

# **Markscheme**

**Mock examination — Set 2**

**Computer science**

**Standard level**

**Paper 2**

This markscheme is **confidential** and for the exclusive use of the teachers marking this mock examination.

It must **not** be shared with candidates before the mock examination has been sat.

### General marking instructions

1. Follow the markscheme provided, award only whole marks and mark only in **RED**.
2. Where a mark is awarded, a tick/check (✓) **must** be placed in the text at the **precise point** where it becomes clear that the candidate deserves the mark. **One tick to be shown for each mark awarded.**
3. Sometimes, careful consideration is required to decide whether or not to award a mark. In these cases write a brief comment in the margin to support your decision, especially for moderation purposes. It should be remembered that the script may be returned to the candidate.
4. Personal codes/notations are unacceptable.
5. Where an answer to a part question is worth no marks but the candidate has attempted the part question, write a zero next to the answer. Where the candidate has not attempted the part question, write “NR”.
6. Ensure that you have viewed **every** page including any additional sheets.
7. A mark should not be awarded where there is contradiction within an answer. Write “CON” next to the contradiction.
8. **If a candidate has attempted more than one Option** within a paper, mark all the candidate’s work and award the marks for the higher scoring Option.

## Subject details: Computer Science SL Paper 2 markscheme (Mock examination — Set 2)

### Mark allocation

Candidates are required to answer **all** questions in **one** Option. Total 50 marks.

### General

A markscheme often has more specific points worthy of a mark than the total allows. This is intentional. Do not award more than the maximum marks allowed for that part of a question.

When deciding upon alternative answers by candidates to those given in the markscheme, consider the following points:

- Each statement worth one point has a separate line and the end is signified by means of a semi-colon (;).
- An alternative answer or wording is indicated in the markscheme by a “/”; either wording can be accepted.
- Words in ( ... ) in the markscheme are not necessary to gain the mark.
- If the candidate’s answer has the same meaning or can be clearly interpreted as being the same as that in the markscheme then award the mark.
- Mark positively. Give candidates credit for what they have achieved and for what they have got correct, rather than penalizing them for what they have not achieved or what they have got wrong.
- Remember that many candidates are writing in a second language; be forgiving of minor linguistic slips. In this subject effective communication is more important than grammatical accuracy.
- Occasionally, a part of a question may require a calculation whose answer is required for subsequent parts. If an error is made in the first part then it should be penalized. However, if the incorrect answer is used correctly in subsequent parts then **follow through** marks should be awarded. Indicate this with “FT”.

### General guidance

| Issue                                                                     | Guidance                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|---------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Answering more than the quantity of responses prescribed in the questions | <ul style="list-style-type: none"><li>• In the case of an “identify” question, read all answers and mark positively up to the maximum marks. Disregard incorrect answers.</li><li>• In the case of a “describe” question, which asks for a certain number of facts eg “describe two kinds”, mark the first two correct answers. This could include two descriptions, one description and one identification, or two identifications.</li><li>• In the case of an “explain” question, which asks for a specified number of explanations eg “explain two reasons ...”, mark the first two correct answers. This could include two full explanations, one explanation, one partial explanation etc.</li></ul> |

## Option A — Java

### 1. (a) Award [3] as follows:

correct values of  $i$  and correct TRUE/FALSE values in the  $i \leq N$  ? column;  
 correct TRUE/FALSE values in both remainder columns;  
 correct values of total and correct output;

| $i$ | $i \leq N$ ? | $i \% 3 = 0$ ? | $i \% 5 = 0$ ? | total | output |
|-----|--------------|----------------|----------------|-------|--------|
| 1   | TRUE         | FALSE          | FALSE          | 0     |        |
| 2   | TRUE         | FALSE          | FALSE          |       |        |
| 3   | TRUE         | TRUE           |                | 3     |        |
| 4   | TRUE         | FALSE          | FALSE          |       |        |
| 5   | TRUE         | FALSE          | TRUE           | 8     |        |
| 6   | TRUE         | TRUE           |                | 14    |        |
| 7   | TRUE         | FALSE          | FALSE          |       |        |
| 8   | FALSE        |                |                |       | 14     |

*Note: When  $i$  is 3 or 6, the  $i \% 5 = 0$  ? decision is not reached; do not penalize a value written in that column for these rows. Accept any layout in which the values appear in the correct order. Award FT for later values that follow correctly from an earlier error.*

