Name:

Student ID: Cohort:

10.009 The Digital World
Term 3. 2018.
Final Exam
April 26, 2018 (Thursday)
Actual Duration: 9:00am- 11:00am (2 hours)
Room: Cohort Classroom
Most recent update: April 17, 2018

- Write your name and student ID at the top of this page.
- This exam has two parts. You can secure a maximum of 100 points.
- For Part A question 1, write your answers **on this exam paper**.
- For Part A question 2, write your answers **on the bubble sheet given**.
- For Part B (questions 3-7), submit your solutions to **Vocareum**. Template files may be provided in Vocareum.
- You may consult all material on your laptop and in your notes. You may also use any paper-based materials(s) as a reference.
- **You are not allowed to use any Internet-accessing or communicating device during the exam.**
- **You are not allowed to consult anyone inside or outside of the classroom other than the proctors in the examination room.**
- Use your desktop IDE to test your programs. Enter your program into Vocareum regularly so that Vocareum can save your work. You have unlimited number of submissions, so make sure you submit early to Vocareum.
- All answers will be graded manually. You may be able to earn partial credit for questions.
- Good luck!

Summary:

Category	Description	Number of Problems	Total Points
Part A	Written questions	2 (Questions 1-2)	20
Part B	Programming questions	6 (Questions 3-7)	80

Q1	
SubTotal	

SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN
HONOUR CODE

"As a member of the SUTD community, I pledge to always uphold honourable conduct. I will be accountable for my words and actions, and be respectful to those around me."

Introduction to the SUTD Honour Code

What is the SUTD Honour Code?

The SUTD Honour Code was established in conjunction with the school's values and beliefs, in good faith that students are able to discern right from wrong, and to uphold honourable conduct. It is an agreement of trust between the students and the staff and faculty of SUTD, and serves as a moral compass for students to align themselves to. Being in a university that aspires to nurture the leaders of tomorrow, it calls for students to behave honourably, not just solely in their academic endeavours, but also in everyday life.

What the Honour Code encompasses

Integrity & Accountability

To be honourable is to do what is right even when nobody is watching, and to be accountable for the things one does. One should always be accountable for one's words and actions, ensuring that they do not cause harm to others. Putting oneself in a favourable position at the expense of others is a compromise of integrity. We seek to create a community whereby we succeed together, and not at the expense of one another.

Respect

Part of being honourable is also respecting the beliefs and feelings of others, and to never demean or insult them. Should conflicts arise, the aim should always be to settle them in a manner that is non-confrontational, and try to reach a compromise. We will meet people of differing beliefs, backgrounds, opinions, and working styles. Understand that nobody is perfect, learn to accept others for who they are, and learn to appreciate diversity.

Community Responsibility

In addition to that, being honourable also involves showing care and concern for the community. Every individual has a duty to uphold honourable conduct, and to ensure that others in the community do likewise. The actions of others that display immoral or unethical conduct should not be condoned nor ignored. We should encourage each other to behave honourably, so as to build a community where we can trust one another to do what is right.

Student's signature

[this page is left blank]

Part A

Q.1 [10 points]
Study the following code that carries out linear regression on the diabetes dataset. All line numbers mentioned in this question refer to the code below.

```
1   import numpy as np
2   from sklearn import datasets, linear_model
3   from sklearn.model_selection import train_test_split
4   from sklearn.metrics import mean_squared_error
5
6   bunchobject = datasets.load_diabetes()
7   data = bunchobject.data
8
9   x_index = 2
10  x = data[:,np.newaxis, x_index]
11  y = bunchobject.target
12
13  x_train, x_test, y_train, y_test = train_test_split( x , y , test_size =
14  0.4, random_state = 10009 )
15
16  regr = linear_model.LinearRegression()
17  a = regr.fit(x_train, y_train)
18  y_pred = regr.predict(x_test)
19
20  mse = mean_squared_error(y_pred, y_test)
21
22  results = {'coefficients': regr.coef_ ,
23             'intercept': regr.intercept_,
24             'mean squared error': mse}
25
26  print(results)
```

Note: This code with simple explanations is also available in eDimension solely for your convenience. It is available both as an iPython notebook and a python script file. If any discrepancy is spotted, the code printed on this page takes precedence.

a. Answer the following questions. Write your answer on this paper, in the space below. You may need to write python code to obtain these answers, but in this question, you do not need to provide any python code in your answer. (2 points)

   i. Counting from 0, features 4 to 9 in the dataset are named as `s1` to `s6`. These features come from a certain type of measurement taken on diabetes patients.

