Ngoc (Alice) Hua

7.25.2019

W18: Python Fundamental for Data Science

Project 1 Reflective Summary: Sudoku game

I. How to use the project:

Run the .py file to use the project.

The rules to the game are as follow:

1. Each row has all numbers from 1 - 9
2. Each column has all numbers from 1 - 9
3. Each 3x3 grid box has all numbers from 1 - 9

User will be asked to give a numerical input (valid input is any number from 1 to 9) based on their row and column selection.

Each time the user enters a number, the board will be printed with the updated input. However many empty cells are the needed user’s inputs to complete the game.

Once the board is filled. The game will compare the user’s solution to their board’s original solution.

If they have successfully solved the Sudoku, the game will return True. Otherwise, they’ll see False. In this case, they could exit the game or play again.

II. Dicuss challenges I faced and how I overcame them:

Challenge 1: The first challenge building this game is how could the board be built in a way that allows me to extract the row and column, update the item and return the board. I found that I need to take the user’s inputs to update the current board and print it out for each cycle. The print-out board needs to look visually understandable to the user, without commas, quotations or brackets, so I needed to create a DisplayBoard method to do so. Also in regard to asking the same questions till the board is full, I thought of putting the inputs in a while loop with the condition that if the board is not full, which is checked by a class method Board\_not\_Full.

Challenge 2: How to initialize the functions written. Like when to call the functions. I found out that initializing a method requires that you create a new instance of a class, which if I use @staticmethod, then I don’t need to create a new instance. However, I did not need to use it. The challenge is here is how to solve problems the simplest way possible, not be inclined to import multiple libraries or overcomplicate the solutions.

Challenge 3: How to check for answer. Should I write multiple methods to check for each row and each column. I wrote all the methods I thought were necessary but toward the end of the week, I came up with a simpler solution. Instead of checking for each row and column then each box. I can achieve the same goal with one method that checks the filled out board to the solution board.

I wrote the following methods: However, I can achieve the same goal with one method

|  |  |
| --- | --- |
| def is\_valid\_row(row):  counts = [0, 0, 0, 0, 0, 0, 0, 0, 0]  for element in row:  counts[element - 1] += 1  if counts != [1, 1, 1, 1, 1, 1, 1, 1, 1]:  return False  else:  return True    def all\_rows\_valid(board):  num\_valid\_rows = 0  for row in board:  if is\_valid\_row(row):  num\_valid\_row += 1  if num\_valid\_rows == 9:  return True  else:  return False    def is\_valid\_column(column):  counts = [0, 0, 0, 0, 0, 0, 0, 0, 0]  for element in column:  counts[element - 1] += 1  if counts != [1, 1, 1, 1, 1, 1, 1, 1, 1]:  return False  else:  return True    def all\_columns\_valid(board):  num\_valid\_column = 0  for column in board:  if is\_valid\_column(column):  num\_valid\_column += 1  if num\_valid\_column == 9:  return True  else:  return False  def get\_box\_elements(board, box\_num):  elements = []  x\_lo, y\_lo = define\_limits(box\_num)  for row in range(3 \* x\_lo, 3 \* x\_lo + 3):  for col in range(3 \* y\_lo, 3 \* y\_lo + 3):  elements.append(board[row][col])  return elements  def box\_is\_valid(board, box\_num):  values = get\_box\_elements(board, box\_num)  num\_values = [int(x) for x in values]  print(num\_values)  return is\_valid\_row(num\_values)  def is\_valid\_board(board):  if all\_rows\_valid(board) and all\_columns\_valid(board) and all\_boxes\_valid(board):  return True  else:  return False | def BoardsAreEqual(board1, board2):  """Returns true if board1 and board2 are identical false otherwise"""  for row\_i in range(0, 9):  for col\_i in range(0, 9):  if board1[row\_i][col\_i] != board2[row\_i][col\_i]:  return False  return True |

Update: The 24-hour period update

I created more levels of the game adding to level easy are medium, hard, and diabolical.

First, I wrote the boards and all the board level solutions.

Then I want to create classes for the other levels using inheritance rule and reuse the code from the existing class Board for level easy.

Challenge 4:

I learned to use inheritance as my solution and arranged my code in the other classes related to board so that they can inherit the board’s method attributes. This solution saves me from having to rewrite all the methods.

Challenge 5:

My while loop doesn’t recognize what board it is that the user is playing so it keeps updating the original easy board. So I needed to set a bigger if else statement to make sure that the loop will execute under a certain user’s input for the game level.

Challenge 6:

Setting the response to the check method on filled out board. I have to create instances of check classes to correspond with the board level and pass in both the filled-out board and the board solution. Carefully making sure that each level is checked accordingly and print out an easy to understand response statement and close the game.