### (b) Award [2 max]

$O(N)$  / linear time;  
 ...because the loop is repeated  $N$  times / once for each value of  $i$  from 1 to  $N$ ;  
 ...and each pass carries out a fixed (maximum) number of steps / at most three decisions and two assignments;  
 ...so doubling  $N$  (about) doubles the time taken;

*Note: Award [1] for  $O(N)$  and [1] for a valid description linked to the number of passes through the loop. Do not accept  $O(N^2)$ ,  $O(\log N)$  or  $O(1)$  for the first mark; a correct description of the loop can still gain [1].*

### (c) Award [2] as follows: [1] for each step.

Step 1: abstraction;  
 Step 2: pattern recognition;

*Note: Accept generalization for Step 2. Do not accept decomposition or algorithmic design for either step.*

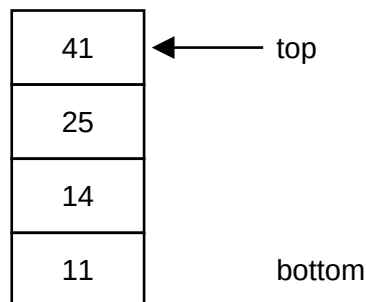
### (d) Award [3] as follows:

pass 1: the smallest value, 8, is found (by comparing all the values) and swapped with the first value, 29, giving 8, 14, 37, 29, 21;  
 pass 2: the smallest of the values at index 1 to 4 is 14, which is already at index 1, so it is not moved / the list is unchanged;  
 pass 3: the smallest of the values at index 2 to 4 is 21, which is swapped with 37, giving 8, 14, 21, 29, 37, and pass 4 leaves 29 at index 3, so the list is sorted;

*Note: Award [1 max] for a correct general description of a selection sort that is not applied to this list. Accept the lists after each pass, provided that it is clear that the smallest remaining value is selected on each pass. Award FT for later passes that follow correctly from an earlier error. Accept a selection sort that selects the largest value on each pass: 37 is swapped with 21, giving 29, 14, 21, 8, 37 [1]; 29 is swapped with 8, giving 8, 14, 21, 29, 37 [1]; 21 and 14 are already in place, so the list is sorted [1]. A description of a bubble sort (comparing and swapping adjacent values) gains no marks.*

(e) Award [2].

correct trays in the correct order, from bottom to top 11, 14, 25, 41 [1];  
correct indication of the top (41) [1];



*Note: Do not award the first mark if 18 or 30 is shown in the stack. Accept the stack drawn horizontally, provided that the top is clearly labelled.*

(f) Award [3] as follows: [1] for each correct line (or equivalent).

repeat until isEmpty() is true / while isEmpty() is false / size() times: pop a tray number and push it onto the second stack;

compare each popped number with the largest number found so far, and keep the larger (the first number popped / peek() gives the starting value); output the largest once every number has been checked;

then repeat until the second stack is empty: pop a number from the second stack and push it back onto trays;

*Note: Answer must demonstrate an algorithmic approach. Pushing the numbers back from the second stack restores their original order. Storing the numbers in an array, list or queue rather than in the second stack does not gain the first or the third mark, because only the operations in the table may be used.*

2. (a) Award **[2 max]**. Award **[1]** for a reason and **[1]** for an expansion.

total is only needed inside showTotals() / no other method uses total;  
...so other methods cannot change it (by mistake) / the value cannot be affected by code elsewhere in the program;

a local variable only exists while the method is running;  
...so its memory is released when the method ends / it is created again, with a new value, each time the method is called;

the same name can be used for variables in other methods (without a conflict);  
...so the method is self-contained / easier to reuse/test/debug;

*Note: Award **[1 max]** for a definition of local and global variables that does not give a reason.*

- (b) Award **[2]** as follows: **[1]** for each correct line.

glass: 140;  
paper: 350;

*Note: paper: 210 (the intended total) does not gain the second mark, because total is not reset. Ignore minor differences in spacing or punctuation.*

- (c) Award **[3 max]**

the statement outputs the value of total at the start of the calculation for each material / before the masses of each material are added;  
for m = 0 it outputs total = 0, but for m = 1 it outputs total = 140 (the total for glass) instead of 0;

so the programmer can see that total is not reset to 0 for each material / the totals of the earlier materials are carried over;  
move the statement int total = 0; inside the outer loop, before the inner loop / add total = 0; as the first statement in the outer loop;

