses for one week in two global variables, `materials` and `mass`. They are declared in the program's main class, outside any method:

```
static String[] materials = {"glass", "paper", "plastic", "metal"};
static int[][] mass = new int[4][7];
```

`mass[m][d]` holds the mass of material `m` collected on day `d`, where material 0 is glass and day 0 is Monday. For example, `mass[1][4] = 25` shows that 25 kg of paper was collected on day 4 (Friday).

Table 3 shows the contents of `mass` for one week.

Table 3: Mass, in kg, of each material collected on each day

Material	m	d = 0	d = 1	d = 2	d = 3	d = 4	d = 5	d = 6
glass	0	20	15	25	10	30	20	20
paper	1	35	40	20	30	25	45	15
plastic	2	10	25	30	15	20	35	40
metal	3	5	10	15	25	10	15	30

The method `showTotals()` is intended to output the total mass of each material collected during the week:

```
public static void showTotals()
{
    int total = 0;
    for (int m = 0; m < 4; m++)
    {
        for (int d = 0; d < 7; d++)
        {
            total = total + mass[m][d];
        }
        System.out.println (materials[m] + ": " + total);
    }
}
```

The variable `total` is a local variable of `showTotals()`.

- (a) Outline why `total` should be a local variable, rather than a global variable. [2]
- (b) State the **first two** lines of output when `showTotals()` runs with the data in **Table 3**. [2]

(Option A continues on the following page)

(Option A, question 2 continued)

The output of `showTotals()` is not correct. To find the error, a programmer adds the following statement as the first statement inside the outer loop, before the inner loop:

```
System.out.println ("m = " + m + ", total = " + total);
```

- (c) Describe how the output of this statement would help the programmer to find **and** correct the error. [3]

At the end of each day, the method `saveDay()` is passed the number of that day. It appends **one** line to the end of the existing text file `week.txt`. The line holds the day number, followed by the masses of glass, paper, plastic and metal collected on that day, all separated by commas, and ends with a new line. For example, using the data in **Table 3**, `saveDay(5)` appends this line:

5, 20, 45, 35, 15

The file is currently closed.

- (d) Construct the method `saveDay()`. [3]

The method `dailyBest()` uses the global variables `materials` and `mass`. For each day, it outputs the day number and the name of the material with the greatest mass collected on that day. For example, using the data in **Table 3**, the first line of output is:

Day 0: paper

- (e) Construct the method `dailyBest()`.

Assume that, on each day, only one material has the greatest mass.

Library methods that sort values or find a maximum, such as `Arrays.sort()`, `Collections.max()` and `Math.max()`, must not be used. [5]

(Option A continues on the following page)

(Option A continued)

3. A school library has eight study rooms that students can book. Each room is booked in one-hour slots, and there are 10 slots each day: slot 0 starts at 9:00, slot 1 starts at 10:00, and so on, up to slot 9, which starts at 18:00. The library uses object-oriented programming (OOP) to manage the bookings: each room is represented by a StudyRoom object.

The StudyRoom class is partially shown:

```
public class StudyRoom
{
    private static int firstHour = 9; // slot 0 starts at 9:00
    private String name;              // name of the room, e.g. "Room B"
    private int capacity;             // maximum number of students
    private boolean[] booked;        // true if slot s is booked

    public StudyRoom(...)
    {
        // code missing
    }

    public String getName()
    {
        return this.name;
    }

    public int getCapacity()
    {
        return this.capacity;
    }

    public boolean isFree(int s)      // true if slot s is not booked
    {
        return this.booked[s] == false;
    }

    public boolean book(int s)        // books slot s if it is free
    {
        if (this.booked[s] == true)
            return false;
        this.booked[s] = true;
        return true;
    }

    public static String slotTime(int s) // start time of slot s
    {
        return (firstHour + s) + ":00";
    }

    public int firstFree()             // returns the first free slot
    {
        // code missing
    }
}
```

(Option A continues on the following page)

Turn over

(Option A, question 3 continued)

The following statement creates the `StudyRoom` object for Room B, which can hold up to 6 students:

```
StudyRoom room1 = new StudyRoom("Room B", 6);
```

The constructor is passed the name of the room and its capacity. All 10 slots of a new room must be free.

- (a) Construct the constructor for the `StudyRoom` class. [2]

The method `slotTime()` is declared as `static`, but the method `isFree()` is not.

- (b) Distinguish between `slotTime()` and `isFree()` in terms of how each method is called and the data that each method can use. [2]

The method `firstFree()` returns the number of the first slot that has not been booked, or `-1` if every slot has been booked. For example, if only slots 0, 1 and 3 of a room have been booked, `firstFree()` returns 2.

- (c) Construct the method `firstFree()`.

