NATIONAL UNIVERSITY OF SINGAPORE

CS1010S—Programming Methodology

2019/2020 Semester 2

Time Allowed: 1 hour

INSTRUCTIONS TO STUDENTS

1. The assessment paper contains **FIVE (5) questions** and comprises **NINE (9) pages** including this cover page.
2. Weightage of each question is given in square brackets. The maximum attainable score is **50**.
3. This is a **CLOSED** book assessment, but you are allowed to bring **ONE** double-sided A4 sheet of notes for this assessment.
4. Enter all your answers in the correct space provided on Examplify.
5. **Marks may be deducted** for excessively long code. A general guide would be not more than twice the length of our model answers.
6. Common `List` and `Dictionary` methods are listed in the Appendix for your reference.

Question 1: Python Expressions [10 marks]

There are two 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**.

A.

```
lst = [[1], [2, 2], [3, 3, 3]]

def f(lst):
    for i in lst.copy():
        if len(i) < 2:
            i.append(1)
        if sum(i) < 5:
            i.pop()
        else:
            lst.extend(i)
        print(lst)
    return lst

print(lst is f(lst))
print(lst)
```

[5 marks]

B.

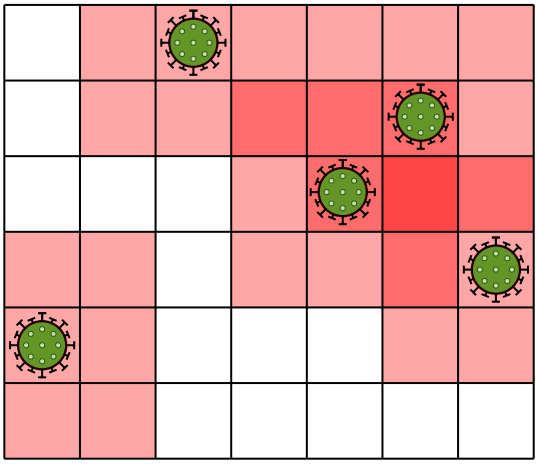
```
s = 'mississipi'
d = {}
while s:
    if s[0] not in d:
        d[s[0]] = s[1]
    else:
        d.pop(s[0])
    s = s[1:]
    print(d)
```

[5 marks]

Question 2: COVID Sweeper [15 marks]

Background (OK to skip) *COVID Sweeper is a game invented by Ng Keng Hwee which is a blatant rip-off of Minesweeper. It basically works the same but with mines replaced by coronaviruses.* *Source: Dr. Leong*

Viruses are scattered within a rectangular grid:



The shaded red cells show the infectious zones for each virus, which are the 8 adjacent cells of a virus, as well as the middle cell containing the virus itself. Darker reds shows overlapping infectious zones from two or more viruses around that cell.

A. Suppose the position of each virus in the grid is given as a sequence of `(row, col)` pairs. For example, for the grid shown above, the sequence will be:

```
seq = ((0, 2), (1, 5), (2, 4), (3, 6), (4, 0))
```

The function `adjacent_viruses(seq, row, col)` takes as input the positions of the viruses, a row and column number. It returns the number of viruses surrounding and within the cell in the given row and column. For example:

```
>>> adjacent_viruses(seq, 2, 5)
3
>>> adjacent_viruses(seq, 1, 5)
2
>>> adjacent_viruses(seq, 4, 4)
0
```

Provide an implementation for the function `adjacent_viruses`. You may assume that the inputs are always valid and will not exceed the bounds of the grid. [5 marks]

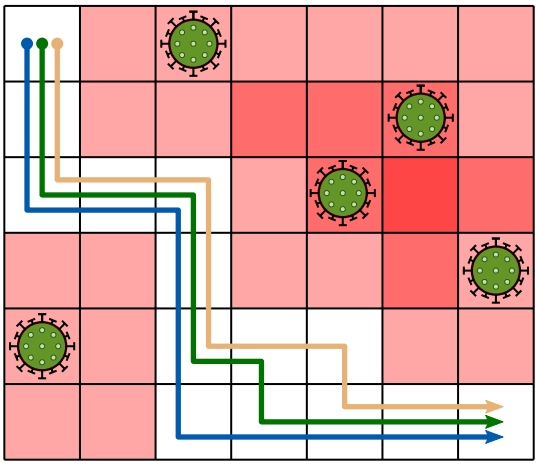
B. Knowing the position of each virus is good, but having a map of the infectious zone is better.