*Note: Award **[2 max]** if no correct correction is given. Accept answers that refer to the values output for m = 2 (350) or m = 3 (525).*

- (d) Award **[3]** as follows:

Award [1] for opening week.txt in append mode (eg new FileWriter("week.txt", true));  
Award [1] for writing the day number and the four masses for that day (mass[0][d] to mass[3][d]), separated by commas and followed by a new line;  
Award [1] for closing the file;

### Example answer

```
public static void saveDay(int d) throws IOException
{
    FileWriter writer = new FileWriter("week.txt", true);
    String line = "" + d;
    for (int m = 0; m < 4; m++)
    {
        line = line + "," + mass[m][d];
    }
    writer.write(line + "\n");
    writer.close();
}
```

### Alternative answer

```
public static void saveDay(int d) throws IOException
{
    PrintWriter out = new PrintWriter(new FileWriter("week.txt", true));
    out.println(d + "," + mass[0][d] + "," + mass[1][d] + ","
               + mass[2][d] + "," + mass[3][d]);
    out.close();
}
```

*Note: Opening the file without append mode (eg new FileWriter("week.txt")) does not gain the first mark, because it deletes the lines already in the file. The masses may be written using a loop or one at a time. mass[d][m] does not gain the second mark, and nor does a trailing comma after the last mass (eg 5,20,45,35,15,), because the format is given. Penalize write(int) (which writes a character code, not the digits) or joining with char literals such as ',' (which adds character codes) once only. Accept a BufferedWriter with newLine(), or Files.writeString() with StandardOpenOption.APPEND. Accept try-with-resources (try (FileWriter writer = ...) { ... }) for the closing mark. Exception handling (throws IOException or try/catch) is not required for any mark. Ignore minor syntax errors; penalize a repeated error once only.*

(e) Award **[5]** as follows:

Award [1] for an outer loop through the 7 days;  
Award [1] for an inner loop through the 4 materials that accesses mass[m][d] for that day;  
Award [1] for initializing best (eg best = 0) for each day, inside the outer loop and before the inner loop, together with the greatest mass if it is stored (eg greatest = mass[0][d]);  
Award [1] for comparing mass[m][d] with the greatest mass so far (eg greatest or mass[best][d]) and updating best (and the greatest mass, if it is stored separately);  
Award [1] for outputting the day number and materials[best] once for each day, after the inner loop;

### Example answer

```
public static void dailyBest()
{
    for (int d = 0; d < 7; d++)
    {
        int greatest = -1;      // or mass[0][d] with m from 1
        int best = 0;
        for (int m = 0; m < 4; m++)
        {
            if (mass[m][d] > greatest)
            {
                greatest = mass[m][d];
                best = m;
            }
        }
        System.out.println ("Day " + d + ": " + materials[best]);
    }
}
```

*Note: Accept an answer that loops through the materials in the outer loop, stores the greatest mass and its material for each day in two arrays of size 7, and then outputs them; award the equivalent marks. Accept tracking only the index of the best material: best = 0, then if (mass[m][d] > mass[best][d]) best = m; A loop that starts at m = 1 is correct only if, for each day, best is initialized to 0 (and the greatest mass, if stored, to mass[0][d]). best may be initialized once, before the outer loop, only if the inner loop starts at m = 0 and either only the index is tracked or the greatest mass starts at -1 for each day. Initializing greatest = mass[0][d] each day but best only once does not gain the third mark. Ignore the format of the output. An answer that uses a library method that is not permitted (including Math.max()) gains **[3 max]**. Ignore minor syntax errors; penalize a repeated error once only.*

3. (a) Award **[2]** as follows:

Award [1] for a constructor header with a String and an int parameter that assigns them to the attributes name and capacity;

Award [1] for assigning a new boolean array of 10 elements to the attribute booked (this.booked = new boolean[10] / booked = new boolean[10]);

**Example answer**

```
public StudyRoom(String name, int capacity)
{
    this.name = name;
    this.capacity = capacity;
    this.booked = new boolean[10];    // every element is false
}
```

*Note: Do not award the array mark if a new local array is declared (boolean[] booked = new boolean[10];), because it shadows the attribute, which is left as null. Similarly, declaring local variables such as String name = n; does not gain the first mark. Setting each element of booked to false with a loop is not required, as a new boolean array holds false in every element. this. is not required if the parameter names differ from the attribute names. Ignore minor syntax errors; penalize a repeated error once only.*

(b) Award **[2 max]**. Award **[1]** for a valid point about slotTime() and **[1]** for a contrasting point about isFree().

slotTime() belongs to the class / is called using the class name, without a StudyRoom object (eg StudyRoom.slotTime(2));

isFree() belongs to each object / must be called on a particular StudyRoom object (eg room1.isFree(2));

slotTime() can only use static attributes, such as firstHour (and its parameter) / cannot use the attributes of a room, such as booked;

isFree() uses the attributes (booked) of the object on which it is called;

slotTime() gives the same result for every room / does not depend on the data of any room;

isFree() can give different results for different rooms;

(c) Award **[3]** as follows:

Award [1] for a loop through all 10 slots;

Award [1] for finding the first (lowest-numbered) slot that is not booked (booked[s] == false / isFree(s)) and returning its number;

Award [1] for returning -1 only when no slot is free, after all 10 slots have been checked;

**Example answer**

```
public int firstFree()
{
    for (int s = 0; s < 10; s++)    // allow s < booked.length
    {
        if (booked[s] == false)    // allow if (!booked[s])
            return s;
    }
    return -1;
}
```

*Note: Accept any loop that returns the lowest-numbered free slot, eg a loop from slot 9 down to 0 that records each free slot, or a variable that records the first free slot with a single return after the loop. Returning -1 inside the loop (eg in an else branch) does not gain the third mark. An answer that returns the last (highest-numbered) free slot does not gain the second mark. Using stream methods (not permitted by the question) gains **[1 max]**. Ignore minor syntax errors; penalize a repeated error once only.*

(d) Award **[3]** as follows:

Award [1] for the class name and all four attributes, with their data types and the private visibility symbol (-);

Award [1] for all six methods in the class code, with their parameters, return types and the public visibility symbol (+);

Award [1] for the new method cancel() with an int parameter and a return type (eg void or boolean);

| StudyRoom                                                                                                                                                                                                                            |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <u>- firstHour: int</u><br>- name: String<br>- capacity: int<br>- booked: boolean[]                                                                                                                                                  |
| + StudyRoom(name: String, capacity: int)<br>+ getName(): String<br>+ getCapacity(): int<br>+ isFree(s: int): boolean<br>+ book(s: int): boolean<br><u>+ slotTime(s: int): String</u><br>+ firstFree(): int<br>+ cancel(s: int): void |

*Note: Allow one minor slip (eg one missing data type) within each of the first two marks. The constructor does not need to be shown. Static members (firstHour and slotTime()) may be underlined, but this is not required. Accept other clear notations for data types (eg int capacity).*

(e) Award **[2 max]**

booked is private, so code outside the class cannot access booked directly / only the methods of StudyRoom use booked;

...code outside the class uses only the public methods, such as book(), isFree() and firstFree();

...only the code inside these methods has to be rewritten; their names, parameters and return values stay the same;

...so the driver code still works without any change / the change is hidden from the rest of the program;

*Note: Award **[1 max]** for a definition of information hiding that is not linked to the change in the way the bookings are stored.*

- (f) *Award [3 max]. Award [1] for each valid point that is linked to the booking system.*

#### **Advantages of StudyRoom objects**

each object keeps the data for one room (name, capacity, bookings) together with the methods that use it / models a real room;  
the class is written once and used for all eight rooms / a new room is added by creating another object, without changing the class;  
the bookings can only be changed through methods such as book(), which checks that a slot is free, so a slot cannot be double-booked by mistake;  
listing the free rooms is still simple: a loop calls isFree(slot) on each room, so no new code is needed inside the class;  
the class can be tested and maintained separately from the driver code / different programmers can work on the class and the driver;

#### **Disadvantages of StudyRoom objects**

the 2D array answers the new request directly, by reading one column, whereas the objects need a method call on every room;  
more code and design time are needed than for a single 2D array, which may not be justified for only eight rooms;  
objects and method calls use more memory/processing time than reading one array directly;  
the programmers must understand OOP concepts / the program may be harder for a beginner to follow;

*Note: Award [2 max] if only advantages or only disadvantages are given. Award [1 max] for points that are not linked to the booking system.*

- (g) *Award [5] as follows:*

Award [1] for a loop through all 8 elements of rooms;  
Award [1] for calling firstFree() for each room and comparing the result with -1 (or searching slots 0 to 9 of each room with isFree());  
Award [1] for outputting getName() and the start time of the first free slot, using slotTime() (eg StudyRoom.slotTime(f)), when a slot is free;  
Award [1] for outputting getName() and Fully booked when no slot is free;  
Award [1] for counting the rooms that are fully booked and outputting the count once, after the loop;

### Example answer

```
public static void showFirstFree(StudyRoom[] rooms)
{
    int full = 0;
    for (int i = 0; i < 8; i++)           // allow i < rooms.length
    {
        int f = rooms[i].firstFree();
        if (f == -1)
        {
            System.out.println (rooms[i].getName() + " Fully booked");
            full = full + 1;
        }
        else
            System.out.println (rooms[i].getName() + " "
                                + StudyRoom.slotTime(f));
    }
    System.out.println ("Fully booked rooms: " + full);
}
```

*Note: Accept a search of slots 0 to 9 using isFree() in place of firstFree() for the second mark. Accept rooms[i].slotTime(f), and accept (9 + f) + ":00" for the start time. Calling firstFree() more than once for a room is acceptable. Outputting the count inside the loop does not gain the fifth mark. Ignore the format of the output. Ignore minor syntax errors; penalize a repeated error once only.*

## Option B — Python

### 4. (a) Award [3] as follows:

correct values of  $i$  and correct TRUE/FALSE values in the  $i \leq N$  ? column;  
 correct TRUE/FALSE values in both remainder columns;  
 correct values of total and correct output;

| $i$ | $i \leq N$ ? | $i \% 3 = 0$ ? | $i \% 5 = 0$ ? | total | output |
|-----|--------------|----------------|----------------|-------|--------|
| 1   | TRUE         | FALSE          | FALSE          | 0     |        |
| 2   | TRUE         | FALSE          | FALSE          |       |        |
| 3   | TRUE         | TRUE           |                | 3     |        |
| 4   | TRUE         | FALSE          | FALSE          |       |        |
| 5   | TRUE         | FALSE          | TRUE           | 8     |        |
| 6   | TRUE         | TRUE           |                | 14    |        |
| 7   | TRUE         | FALSE          | FALSE          |       |        |
| 8   | FALSE        |                |                |       | 14     |

*Note: When  $i$  is 3 or 6, the  $i \% 5 = 0$  ? decision is not reached; do not penalize a value written in that column for these rows. Accept any layout in which the values appear in the correct order. Award FT for later values that follow correctly from an earlier error.*

### (b) Award [2 max]

$O(N)$  / linear time;  
 ...because the loop is repeated  $N$  times / once for each value of  $i$  from 1 to  $N$ ;  
 ...and each pass carries out a fixed (maximum) number of steps / at most three decisions and two assignments;  
 ...so doubling  $N$  (about) doubles the time taken;

*Note: Award [1] for  $O(N)$  and [1] for a valid description linked to the number of passes through the loop. Do not accept  $O(N^2)$ ,  $O(\log N)$  or  $O(1)$  for the first mark; a correct description of the loop can still gain [1].*

### (c) Award [2] as follows: [1] for each step.

Step 1: abstraction;  
 Step 2: pattern recognition;

*Note: Accept generalization for Step 2. Do not accept decomposition or algorithmic design for either step.*

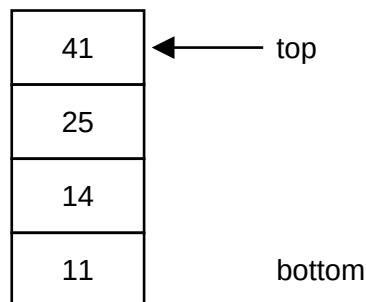
### (d) Award [3] as follows:

pass 1: the smallest value, 8, is found (by comparing all the values) and swapped with the first value, 29, giving 8, 14, 37, 29, 21;  
 pass 2: the smallest of the values at index 1 to 4 is 14, which is already at index 1, so it is not moved / the list is unchanged;  
 pass 3: the smallest of the values at index 2 to 4 is 21, which is swapped with 37, giving 8, 14, 21, 29, 37, and pass 4 leaves 29 at index 3, so the list is sorted;

*Note: Award [1 max] for a correct general description of a selection sort that is not applied to this list. Accept the lists after each pass, provided that it is clear that the smallest remaining value is selected on each pass. Award FT for later passes that follow correctly from an earlier error. Accept a selection sort that selects the largest value on each pass: 37 is swapped with 21, giving 29, 14, 21, 8, 37 [1]; 29 is swapped with 8, giving 8, 14, 21, 29, 37 [1]; 21 and 14 are already in place, so the list is sorted [1]. A description of a bubble sort (comparing and swapping adjacent values) gains no marks.*

(e) Award [2].

correct trays in the correct order, from bottom to top 11, 14, 25, 41 [1];  
correct indication of the top (41) [1];



*Note: Do not award the first mark if 18 or 30 is shown in the stack. Accept the stack drawn horizontally, provided that the top is clearly labelled.*

(f) Award [3] as follows: [1] for each correct line (or equivalent).

repeat until isEmpty() is true / while isEmpty() is false / size() times: pop a tray number and push it onto the second stack;

compare each popped number with the largest number found so far, and keep the larger (the first number popped / peek() gives the starting value); output the largest once every number has been checked;

then repeat until the second stack is empty: pop a number from the second stack and push it back onto trays;

*Note: Answer must demonstrate an algorithmic approach. Pushing the numbers back from the second stack restores their original order. Storing the numbers in an array, list or queue rather than in the second stack does not gain the first or the third mark, because only the operations in the table may be used.*

5. (a) Award **[2 max]**. Award **[1]** for a reason and **[1]** for an expansion.

total is only needed inside show\_totals() / no other function uses total;  
...so other functions cannot change it (by mistake) / the value cannot be affected by code elsewhere in the program;

a local variable only exists while the function is running;  
...so its memory is released when the function ends / it is created again, with a new value, each time the function is called;

the same name can be used for variables in other functions (without a conflict);  
...so the function is self-contained / easier to reuse/test/debug;

*Note: Award **[1 max]** for a definition of local and global variables that does not give a reason.*

- (b) Award **[2]** as follows: **[1]** for each correct line.

glass: 140;  
paper: 350;

*Note: paper: 210 (the intended total) does not gain the second mark, because total is not reset. Ignore minor differences in spacing or punctuation.*

- (c) Award **[3 max]**

the statement outputs the value of total at the start of the calculation for each material / before the masses of each material are added;  
for m = 0 it outputs total = 0, but for m = 1 it outputs total = 140 (the total for glass) instead of 0;  
so the programmer can see that total is not reset to 0 for each material / the totals of the earlier materials are carried over;  
move the statement total = 0 inside the outer loop, before the inner loop / add total = 0 as the first statement in the outer loop;

*Note: Award **[2 max]** if no correct correction is given. Accept answers that refer to the values output for m = 2 (350) or m = 3 (525).*

- (d) Award **[3]** as follows:

Award [1] for opening week.txt in append mode (open("week.txt", "a"));  
Award [1] for writing the day number and the four masses for that day (mass[0][d] to mass[3][d]), separated by commas and followed by a new line;  
Award [1] for closing the file;

#### Example answer

```
def save_day(d):  
    file = open("week.txt", "a")  
    line = str(d)  
    for m in range(0, 4):  
        line = line + "," + str(mass[m][d])  
    file.write(line + "\n")  
    file.close()
```

### Alternative answer

```
def save_day(d):
    with open("week.txt", "a") as file:
        file.write(str(d) + "," + str(mass[0][d]) + ","
                  + str(mass[1][d]) + "," + str(mass[2][d])
                  + "," + str(mass[3][d]) + "\n")
```

*Note: Opening the file in write mode ("w") does not gain the first mark, because it deletes the lines already in the file. Accept with open("week.txt", "a") as file: for both the opening and the closing marks. The masses may be written using a loop or one at a time; accept print(line, file=file). mass[d][m] does not gain the second mark, and nor does a trailing comma after the last mass (eg 5,20,45,35,15,), because the format is given. Penalize a missing str() (which raises a TypeError) once only. Exception handling is not required for any mark. Ignore minor syntax errors; penalize a repeated error once only.*

(e) Award **[5]** as follows:

Award [1] for an outer loop through the 7 days;  
Award [1] for an inner loop through the 4 materials that accesses mass[m][d] for that day;  
Award [1] for initializing best (eg best = 0) for each day, inside the outer loop and before the inner loop, together with the greatest mass if it is stored (eg greatest = mass[0][d]);  
Award [1] for comparing mass[m][d] with the greatest mass so far (eg greatest or mass[best][d]) and updating best (and the greatest mass, if it is stored separately);  
Award [1] for outputting the day number and materials[best] once for each day, after the inner loop;

### Example answer

```
def daily_best():
    for d in range (0, 7):
        greatest = -1          # or mass[0][d] with m from 1
        best = 0
        for m in range (0, 4):
            if mass[m][d] > greatest:
                greatest = mass[m][d]
                best = m
        print ("Day " + str(d) + ": " + materials[best])
```

*Note: Accept an answer that loops through the materials in the outer loop, stores the greatest mass and its material for each day in two lists of size 7, and then outputs them; award the equivalent marks. Accept tracking only the index of the best material: best = 0, then if mass[m][d] > mass[best][d]: best = m. A loop that starts at m = 1 is correct only if, for each day, best is initialized to 0 (and the greatest mass, if stored, to mass[0][d]). best may be initialized once, before the outer loop, only if the inner loop starts at m = 0 and either only the index is tracked or the greatest mass starts at -1 for each day. Initializing greatest = mass[0][d] each day but best only once does not gain the third mark. Ignore the format of the output. An answer that uses a function or method that is not permitted gains **[3 max]**. Ignore minor syntax errors; penalize a repeated error once only.*

6. (a) Award **[2]** as follows:

Award [1] for a header `def __init__(self, name, capacity)` that assigns the parameters to `self._name` and `self._capacity`;

Award [1] for assigning a list of 10 elements that are all False to `self._booked`;

**Example answer**

```
def __init__(self, name, capacity):
    self._name = name
    self._capacity = capacity
    self._booked = [False] * 10
```

**Alternative answer**

```
def __init__(self, name, capacity):
    self._name = name
    self._capacity = capacity
    self._booked = []
    for s in range(0, 10):
        self._booked.append(False)
```

*Note: An empty list (`self._booked = []`) does not gain the second mark, and nor does a local variable without `self`. (`booked = [False] * 10`), because `book()` and `is_free()` access `self._booked[s]`. Similarly, assigning `name` and `capacity` without `self` does not gain the first mark. Ignore minor syntax errors; penalize a repeated error once only.*

(b) Award **[2 max]**. Award **[1]** for a valid point about `slot_time()` and **[1]** for a contrasting point about `is_free()`.

`slot_time()` belongs to the class / is called using the class name, without a `StudyRoom` object (eg `StudyRoom.slot_time(2)`);

`is_free()` belongs to each object / must be called on a particular `StudyRoom` object (eg `room1.is_free(2)`);

`slot_time()` has no `self` parameter, so it can only use class attributes, such as `_first_hour` (and its parameter) / cannot use the attributes of a room, such as `self._booked`;

`is_free()` uses the attributes (`self._booked`) of the object on which it is called (`self`);

`slot_time()` gives the same result for every room / does not depend on the data of any room;

`is_free()` can give different results for different rooms;

(c) Award **[3]** as follows:

Award [1] for a loop through all 10 slots;

Award [1] for finding the first (lowest-numbered) slot that is not booked (`self._booked[s] == False` / `self.is_free(s)`) and returning its number;

Award [1] for returning -1 only when no slot is free, after all 10 slots have been checked;

**Example answer**

```
def first_free(self):
    for s in range(0, 10):          # allow range (len(self._booked))
        if self._booked[s] == False:
            return s
    return -1
```

### Alternative answer

```
def first_free(self):
    first = -1
    s = 0
    while s < 10 and first == -1:
        if self._booked[s] == False:
            first = s
        s = s + 1
    return first
```

*Note: Accept any loop that returns the lowest-numbered free slot, eg a loop from slot 9 down to 0 that records each free slot, or an enumerate() loop. Returning -1 inside the loop (eg in an else branch) does not gain the third mark. An answer that returns the last (highest-numbered) free slot does not gain the second mark. Using index() or next() (not permitted by the question) gains **[1 max]**. Ignore minor syntax errors; penalize a repeated error once only.*