Stream methods, such as `IntStream.range()`, must not be used. [3]

The library decides to add a method, `cancel()`, which is passed a slot number and makes that slot free again.

- (d) Construct the UML class diagram of the `StudyRoom` class after `cancel()` has been added. [3]

The attribute `booked` is declared as `private`.

The library's programmers decide to change the way in which the bookings are stored inside the class. For example, the bookings for a room could be stored in a single string such as `"--XX-X----`", where `X` shows a booked slot.

- (e) Outline how information hiding means that code outside the `StudyRoom` class, such as the driver code, does not need to be changed. [2]

The library also wants to list every room that is free in a given slot. Instead of using `StudyRoom` objects, the bookings could be stored in one 2D array, `bookings`, with one row for each room and one column for each slot.

- (f) Evaluate the use of `StudyRoom` objects, rather than the 2D array `bookings`, for this booking system. [3]

(Option A continues on the following page)

(Option A, question 3 continued)

An array, `rooms`, of size 8 holds the `StudyRoom` objects for the library's 8 study rooms. The driver class includes the method `showFirstFree(rooms)`. For each room in `rooms`, it outputs:

- the name of the room and the start time of its first free slot, for example `Room B 11:00`, if the room has a free slot
- the name of the room and `Fully booked`, if every slot in that room has been booked.

Finally, it outputs the number of rooms that are fully booked.

(g) Construct the method `showFirstFree()`.

You may use any of the methods in the `StudyRoom` class.

[5]

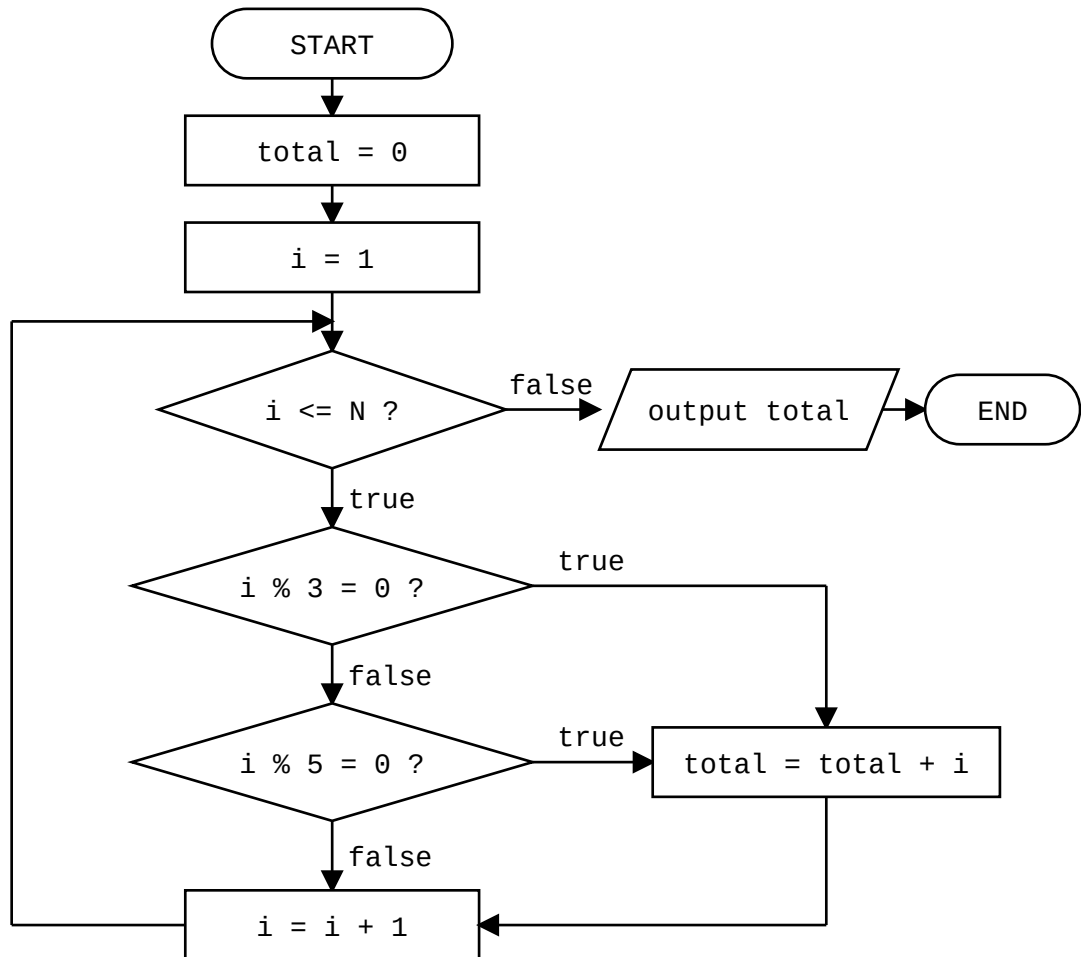
End of Option A

Option B — Python

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

4. **Figure 2** shows a flowchart for an algorithm. A whole number, N , is input when the algorithm starts. In the flowchart, $\%$ gives the remainder after a division. For example, $7 \% 3$ is 1.

