UNIVERSITY OF SASKATCHEWAN

Department of Computer Science
176 Thorvaldson Building
110 Science Place, Saskatoon, SK, S7N 5C9, Canada
Telephine: (306) 966-4886, Facimile: (306) 966-4884

Assignment 2

Testing And More

Date Due: June 16, 2023, 11:59pm **Total Marks: 55**

General Instructions

- **This assignment is individual work.** You may discuss questions and problems with anyone, but the work you hand in for this assignment must be your own work.
- **Assignments are being checked for plagiarism.** We are using state-of-the-art software to compare every pair of student submissions.
- Each question indicates what to hand in. You must give your document the name we prescribe for each question, usually in the form aNqM, meaning Assignment N, Question M.
- Make sure your name and student number appear at the top of every document you hand in. These conventions assist the markers in their work. Failure to follow these conventions will result in needless effort by the markers, and a deduction of grades for you.
- Do not submit folders, or zip files, even if you think it will help. It might help you, but it adds an extra step for the markers.
- Programs must be written in Python 3.
- **Assignments must be submitted to Moodle.** There is a link on the course webpage that shows you how to do this.
- **Moodle will not let you submit work after the assignment deadline.** It is advisable to hand in each answer that you are happy with as you go. You can always revise and resubmit as many times as you like before the deadline; only your most recent submission will be graded.
- Read the purpose of each question. Read the Evaluation section of each question.

Question 1 (10 points):

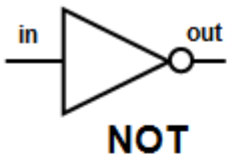
Purpose: To demonstrate the relation between logic and the lower level behaviour of a computer.

Degree of Difficulty: Moderate

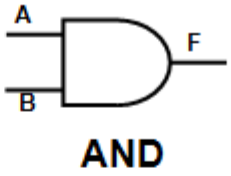
A *proposition* is a statement, *propositional logic* examines how the statements interact with each other. *Truth tables* can be used to examine all possibilities of statements and what their outcomes will be. When outcomes of two different truth tables with the same input are the same we say the statements are *logically equivalent*. An example of this is *double negation* $\neg\neg A$ is logically equivalent to A (Create the truth tables if you are curious).

We represent information using $1's$ and $0's$ and this information goes on to create more complicated calculations such as running a calculation on a calculator, or running your operating system. The $1's$ and $0's$ are trasnformed from input wires using *logic gates*.

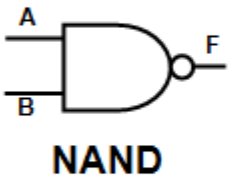
Our boolean operators 'NOT AND OR' can be represented using logic gates and the corresoponding truth tables.

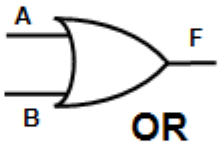


Input	Output
I	F
0	1
1	0



Inputs		Output
A	B	F
0	0	0
1	0	0
0	1	0
1	1	1





Inputs		Output
A	B	F
0	0	0
1	0	1
0	1	1
1	1	1

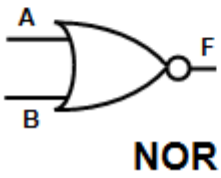


Figure 1: Logic gates and their corresoponding turth tables

Answer the following;

(a) Using logic gates $NOT\ AND\ OR$ to create the circuit representing $\neg(A \vee B)$.

(b) Using logic gates $NOT\ AND\ OR$ to create the circuit representing $\neg A \wedge \neg B$.

(c) Create the truth table for $NAND$ (not and) is it logically equivalent to $\neg A \wedge \neg B$? Why or why not?

(d) Create the truth table for NOR (not or) is it logically equivalent to $\neg A \wedge \neg B$? Why or why not?

Circuit Diagrams: You can draw by hand or use one of the following simulators and take a screen shot of the result, LogicGateSimulator1 LogicGateSimulator2.
Make sure you can verify behaviour is expected based on input.

What to Hand In

- Your answers to the questions will be submitted as a pdf named `a2q1.pdf`.

Be sure to include your name, NSID, student number, course number and section number at the top of all documents.

WARNING: Marks can be deducted if instructions are not followed, or if the answers are not legible.

Evaluation

WARNING: Marks can be deducted if instructions are not followed.

