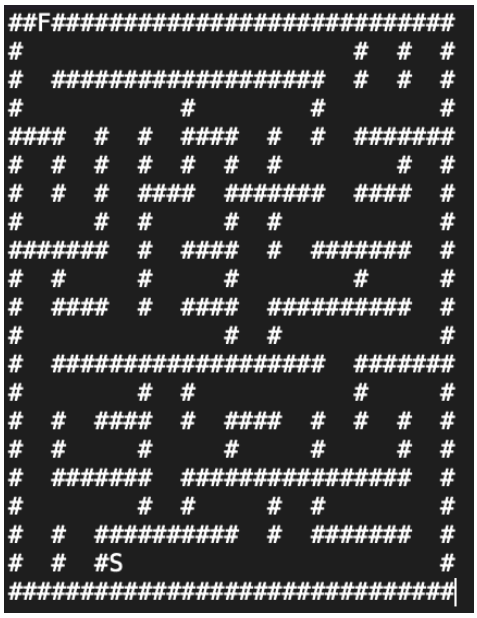
Python and Mathematics for Machine Learning

Assignment - Recursion:

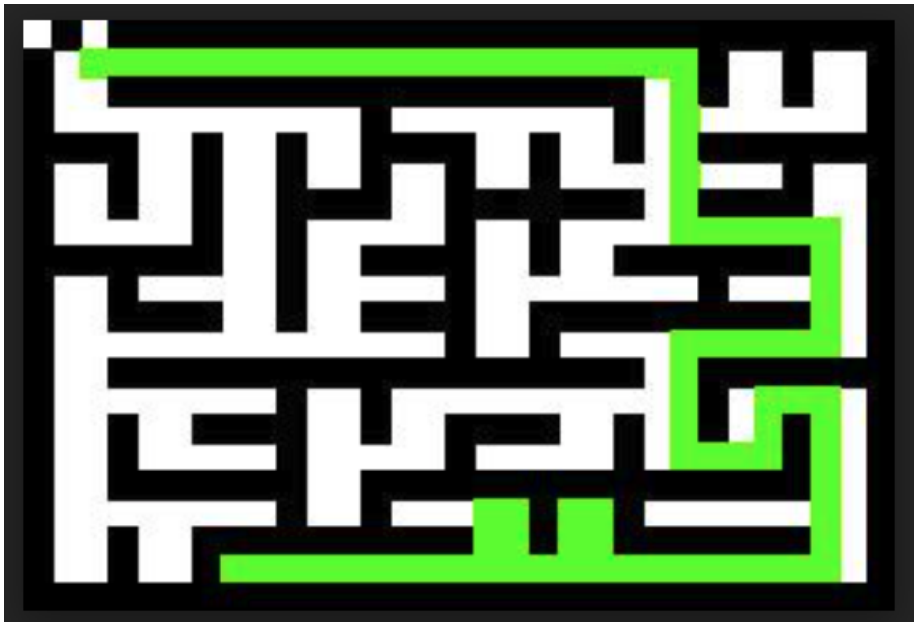
Write a Python program which creates a recursive 'maze solver'. A reference Colab script and companion test files have been included in this assignment module. Your program must satisfy the following requirements:

1. Note, a reference Colab script and companion test files have been included in this assignment module.
2. Reads a text file containing a maze 'map' comprised of symbols which define walls (#), hallways (white space), start marker (S) and finish marker (F), and creates a Python 2-D list of the map. You may want to create a function to do this. An example of a maze map is shown below:



3. Create a recursive function with the following API: *def solveFrom(maze, rowPos, colPos):* where maze is the 2-D list containing the maze map, rowPos is the row index in the maze map, colPos is the column position in the maze map. Return a boolean indicating whether a wall, visited hallway (e.g. a hallway that the algorithm has experienced in a previous recursive iteration), or finish marker has been encountered.
4. Main program to initiate the maze solver recursive function, i.e. call to solveFrom(maze_map, start_row, start_col)
5. Writes an image file using the Python Image and ImageDraw modules from the PIL library (see the reference Colab script for information on these modules)
6. Upload your Colab script, screenshots of your console output, and resultant image file showing the path from the start to finish markers. An example of the output image file for the maze_large map is shown below.

Python and Mathematics for Machine Learning



Assignment - Web Scraper:
Write a Python program which creates a 'web scraper' to access and retrieve real-time data from the US Geological Service (USGS) reflecting the latest active earthquakes around the world which are equal or above a user input magnitude.