Figure 2: Flowchart for a summing algorithm



- (a) Copy and complete the trace table for this flowchart, where $N = 7$.

[3]

i	$i \leq N ?$	$i \% 3 = 0 ?$	$i \% 5 = 0 ?$	total	output
1				0	

Big O notation is used to describe the efficiency of an algorithm.

- (b) Using Big O notation, describe how the time taken by the algorithm in **Figure 2** grows as N increases.

[2]

(Option B continues on the following page)

(Option B, question 4 continued)

Computational thinking can be used to design an algorithm before any code is written.

A bakery wants a program that calculates how much of each ingredient it needs to order every week. The programmer applies computational thinking in the following two steps:

- Step 1: The programmer records only the mass of each ingredient needed to bake one loaf of each type, and ignores details such as the shape of each loaf and the design of the bags in which the loaves are sold.
- Step 2: The programmer notices that the mass of every ingredient is calculated in the same way: the number of loaves to be baked is multiplied by the mass of that ingredient needed for one loaf.

- (c) Identify the computational thinking concept that has been applied in each of these steps.

[2]

Table 4 shows a list of values.

Table 4: A list of values

Index	0	1	2	3	4
Value	29	14	37	8	21

- (d) Describe the steps that a selection sort would follow to sort this list into ascending order.

[3]

(Option B continues on the following page)

(Option B, question 4 continued)

In a school canteen, clean trays are placed in a dispenser, and the last tray put in is the first tray taken out. A program models the dispenser as a stack, `trays`, in which each tray is represented by its number.

Table 5 shows the operations available on `trays`.

Table 5: Operations available on the stack, `trays`

Operation	Description
<code>push(x)</code>	Adds tray <code>x</code> to the top of the stack.
<code>pop()</code>	Removes the tray at the top of the stack and returns it.
<code>peek()</code>	Returns the tray at the top of the stack without removing it from the stack.
<code>isEmpty()</code>	Returns true if the stack is empty, else it returns false.
<code>size()</code>	Returns the number of trays in the stack.

At the start of lunch, `trays` holds three trays: tray 11 at the bottom, then tray 14, then tray 18 at the top. The following operations are then carried out, in order:

```
pop()
push(25)
push(30)
size()
pop()
peek()
push(41)
isEmpty()
```

- (e) Sketch `trays` after these operations have been carried out.

Your sketch should show which tray is at the top of the stack.

[2]

At the end of lunch, `trays` holds the numbers of the trays that are left. A second, empty stack, with the same operations, is also available.

- (f) Using only the operations shown in **Table 5**, describe the steps that will output the largest tray number in `trays`. At the end, `trays` must contain the same trays, in the same order, as at the start.

Assume that `trays` is not empty.

[3]

(Option B continues on the following page)

(Option B continued)

5. A recycling centre collects four materials: glass, paper, plastic and metal. At the end of each day, the mass of each material collected that day is recorded in kilograms, as a whole number. The centre is open every day.

A program stores the masses for one week in two global variables, `materials` and `mass`. They are declared outside any function:

```
materials = ["glass", "paper", "plastic", "metal"]
mass = [[0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0]]
```

`mass[m][d]` holds the mass of material `m` collected on day `d`, where material 0 is glass and day 0 is Monday. For example, `mass[1][4] = 25` shows that 25 kg of paper was collected on day 4 (Friday).

Table 6 shows the contents of `mass` for one week.

Table 6: Mass, in kg, of each material collected on each day

Material	m	d = 0	d = 1	d = 2	d = 3	d = 4	d = 5	d = 6
glass	0	20	15	25	10	30	20	20
paper	1	35	40	20	30	25	45	15
plastic	2	10	25	30	15	20	35	40
metal	3	5	10	15	25	10	15	30

The function `show_totals()` is intended to output the total mass of each material collected during the week:

```
def show_totals():
    total = 0
    for m in range(0, 4):
        for d in range(0, 7):
            total = total + mass[m][d]
            print(materials[m] + ":", total)
```

The variable `total` is a local variable of `show_totals()`.