- 2 marks: Correctly creating logic circuit for $\neg(A \vee B)$.
- 2 marks: Correctly creating logic circuit for $\neg A \wedge \neg B$.
- 2 marks: Truth Table for $NAND$.
- 2 marks: Truth Table for NOR.
- 2 marks: Properly analyzing logical equivalency.

Further Resources

- Propsitional Logic
  - Discrete Mathematics - Propositional Logic
  - Internet Encyclopedia of Philosophy - Propositional Logic
- Logic Gate Simulators
  - LogicGateSimulator1
  - LogicGateSimulator2
- Logic Gates
  - Logic Gates and ALU
  - 101 Computing - 8 Bit ALU with Logic Gates
  - Wikipedia - ALU
  - Khan Academy - Logic Gates

Question 2 (15 points):

Purpose: Emphasize the difference between black box and white box testing.

Degree of Difficulty: Easy

Chapter's 6 and 7 of the readings discuss the test driven development (tdd) and the importance of testing. You are expected to implement **test cases** for the functions provided, and identify which are white box testable and which are black box testable.
Your test cases should not make the program crash, but instead alert the user of what tests were unsuccessful, and how many were ran.
You are not expected to implement any functions within this question.

Do not forget to `commit` throughout this process.

Using the provided file (`boxing.py`) do the following;

(a) Add the provided file(s) to your git repo.

(b) Rename the provided file to `a2q2.py`

(c) Create unit tests for each function.
- pre-conditions, post conditions, return values
- edge cases, conditional branching
- values expected to be valid, values expected to be invalid
- unique test cases (not test cases covering more than 1 equivalence class).

(d) Create a test driver that will run only the white box test cases.

(e) Create a test driver that will run only the black box test cases.

(f) Create a test driver that will test both the white box and black box test cases.

Provided Files

- `boxing.py`

If you change provided code:

- Highlight you are making a #CHANGE
- Explain why you're making the change.
- Comment out original code.
- Write your code.
- `commit` your change.

What to Hand In

- Your code changes will be submitted as a python file named `a2q2.py`.
- Your git log relating to this question named `a2q2.log`.

Be sure to include your name, NSID, student number, course number and section number at the top of all documents.
WARNING: Marks can be deducted if instructions are not followed, or if the answers are not legible.

Evaluation

WARNING: Marks can be deducted if instructions are not followed.

- 3 marks: gcd() test cases.
- 3 marks: replace() test cases.
- 3 marks: grade_letter() test cases.
- 3 marks: sort_students_into_grades() test cases.
- 1 mark: Create easy to run test drivers. You are expected to write your test drivers yourself for this assingment.
- 1 mark: Properly identify white box test cases and black box test cases.
- 1 mark: Git log present, and relates to this question.

Further Resources

- W3Schools - Assert
- Wikipedia - Test Driven Development
- Free Code Camp - Inro to TDD
- Test Driven - Modern TDD with Python

Question 3 (15 points):

Purpose: Become comfortable utilizing the debugger.

Degree of Difficulty: Moderate

One of the advantages with IDE's is that debugging is typically built into the GUI (graphical user interface) making it easier to step through programs and pin point where errors are occurring and provide insight as to why.
The test cases should not make the program crash, but instead alter the user of what tests were unsuccessful, and how many were ran. Do not forget to `commit` throughout this process.

Use Chapter's 6 and 7 of the readings and the debugger to help solve this assignment.
Using the provided file (texttttWheresWrong.py), the provided test cases and the debugger do the following;

- Add the provided file(s) to your git repo.
- Rename the provided file to `a2q3.py` and `commit`.
- Use debugger and test cases to
  (a) Identify and Fix the Error(s)
    - copy_list_of_lists()
    - copy_dict_of_dicts()
    - For both functions comment what the error is and why it is occurring as part of the #CHANGE
  (b) Implement Function(s)
    - deep_copy_list_of_dicts()
    - remove_from_2DList()
    - filter_from_2DList()

Note you are not responsible for copying values beyond a depth of 2, therefore do not worry if a list of list contains lists or dicts, similarily with dictionaries.

Provided Files

- `WheresWrong.py`

If you change provided code:

- Highlight you are making a #CHANGE
- Explain why you're making the change.
- Comment out original code.
- Write your code.
- `commit` your change.

If you create more test cases explain testitwhy.

What to Hand In

- Your code changes will be submitted as a python file named `a2q3.py`.
- Your git log relating to this question named `a2q3.log`.

Be sure to include your name, NSID, student number, course number and section number at the top of all documents. WARNING: Marks can be deducted if instructions are not followed, or if the answers are not legible.

Evaluation

WARNING: Marks can be deducted if instructions are not followed.

- 3 marks: Part A copy_list_of_lists()
  - 1 mark: Properly identify what the error is/ why it is occurring
  - 2 mark: Fix error
- 3 marks: Part A copy_dict_of_dicts()
  - 1 mark: Properly identify what the error is/ why it is occurring
  - 2 mark: Fix error
- 3 marks: Part B implementation of deep_copy_list_of_dicts()
- 3 marks: Part B implementation of remove_from_2DList()
- 3 marks: Part B implementation of filter_from_2DList()

Further Resources

- Real Python - Pass by Reference
- Stack Abuse - Check if Value is List
- Jetbrains - Debugging Python Code
- Jetbrains - Basic Code Debugging
- Better Programming - Mastering PyCharm Debugger
- W3Schools - Assert
- Wikipedia - Test Driven Development
- Free Code Camp - Inro to TDD
- Test Driven - Modern TDD with Python

Question 4 (15 points):

Purpose: Emphasize test test driven.

Degree of Difficulty: Moderate

Chapter's 6 and 7 of the readings discuss the test driven development (tdd) and the importance of testing. The test cases *intentionally* crash the program when test cases do not pass, this is to force test driven development to occur.
You are expected to incrementally implement functions within this question, based on the currently failing test case.

This question touches upon Chapter 8 and the idea of abstract data types (adts). The provided question examines people as part of a community, and uses functions to alter or calculate information based on the community.
`community` is a list of records (dictionaries), each record represents a person within the community.
`person` is a dictionary that will *always* contain the following key value pairs `"name": str` `"friends": list`, and `"foes": list`.
The lists related to the keys "friends" and "foes" contain names (`str`) of names either within the community or outside.
In real life people are complicated fickle beings, we will similarly represent this within our community adt! A person can view someone as a friend while the other does not feel the same way, or person can view someone as both friend and foe.
As time progress people drift apart connections are lost, both good and bad.
Implement functions that

- Take in communities and either change them, or recognize characteristics within the community or the people.
- List all person adts that have no foes
- Removes a person adt from the community and remove's the person's name from other people's lists
- Confirms if two people are from the same community and see each other as friends.
- Lists all person adt's someone is truly friends with.

DO NOT ALTER EXISTING TEST CASES, unless you validate *why*.
Do not forget to `commit` throughout this process.

Using the provided file (`tdd.py`), the function docstring, and the tests do the following:

- Add the provided file(s) to your git repo.
- Rename the provided file to `a2q4.py`
- Develop code using tdd methodology;
  1. Run the file
  2. Examine what test case has failed, and why
  3. Develop code to resolve failing test case
  4. Run the file again
  5. If original failing test now passed; `commit` code and go to Step 1.
  6. Else Step 2
  7. Repeat until all test cases pass, for all functions.

Provided Files

- `tdd.py`

If you change provided code:

- Highlight you are making a #CHANGE
- Explain why you're making the change.
- Comment out original code.
- Write your code.
- `commit` your change.

If you choose to add more test cases explain `why`.

What to Hand In

- Your code changes will be submitted as a python file named `a2q4.py`.
- Your git log relating to this question named `a2q4.log`.

Be sure to include your name, NSID, student number, course number and section number at the top of all documents.

WARNING: Marks can be deducted if instructions are not followed, or if the answers are not legible.

Evaluation

WARNING: Marks can be deducted if instructions are not followed.

- 2 marks: get_patient_people()
- 4 marks: leave_community()
- 3 marks: are_community_besties()
- 3 marks: get_all_community_besties()
- 1 mark: Further testing
- 2 marks: Git log present, with tdd evident within `commit` messages

Further Resources

- Python - __main__
- Jetbrains - Debugging Python Code
- Jetbrains - Basic Code Debugging
- Better Programming - Mastering PyCharm Debugger
- W3Schools - Assert
- Wikipedia - Test Driven Development
- Free Code Camp - Inro to TDD
- Test Driven - Modern TDD with Python