Worksheet 02

Name:
UID:

Topics

- Effective Programming

Effective Programming

a) What is a drawback of the top down approach?

If there are changes in requirements or unforeseen problems at the lower levels, it might be difficult to adapt the overall design without significant reworking.

b) What is a drawback of the bottom up approach?

Since the focus is on the details and specific components from the beginning, there's a risk of losing sight of the overall system architecture and objectives. This can result in a system that works well in parts but fails to meet the overall requirements or goals.

c) What are 3 things you can do to have a better debugging experience?

1. Read the error
2. Re-read the code
3. Sanity check - is everything doing what its supposed to do?

d) (Optional) Follow along with the live coding. You can write your code here:

In [1]:
```
class Board:

    def __init__(self):
        self.board = [["-" for _ in range(8)] for _ in range(8)]

    def __repr__(self):
        res = ""
        for row in range(8):
            for col in range(8):
                res+= self.board[row][col]
                res += " "
            res += "\n"
        return res

    def set_queen_at(self, row, col):
        self.board[row][col] = "Q"

    def unset_queen_on_row(self, row):
        self.board[row] = ["-" for _ in range(8)]

    def find_solution(self):

        row = 0
        col = 0
        while(row < 8):
            # we are searching for a solution
            if (self.is_valid_move(row, col)):
                self.set_queen_at(row, col)
                row += 1
                col += 0

            else:
                col += 1
                if (col >= 8):
                    # we weren't able to place a queen on this row
                    # we need to backtrack and adjust the postion
                    # of the queen on the previous row
                    col = self.get_queen_on_row(row - 1)
                    col += 1
                    row -= 1

        # we have found the solution
        print("Found a solution: ")
        print(self)


test = Board()
print(test)
test.set_queen_at(1, 1)
print(test)
test.unset_queen_on_row(1)
print(test)
```

```
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _

_ _ _ _ _ _ _ _
_ Q _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _

_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _
```

Exercise

This exercise will use the Titanic dataset (https://www.kaggle.com/c/titanic/data) (https://www.kaggle.com/c/titanic/data (https://www.kaggle.com/c/titanic/data)). Download the file named `train.csv` and place it in the same folder as this notebook.

The goal of this exercise is to practice using pandas (https://pypi.org/project/pandas/) methods. If your:

1. code is taking a long time to run
2. code involves for loops or while loops
3. code spans multiple lines

look through the pandas documentation for alternatives. This cheat sheet (https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf) may come in handy.

a) Complete the code below to read in a filepath to the `train.csv` and returns the DataFrame.

In [25]:
```
import pandas as pd

#Reading the csv
df = pd.read_csv("train.csv")
df.describe()
```

Out[25]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

b) Complete the code so it returns the number of rows that have at least one empty column value

In [19]:
```
print("there are " +  str(df.isna().any(axis=1).sum()) + " rows with at
```

```
there are 708 rows with at least one empty value
```

c) Complete the code below to remove all columns with more than 200 NaN values

In [26]:
```
df.columns

# Identify columns with more than 200 NaN values
mask = df.isna().sum() > 200

# Converting to a list (so that pandas dosent do a future warning)
columns_to_drop = df.columns[mask].tolist()

# Drop these columns
df.drop(columns_to_drop, axis=1, inplace=True)

df.columns
```

Out[26]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibS
p',
       'Parch', 'Ticket', 'Fare', 'Embarked'],
      dtype='object')
```

d) Complete the code below to replaces `male` with 0 and `female` with 1

In [27]:
```
df['Sex'] = df['Sex'].replace({'male': '0', 'female': '1'})

# or df['Sex'] = df['Sex'].str.replace('male', '0').str.replace('female',

df.head()
```

Out[27]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarl
0	1	0	3	Braund, Mr. Owen Harris	0	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	1	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	35.0	1	0	113803	53.1000	
4	5	0	3	Allen, Mr. William Henry	0	35.0	0	0	373450	8.0500	

e) Complete the code below to add four columns `First Name`, `Middle Name`, `Last Name`, and `Title` corresponding to the value in the `name` column.

For example: `Braund, Mr. Owen Harris` would be:

First Name	Middle Name	Last Name	Title
Owen	Harris	Braund	Mr

Anything not clearly one of the above 4 categories can be ignored.

In [28]:

```
# Define a regular expression pattern to extract the names
# The pattern assumes the format 'Last Name, Title. First_Name Middle_Na
pattern = r'(?P<Last_Name>[^,]+), (?P<Title>\w+)\. (?P<First_Name>\w+)(?