      State the type of measurement taken on patients that are given in features `s1` to `s6`.

      Hint: This information can be found in the dataset description.

   ii. State the data type of the target variable (numerical / categorical / text) and justify your answer.

b. In this part, some tasks are given. You have to add new lines of code or modify existing code to the code given in this question. **In the space below**, write the necessary python code to accomplish the tasks and state the line(s) where this modification or addition has to take place. (4 points)

   i. Calculate the median body mass index and display it on the screen.
   ii. Include the body mass index, average blood pressure and s1, in this particular order, as independent variables in the linear regression.

c. Explain why it is necessary to calculate the mean-squared error at Line 20. A brief description comprising of two to three sentences will be sufficient. (2 points)

d. After the code is modified in part b(ii), the output at Line 26 is as follows.

```
{'coefficients': array([ 815.74078727,  340.98099507,  123.02175425]),
'intercept': 152.6459746994903, 'mean squared error': 3362.486106934875
8}
```

   Write the mathematical formula that represents the linear regression model built by this code. (2 points)

Your Answer:

Your Answer (Continued):

[this page is left blank]

Q.2 [10 points]

For all the following questions, please select a single answer from the given choices.

Write your answer on the special sheet given for Multiple Choice Questions.

Please look at the program below to answer questions (1), (2), and (3):

```
class Number():

    '''Method accepts a tuple of two numbers as "values" '''
    def __init__(self, values):
         self.values = values
         self.real = values[0]
         self.imaginary = values[1]

    def __str__(self):
        return "The number is: " + str(self.values)


class FloatingPointNumber(Number):

    '''Method accepts a FloatingPointNumber object as "compare_to" '''
    def __eq__(self,compare_to):
       self.epsilon = 0.1
       if abs(self.real-compare_to.real) < self.epsilon:
           return True
       else:
           return False


class IntegerNumber(FloatingPointNumber):

    '''Method accepts an IntegerNumber object as "compare_to" '''
    def __eq__(self,compare_to):
       if self.real == compare_to.real:
           return True
       else:
           return False

class ComplexNumber(Number):

    '''Method accepts a ComplexNumber object as "compare_to" '''
    def __eq__(self,compare_to):
       if self.real == compare_to.real and self.imaginary ==
compare_to.imaginary:
           return True
       else:
           return False
```

(1) (bubble sheet **q1**) For print(float1 == float2) to display True, what should a and b be? (2 points)

```
float1 = FloatingPointNumber((a,0))
float2 = FloatingPointNumber((b,0))
```

```
I.    a = 1.03
      b = 1.12
II.   a = 1.09
      b = 1.01
III.  a = 1.12
      b = 1.01
```

a. II only
b. I and II only
c. III only
d. I, II, and III

(2) (bubble sheet **q2**) In the context of the code given above, which of the following statements best describes the relation between `IntegerNumber` and `FloatingPointNumber`? (2 points)

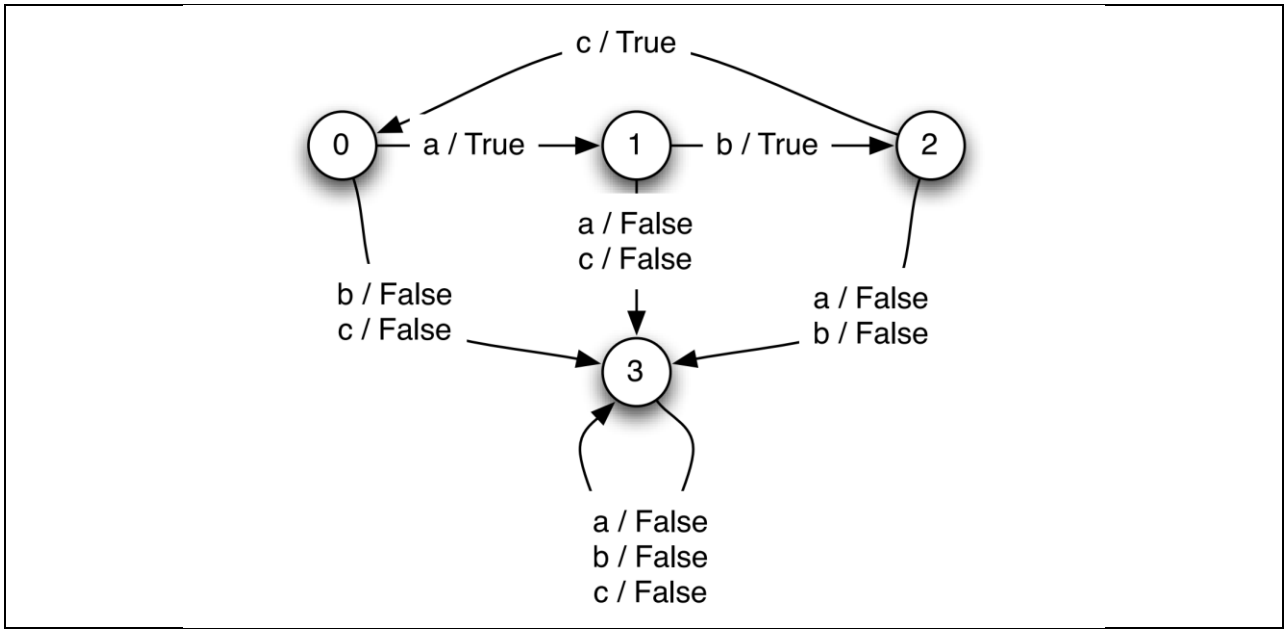
a. `FloatingPointNumber` is a subclass of `IntegerNumber`
b. `Number` is a subclass of `FloatingPointNumber`
c. `IntegerNumber` is a subclass of `FloatingPointNumber`
d. `ComplexNumber` inherits from `IntegerNumber`

(3) (bubble sheet **q3**) In the `ComplexNumber` class, the `__str__(self)` method of the `Number` class is: (2 points)

a. Overridden
b. Overloaded
c. Inherited
d. Not defined

[The question continues on the next page.]

Please look at the diagram below to answer question (4):



(4) (bubble sheet **q4**) What will be the state and output at the end of processing the input string: ('a', 'b', 'c', 'a', 'c', 'a', 'b')? The starting state is state 0. (2 points)

a. State = 3, Output = False
b. State = 3, Output = True
c. State = 2, Output = False
d. State = 2, Output = True

Please look at the code below to answer question (5):

```
from libdw import sm

class MySM(sm.SM):
    def __init__(self, start_state):
        self.start_state=start_state

    def get_next_values(self, current_state, input):
            next_state = current_state + input
            output = current_state + input
            return next_state, output

my_sm = MySM(100)
my_sm.start()
print(my_sm.transduce([10, 20 ,30, 60, zz ]))
```

(5) (bubble sheet **q5**) When transduce is completed, the very last output given by the state machine is 300. What is `zz`? (2 points)

a. zz = 300
b. zz = 180
c. zz = 120
d. zz = 80

Part B

Q.3 [10 points]

Book-keeping: In book-keeping, one form of asset is **Accounts Receivable** and one form of liability is **Accounts Payable**.

- **Accounts receivable** is money owed to your company by your clients.
- **Accounts payable** is money that your company owes to your suppliers.

The **Net equity** = **Total Accounts Receivable** – **Total Accounts Payable**. Write a function `equity(f)` that:

- Takes as input a dictionary `f` that models the accounting journal. The keys are the company names and the corresponding values are a list with two values. The zeroth value is the accounts receivable and the first value is the accounts payable. For example, given the following dictionary:

  {'Y': [100,20],'Z': [0,90]}

  It says that with company Y, the accounts receivable is 100 units and the accounts payable is 20 units. With company Z, there is no accounts receivable and the accounts payable is 90 units.
- Uses the information in the dictionary returned by ledger to calculate the net equity
- Returns a tuple with total receivable, total payable and net equity (see test code below)

Test Code:	**Output:**

`f1={'A': [100, 0], 'B': [100, 0], 'C': [100, 0]}`
`f2={'M': [30, 20], 'N': [50, 70], 'O': [60, 80]}`
`f3={'J': [0, 30], 'K': [0, 20], 'L': [0, 40]}`
 	
`print(equity(f1))`
`print(equity(f2))`
`print(equity(f3))`
	`(300, 0, 300)`
	`(140, 170, -30)`
	`(0, 90, -90)`

Q.4 [15 points]

Write a function called `wordcount(s)` that takes as input, a string `s`, and returns as output, a list of counts of **distinct, unique words** that appear in each **line** contained in `s`.

A **line** is defined as a set of characters terminated by the newline character `'\n'`.
Examples:

- `s = 'Hello hello',s` contains one line
- `s = 'Hello \nWorld', s` contains two lines. The individual lines are:
  - `'Hello '`
  - `'World'`
- `s = 'Hello \n\nWorld',s` contains three lines. The individual lines are:
  - `'Hello '`
  - `''` (empty string)
  - `'World'`

The input string is made up of **ASCII** characters, and valid words contain only the letters of the English alphabet. Two words are distinct and unique only if they are the same word, case included. The words need not mean anything. If a line contains no words return `None` in place of the word count for that line.

Examples:

- For `s = 'As as'`, the word count is `2`
- For `s = 'as as '`, the word count is `1`.
- For `s = 'abcde ggg '`, the word count is `2`.
- For `s = ''`, the word count is `None`.

Punctuation characters from the set (`':', ';', ',', '.'` and `'-'`) may appear in the string and **are to be ignored**. For example, the string "hello; hello- hello." will give the same count as the string "hello hello hello", which is one (1) unique word. You may assume that words are separated by white space, even if there is punctuation in between them. Thus substrings of the type `'abc;def'` will not be present in the input.

Test Code:	**Output:**
`print(wordcount('Tom Dick Harry'))`	`[3]`
`print(wordcount("Hello world\nHello Hello"))`	`[2, 1]`
`print(wordcount ("Hello  hello"))`	`[2]`
`print(wordcount ("Hello  World"))`	`[2]`
`print(wordcount ("Hello, Hello World"))`	`[2]`
`print(wordcount ("Hello \nWorld"))`	`[1,1]`
`print(wordcount ("Hello \nWorld\n"))`	`[1,1,None]`
`print(wordcount ("Hello \n\nWorld"))`	`[1, None, 1]`

`print(wordcount ("asdasf \n \n \n\nasdasfd;;`
`asfdasd\n Hello hello hello hello world world`
`world"))`
	`[1, None, None, None,`
	`2, 3]`

`print(wordcount ("Hello, world\nHello. Hello\n."))`	`[2,1,None]`

Q.5: Objects and Classes [Total: 25 points]

You are required to define a class named `Person`, which will record some basic data about individuals.

Your task is divided into several sub-tasks (A-D) of increasing difficulty. You should complete Task A and Task B first, but Tasks C and D can be attempted in either order.

Note that your submission to Vocareum should consist of one Person class only, regardless of how many of the tasks you manage to complete. In other words, if you manage to complete only Task A, B and C, submit only one Person class that meets the requirements of these tasks. Do NOT provide a different class for each sub-task.

Task A (6 points): Implement an `__init__` constructor method for initialising the attributes of your `Person` class. Initially, these should consist of:

- `name`, which stores the Person's name (a string);
- `age`, which stores the Person's age (an integer);
- `contact_details`, which stores a dictionary of contact data.

The default value of `name` should be `'Unknown'`; the default value of `age` should be `0`; and the default value of `contact_details` should be a new dictionary with the keys/values as shown below:

```
{'phone':'+65 0000 0000', 'email':'nobody@nowhere.com.sg'}
```

Test code (Task A):	**Output:**

Test Case #A-1
`p = Person()`
`print(p.name, p.age)`
`print(p.contact_details)`
	`Unknown 0`
	`{'phone': '+65 0000 0000', 'email': 'nobody@nowhere.com.sg'}`

Test Case #A-2
`d = {'phone': '+65 8888 8888',`
`  'email': 'chicken@floss.com.sg'}`
`p = Person('Charlie the Chicken', 88, d)`
`print(p.name, p.age)`
`print(p.contact_details)`
	`Charlie the Chicken 88`
	`{'phone': '+65 8888 8888', 'email': 'chicken@floss.com.sg'}`

Task B (7 points): replace the `name` and `age` attributes of `Person` with **properties**, along with their associated setter and getter methods.

The setter method for the property `name` should only update the name if the input is of type string, and is of length one or more.

The setter method for the property `age` should only update the age if the input is of type integer, and is a natural number (i.e. a positive integer, including 0).

You can assume that the constructor (`__init__`) will only ever be provided with valid inputs.

Hint #1: the setters/getters will need to work with new "private" attributes, e.g. _name, _age.

Hint #2: the Boolean expression `isinstance(inp, str)` *returns* `True` *if* `inp` *is of type* `str`*, and returns* `False` *otherwise.*

Test code (Task B):	**Output:**

Test Case #B-1
`p = Person()`
`p.name = ""`
`print(p.name)`
	`Unknown`

Test Case #B-2
`p = Person()`
`p.age = 88`
`p.age = 'this is, in fact, a string'`
`print(p.age)`
	`88`

[The question continues on the next page.]

Task C (7 points – Harder): create a property in `Person` called `email`.

The getter method for `email` should return the value of `contact_details['email']`.

The setter method for `email` should update the value of `contact_details['email']` with the given input, but ONLY if all of the following conditions are true of that input:

- it is a string;
- it contains **exactly one** "@" symbol;
- all characters before/after the "@" are either lower/uppercase letters, digits, ".", or "_";
- after the "@" symbol, it contains **at least one** "."

Test code (Task C):	**Output:**

Test Case #C-1
`p = Person()`
`print(p.email)`
`print(p.email ==`
`  p.contact_details['email'])`
	`nobody@nowhere.com.sg`
	`True`

Test Case #C-2
`p = Person()`
`p.email = 'chicken@floss.com.sg'`
`p.email = 'nasi.kukus'`
`p.email = 'ayam.goreng@berempah'`
`p.email = 'sedap!@makanan.com.sg'`
`print(p.email)`
	`chicken@floss.com.sg`

Task D (5 points – Challenging): extend your `Person` class with a property called `mother`. This property will be used to store/retrieve a reference to the person's mother (itself a `Person` object), or the special value `None` if they do not have one.

Your `__init__` method should initialise `mother` with a default value of `None`.

The getter for `mother` should return the stored value, i.e. an object reference or `None`.

The setter for `mother` should store some given input *m*, but ONLY if all of the following conditions are true:

- *m* is `None`, or *m* is a reference to an object of class `Person`;
- *m* is NOT the same object as the current one (`self`);
- the current object (`self`) is NOT an ancestor of *m* (i.e. `self` is not the mother of *m*, or their grandmother, or their great grandmother, or …)

Hint: you might find the `is` *operator useful for comparing objects.*

Test code (Task D):	**Output:**

Test Case #D-1
`p = Person()`
`print(p.mother)`
	`None`

Test Case #D-2
`p = Person()`
`p.mother = 'Mary Poppins'`
`print(p.mother)`
	`None`

Test Case #D-3
`p = Person()`
`p.mother = p`
`print(p.mother)`
	`None`

Test Case #D-4
`p = Person()`
`q = Person('Charlie the Chicken', 88)`
`p.mother = q`
`print(p.mother.name)`
	`Charlie the Chicken`

Test Case #D-5
`p = Person()`
`q = Person('Charlie the Chicken', 88)`
`r = Person('Alison', 22)`
`s = Person('Eileen', 55)`
`q.mother = r`
`r.mother = s`
`s.mother = p`
`p.mother = q`
`print(q.mother.name)`
`print(r.mother.name)`
`print(s.mother.name)`
`print(p.mother)`
	`Alison`
	`Eileen`
	`Unknown`
	`None`

Q.6 State Machines [15 points] (Moderate to Challenging)

(15 pts) Write a State Machine class called VacuumRobot. The robot moves around a rectangular floor in a counter clockwise manner from the wall to the inside of the room. The robot completes its job by covering all tiles in the rectangular floor and stops at the last tile it covers. The robot has the following specifications:

a. The robot must always start at the bottom right (0,0) <start> of the floor map and it always faces up. Please see the table in Example 1 below.
b. The robot can only move forward. Thus "facing up" means that the robot's y-coordinate will increase by 1 during its next move, and "facing left" means the robot's x-coordinate will decrease by 1 during its next move.
c. The robot has one obstacle sensor that gives either True or False. True means that there is a wall or an obstacle in front of the robot. False means there is no wall or obstacle.
d. The robot outputs its current position as a tuple (x,y). The robot ends inside the room at the last tile it covers (x,y) <end>.

