NATIONAL UNIVERSITY OF SINGAPORE

CS1010S—Programming Methodology

2018/2019 Semester 1

Time Allowed: 2 hours

INSTRUCTIONS TO STUDENTS

1. Please write your Student Number only. Do not write your name.
2. The assessment paper contains **FIVE (5) questions** and comprises **TWELVE (12) pages** including this cover page.
3. Weightage of each question is given in square brackets. The maximum attainable score is 100.
4. This is a **CLOSED** book assessment, but you are allowed to bring **ONE** double-sided A4 sheet of notes for this assessment.
5. Write all your answers in the space provided in the **ANSWER BOOKLET**.
6. You are allowed to write with pencils, as long as it is legible.
7. **Marks may be deducted** for i) unrecognisable handwriting, and/or ii) excessively long code. A general guide would be not more than twice the length of our model answers.
8. Common `List` and `Dictionary` methods are listed in the Appendix for your reference.

This page is intentionally left blank.

It may be used as scratch paper.

Question 1: Python Expressions [30 marks]

There are several parts to this problem. Answer each part **independently and separately**. In each part, one or more Python expressions are entered into the interpreter (Python shell). Determine the response printed by the interpreter for the final expression entered and **write the exact output in the answer box**. If the interpreter produces an error message, or enters an infinite loop, explain why and **clearly state the responsible evaluation step**.

The code is replicated on the answer booklet. You may show your workings **outside the answer box** in the space beside the code. Partial marks will be awarded for workings if the final answer is wrong.

A.

```
x = 3
y = 4
def f(y):
    return g(x) + y
def g(y):
    return x + y
print(f(5))
```

[5 marks]

B.

```
for i in range(10):
    if i % 2 == 0:
        continue
        print(i)
    elif i % 3 == 0:
        i += 10
    elif i % 6 == 0:
        break
    print(i)
```

[5 marks]

C.

```
a = ["CS", 1010, "U"]
b = [a[:2], "S"]
c = b.copy()
c[0][1] = 2020
print(b + a[1:])
```

[5 marks]

D.

```
def scramble(s):
    if len(s) == 0:
        return ""
    else:
        mid = scramble(s[1:-1])
        return s[-1] + mid + s[0]
print(scramble("PU_EKAW"))
```

[5 marks]

E.

```
def hammer_time(*mc):
    try:
        return mc[0] + mc[-1]
    except TypeError:
        return str(mc[0]) + str(mc[-1])
    except IndexError:
        return mc
    except Exception:
        return "Can't touch this"
print( hammer_time(1, "2") )
print( hammer_time( [3, 4, 5] ) )
print( hammer_time() )
```

[5 marks]

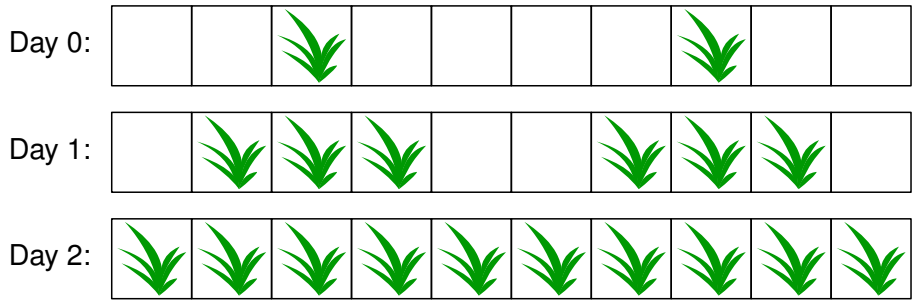
F.

```
def foo(x):
    return lambda y: x(y**2)
def bar(x):
    return lambda y: y(y(x))
print( bar(0)(foo(lambda x: x+3)) )
```

[5 marks]

Question 2: Weeding Weeds [25 marks]

A linear field has some weed grass growing at some plots. The weed grass is so fast growing that after one day, it will have spread to the adjacent plots. The figure below shows this process over two days in a given field:



