i **2021 Testing Aug Intro**

Software Testing Exam.

This exam is an open book exam, and you may use any resources that you want from the internet for reference. The only requirement is that work alone, and that you do not directly copy or plagiarise text from the internet or other sources.

Due to the online nature of the exam, this is a very different exam from previous exams in the course. Nonetheless, I believe that it tests the same skills and knowledge as the previous exams. If you are well prepared for the exam, and you have gone through previous exams then you should be well prepared for this exam.

In section 1 of the exam you are to do some test driven development. I prefer that you write the code in Python, but you can use any programming language that you feel comfortable with.

Be careful when you open links that you open the link in a new tab or window so that you do not accidentally navigate out of the exam system.

If you do not Python, then installed there are online python environments such as

- Progamiz
- Online Python
- Tutorial Sport

The two sections have equal weight on your final mark. I suggest that you spend half of your time on each section. In section two you will have to spend sometime trying to understand the code and the specification. You will not only be graded on test cases that you provide, but on the quality of your justifications.

For your reference all the links as naked text are included below:

- [Progamiz](https://www.programiz.com/python-programming/online-compiler/)
- [Online Python](https://www.online-python.com/)
- [Tutorial Sport](https://www.tutorialspoint.com/execute_python_online.php)
- [Basic Rules](https://en.wikipedia.org/wiki/Comma-separated_values#Basic_rules)
- [Hamiltonian cycle](https://en.wikipedia.org/wiki/Hamiltonian_path)
- [TheAlgorithms] (https://github.com/TheAlgorithms/Python/blob/master/backtracking/hamiltonian_cycle.py)
- [unit testing framework](https://docs.python.org/3/library/unittest.html)

Online availability

I will be online 8:00 until 9:00 and 12:00 until 13:00. You are advised to read the whole exam and ask me any clarification questions early on the exam. I can be contacted via email at justin.pearson@it.uu.se

1 2021 Software Testing Aug TDD

In this section you are to do some test driven development. As I stated before you can use any programming language that feel comfortable with, but I prefer Python.

You are to implement (as much as you can in the time span of the exam) a simple calculator. Your calculator should take a string such as "(1 + 2)*4" and produce an integer representing the answer (in this case 12).

You are not allowed to use any external libraries and parsing tools. Your parser should be implemented using the basic string manipulation facilities in the programming language.

You are to construct a sequence of test cases that gives you a working parser. The goal of TDD is to write tests before you write code, and come up with a sequence of tests that grows your code slowly. Do not forget that refactoring is an important part the development process.

For each test case you are also to provide comments explaining why you picked that test case.

Due to the limitation of this online exam system, you can only be given one code editor box per question. I advise that name your sequence of functions calc_1 , calc_2 , ...

Skriv in ditt svar här

```
1
```

Maximum marks: 5

ⁱ 2021 Software Testing Aug Section 2 introduction

The questions in this section concern the following code to find a [Hamiltonian cycle](#) in a graph. You should spend sometime understanding the problem, and sometime understanding the code.

The code is taken from [TheAlgorithms](#) github repository. The code is repeated here

```python
from typing import List


def valid_connection(
    graph: List[List[int]], next_ver: int, curr_ind: int, path: List[int]
) -> bool:
    """
    Checks whether it is possible to add next into path by validating 2 statements
    1. There should be path between current and next vertex
    2. Next vertex should not be in path
    If both validations succeeds we return True saying that it is possible to connect
    this vertices either we return False

    Case 1:Use exact graph as in main function, with initialized values
    >>> graph = [[0, 1, 0, 1, 0],
    ...          [1, 0, 1, 1, 1],
    ...          [0, 1, 0, 0, 1],
    ...          [1, 1, 0, 0, 1],
    ...          [0, 1, 1, 1, 0]]
    >>> path = [0, -1, -1, -1, -1, 0]
    >>> curr_ind = 1
    >>> next_ver = 1
    >>> valid_connection(graph, next_ver, curr_ind, path)
    True

    Case 2: Same graph, but trying to connect to node that is already in path
    >>> path = [0, 1, 2, 4, -1, 0]
    >>> curr_ind = 4
    >>> next_ver = 1
    >>> valid_connection(graph, next_ver, curr_ind, path)
    False
    """

    # 1. Validate that path exists between current and next vertices
    if graph[path[curr_ind - 1]][next_ver] == 0:
        return False

    # 2. Validate that next vertex is not already in path
    return not any(vertex == next_ver for vertex in path)


def util_hamilton_cycle(graph: List[List[int]], path: List[int], curr_ind: int) -> bool:
    """
    Pseudo-Code
    Base Case:
    1. Check if we visited all of vertices
        1.1 If last visited vertex has path to starting vertex return True either
            return False
    Recursive Step:
    2. Iterate over each vertex
        Check if next vertex is valid for transiting from current vertex
            2.1 Remember next vertex as next transition
            2.2 Do recursive call and check if going to this vertex solves problem
            2.3 If next vertex leads to solution return True
            2.4 Else backtrack, delete remembered vertex

    Case 1: Use exact graph as in main function, with initialized values
    >>> graph = [[0, 1, 0, 1, 0],
    ...          [1, 0, 1, 1, 1],
    ...          [0, 1, 0, 0, 1],
```

```
...              [1, 1, 0, 0, 1],
...              [0, 1, 1, 1, 0]]
>>> path = [0, -1, -1, -1, -1, 0]
>>> curr_ind = 1
>>> util_hamilton_cycle(graph, path, curr_ind)
True
>>> print(path)
[0, 1, 2, 4, 3, 0]

Case 2: Use exact graph as in previous case, but in the properties taken from
    middle of calculation
>>> graph = [[0, 1, 0, 1, 0],
...          [1, 0, 1, 1, 1],
...          [0, 1, 0, 0, 1],
...          [1, 1, 0, 0, 1],
...          [0, 1, 1, 1, 0]]
>>> path = [0, 1, 2, -1, -1, 0]
>>> curr_ind = 3
>>> util_hamilton_cycle(graph, path, curr_ind)
True
>>> print(path)
[0, 1, 2, 4, 3, 0]
"""

# Base Case
if curr_ind == len(graph):
    # return whether path exists between current and starting vertices
    return graph[path[curr_ind - 1]][path[0]] == 1

# Recursive Step
for next in range(0, len(graph)):
    if valid_connection(graph, next, curr_ind, path):
        # Insert current vertex  into path as next transition
        path[curr_ind] = next
        # Validate created path
        if util_hamilton_cycle(graph, path, curr_ind + 1):
            return True
        # Backtrack
        path[curr_ind] = -1
return False


def hamilton_cycle(graph: List[List[int]], start_index: int = 0) -> List[int]:
    r"""
    Wrapper function to call subroutine called util_hamilton_cycle,
    which will either return array of vertices indicating hamiltonian cycle
    or an empty list indicating that hamiltonian cycle was not found.
    Case 1:
    Following graph consists of 5 edges.
    If we look closely, we can see that there are multiple Hamiltonian cycles.
    For example one result is when we iterate like:
    (0)->(1)->(2)->(4)->(3)->(0)

    (0)---(1)---(2)
     |   /   \   |
     |  /     \  |
     | /       \ |
     |/         \|
    (3)---------(4)
    >>> graph = [[0, 1, 0, 1, 0],
    ...          [1, 0, 1, 1, 1],
    ...          [0, 1, 0, 0, 1],
    ...          [1, 1, 0, 0, 1],
    ...          [0, 1, 1, 1, 0]]
    >>> hamilton_cycle(graph)
    [0, 1, 2, 4, 3, 0]

    Case 2:
    Same Graph as it was in Case 1, changed starting index from default to 3
```

```
(0)---(1)---(2)
 |   /   \   |
 |  /     \  |
 | /       \ |
 |/         \|
(3)---------(4)
>>> graph = [[0, 1, 0, 1, 0],
...          [1, 0, 1, 1, 1],
...          [0, 1, 0, 0, 1],
...          [1, 1, 0, 0, 1],
...          [0, 1, 1, 1, 0]]
>>> hamilton_cycle(graph, 3)
[3, 0, 1, 2, 4, 3]

Case 3:
Following Graph is exactly what it was before, but edge 3-4 is removed.
Result is that there is no Hamiltonian Cycle anymore.

(0)---(1)---(2)
 |   /   \   |
 |  /     \  |
 | /       \ |
 |/         \|
(3)         (4)
>>> graph = [[0, 1, 0, 1, 0],
...          [1, 0, 1, 1, 1],
...          [0, 1, 0, 0, 1],
...          [1, 1, 0, 0, 0],
...          [0, 1, 1, 0, 0]]
>>> hamilton_cycle(graph,4)
[]
"""

# Initialize path with -1, indicating that we have not visited them yet
path = [-1] * (len(graph) + 1)
# initialize start and end of path with starting index
path[0] = path[-1] = start_index
# evaluate and if we find answer return path either return empty array
return path if util_hamilton_cycle(graph, path, 1) else []
```

This question has 2 parts and all concern the same piece of code. You should have the code open in a separate browser window for reference. You are free to download the code and run it. You must write all your tests as unit test using the Python [unit testing framework](#). For each test you are to provide comments explaining what you are testing. Due to the nature of this online exam platform, I can only give you a code window or a text window and not both. For some questions it might help for you to draw a control flow graph to derive your test cases, but you do not need to submit a control flow graph.

Each question is about coverage, and you should provide justification (in the comments) what sort of coverage you have achieved and justification of why you have achieved that coverage.

2 **2021 Software Testing Aug Section 2 1 a**

Just based on the specification of the function `valid_connection` write some white box tests. You should provide 4 test cases that cover different parts of the specification. Justification can be supplied as comments in your code.

Fill in your answer here

```
1
```

Maximum marks: 5

³ 2021 Software testing exam Aug Section 2 part 1 b

Consider the function `hamilton_cycle`. You are to come up with a partition using the functional based approach. Define your partition, explain what you are trying to test, and try justify that you actually have a partition. You are free to google and use any facts or examples about Hamiltonian cycles. If you take an example from the internet then give a link to the source.

7/9

Fill in your answer here

| Format ▾ | B | I | U | x₂ | x² | I | ⎘ | ⎗ | ↶ | ↷ | ↺ | ≟ | ≔ | Ω | ⊞ | ✏ | Σ |

Words: 0

Maximum marks: 5

4 ## 2021 Software Testing Aug Section 2 part 1 c

Again consider the function `valid_connection`. You are to come up with a partition of the input domain using the interface based approach. You have to explain what you are trying to test with your partition

Fill in your answer here

Format ▾	B *I* U x₂ x²	Iₓ	copy paste	↶ ↷ ↺	☰ ☰	Ω ⊞	✎	Σ

Words: 0

Maximum marks: 5

5 ## 2021 Software Testing Aug Section 2 part 2

This question concerns the function `util_hamilton_cycle`. The function is recursive and has a loop. You are to provide some sort of path coverage. Choose an appropriate type of path coverage, and give test cases that achieve that level of path coverage. For each test case give the paths that code executes. You can refer to the line numbers on the [web interface to the github repo](#).

Fill in your answer here

```
1
```

Maximum marks: 5

This question concerns the function `util_hamilton_cycle`. The function is recursive and has a loop. You are to provide some sort of path coverage. Choose an appropriate type of path coverage, and give test cases that achieve that level of path coverage. For each test case give the paths that code executes. You can refer to the line numbers on the [web interface to the github repo](#).