NATIONAL UNIVERSITY OF SINGAPORE

Department of Computer Science, School of Computing

IT5001—Software Development Fundamentals

SELF-DIAGNOSTIC ASSESSMENT

Question Booklet

Time allowed: 2 hours

PREAMBLE

This is a self-diagnostic assessment of your exposure and mastery of basic computing methodologies to help you determine whether your programming competency is sufficient for replacing IT5001 with another core course.

This test should take about two hours, self-timed. We suggest that you to take this self-diagnostic assessment as a mock assessment, only after you have sufficient preparation.

Note that the format and structure of the actual proficiency test may be different to this assessment.

Please read the following instructions carefully before attempting this self-diagnostic assessment.

INSTRUCTIONS TO CANDIDATES (*please read carefully*):

1. This question booklet comprises **31 questions** and **14 printed pages** including the cover page. All questions must be answered correctly to obtain the full score.
2. This is a **closed-book** assessment. You are **not** allowed to refer to any materials and access any resources (online or otherwise), except one A4-sized double-sided reference sheet, written or printed. You may also use one blank A4-sized paper for your rough work.
3. **Mobile phones and smartwatches should be turned off.**
4. You are **not allowed** to use any electronic devices other than a non-programmable calculator.
5. The total attainable score is **100**. The assessment constitutes **100%** of your overall grade.
6. You **cannot** communicate with anyone other than the invigilators throughout the assessment.
7. **You must attempt the assessment on your own**. The University takes a zero-tolerance approach towards plagiarism and cheating.

EXPRESSION EVALUATION [15 marks]

There are several questions in this section. Answer each question **independently and separately**. In each question, a complete program stored in a `.py` file is given to you, assuming there are no other imports. Determine the output shown in the console upon program execution.

Question 1 [1 mark].

```
print(max(-1, 2, -3))
```

(A) `-1`
(B) `2`
(C) `-3`
(D) `None`
(E) Evaluating this expression yields an error

Question 2 [1 mark].

```
print(1 + 6 // 4)
```

(A) `2`
(B) `2.0`
(C) `2.5`
(D) `None`
(E) Evaluating this expression yields an error

Question 3 [1 mark].

```
print('abcde'[-1:7])
```

(A) A blank line is shown in the console
(B) `e`
(C) `edcba`
(D) `None`
(E) Evaluating this expression yields an error

Question 4 [1 mark].

```
print(bool('False'))
```

(A) `True`
(B) `False`
(C) `None`
(D) Evaluating this expression yields an error
(E) None of the above

Question 5 [1 mark].

```
print('' in 'abc')
```

(A) `True`
(B) `False`
(C) `0`
(D) `None`
(E) Evaluating this expression yields an error

Question 6 [1 mark].

```
print([1, 3] * 2)
```

(A) `[2, 6]`
(B) `[1, 3, 2]`
(C) `[1, 3, 1, 3]`
(D) `None`
(E) Evaluating this expression yields an error

Question 7 [1 mark].

```
print((1, [2], 3)[1])
```

(A) `[1]`
(B) `[2]`
(C) `[3]`
(D) 2
(E) Evaluating this expression yields an error

Question 8 [1 mark].

```
print([[1, 2], [3, 4, 5]][1][2])
```

(A) 5
(B) `[1, 2]`
(C) `[3, 4, 5]`
(D) None
(E) Evaluating this expression yields an error

Question 9 [1 mark].

```
print(sorted('abracadabra')[:3])
```

(A) aaa
(B) `['a', 'a', 'a']`
(C) `['a', 'b', 'r']`
(D) `['abracadabra']`
(E) Evaluating this expression yields an error

Question 10 [1 mark].

```
print({1: 2, 2: 1}[1] + {3: 4, 0: 1}[0])
```

(A) 3
(B) 4
(C) 5
(D) 6
(E) Evaluating this expression yields an error

Question 11 [1 mark].

```
print({1: 2, 3: 4}.get(2))
```

(A) 1
(B) 2
(C) 3
(D) None
(E) Evaluating this expression yields an error

Question 12 [1 mark].

```
print({1: {2: {3: 4}}}[{1: 2, 3: 4}[1]])
```

(A) 1
(B) `{3: 4}`
(C) `{2: {3: 4}}`
(D) None
(E) Evaluating this expression yields an error

Question 13 [1 mark].

```
print([i - 1 for i in [1, 2]])
```

(A) `[0, 1]`
(B) `[1, 2]`
(C) `[]`
(D) None
(E) Evaluating this expression yields an error

Question 14 [1 mark].

```
print([i for i in [0, 1, 2] if i - 1])
```

(A) `[0, 2]`
(B) `[1, 2]`
(C) `[-1, 1]`
(D) `[]`
(E) Evaluating this expression yields an error

Question 15 [1 mark].

```
a = map(int, '123')
print(max(a) + min(a))
```

(A) `31`
(B) `4`
(C) `[3, 1]`
(D) `0`
(E) Evaluating this expression yields an error

Proceed to the next page...

MULTIPLE STATEMENT QUESTIONS [15 marks]

There are several questions in this section. Answer each question independently and separately.

In each of these questions you are given a statement and offered several options. For each of these, choose the most appropriate option.

Question 16 [3 marks]. Which of the following statements is true of lists and tuples?

(A) Lists are immutable (cannot be mutated), tuples are mutable (can be mutated)

(B) Where `tp` is a tuple, an operation like `tp += (1,)` implicitly calls the `extend` method of tuples

(C) `len(([1, 2, (3, 4)],))` evaluates to `4` because the length of a list/tuple is the sum of the lengths of its elements (the length of non-sequence objects like integers is `1`)

(D) A list concatenated with a tuple gives a list

(E) None of the above

Question 17 [3 marks]. Observe the following code snippet and some remarks about it:

```
1 a = int(input())
2 cond = a == 1
3 if cond == True:
4   print(1)
5 else:
6   print('not 1')
```

1. Line 3 can be replaced with `if cond:` and the code snippet would behave identically
2. Upon execution of this snippet, a `TypeError` will be raised from line 4 because the `print` function accepts strings only
3. Upon execution of this snippet, a `ValueError` will be raised from line 1 if the user enters a floating point number into the console

Which of the remarks is/are true?

(A) Only 1 is correct

(B) Only 2 is correct

(C) Only 3 is correct

(D) 1 and 2 are correct, but not 3

(E) 1 and 3 are correct, but not 2

Question 18 [3 marks]. Observe the following memoized implementation of the fibonacci function:

```
memo = {}
def fib(n):
  if n == 0 or n == 1:
    return 1
  if n in memo:
    return memo[n]
  z = fib(n - 1) + fib(n - 2)
  memo[n] = z
  return z
```

Which of the following statements is true of `fib`?

(A) Memoizing `fib` using a list is just as (if not more) efficient than memoizing it using a dictionary

(B) Evaluating `fib(-1)` creates an infinite loop

(C) No implementation of `fib` using loops can outperform this recursive implementation by any metric

(D) Memoizing `fib` this way actually makes `fib` run slower than without the memoization

(E) None of the above

Question 19 [3 marks]. Observe the `Duck` class:

```
class Duck:
  def __init__(self):
    self.sound = 'quack'
```

Which of the following statements is true of `Duck`?

(A) Instances of `Duck` are actually dictionaries, therefore `Duck()['sound']` evaluates to `'quack'`

(B) If we define a class `BrownDuck` that inherits `Duck`, then all instances of `BrownDuck` will also have an attribute called `sound`

(C) The expression `Duck().sound` will always evaluate to `'quack'`

(D) Because `Duck` does not define the `__str__` method, the evaluating the expression `str(Duck())` will raise an exception

(E) None of the above

Question 20 [3 marks]. Which of the following is true of exceptions?

(A) Raising exceptions can be useful for detecting errors

(B) Developers should never intentionally raise exceptions because it could crash a program

(C) `finally` clauses in a `try`-`except` block are never needed

(D) Developers should never write `try`-`except` blocks because they hide exceptions

(E) None of the above

Proceed to the next page...

PROGRAM TRACING [20 marks]

There are several questions in this section. Answer each question independently and separately. In each of the following questions in this section, you are given a complete Python program stored in a `.py` file. Determine the output (if any) of the program upon execution, and choose the most appropriate option.

Question 21 [4 marks].

```
def f21(n):
  if n > 5: return 5
  if n > 3: return 3
  if n > 1: return 1
  return 0
print(f21(2))
```

(A) `0`
(B) `1`
(C) `3`
(D) `5`
(E) None of the above

Question 22 [4 marks].

```
def f22(seq):
  if isinstance(seq, str):
    return seq
  return ''.join([f22(i) for i in seq])
print(f22(['a', ['b', ['c']], [['d']]]))
```

(A) `['a', ['b', ['c']], [['d']]]`
(B) `['d', 'c', 'b', 'a']`
(C) abcd
(D) The program does not terminate
(E) None of the above

Question 23 [4 marks].

```
def f23(d):
  acc = {}
  for k, v in d.items():
    if v not in acc:
      acc[v] = []
    acc[v].append(k)
  return acc
print(f23({1: 2, 2: 3, 3: 2, 4: 2, 5: 4}))
```

(A) `[1, 2, 3, 4, 5]`
(B) `{2: 4, 3: 2, 4: 5}`
(C) `{2: [1, 3, 4], 3: [2], 4: [5]}`
(D) The program does not terminate
(E) None of the above

Question 24 [4 marks].

```
f = lambda y: lambda x: x[-y]
ls = [['i', 'am'], ['an', 'SoC'], [], ['student']]
_ = map(f, range(1, 5))
_ = map(lambda f: f(ls), _)
_ = map(' '.join, _)
_ = filter(bool, _)
res = ' '.join(_)
print(res)
```

(A) `student an SoC i am`

(B) `i am an SoC student`

(C) `i am an SoC  student`

(D) `student SoCanami`

(E) None of the above

Question 25 [4 marks].

```
class Entity:
  def reset(self):
      self.uuid = 0

class Named(Entity):
  def reset(self):
    self.name = ''
    super().reset()

class WithEmail(Entity):
  def reset(self):
    self.email = ''
    super().reset()

class User(Named, WithEmail):
  def __init__(self):
    self.name = 'Bob'
    self.email = 'bob@gmail.com'
    self.uuid = 123
  def reset(self):
    super().reset()

bob = User()
bob.reset()
print([bob.uuid, bob.name, bob.email])
```

(A) `[0, '', 'bob@gmail.com']`

(B) `[123, '', 'bob@gmail.com']`

(C) `[0, '', '']`

(D) A `TypeError` is raised

(E) None of the above

Proceed to the next page...

PROGRAMMING [50 marks]

There are several questions in this section. Answer each question **independently** and **separately**. Each question poses a programming problem to solve, and provides a program template with blanks. Your task for each question is to replace each blank with a valid Python expression/statement so that your solution solves the problem correctly.

Question 26 [5 marks]. There are n cities numbered $0, 1, \ldots, n-1$, arranged in a circle where for each i, city i is connected to cities $i-1$ and $i+1$ (cities 0 and $n-1$ are also connected to each other). You are also given a list A consisting of n nonnegative integers, indicating that the cost of entering city i is $\$A[i]$.

The `cheapest` function receives list A and two cities x and y such that $0 \leq x, y < |A|$ ($|A|$ is the length of A), and as a result returns the least amount of money needed to travel from city x to y (the cost of x is not included in the total cost). Example runs and an incomplete implementation of `cheapest` are given below. Replace each blank with a valid Python expression/statement so that your solution solves the problem correctly.