# Extract the parts of the names into a new DataFrame
names_df = df['Name'].str.extract(pattern)

# Assign the extracted columns to the original DataFrame
df[['First_Name', 'Middle_Name', 'Last_Name', 'Title']] = names_df[['Firs

# Show the resulting DataFrame
df.head()
```

Out[28]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarl
0	1	0	3	Braund, Mr. Owen Harris	0	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	1	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	35.0	1	0	113803	53.1000	
4	5	0	3	Allen, Mr. William Henry	0	35.0	0	0	373450	8.0500	

f) Complete the code below to replace all missing ages with the average age

In [32]:
```
import numpy as np
df['Age'] = df['Age'].replace(np.nan,df['Age'].mean())
df.head()
```

Out[32]:

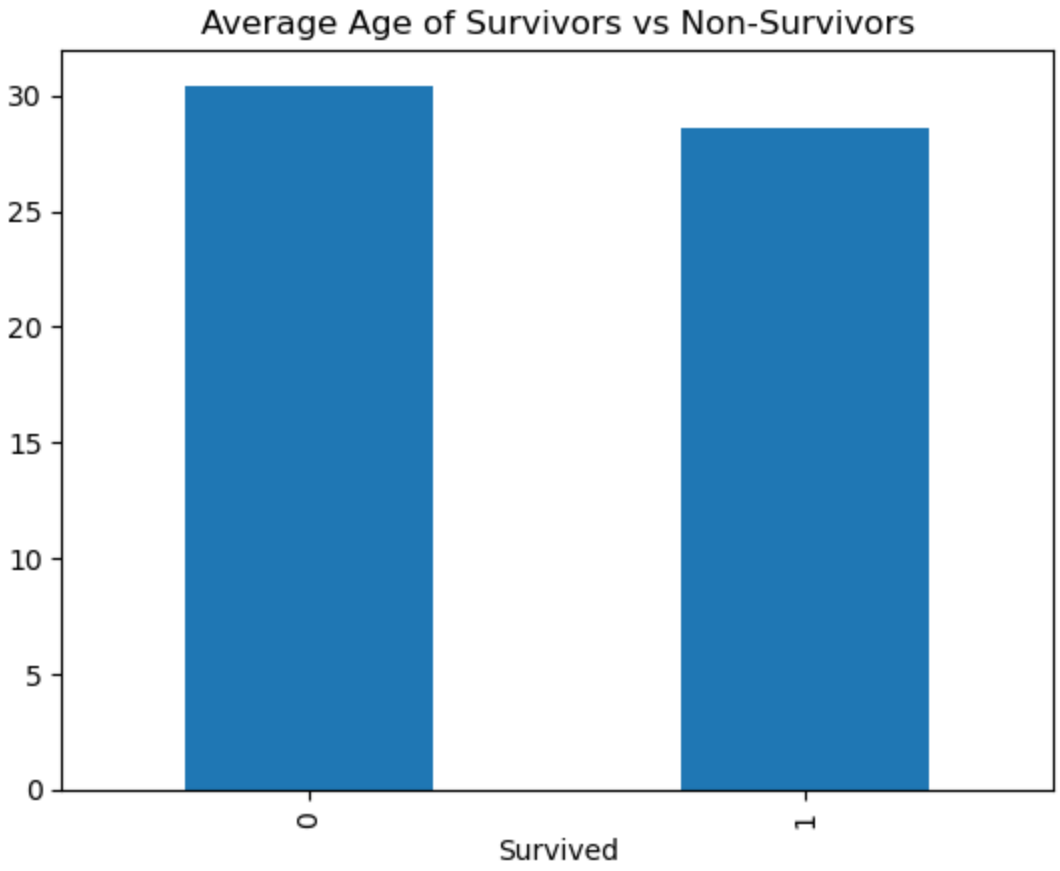
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Embarl
0	1	0	3	Braund, Mr. Owen Harris	0	22.0	1	0	A/5 21171	7.2500	
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	1	38.0	1	0	PC 17599	71.2833	
2	3	1	3	Heikkinen, Miss. Laina	1	26.0	0	0	STON/O2. 3101282	7.9250	
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	1	35.0	1	0	113803	53.1000	
4	5	0	3	Allen, Mr. William Henry	0	35.0	0	0	373450	8.0500	

g) Plot a bar chart of the average age of those that survived and did not survive. Briefly comment on what you observe.

In [34]:
```
# Group the DataFrame by 'Survived' and calculate the mean age for each g
average_age_by_survival = df.groupby('Survived')['Age'].mean()

# Plot the results using a bar chart
average_age_by_survival.plot(kind='bar', title='Average Age of Survivors
```

Out[34]:
```
<Axes: title={'center': 'Average Age of Survivors vs Non-Survivors'}, x
label='Survived'>
```



the average age of the people who survived and didnt is about the same, dont see agism

In [ ]:

In [ ]: