# Assignment 1: Streamlining Healthcare Operations by Automating a Decision Rule

**Learning Objectives:**

- Implementing a decision support system for streamlining decision making.
- Writing code to interact with users via `input` and `print`.
- Use `if` statements to control the flow of execution.
- Properly documenting code with comments and docstrings.

## Assignment Description

One major problem in modern healthcare is the undue variation in clinical practice, as doctors may idiosyncratically prescribe widely different treatments for the same symptoms, even though the academic research on the condition has clear recommendations of best practices. This inefficiency has been documented in various research articles attached to this assignment. (Wennberg 2002, Lavelle et al. 2015)

One potential solution for a medical center to reduce such variation and standardize care using evidence-based best practices is to implement an automated decision support tool. An example of a simple tool that has been successfully implemented for the triaging of patients with symptoms of heart attack is the following decision tree designed by Dr. Lee Goldman and colleagues. The following figure is taken from Reilly et al. (2002), who demonstrate in a large field experiment the effectiveness this tool in improving physician decisions in the Emergency Department (ED).

Your assignment is to submit a `decisionTree.py` module containing a function called `implement()`, such that each call to this function guides a medical professional to triage an incoming patient displaying symptoms of heart attack. For example, if the incoming patient has ECG evidence of acute myocardial infarction (the leading cause of death in the US), then the assessed risk level would be high according to the above. However, if the incoming patient has no ECG evidence of acute MI or acute ischemia, and only one of the three urgent factors are present, then the patient is deemd low risk and referred to the inpatient telemetry unit. The following section describes the specifications in detail and give examples.

## Detailed Specification

The submitted `decisionTree.py` module should contain one function called `implement`, with no input parameters. Each call to this function performs the following:

- Display an introductory prompt: "Clinical Decision Support for Suspected Acute Cardiac Ischemia V1.0"
- Display the first question: "Q1: Is there ECG evidence of acute Myocardial Infarction (MI)? (Y/N)"
- If the user responds by typing "Y" then proceed to recommend high risk as in the decision tree. If the user responds with "N" then proceed to the other branch of the tree.
- Continue as above to go through the questions of the tree until a recommendation.

The text of the other questions are "Q2: Is there ECG evidence of acute Ischemia? (Y/N)", and "Q3: How many urgent factors are present? (0/1/2/3)". For the last question, the possible responses are 0, 1, 2 or 3 (keep in mind the inputs are strings). If the user does not respond exactly as specified, then the code should say "Invalid response. Please restart." For this version, do not implement input correction. For example, even for the Yes/No question, the only valid

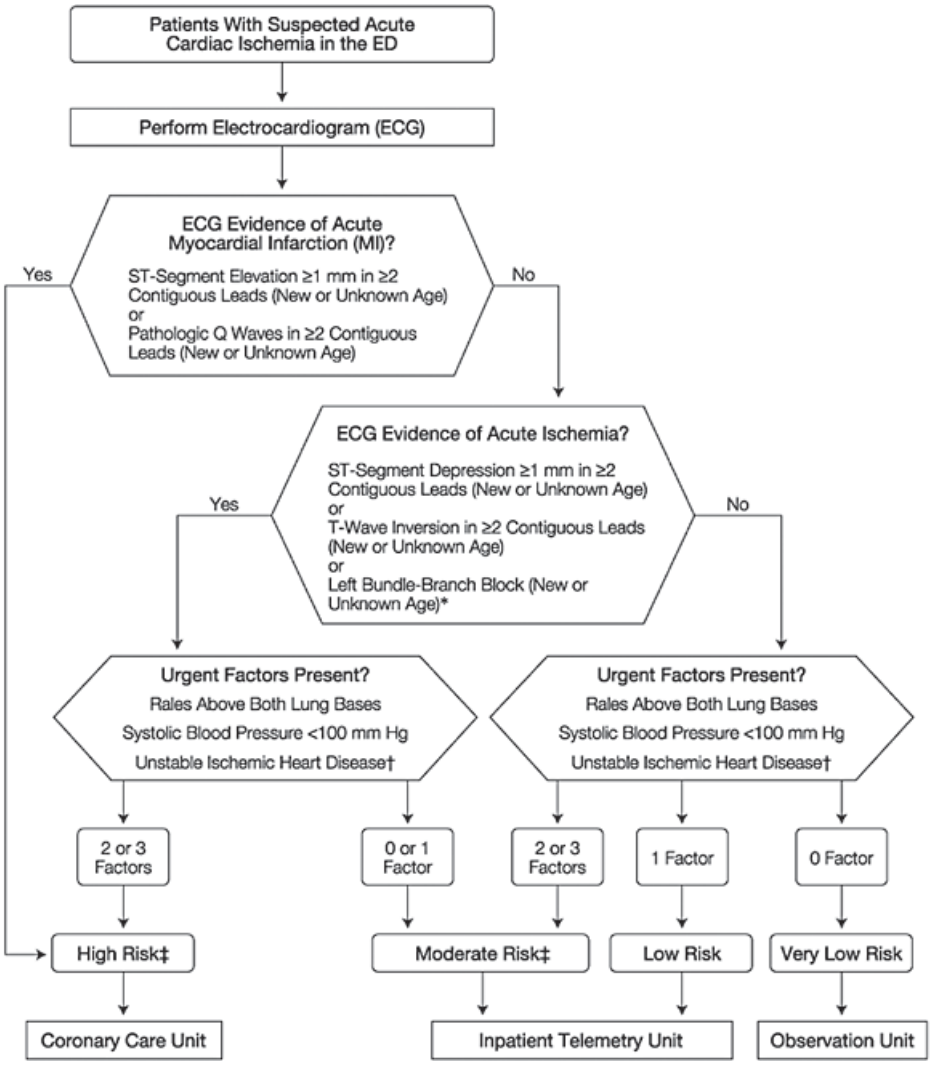


Figure 1. Flow chart of the decision tree used to triage patients in the emergency department, taken from Reilly et al. (2002).

inputs are "Y" and "N" and anything else is invalid, including "y", "n", and "Yes" and "No". For the numerical question, only "0" "1" "2" and "3" are valid.

Note that in displaying questions, you can assume the user knows what are the possible evidences of acute MI and acute ischemia for Q1 and Q2, and what are the three urgent factors in Q3. The possible recommendations are:

- "Diagnosis: Patient has HIGH risk, refer to Coronary Care Unit."
- "Diagnosis: Patient has MODERATE risk, refer to Inpatient Telemetry Unit."
- "Diagnosis: Patient has LOW risk, refer to Inpatient Telemetry Unit."
- "Diagnosis: Patient has VERY LOW risk, refer to Observation Unit."

See below for a few sample executions. (Your `decisionTree.py` module should behave exactly in the same way when called as follows).

```
[1]: import decisionTree as tree
     tree.implement()

Clinical Decision Support for Suspected Acute Cardiac Ischemia V1.0
Q1: Is there ECG evidence of acute Myocardial Infarction (MI)? (Y/N)
```

```
Y
Diagnosis: Patient has HIGH risk, refer to Coronary Care Unit.
```

```
[8]: tree.implement()
```

```
Clinical Decision Support for Suspected Acute Cardiac Ischemia V1.0
Q1: Is there ECG evidence of acute Myocardial Infarction (MI)? (Y/N)
N
Q2: Is there ECG evidence of acute Ischemia? (Y/N)
Y
Q3: How many urgent factors are present? (0/1/2/3)
1
Diagnosis: Patient has MODERATE risk, refer to Inpatient Telemetry Unit.
```

```
[4]: tree.implement()
```

```
Clinical Decision Support for Suspected Acute Cardiac Ischemia V1.0
Q1: Is there ECG evidence of acute Myocardial Infarction (MI)? (Y/N)
y
Invalid response. Please restart.
```

```
[5]: tree.implement()
```

```
Clinical Decision Support for Suspected Acute Cardiac Ischemia V1.0
Q1: Is there ECG evidence of acute Myocardial Infarction (MI)? (Y/N)
N
Q2: Is there ECG evidence of acute Ischemia? (Y/N)
N
Q3: How many urgent factors are present? (0/1/2/3)
4
Invalid response. Please restart.
```

In writing your code, you should put comments at the top of the `decisionTree.py` module to indicate your name and date of completion. You should also use suitable variable names and include comments as needed in the `implement` function to make it readable by other analysts. Moreover, you should include a docstring when defining `implement`, so that the following text is displayed when someone runs `help(tree.implement)`.

```
[7]: help(tree.implement)
```

```
Help on function implement in module decisionTree:

implement()
    This is a function that implements a decision tree for triaging
    suspected patients for acute cardiac ischemia using the command line.
    The tree is developed by Lee Goldman, and tested in a field study in
    the article: https://jamanetwork.com/journals/jama/fullarticle/195118
```

## Grading Rubric

The grade will be out of 12, with a maximum of 3 points allocated to each of the following four categories. For each category, 3 points correspond to being perfect in that category. There is a half point deduction for each minor error and a one point deduction for each major error.

The four categories are:

- **Syntax**: Code runs without raising an Exception regardless of user input.
- **Input/Output**: Code displays the texts exactly as described in the specifications above and correctly elicits user inputs; proper use of `print` and `input` statements.
- **Logic**: Code correctly implements the decision tree and user response leads to the correct output as described in the specifications; proper use of `if`, `elif` and `else` statements.
- **Readability**: Code contains the specified docstring in the function definition, and the identification of the author at the top. Code also includes appropriate comments and variable names so another analyst can easily understand it, with all necessary explanations contained in the code itself.