Example runs:

```
>>> A = [5, 4, 3, 4, 5]
>>> cheapest(A, 1, 4)
10 # 1 -> 0 -> 4
>>> cheapest(A, 0, 2)
7 # 0 -> 1 -> 2
>>> cheapest(A, 3, 3)
0 # 3
```

Template:

```
def cheapest(A, x, y):
    a = __blank_1__
    b = __blank_2__
    return min(a, b)
```

Question 27 [8 marks]. Your task is to write a function `deep_dup(seq)` to duplicate all the singleton items (in this case, the integers) of an arbitrarily deeply-nested list. You may assume that the input `seq` is a list that will not contain anything other than integers and/or lists. Example runs and an incomplete implementation of `deep_dup` is given below. Replace each blank with a valid Python expression/statement so that your solution solves the problem correctly.

Example runs:

```
>>> deep_dup([1, [6, [9]], 8, 4])
[1, 1, [6, 6, [9, 9]], 8, 8, 4, 4]
>>> deep_dup([[[[8]]], 1, [[9]], 4, 5])
[[[[8, 8]]], 1, 1, [[9, 9]], 4, 4, 5, 5]
```

Template:

```
def deep_dup(seq):
    if seq == []:
        return seq
    if __blank_3__:
        return __blank_4__
    return __blank_5__
```

Question 28 [15 marks]. The following three sub-questions pose the same problem, but their solutions are to be written in a different programming style.

Question 28 (i) [5 marks]. Your task is to define a function `weighted_sum_i(num_str, weight)`, which receives a **string** `num_str` with length n greater than or equal to 0 and consisting only of digits, and a **list of integers** `weight` also with length n, and produces the weighted sum of the digits in `num_str` with weights `weight` as an **integer**. You may assume that `num_str` and `weight` will always have equal lengths, and that `num_str` will only contain digits. **Your function must use loops and cannot be recursive**. Example runs and an incomplete implementation are given below. Replace each blank with a valid Python expression/statement so that `weighted_sum_i` runs correctly.

Example runs:

```
>>> weight = [2, 7, 6, 5, 4, 3, 2]
>>> weighted_sum_i('1234567', weight)
106
>>> weighted_sum_i('1111111', weight)
29
>>> weighted_sum_i('222', [1, 2, 3])
12
```

Template:

```
def weighted_sum_i(num_str, weight):
    output = 0
    for i in range(len(num_str)):
        output += __blank_6__
    return output
```

Question 28 (ii) [5 marks]. Now define the function `weighted_sum_r` that behaves identically to `weighted_sum_i` except that **it is defined recursively without containing loops or any form of for-comprehension**. Your implementation cannot invoke `weighted_sum_i` or `weighted_sum_1`. An incomplete implementation of `weighted_sum_r` is given below. Replace each blank with a valid Python expression/statement so that `weighted_sum_r` runs correctly.

```
def weighted_sum_r(num_str, weight):
    if __blank_7__:
        return 0
    return __blank_8__
```

Question 28 (iii) [5 marks]. Now define the function `weighted_sum_1` that behaves identically to `weighted_sum_i` except that **it is defined with a single return statement**. Your implementation cannot invoke `weighted_sum_i` or `weighted_sum_r`. An incomplete implementation of `weighted_sum_1` is given below. Replace each blank with a valid Python expression/statement so that `weighted_sum_1` runs correctly.

```
def weighted_sum_1(num_str, weight):
    return __blank_9__
```

Proceed to the next page...

Question 29 [6 marks]. Your task is to write a function `create_guess_game(n)` that takes in an integer `n` and returns a *function* that will play the guessing game. The returned function will take in another integer `x` and thereafter returns

- `'bingo'` if `x` is equal to `n`
- `'too big'` if `x` is greater than `n`
- `'too small'` if `x` is smaller than `n`

Example runs and an incomplete implementation of `create_guess_game` follow. Replace the blank with a valid Python expression so that `create_guess_game` works correctly.