The input and output of the state machine is as follows:

- Input: obstacle sensor (True or False as specified above)
- Output: current position of the robot as a tuple (x,y)

The examples below gives the floor map. Each cell in the floor map represents a tile on the floor that the robot is to cover. Within each cell, information on the robot's coordinates (x,y), the direction it is facing and what its front sensor detects is given.
For example:

- (0,1) left |True|: this means that if the robot arrives at this tile, its current position is at x = 0, y = 1, it is going to move to the left, and currently it does detect a wall.
- (-2,0) right |False|: this means that if the robot arrives at this tile, its current position is at x = -2, y = 0, it is going to move to the right, and currently it does NOT detect any wall.

Hint:

- One possible solution for the state of the machine is as follows:
  `next_state = fstate, x, y, last_movement, boundary`
- `fstate`: is either 'find boundary' or 'inside'. The machine is in the 'find boundary' state when it goes around the wall to find the boundary of the floor. The machine is in the 'inside' state when the machine is no longer at the boundary but moving in the inside.
- `(x,y)` is the position of the robot as shown in the floor map example below.
- `last_movement` can either be 'up', 'down', 'left', or 'right'. It indicates what was the movement of the robot in the previous time step.
- `boundary` is to record the floor boundary. It should store what is the maximum and minimum position of the boundary, or the corners of the floor.

You can ignore the hint above if you have a better solution.

Note: Your get_next_values method must not change the state of the state machine or even the attributes of the instance. All that the state machine has to remember must be passed on to the next state. This means that you are not allowed to create additional attributes to store the state.

Example 1:
Floor Map:

(-3,1) down |True|	(-2,1) left |False|	(-1,1) left |False|	(0,1) left |True|
(-3,0) right |True|	(-2,0) right |False|	(-1,0) stop |False|<end>	**(0,0) up |False| <start>**

Example 2:
Floor Map:

(-3,2) down |True|	(-2,2) left |False|	(-1,2) left |False|	(0,2) left |True|
(-3,1) down |False|	(-2,1) stop |False| <end>	(-1,1) left |False|	(0,1) up |False|
(-3,0) right |True|	(-2,0) right |False|	(-1,0) up |False|	**(0,0) up |False| <start>**

Example 3:
Floor Map:

(-3,3) down |True|	(-2,3) left |False|	(-1,3) left |False|	(0,3) left |True|
(-3,2) down |False|	(-2,2) down |False|	(-1,2) left |False|	(0,2) up |False|
(-3,1) down |False|	(-2,1) stop |False| <end>	(-1,1) up |False|	(0,1) up |False|
(-3,0) right |True|	(-2,0) right |False|	(-1,0) up |False|	**(0,0) up |False| <start>**

Test Code:

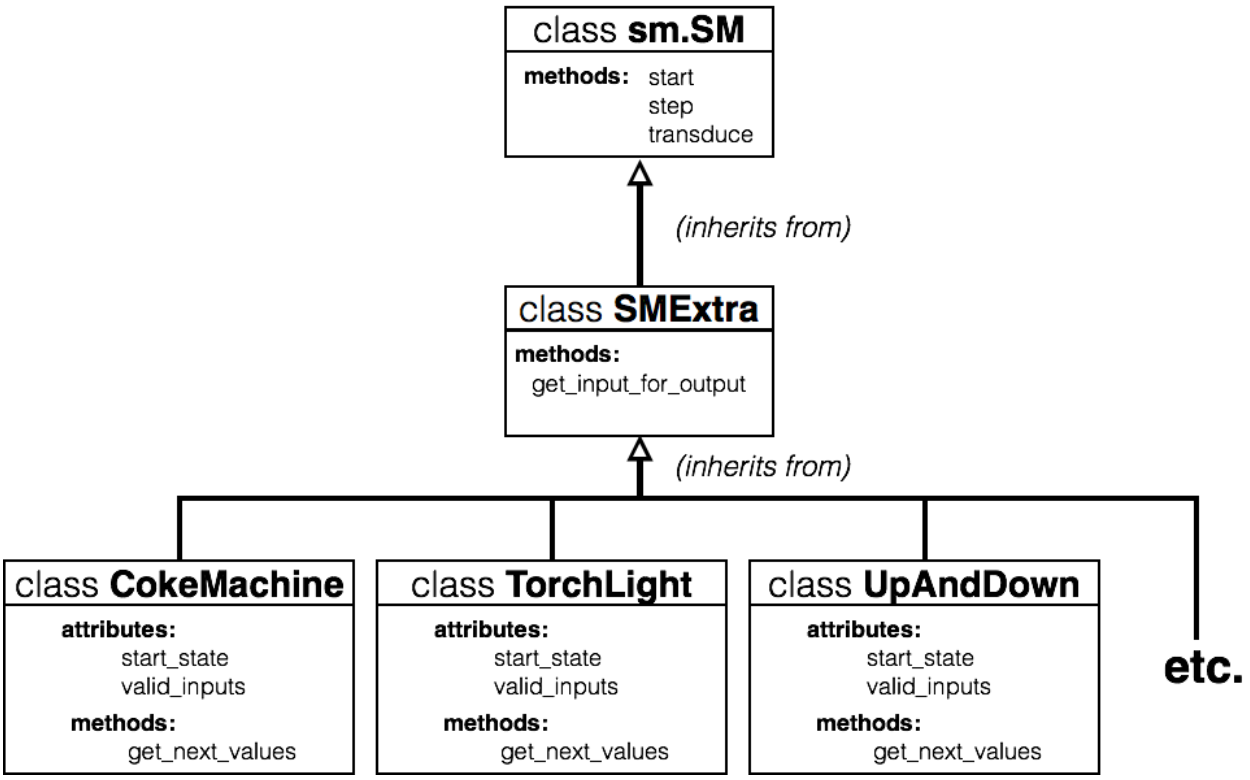
```
v = VacuumRobot()
print('Test 1:')
inp = [False, True, False, False, True, True,
False, False]
ans = v.transduce(inp)
print(ans)

print('Test 2:')
v = VacuumRobot()
inp = [False, False, True, False, False, True,
False, True, False, False, False, False]
ans = v.transduce(inp)
print(ans)

print('Test 3:')
v = VacuumRobot()
inp = [False, False, False, True, False, False,
True, False, False, True, False, False, False,
False,False, False]
ans = v.transduce(inp)
print(ans)

print('Test 4:')
v = VacuumRobot()
v.start()
inp = [False, True, False, False, True, True,
False, False]
state = v.state
for i in inp:
       ns, o = v.get_next_values(state, i)
       state = ns
status = v.state == v.start_state
print('State:', v.state)
print('Not Modified:',status)
```

Output:

```
Test 1:
[(0, 0), (0, 1), (-1, 1), (-2,
1), (-3, 1), (-3, 0), (-2, 0),
(-1, 0)]


Test 2:
[(0, 0), (0, 1), (0, 2), (-1,
2), (-2, 2), (-3, 2), (-3, 1),
(-3, 0), (-2, 0), (-1, 0), (-1,
1), (-2, 1)]


Test 3:
[(0, 0), (0, 1), (0, 2), (0, 3),
(-1, 3), (-2, 3), (-3, 3), (-3,
2), (-3, 1), (-3, 0), (-2, 0),
(-1, 0), (-1, 1), (-1, 2), (-2,
2), (-2, 1)]


Test 4:
State: (this should be the same
as your start state)
Not Modified: True
```

Q.7 Challenge Question: Inheritance and State Machines [15 points]

Task: You are required to define a class named `SMExtra`. Your class will inherit from the general state machine class `sm.SM` in `libdw`, and extend it with some additional functionality. State machine classes will then inherit from `SMExtra`, instead of `sm.SM`, as depicted below:



In `SMExtra`, define the method `get_input_for_output(self, output_list)`, where `self` is (as usual) the object reference of the state machine the method is called on, and `output_list` is a **list of output symbols** from the state machine. The method should return the **list of input symbols** that is required for the state machine to transduce (i.e. generate) the given `output_list`. If such a list of input symbols does not exist, the method should return `[]`.

Take note of the following four requirements:

- Your `get_input_for_output` method should only be implemented ONCE, in `SMExtra`; state machines may access the method only via inheritance.
- You can assume that every state machine object inheriting from `SMExtra` will have an attribute named `valid_inputs` that contains a list of all the possible inputs to the state machine.

- Of the various methods that can be called on `SMExtra` objects, your implementation may ONLY use `get_next_values`, `start`, `step`, or `transduce`. Note that three of these methods actually run the state machine**: if you use them, you must reset the state machine before returning the list of input symbols.**
- Assume that a given output list is transduced from **at most one** input list, and that there is only ever one path through the state machine to produce it.