A field containing weeds that grows over two days

Suppose the field is modelled as a `List` of `Bool`, with each index representing a plot and the value `True` indicating a weed grass is growing in the plot, and `False` otherwise. Thus, the above-mentioned field on Day 0 will be represented by:

```
[False, False, True, False, False, False, False, True, False, False]
```

A. Implement the function `grow(field)` which takes as input a field in this representation, and **modifies** it to the new state after a day of growth. [6 marks]

Another way to represent the field is to number the plots in sequential order, and maintain a list of the plots that contains weeds. A possible numbering for the above-mentioned field is to have the fourth plot be number 0, then the representation of the weeds on Day 0 is the list `[-1, 4]` and on Day 1 it will be `[-2, -1, 0, 3, 4, 5]`.

B. List one advantage that this second representation has over the first. [2 marks]

Farmer Stone came up with the following implementation of the function `grow` to work with the second representation:

```
1  def grow(field):
2      new = []
3      for plot in field:              # Changes suggested in part E
4          if plot-1 not in new:       #  new[-1] < plot-1
5              new.append(plot-1)
6          elif plot not in new:       #  new[-1] < plot
7              new.append(plot)
8          elif plot+1 not in new:     #  new[-1] < plot+1
9              new.append(plot+1)
10     field = new
```

C. **State and explain** the order of growth in terms of time and space for Farmer Stone's implementation of `grow`. [2 marks]

D. When Farmer Stone ran his function, he realised that it is not working as intended. Explain the error(s) and suggest how they can be fixed.

You need not rewrite the function if you can describe what needs to be fixed in words. [5 marks]

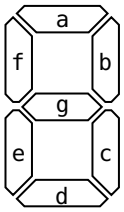
E. Your friend Mary Jane thinks Farmer Stone's algorithm is slow and she can do better. She suggests changing the condition in line 4 from `plot-1 not in new` to `new[-1] < plot-1`, as well as for lines 6 and 8 as shown in comments in the code.

Do you agree with her? Please **explain why** and if true, **how** the modification works, as well as **any assumptions** made. [5 marks]

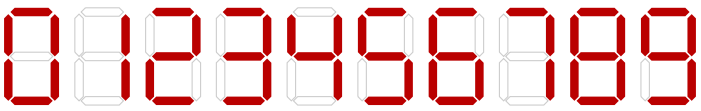
F. Farmer Stone sprays some herbicide on the fields which kills the weed on the plots that the herbicide lands on. Implement the function `kill(field, plots)` which takes as input the field using the second representation, and a sequence of integers representing plots that the herbicide lands on. The function modifies the field to the new state of that after the herbicide was applied to the given plots. [5 marks]

Question 3: *n*-Segment Display [25 marks]

A seven-segment display is a form of electronic display device to display Arabic numerals as well as some Latin characters. Each display unit is made up of seven segments, commonly labelled with letters '`a`' through '`f`' as shown on the right.



Some combination of the segments are turned on to produce a digit or character, for example, the digits 0–9 as shown:



One way of representing the state of a seven-segment display unit is to use a string containing the labels of the segments that are lit. So a unit displaying the digit `0` will be represented as `'abcdef'`, the digit `1` as `'bc'`, the digit `2` as `'abdeg'`, etc.

It is also possible to generalise this to an n-segment display that uses n segments to display characters by simply using more letters to represent the extra segments.

Note that the letters in the string representation need not follow any order.

A. Propose another way in which the state of a n-segment display unit can be represented in Python. [2 marks]

A Python dictionary *D*, contains the mapping of characters to the lit state of a n-segment display. In other words, the keys of *D* contains characters or digits, with the values being the corresponding segment lit state that displays the key. For example, a dictionary representing numeric digits using seven-segment displays might begin like:

```
{'0': 'abcdef', '1': 'bc', '2': 'abged', '3': 'abdcg', '4': 'fgbc', ...}
```

Note that you should not assume that the number of segments of the display are fixed and that it might be possible to display other characters in the display.

B. A inverse of the above-mentioned dictionary *D* would instead have its keys be the segments on the display and the values the corresponding characters that would require the segment to be lit. For example, segment 'a' is lit if the display were displaying digits 0, 2, 3, 5, 6, 7, 8, 9. Thus, the inverse dictionary would have an entry `'a': '02356789'`

Implement the function `inverse` that takes in the former dictionary mapping *D* and returns the latter dictionary mapping of the state of a n-segment display. [5 marks]

C. Implement the function `character(state, mapping)` that takes in a state of a n-segment display using our string representation, and a dictionary mapping *D*. It returns the character that is being displayed if a valid mapping exists. Otherwise, it returns `None`.
[5 marks]

D. Is your implementation in Part C faster than $O(\texttt{len}(\texttt{mapping}))$ time? Explain why and if no, what is needed for you to achieve it? [3 marks]

You observe a seven-segment display and see that the segments that are lit do not correspond to any know character or digit in *D*. You suspect that some segments are damaged and remain off even though they should have been lit, giving you a partial display.

For example, if the state `'adg'` is observed, it could mean that either 2, 3, 5, 6, 8 or 9 might be displayed.

E. Implement the function `guess(state, mapping)` that takes an observed state of a display and the dictionary mapping *D*. It returns a list of possible characters that is being displayed with the assumption that there can be damaged segments that are always off. [5 marks]

F. **[Warning: Challenging]** Now suppose you know for certain which segments are really damaged. You should be able to narrow the possibilities of the character being displayed.

For example, if you observe segments `adeg` being displayed, and know for certain that only segments `bf` are damaged, then you can rule out digits '`8`' and '`6` because segment `c` would have been lit. The only possible remaining digit is '`2`'.

Implement the function `better_guess(state, mapping, damaged)` that takes in the same inputs as the previous function, with the addition of a string of damaged segments. It should return a list of possible characters that might be displayed. [5 marks]

Question 4: Pickup Truck [16 marks]

"A pickup truck is a light-duty truck having an enclosed cab and an open cargo area with low sides and tailgate. Once a work tool with few creature comforts, in the 1950s, consumers began purchasing pickups for lifestyle reasons, and by the 1990s, less than 15% of owners reported use in work as the pickup truck's primary purpose. Today in North America, the pickup is mostly used like a passenger car and accounts for about 18% of total vehicles sold in the US."
– Wikipedia

In this question, we are going to model cars, trucks, and pickup trucks. Consider the implementation of `Vehicle` and `Car` below.

```
1  class Vehicle:
2      def __init__(self, name, mpg):
3          self.name = name
4          self.mpg  = mpg
5          self.fuel = 16
6
7     def fuel_used(self, mile):
8         return mile/self.mpg
9
10    def refuel(self, fuel):
11        self.fuel += fuel
12
13     def drive(self, mile):
14         fuel_used = self.fuel_used(mile)
15         if fuel_used > self.fuel:
16             print("Not enough fuel")
17             return False
18         else:
19             self.fuel -= fuel_used
20             print("Used " + str(fuel_used) + " gallon of fuel")
21             return True
22
23 class Car(Vehicle):
24     def __init__(self, name, mpg):
25         super().__init__(name, mpg)
26         self.passengers = 0
27
28     def board(self, passengers):
29         self.passengers += passengers
30
31     def alight(self, passengers):
32         self.passengers -= passengers
33
34     def fuel_used(self, mile):
35         return super().fuel_used(mile) + self.passengers*0.1*mile
```

A `Vehicle` is initialized with two inputs: its name (**`str`**) and its fuel consumption as mile per gallon (**`float`**). It is also initialized as having 16 gallons of fuel. A `Car` is a `Vehicle` that can carry passengers. It is initialized with two inputs: its name (**`str`**) and its fuel consumption as mile per gallon (**`float`**). It is also initialized as having 16 gallons of fuel.

The amount of fuel used in `Vehicle` and `Car` are different. For a `Vehicle`, the amount of fuel used is simply dependent on its fuel consumption. For a `Car`, it consumes an *additional* 0.1 gallon per mile for every passenger.

A. Your good friend, Takumi Fujiwara, tells you that the code for `Car` does not work because it does not have a function for `drive`. Even if it calls the `drive` from `Vehicle`, it will call the incorrect `fuel_used` function. Is he correct? If yes, write a correction for the code. If no, please explain why. [4 marks]

Takumi Fujiwara then shows you a different type of `Vehicle` called `Truck`. A `Truck` is initialized with two inputs: its name (**`str`**) and its fuel consumption as mile per gallon (**`float`**). A `Truck` behaves like a `Vehicle` except that you can load and unload cargo. To simplify, the methods `load` and `unload` take in an **`int`** which is the weight of the cargo to load or unload in kilogram.

Furthermore, the amount of fuel used will depend on the weight of the cargo carried. For every 1 kg of cargo, the `Truck` fuel consumption *increases* by 0.1%.

An example run is given below.

```
>>> truck = Truck("Might-E", 25)  # Might-E truck, 25mpg
>>> truck.drive(100)
'Used 4.0 gallon of fuel'         # 1/25 * 100
True

>>> truck.load(600)               # Load 600kg of cargo
>>> truck.drive(100)              # Uses 60% more fuel
'Used 6.4 gallon of fuel'
True

>>> truck.drive(100)              # Not enough fuel
'Not enough fuel'
False

>>> truck.unload(200)             # Unload 200kg of cargo
>>> truck.drive(100)              # Uses 40% more fuel
'Used 5.6 gallon of fuel'
True
```

B. Using OOP, provide a **minimal** implementation of the class `Truck`. [6 marks]

Now, a `Pickup_Truck` is a mix of `Car` and `Truck` in a sense that it can be used to carry both passengers and items. The fuel usage will then depend on both the number of passengers and the weight of the items. Like a `Truck`, for every kilogram of item, the `Pickup_Truck` fuel consumption increases by 0.1%. Similarly, like a `Car`, for every passenger `Pickup_Truck` uses 0.1 additional gallons of fuel per mile.

Takumi's Father, Bunta, suggested that the code for `Pickup_Truck` can be simply implemented as:

```
class Pickup_Truck(Car, Truck):
    pass
```

Takumi thinks this is nonsense and raises the following issues:

i) The class is empty, so the `Pickup_Truck` has no functionality.

ii) A pickup truck is primarily a truck, and secondarily a car. Thus, the class should be defined as `class Pickup_Truck(Truck, Car):` instead.

iii) After fixing the issues above, the `drive` function is not able to call the both `fuel_used()` functions in both `Car` and `Truck` classes. So it will only calculate the fuel consumption as a `Car` or `Truck`, but not both.

C. Address **each** of Takumi's concerns by explaining why he is right or wrong. [6 marks]

Question 5: 42 and the Meaning of Life [4 marks]

Either: (a) explain how you think some of what you have learnt in CS1010S will be helpful for you for the rest of your life and/or studies at NUS; or (b) tell us an interesting story about your experience with CS1010S this semester. [4 marks]

— END OF PAPER —

Appendix

Parts of the Python documentation is given here for your reference.

List Methods

- `list.append(x)` Add an item to the end of the list.
- `list.extend(iterable)` Extend the list by appending all the items from the iterable.
- `list.insert(i, x)` Insert an item at a given position.
- `list.remove(x)` Remove the first item from the list whose value is *x*. It is an error if there is no such item.
- `list.pop([i])` Remove the item at the given position in the list, and return it. If no index is specified, removes and returns the last item in the list.
- `list.clear()` Remove all items from the list
- `list.index(x)` Return zero-based index in the list of the first item whose value is *x*. Raises a `ValueError` if there is no such item.
- `list.count(x)` Return the number of times x appears in the list.
- `list.sort(key=None, reverse=False)` Sort the items of the list in place.
- `list.reverse()` Reverse the elements of the list in place.
- `list.copy()` Return a shallow copy of the list.

Dictionary Methods

- `dict.clear()` Remove all items from the dictionary.
- `dict.copy()` Return a shallow copy of the dictionary.
- `dict.items()` Return a new view of the dictionary's items (`(key, value)` pairs).
- `dict.keys()` Return a new view of the dictionary's keys.
- `dict.pop(key[, default])` If *key* is in the dictionary, remove it and return its value, else return *default*. If *default* is not given and *key* is not in the dictionary, a `KeyError` is raised.
- `dict.update([other])` Update the dictionary with the key/value pairs from *other*, overwriting existing keys. Return `None`.
- `dict.values()` Return a new view of the dictionary's values.

Scratch Paper

Scratch Paper

— H A P P Y H O L I D A Y S ! —

CS1010S — Programming Methodology
School of Computing
National University of Singapore

Final Assessment — Answer Sheet

2018/2019 Semester 1

Time allowed: 2 hours

Student No:	**A**								

Instructions (please read carefully):

1. Write down your **student number** on this answer sheet. DO NOT WRITE YOUR NAME!
2. This answer booklet comprises **TWELVE (12) pages**, including this cover page.
3. All questions must be answered in the space provided; no extra sheets will be accepted as answers. You may use the extra page behind this cover page if you need more space for your answers.
4. You must submit only the **ANSWER SHEET** and no other documents. The question set may be used as scratch paper.
5. An excerpt of the question may be provided to aid you in answering in the correct box. It is not the exact question. You should still refer to the original question in the question booklet.
6. You are allowed to use pencils, ball-pens or fountain pens, as you like as long as it is legible (no red color, please).
7. **Marks may be deducted** for i) unrecognisable handwriting, and/or ii) excessively long code. A general guide would be not more than twice the length of our model answers.

For Examiner's Use Only

Question	Marks
Q1	/ 30
Q2	/ 25
Q3	/ 25
Q4	/ 16
Q5	/ 4
Total	/100

This page is intentionally left blank.
Use it ONLY if you need extra space for your answers, and indicate the **question number clearly** as well as in the original answer box. **Do NOT** use it for your rough work.

Question 1A

[5 marks]

```
x = 3
y = 4
def f(y):
    return g(x) + y
def g(y):
    return x + y
print(f(5))
```

Question 1B

[5 marks]

```
for i in range(10):
    if i % 2 == 0:
        continue
        print(i)
    elif i % 3 == 0:
        i += 10
    elif i % 6 == 0:
        break
    print(i)
```

Question 1C

[5 marks]

```
a = ["CS", 1010, "U"]
b = [a[:2], "S"]
c = b.copy()
c[0][1] = 2020
print(b + a[1:])
```

Question 1D

[5 marks]

```
def scramble(s):
    if len(s) == 0:
        return ""
    else:
        mid = scramble(s[1:-1])
        return s[-1] + mid + s[0]
print(scramble("PU_EKAW"))
```

Question 1E

[5 marks]

```
def hammer_time(*mc):
    try:
        return mc[0] + mc[-1]
    except TypeError:
        return str(mc[0]) + str(mc[-1])
    except IndexError:
        return mc
    except Exception:
        return "Can't touch this"
print( hammer_time(1, "2") )
print( hammer_time( [3, 4, 5] ) )
print( hammer_time() )
```

Question 1F

[5 marks]

```
def foo(x):
    return lambda y: x(y**2)
def bar(x):
    return lambda y: y(y(x))
print( bar(0)(foo(lambda x: x+3)) )
```

Question 2A [6 marks]

```
def grow(field):
```

Question 2B List one advantage of the second representation [2 marks]

Question 2C Explain the order of growth for Farmer Stone's implementation [2 marks]

Question 2D Explain the error(s) and suggest how to fix the implementation [5 marks]

Question 2E Do you agree with Mary Jane's modification? Explain [5 marks]

Question 2F [5 marks]

```
def kill(field, plots):
```

Question 3A Propose another representation [2 marks]

Question 3B [5 marks]

```
def inverse(mapping):
```

Question 3C [5 marks]

```
def character(state, mapping):
```

Question 3D Is your implementation faster than linear time? Explain [3 marks]

Question 3E [5 marks]

```
def guess(state, mapping):
```

Question 3F [5 marks]

```
def better_guess(state, mapping, damaged):
```

Question 4A Explain why Takumi is correct/wrong [4 marks]

Question 4B Provide a minimal OOP implementation of the class [6 marks]

Question 4C Address each of Takumi's concerns [6 marks]

i)

ii)

iii)

Question 5 42 and the Meaning of Life [4 marks]

This page is intentionally left blank.
Use it ONLY if you need extra space for your answers, and indicate the **question number clearly** as well as in the original answer box. **Do NOT** use it for your rough work.

— END OF ANSWER SHEET —

Question 1A

[5 marks]

```
x = 3
y = 4
def f(y):
    return g(x) + y
def g(y):
    return x + y
print(f(5))
```

```
11
```

Question 1B

[5 marks]

```
for i in range(10):
    if i % 2 == 0:
        continue
        print(i)
    elif i % 3 == 0:
        i += 10
    elif i % 6 == 0:
        break
    print(i)
```

```
1
13
5
7
19
```

Question 1C

[5 marks]

```
a = ["CS", 1010, "U"]
b = [a[:2], "S"]
c = b.copy()
c[0][1] = 2020
print(b + a[1:])
```

```
[['CS', 2020], 'S', 1010, 'U']
```

Question 1D

[5 marks]

```
def scramble(s):
    if len(s) == 0:
        return ""
    else:
        mid = scramble(s[1:-1])
        return s[-1] + mid + s[0]
print(scramble("PU_EKAW"))
```

`'WAKEE_UP'` both with or without quotes are accepted

Question 1E

[5 marks]

```
def hammer_time(*mc):
    try:
        return mc[0] + mc[-1]
    except TypeError:
        return str(mc[0]) + str(mc[-1])
    except IndexError:
        return mc
    except Exception:
        return "Can't touch this"
print( hammer_time(1, "2") )
print( hammer_time( [3, 4, 5] ) )
print( hammer_time() )
```

```
12
[3, 4, 5, 3, 4, 5]
()
```

Question 1F

[5 marks]

```
def foo(x):
    return lambda y: x(y**2)
def bar(x):
    return lambda y: y(y(x))
print( bar(0)(foo(lambda x: x+3)) )
```

```
12
```

Question 2A [6 marks]

```
def grow(field):
    # grow to the left
    for i in range(0, len(field)-1):
        if field[i+1]:
            field[i] = True
    # grow to the right
    for i in range(len(field)-1, 1, -1):
        if field[i-1]:
            field[i] = True
```

Be careful not to grow new weeds from the new sprouts.

Question 2B List one advantage of the second representation [2 marks]

1. It uses less space as it contains only plots with weeds.
2. The field is of unlimited size and is not constrained to the size of the list.

Any of the above is acceptable

Question 2C Explain the order of growth for Farmer Stone's implementation [2 marks]

Both time and space is $O(\texttt{len}(\texttt{field})^2))$ due to the way duplicates are checked before inserting new elements. The inbuilt `in` operation does a linear search which takes $O(\texttt{len}(\texttt{new}))$, and this search is done for before every append to the list. This incurs $O(1+2+3+...n) = O(n^2)$ where $n = \texttt{len}(\texttt{field})$.

Question 2D Explain the error(s) and suggest how to fix the implementation [5 marks]

Lines 6 and 8 should not be an `elif`. It should be an `if` as the conditions should be independent because up to 3 weeds can grow from one weed. Otherwise, the conditions will be dependent and only one weed is appended.

Line 10 does not modify the input `field`. It reassigns the local argument to the new list, not the original input variable. To correct, replace line 10 with the following:

```
field.clear()
field.extend(new)
```

Question 2E Do you agree with Mary Jane's modification? Explain [5 marks]

Yes she is correct. This removes the need to do a linear search to find duplicates when appending a new element. The assumption is that `field` must be sorted in ascending order and that `new` has to be initialised to `field[0]-1` in line 2.

This works because the elements in `field` are examined in increasing order and any smaller elements that the current would have already been appended earlier into `new`.

Question 2F [5 marks]

```
def kill(field, plots):
    for plot in plots:
        if plot in field:
            field.remove(plot)
```

Question 3A Propose another representation [2 marks]

1. Use a list of length n of `Bool` values, where each element represents a segment, and the value `True` means it is lit, and `False` otherwise.
2. Each segment can be assigned with a value of different power-of-two (or >1), and the state can be the sum of the values of segments that are lit.
3. Each segment can be assigned a different prime number and the state can be the product of the segments that are lit.

Any 1 of the 3 schemes are sufficient

Question 3B [5 marks]

```
def inverse(mapping):
    d = {}
    for k, v in mapping.items():
        for seg in v:
         if seg not in d:
             d[seg] = ''
         d[seg] += k
    return d
```

Question 3C [5 marks]

```
def character(state, mapping):
    for k, v in mapping.items():
        if v == state:
            return k
    # implicit None is returned
```

Question 3D Is your implementation faster than linear time? Explain [3 marks]

No, it is linear time. It is not possible to go faster because we are searching in the values of the dictionary, and the only way is to do a linear search. We can achieve $O(1)$ timing by building a reverse dictionary that maps the values to the keys and use it.

Question 3E [5 marks]

```
def guess(state, mapping):
    chars = list(mappings.keys())
    for seg in state:
        chars = list(filter(lambda c: seg in mapping[c], chars))
    return chars
```

Question 3F [5 marks]

The conditions each possible character c have to meet are

1. $\forall i \in \texttt{state} \rightarrow i \in \texttt{mapping[c]}$
2. $\forall i \in \texttt{mapping[c]} \rightarrow i \in \texttt{state} \cup \texttt{damaged}$

```
def better_guess(state, mapping, damaged):
    chars = guess(state, mapping)  # condition 1
    state += damaged               # state ∪ damaged
    for c in chars.copy():         # }
        for seg in mapping[c]:     # } condition 2
            if seg not in state:   # }
                chars.remove(c)    # remove if not met
                break;
     return chars
```

Question 4A Explain why Takumi is correct/wrong [4 marks]

He is wrong on both claims. The class `Car` inherits `Vehicle`. Hence, it will inherit the function `drive`. Furthermore, if the instance created is `Car`, `self.fuel_usage` will call the function defined in `Car`.

Question 4B Provide a minimal OOP implementation of the class [6 marks]

```
class Truck(Vehicle):
  def __init__(self, name, mpg):
    super().__init__(name, mpg)
    self.weight = 0

  def load(self, weight):
    self.weight += weight

  def unload(self, weight):
    self.weight -= weight

  def fuel_used(self, mile):
    return super().fuel_used(mile) * (1+ self.weight/1000)
```

Question 4C Address each of Takumi's concerns [6 marks]

i) There is no need to define any function in `Pickup_Truck` because all the functions can be inherited from both the super classes.

ii) It depends on certain assumptions:

If the 0.1% increase includes the additional consumption from each passenger, then Takumi is correct. Truck's `super().fuel_used` will call `Car.fuel_used()` to get the total consumption.

If we assume that the percentage increase is on the base consumption, then Bunta's answer is correct. Truck's `super().fuel_used` should call `Vehicle.fuel_used()` to get the base mpg.

If `Pickup_Truck.fuel_used` is computed using `self.mpg`, then there is no difference in the order of inheritance.

Note that the constructor (`__init__`) in both `Car` and `Truck` perform independent tasks, so it does not matter which order they are called.

iii) `drive` will call `super().fuel_used(mile)` which will call `Car.fuel_used()`. Then the `super().fuel_used(mile)` in `Car` will call `Truck.fuel_used(mile)` because both `Car` and `Truck` inherit from a common superclass `Vehicle`. So all the subclasses' `fuel_used` must be called first before the superclass function can be called. Thus, `Truck.fuel_used` will finally call `Vehicle.fuel_used()`. Now the `fuel_used` methods in all classes will be called.

Question 5 42 and the Meaning of Life [4 marks]

The student will be awarded points as long as he/she is coherent and doesn't say something obviously wrong.