- A reference Colab script has been included in this assignment module.
- The feed is from the USGS database here: https://earthquake.usgs.gov/earthquakes/feed/. You should become familiar with this site.
- The format of the feed summary is here: https://earthquake.usgs.gov/earthquakes/feed/v1.0/geojson.php. You should become familiar with the fields for the JSON data.

Your program must satisfy the following requirements:

a. Prompt the user for the magnitude earthquake from which the USGS data will be accessed.
b. Convert the retrieved data into a user-readable format. The data is in JSON format.
c. Write a function with API: *def printResults(data):* where data is . Print the events recorded which are equal or greater than the earthquake magnitude you entered. You may print out th eresults directly from the function or return a string which is printed from the runner.

d. Modify your function to return a data structure that can be formatted and output to a CSV file.
e. Upload your Colab script, screenshots of your console outputs, and your CSV file.

Assignment - Card Deck Simulation

Part 1 - Deck of Cards

In Part 1, you will be implementing the classes and models to simulate a deck of cards. A reference Colab script has been provided as a guide to help with the implementation. Your program must satisfy the following requirements:

1. Generate a deck of cards (called myDeck) as a list (52 items in the list), where each card is implemented as a dictionary consisting of its suit (hearts, spades, diamonds, clubs), rank (Ace, '2', '3', … , '10', 'Jack', 'Queen', 'King'), and point value (1, 2, 3, …, 10, 10, 10, 10). Set the Ace to point value 1 (not 11)
2. Your code must pass the doctest module embedded in the Colab script
3. Implement the following classes and runner to test your code.

```
#Python Object Oriented Programming Project
#DeckOcards - Part A

#Class API's:
#class Card:
#  def __init__(self, cardRank, cardSuit, cardPointVal):
#  def getSuit(self):
#  def getRank(self):
#  def getPointVal(self):

#class Deck:
#  def __init__(self, suits, ranks, values):
#     __cards = []
#  def getCard(self, index):
#  def setCard(pos, Card):
#
#Program Runner
#Init suits, ranks, point values for the cards, create myDeck
#  mySuits = ['hearts', 'diamonds', 'clubs', 'spades']
#  myRanks = ['ace', '2', '3', '4', '5', '6', '7', '8', '9', '10', \
#           'jack', 'queen', 'king']
#  myPointVals = [1,2,3,4,5,6,7,8,9,10,10,10,10]
```

Part 2 - Card Shuffler

In Part 2, you will be implementing two functions to simulate shuffling a deck of cards. The implementations will include a perfect interleaved shuffle and a random order shuffle. A reference Colab script has been provided as a guide to help with the implementation. Your program must satisfy the following requirements:

- Part 2a: Create a shuffler function to rearrange the cards using a 'perfect shuffle',
  - Interleave the deck exactly in half. In other words, your shuffled deck should contain card index 0, 26, 1, 27, 2, 28, … 25, 51)
  - Your code must pass the doctest module embedded in the Colab script

```
#Shuffle the deck using perfect interleaved shuffle
#Define shuffDeckPerf init with 0's with length suits*ranks
#Initialize shuffDeckPerf index == 0
#Loop over len(myDeck)/2 (0 index to midpoint in myDeck)
   #Assign shuffDeckPerf at its index to myDeck at loop index
   #Increment shuffDeckPerf index by 2 (every other spot)
#Initialize shuffDeckPerf index == 1
#Loop over len(myDeck)/2 (midpoint to end)
   #Assign shuffDeckPerf at its index to myDeck at loop index
   #Increment shuffDeckPerf index by 2 (every other spot)
```

- Part 2b: Create a shuffler function to rearrange the cards in a random order
  - Create a user input to set how many shuffles will be performed
  - Swap cards in your deck "in-place", i.e. do not assemble a new list of cards
  - Your code must pass the doctest module embedded in the Colab script

Part 3 - Blackjack

In Part 3, you will be implementing the classes and methods needed to simulate the game “Blackjack” using the card deck you created in the previous parts. A reference Colab script has been provided as a guide to help with the implementation. Your program must satisfy the following requirements:

- Use the classes for the cards and deck from the previous parts in this assignment
- Use the random shuffle function from the previous part in this assignment
- Write a “deal” function to deal the cards to the dealer and player
- Write a “hit” or “stay” function for the player and dealer (user input for player)
- Display the winner, whether the dealer or player busted (went over 21), and prompt the player to play again. If playing again, the cards in the deck are replaced and reshuffled.
- Player hits/stays first then dealer stays if >= 16, else hits until >= 16 or busted
- Must have logic to treat Ace as 1 or 11 (use 1 if busted)
- If a user or dealer is dealt a Jack and Ace then they win and game is over