The provided starter code for this question contains three simple state machine classes (`CokeMachine`, `TorchLight`, `UpAndDown`) that all inherit from `SMExtra`, and which can be used for testing. Diagrams of these three state machines are provided overleaf.

Test code:	**Output:**

Test Case #1
`s = TorchLight()`
`print(s.get_input_for_output([]))`
`print(s.get_input_for_output(['off']))`
`print(s.get_input_for_output(['on']))`
`print(s.get_input_for_output(['on', 'off'] * 4))`
	
	`[]`
	`[]`
	`['push']`
	`['push', 'push', 'push',`
	`'push', 'push', 'push',`
	`'push', 'push']`

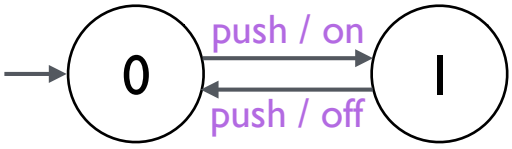
Test Case #2
`s = TorchLight()`
`print('TorchLight:')`
`s.start()`
`s.get_input_for_output(['on','off','on'])`
`print(s.state == s.start_state)`
	
	`True`

Test Case #3
`s = UpAndDown()`
`print(s.get_input_for_output([]))`
`print(s.get_input_for_output([1,2,3,4,5,6,7,8,9,10]))`
 	
`print(s.get_input_for_output([1,2,3,2,1,0,1,2,3,4,5]))`
 	
`print(s.get_input_for_output([1,2,3,2,1,0,1,0]))`
	
	`[]`
	`['up', 'up', 'up', 'up', 'up',`
	`'up', 'up', 'up', 'up', 'up']`
	`['up', 'up', 'up', 'down',`
	`'down', 'down', 'up', 'up',`
	`'up', 'up', 'up']`
	`['up', 'up', 'up', 'down',`
	`'down', 'down', 'up', 'down']`

Test Case #4
`s = CokeMachine()`
`print(s.get_input_for_output([]))`
`print(s.get_input_for_output([(0,'coke',0)]))`
`print(s.get_input_for_output([(0,'--',20)]))`
`print(s.get_input_for_output([(0,'coke',0),`
`(0,'coke',0), (0,'coke',0), (0,'coke',0),`
`(0,'coke',0)]))`
`print(s.get_input_for_output([(50,'--',0),`
`(0,'coke',50)] * 4))`
`print(s.get_input_for_output([(50,'--',0),`
`(50,'--',10), (0,'coke',0)]))
	
	`[]`
	`[100]`
	`[20]`
	`[100, 100, 100, 100, 100]`
	
	`[50, 100, 50, 100, 50, 100,`
	`50, 100]`
	`[50, 10, 50]`

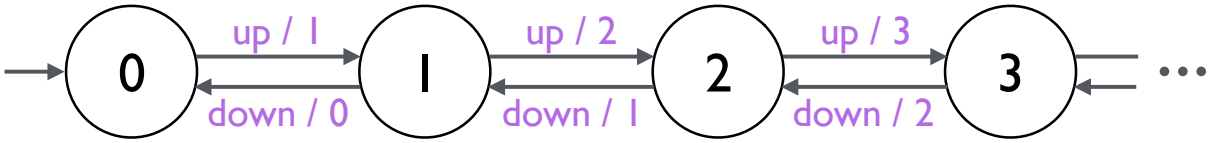
[Diagrams of the example state machines are provided on the next page.]

The TorchLight state machine:



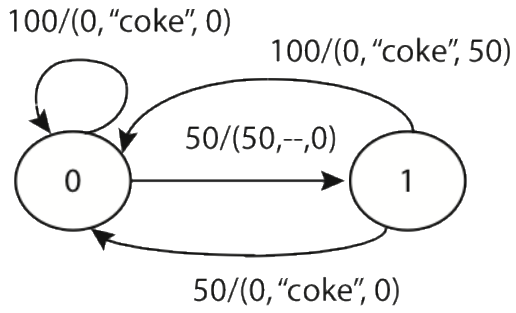
The UpAndDown state machine:

Note: this state machine has an infinite number of states. We depict only the first four. See the starter code for the full implementation.



The CokeMachine state machine:

Note: transitions for rejecting non-50c/$1 coins are not depicted. See the starter code for the full implementation.



End of Exam Paper
[this page is left blank]

[this page is left blank]