Example runs:

```
>>> guess = create_guess_game(50)
>>> guess(61)
'too big'
>>> guess(4)
'too small'
>>> guess(50)
'bingo'
```

Template:

```
def create_guess_game(n):
    return __blank_10__
```

Proceed to the next page...

Question 30 [8 marks]. You are given a dictionary representing parent-child relationships. The keys of the dictionary are parents, and their corresponding values are lists of their immediate children. For example, given the following dictionary:

```
des = {
    'Peter': ['May', 'Carl'],
    'May': ['Frank'],
    'Bob': ['Kelvin', 'Staut'],
    'Frank': ['Emily', 'John']
}
```

we have that May and Carl are Peter's children, Emily and John are Frank's children, and so on. Each parent can have any (nonnegative) number of children.

From this we further define x to be a **descendant of** y if

- x is equal to y; or
- x is a child of some z such that z is a **descendant of** y

By this definition, from `des` above we have that May is a descendant of May, Frank is also a descendant of May because Frank is May's child, and Emily and John are also descendants of May because they are Frank's children.

Your task is to write a function `all_descendants(name, dd)` which receives the name of a person `name` and a descendant dictionary `dd` (like `des` above) and returns a list of all descendants of `name`. The order of the items in your output list is not important. In addition, assume the dictionary `dd` will not produce cyclic descendance, i.e. we will not have the case where Alice is Bob's child and Bob is also Alice's child, for example. (Formally, `dd` will always be such that if x is a **descendant of** y and y is a **descendant of** x, then x is necessarily equal to y; the **descendant of** relationship is guaranteed to be *antisymmetric*.)

Example runs are given below, assuming `des` is as defined above. Replace the blanks in the following incomplete implementation with valid Python expressions/statements so that `all_descendants` works correctly.

Example Runs:

```
>>> all_descendants('May', des)
['May','Frank','Emily','John']
>>> all_descendants('Peter', des)
['Peter','May','Frank','Emily','John','Carl']
>>> all_descendants('Bob', des)
['Bob', 'Kelvin', 'Staut']
```

Template:

```
def all_descendants(name, dd):
    output = __blank_11__
    if __blank_12__:
        return __blank_13__
    for des in __blank_14__:
        __blank_15__
    return output
```

Proceed to the next page...

Question 31 [8 marks]. The following two sub-questions describe the same problem, but their solutions are to be written in a different programming style.

Question 31 (i) [4 marks]. There are $r \times c$ houses a city. Each house has a value and the data is given in an $r \times c$ two-dimensional list of integers. Your task is to survey and compute the total values in a rectangular subarea of the city. For a city with r rows and c columns, you will be given four parameters `r_start`, `r_end`, `c_start` and `c_end` (all within valid bounds of the list). Write a function `sum_2D(m, r_start, r_end, c_start, c_end)` to return the sum of all entries in the list with the row = `i` and column = `j` such that `r_start <= i < r_end` and `c_start <= j < c_end`.

Here are some example output:

```
>> from pprint import pprint
>>> m1 = [[(i + j) % 2 for j in range(12)] for i in range(6)]
>>> pprint(m1)
[[0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
 [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0],
 [0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
 [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0],
 [0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1],
 [1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0]]
>>> print(sum_2D(m1, 1, 3, 0, 6))
6
```

An incomplete implementation of `sum_2D` is given below. Replace each blank with a valid Python expression/statement such that `sum_2D` works correctly.

```
def sum_2D(m, r_start, r_end, c_start, c_end):
    output = 0
    for i in range(__blank_16__):
        for j in range(__blank_17__):
            output += __blank_18__
    return output
```

Question 31 (ii) [4 marks]. Re-write the `sum_2D` function by implementing it with a single **`return`** statement. The template is given below.

```
def sum_2D(m, r_start, r_end, c_start, c_end):
    return __blank_19__
```

– End of Self-Diagnostic Assessment –