The function `adjacent_grid(seq, height, width)` that takes as inputs the sequence of viruses' position of the viruses, and the height and width of the grid. It returns a 2D grid, represented as a tuple of rows, where each row is a tuple of integers indicating number of adjacent viruses for each respective cell. For example:

```
>>> adjacent_grid(seq, 6, 7)
((0, 1, 1, 1, 1, 1, 1),
 (0, 1, 1, 2, 2, 2, 1),
 (0, 0, 0, 1, 2, 3, 2),
 (1, 1, 0, 1, 1, 2, 1),
 (1, 1, 0, 0, 0, 1, 1),
 (1, 1, 0, 0, 0, 0, 0))
```

Provide an implementation for the function `adjacent_grid`. [5 marks]

C. Suppose we want to walk starting from the (0, 0) corner of the grid to the opposite corner, taking care to avoid the infectious zone of each virus. We are only allowed to walk along non-decreasing row and column numbers, i.e. only down or right as viewed in the illustration.



For the given sequence of viruses, there are three possible paths as shown above.

Implement the function `safe_paths(grid)`, which takes as input the grid representation created in Part B. The function returns the number of paths where we can safely get from the (0, 0) corner to the opposing corner. For example:

```
>>> safe_paths(adjacent_grid(seq, 6, 7))
3
```

[5 marks]

Question 3: Counting Faster [15 marks]

Like you, Ben Bitdiddle learnt about counting change in CS1010S:

```
def count_change(amt, denom):
    if amt < 0 or len(denom) == 0:
        return 0
    elif amt == 0:
        return 1
    else:
        return count_change(amt-denom[0], denom) \
                  + count_change(amt, denom[1:])
```

A. Ben's code looks a little different from what we used in class. State the function call with the appropriate inputs to obtain the number of ways to change 1 dollar (100 cents) using 50-cent, 25-cent, 10-cent and 5-cent coin denominations. [2 marks]

B. What is the order of growth in terms of time and space for `count_change`. [2 marks]

C. `count_change` runs too slowly for Ben. He notices that it does a lot of repeated operations. "If only I have some way to remember and reuse past computations," Ben thinks. Then he remembered you have given him some database ADT in the past, and wrote the following code:

```
def create_memo_cc():
    db = create_database()
    def memo_cc(amt, denom):
        if amt < 0 or len(denom) == 0:
            return 0
        elif amt == 0:
            return 1
        elif contains(db, amt, denom):
            return get(db, amt, denom)
        else:
            ans = count_change(amt-denom[0], denom) \
                     + count_change(amt, denom[1:])
            put(db, ans, amt, denom)
            return ans
    return memo_cc
```

Assume the functions provided (highlighted above) works correctly. Unfortunately, the code still runs slowly like before. Describe the problem with Ben's code and how it can be fixed. [3 marks]

D. Provide an implementation which uses a dictionary, for the functions `create_database`, `put`, `get` and `contains` to complete Ben's code. [4 marks]

E. We now use `create_memo_cc` to create a new `count_change` function as follows:

```
>>> count_change = create_memo_cc()
```

What is the order of growth in terms of time and space for this new version of `count_change` in the long term, i.e. after we run `count_change` for *m* times, where *m* is a very large number and for very large values of *amt* while keeping the same coin denominations? [2 marks]

F. Ben realizes that he does not have an infinite number of coins. So he decided to tweak his code to take in a dictionary of available coins. For example, `{50: 2, 25: 2, 10: 2, 5: 2}` means he has two coins of each denomination.