(d) Award **[3]** as follows:

Award [1] for the class name and all four attributes, with their data types and the private visibility symbol (-);

Award [1] for all six methods in the class code, with their parameters, return types and the public visibility symbol (+);

Award [1] for the new method cancel() with an int parameter and a return type (eg None or bool);

| StudyRoom                                                                                                                                                                                                                 |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| - <u>first</u> hour: int<br>- _name: str<br>- _capacity: int<br>- _booked: list                                                                                                                                           |
| + __init__(name: str, capacity: int)<br>+ get_name(): str<br>+ get_capacity(): int<br>+ is_free(s: int): bool<br>+ book(s: int): bool<br>+ <u>slot_time(s: int): str</u><br>+ first_free(): int<br>+ cancel(s: int): None |

*Note: Allow one minor slip (eg one missing data type) within each of the first two marks. The constructor does not need to be shown. Accept names without the underscore (eg - booked: list), list[bool] or bool[] for \_booked, and void for None. Class members (\_first\_hour and slot\_time()) may be underlined, but this is not required.*

(e) **Award [2 max]**

self.\_booked is private, so code outside the class should not access \_booked directly / only the methods of StudyRoom use \_booked;  
...code outside the class uses only the public methods, such as book(), is\_free() and first\_free();  
...only the code inside these methods has to be rewritten; their names, parameters and return values stay the same;  
...so the driver code still works without any change / the change is hidden from the rest of the program;

*Note: Award [1 max] for a definition of information hiding that is not linked to the change in the way the bookings are stored.*

(f) **Award [3 max]. Award [1] for each valid point that is linked to the booking system.**

**Advantages of StudyRoom objects**

each object keeps the data for one room (name, capacity, bookings) together with the methods that use it / models a real room;  
the class is written once and used for all eight rooms / a new room is added by creating another object, without changing the class;  
the bookings can only be changed through methods such as book(), which checks that a slot is free, so a slot cannot be double-booked by mistake;  
listing the free rooms is still simple: a loop calls is\_free(slot) on each room, so no new code is needed inside the class;  
the class can be tested and maintained separately from the driver code / different programmers can work on the class and the driver;

**Disadvantages of StudyRoom objects**

the 2D list answers the new request directly, by reading one column, whereas the objects need a method call on every room;  
more code and design time are needed than for a single 2D list, which may not be justified for only eight rooms;  
objects and method calls use more memory/processing time than reading one list directly;  
the programmers must understand OOP concepts / the program may be harder for a beginner to follow;

*Note: Award [2 max] if only advantages or only disadvantages are given. Award [1 max] for points that are not linked to the booking system.*

(g) **Award [5] as follows:**

Award [1] for a loop through all 8 elements of rooms;  
Award [1] for calling first\_free() for each room and comparing the result with -1 (or searching slots 0 to 9 of each room with is\_free());  
Award [1] for outputting get\_name() and the start time of the first free slot, using slot\_time() (eg StudyRoom.slot\_time(f)), when a slot is free;  
Award [1] for outputting get\_name() and Fully booked when no slot is free;  
Award [1] for counting the rooms that are fully booked and outputting the count once, after the loop;

### Example answer

```
def show_first_free(rooms):
    full = 0
    for i in range (0, 8):                # allow range (len(rooms))
        f = rooms[i].first_free()
        if f == -1:
            print (rooms[i].get_name(), "Fully booked")
            full = full + 1
        else:
            print (rooms[i].get_name(), StudyRoom.slot_time(f))
    print ("Fully booked rooms:", full)
```

### Alternative answer

```
def show_first_free(rooms):
    full = 0
    for room in rooms:
        f = room.first_free()
        if f == -1:
            full = full + 1
            print (room.get_name(), "Fully booked")
        else:
            print (room.get_name(), room.slot_time(f))
    print ("Fully booked rooms:", full)
```

*Note: Accept a search of slots 0 to 9 using `is_free()` in place of `first_free()` for the second mark. Accept `rooms[i].slot_time(f)`, and accept `str(9 + f) + ":00"` for the start time. Calling `first_free()` more than once for a room is acceptable. Outputting the count inside the loop does not gain the fifth mark. Ignore the format of the output. Ignore minor syntax errors; penalize a repeated error once only.*

---