- (a) Outline why `total` should be a local variable, rather than a global variable. [2]
- (b) State the **first two** lines of output when `show_totals()` runs with the data in **Table 6**. [2]

The output of `show_totals()` is not correct. To find the error, a programmer adds the following statement as the first statement inside the outer loop, before the inner loop:

```
print("m =", m, "total =", total)
```

- (c) Describe how the output of this statement would help the programmer to find **and** correct the error. [3]

(Option B continues on the following page)

Turn over

(Option B, question 5 continued)

At the end of each day, the function `save_day()` is passed the number of that day. It appends **one** line to the end of the existing text file `week.txt`. The line holds the day number, followed by the masses of glass, paper, plastic and metal collected on that day, all separated by commas, and ends with a new line. For example, using the data in **Table 6**, `save_day(5)` appends this line:

5, 20, 45, 35, 15

The file is currently closed.

(d) Construct the function `save_day()`. [3]

The function `daily_best()` uses the global variables `materials` and `mass`. For each day, it outputs the day number and the name of the material with the greatest mass collected on that day. For example, using the data in **Table 6**, the first line of output is:

Day 0: paper

(e) Construct the function `daily_best()`.

Assume that, on each day, only one material has the greatest mass.

Functions and methods that sort values or find a maximum, such as `max()`, `sorted()` and `sort()`, must not be used. [5]

(Option B continues on the following page)

(Option B continued)

6. A school library has eight study rooms that students can book. Each room is booked in one-hour slots, and there are 10 slots each day: slot 0 starts at 9:00, slot 1 starts at 10:00, and so on, up to slot 9, which starts at 18:00. The library uses object-oriented programming (OOP) to manage the bookings: each room is represented by a StudyRoom object.

The instance attributes of each StudyRoom object are `self._name` (the name of the room, eg "Room B"), `self._capacity` (the maximum number of students) and `self._booked` (a list of 10 Boolean values, where `self._booked[s]` is True if slot `s` is booked).

The StudyRoom class is partially shown:

```
class StudyRoom():

    _first_hour = 9                # slot 0 starts at 9:00

    def __init__(...):
        # code missing

    def get_name(self):
        return self._name

    def get_capacity(self):
        return self._capacity

    def is_free(self, s):           # True if slot s is not booked
        return self._booked[s] == False

    def book(self, s):             # books slot s if it is free
        if self._booked[s] == True:
            return False
        self._booked[s] = True
        return True

    @staticmethod
    def slot_time(s):              # start time of slot s
        return str(StudyRoom._first_hour + s) + ":00"

    def first_free(self):          # returns the first free slot
        # code missing
```

The following statement creates the StudyRoom object for Room B, which can hold up to 6 students:

```
room1 = StudyRoom("Room B", 6)
```

The constructor is passed the name of the room and its capacity. All 10 slots of a new room must be free.

- (a) Construct the constructor, `__init__()`, for the StudyRoom class. [2]

(Option B continues on the following page)

(Option B, question 6 continued)

The method `slot_time()` is a static method (it is declared using `@staticmethod`), but the method `is_free()` is not.

- (b) Distinguish between `slot_time()` and `is_free()` in terms of how each method is called and the data that each method can use. [2]

The method `first_free()` returns the number of the first slot that has not been booked, or `-1` if every slot has been booked. For example, if only slots 0, 1 and 3 of a room have been booked, `first_free()` returns 2.

- (c) Construct the method `first_free()`.

The `index()` method and the `next()` function must not be used. [3]

The library decides to add a method, `cancel()`, which is passed a slot number and makes that slot free again.

- (d) Construct the UML class diagram of the `StudyRoom` class after `cancel()` has been added. [3]

The underscore character at the start of `self._booked` identifies the variable as being a private attribute of the class.

The library's programmers decide to change the way in which the bookings are stored inside the class. For example, the bookings for a room could be stored in a single string such as `"--XX-X----`", where X shows a booked slot.

- (e) Outline how information hiding means that code outside the `StudyRoom` class, such as the driver code, does not need to be changed. [2]

The library also wants to list every room that is free in a given slot. Instead of using `StudyRoom` objects, the bookings could be stored in one 2D list, `bookings`, with one row for each room and one column for each slot.

- (f) Evaluate the use of `StudyRoom` objects, rather than the 2D list `bookings`, for this booking system. [3]

(Option B continues on the following page)

(Option B, question 6 continued)

A list, `rooms`, holds the `StudyRoom` objects for the library's 8 study rooms. The driver code includes the function `show_first_free(rooms)`. For each room in `rooms`, it outputs:

- the name of the room and the start time of its first free slot, for example `Room B 11:00`, if the room has a free slot
- the name of the room and `Fully booked`, if every slot in that room has been booked.

Finally, it outputs the number of rooms that are fully booked.

(g) Construct the function `show_first_free()`.

You may use any of the methods in the `StudyRoom` class.

[5]

End of Option B