Somehow, he is not able to speed up his new function with the same technique. Suggest a reason why this is so. [2 marks]

Question 4: Airborne Troopers [8 marks]

Consider the following classes:

```
class Trooper:
    def __init__(self, *equipment):
        self.equipment = list(equipment)

    def load(self):
        return sum(map(lambda eq: eq[1], self.equipment))

    def speed(self):
        return 20 - (self.load() // 10)


class JumpJet(Trooper):
    def __init__(self, *equipment):
        super().__init__(('Jetpack', 20), *equipment)
        self.flying = False

    def toggle_jet(self):
        if not self.flying and self.load() > 100:
            print('Too heavy to fly')
        else:
            self.flying = not self.flying

    def speed(self):
        return super().speed() * (10 if self.flying else 1)


class Medic(Trooper):
    ## Incomplete. To be answered in Part B ##
```

A. What is the output of executing the following statements:

```
>>> boba = JumpJet(('Rocket Launcher', 30))

>>> print(boba.speed())
>>> boba.toggle_jet()
>>> print(boba.speed())
```

[3 marks]

B. Your friend Jango wishes to create a `JumpJetMedic` which is both a `JumpJet` and a `Medic` and writes the following code:

```
class JumpJetMedic(JumpJet, Medic):
    pass
```

The following sample execution shows the behaviour of a `JumpJetMedic`:

```
>>> kix = JumpJetMedic(('Medikit', 1))    # kix is a jumpjetting medic
>>> kix.equipment      # medic's equipment includes a stretcher by default
[('Stretcher', 10), ('Jetpack', 20), ('Medikit', 1)]

>>> kix.speed()
17
>>> kix.toggle_jet()
>>> kix.speed()        # kix jump jets to casualty
170
>>> kix.toggle_jet()

>>> rex = Tropper()
>>> rex.weight = 70    # rex weight is 70

>>> kix.evacuate(rex)  # kix now carries rex
>>> kix.load()         # total load increases by rex's weight
101

>>> kix.equipment      # but equipment remains unchanged
[('Stretcher', 10), ('Jetpack', 20), ('Medikit', 1)]

>>> kix.toggle_jet()   # cannot fly while carrying rex
Too heavy to fly
```

Being the lazy guy Jango is, he has left you to implement the class `Medic`. Using OOP principles, provide a minimal implementation of `Medic`, which is a subclass of `Trooper`, that satisfies the above sample execution. Redundant code or methods will be penalised.

[5 marks]

Question 5: 42 and the Meaning of Life [2 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. [2 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.

— H A P P Y H O L I D A Y S ! —

CS1010S — Programming Methodology
School of Computing
National University of Singapore

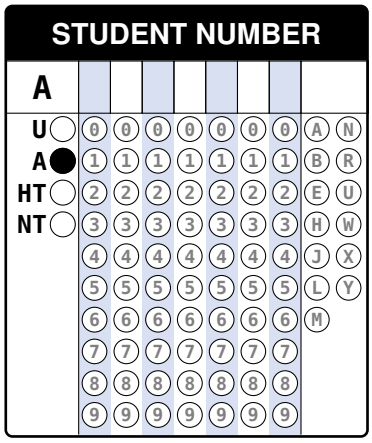
Final Assessment
Answer Sheet

2019/2020 Semester 2

Time allowed: 1 hour

Instructions (please read carefully):

1. Write down your **student number** on the right and using ink or pencil, shade the corresponding circle in the grid for each digit or letter. DO NOT WRITE YOUR NAME!
2. This answer booklet comprises **TEN (10) 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	/ 10
Q2	/ 15
Q3	/ 15
Q4	/ 8
Q5	/ 2
Total	/ 50

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]

```
lst = [[1], [2, 2], [3, 3, 3]]

def f(lst):
    for i in lst.copy():
        if len(i) < 2:
            i.append(1)
        if sum(i) < 5:
            i.pop()
        else:
            lst.extend(i)
        print(lst)
    return lst

print(lst is f(lst))
print(lst)
```

Question 1B

[5 marks]

```
s = 'mississipi'
d = {}
while s:
    if s[0] not in d:
        d[s[0]] = s[1]
    else:
        d.pop(s[0])
    s = s[1:]
    print(d)
```

Question 2A

[5 marks]

```
def adjacent_mines(cases, i, j):
```

Question 2B

[5 marks]

```
def adjacent_grid(cases, row, col):
```

Question 2C

[5 marks]

```
def safe_paths(grid):
```

Question 3A

[2 marks]

```
count_change(100, (50, 25, 10, 5))
```

Question 3B

[2 marks]

Time:

Space:

Question 3C

[3 marks]

Question 3D

[4 marks]

```
def create_database():


def put(db, ans, *n):


def get(db, *n):


def contains(ab, *n):
```

Question 3E

[2 marks]

Time:

Space:

Question 3F

[2 marks]

Question 4A

[3 marks]

Question 4B

[5 marks]

Question 5 42 and the Meaning of Life [2 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]

```
lst = [[1], [2, 2], [3, 3, 3]]

def f(lst):
    for i in lst.copy():
        if len(i) < 2:
            i.append(1)
        if sum(i) < 5:
            i.pop()
        else:
            lst.extend(i)
        print(lst)
    return lst

print(lst is f(lst))
print(lst)
```

```
[[1], [2, 2], [3, 3, 3]]
[[1], [2], [3, 3, 3]]
[[1], [2], [3, 3, 3], 3, 3, 3]
True
[[1], [2], [3, 3, 3], 3, 3, 3]
```

Question 1B

[5 marks]

```
s = 'mississipi'
d = {}
while s:
    if s[0] not in d:
        d[s[0]] = s[1]
    else:
        d.pop(s[0])
    s = s[1:]
    print(d)
```

```
{'m': 'i'}
{'m': 'i', 'i': 's'}
{'m': 'i', 'i': 's', 's': 's'}
{'m': 'i', 'i': 's'}
{'m': 'i'}
{'m': 'i', 's': 's'}
{'m': 'i'}
{'m': 'i', 'i': 'p'}
{'m': 'i', 'i': 'p', 'p': 'i'}
{'m': 'i', 'p': 'i'}
```

Question 2A

[5 marks]

```
def adjacent_mines(cases, i, j):
    mines = []
    for row in range(i-1, i+2):
        for col in range(j-1, j+2):
            if (row, col) in cases:
                mines.append((row, col))
    return len(mines)
```

Question 2B

[5 marks]

```
def adjacent_grid(cases, row, col):
    grid = []
    for i in range(row):
        row = []
        for j in range(col):
            row.append(adjacent_mines(cases, i, j))
        grid.append(tuple(row))
    return tuple(grid)
```

Question 2C

[5 marks]

```
def safe_paths(grid):
    last_row = len(grid)-1
    last_col = len(grid[0])-1
    def helper(i, j):
        if i == last_row and j == last_col:
            return 1
        elif i > last_row or j > last_col or grid[i][j] != 0:
            return 0
        else:
            return helper(i+1, j) + helper(i, j+1)
    return helper(0, 0)
```

Question 3A

[2 marks]

```
count_change(100, (50, 25, 10, 5))
```

Question 3B

[2 marks]

Time: $O(2^{amt})$ will be accepted, though tighter bound would be $O(amt^{len(denom)})$.

Space:$O(amt)$

Question 3C

[3 marks]

`count_change` is called recursively instead of the new `memo_cc`. Modify the line to:

```
ans = memo_cc(amt-denom[0], denom) \
        + memo_cc(amt, denom[1:])
```

Question 3D

[4 marks]

```
def create_database():
    return {}


def put(db, ans, *n):
    db[n] = ans


def get(db, *n):
    return db[n]


def contains(ab, *n):
    return n in db
```

Question 3E

[2 marks]

Time: $O(1)$

Space:$O(amt)$

To check that student understands that after a while all the values will be cached and it will just be a $O(1)$ look up. Space is linear to the inputs as the dictionary will contain previous results.

Question 3F

[2 marks]

Now the number of coins of each denomination is part of the input, and it changes each computation step. Thus the is far fewer repeated computations with identical inputs in the computation step. So there is little speed up as the result cannot just be obtained from a previous run.

Question 4A

[3 marks]

```
15
150
```

Question 4B

[5 marks]

```
class Medic(Trooper):
    def __init__(self, *equipment):
        super().__init__(('Stretcher', 10), *equipment)
        self.carry = None

    def evacuate(self, person):
        self.carry = person

    def load(self):
        return super().load() + (self.carry.weight if self.carry else 0)
```

Question 5 42 and the Meaning of Life [2 marks]

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

This question is designed for students who get stuck with the other questions. They can at least spend some time writing an essay for marks. Points will be awarded for effort.

Make the prof laugh out loud and you get 4 points.