**NOTE: This is an official document by Indexademics. Unless otherwise stated, this document may not be accredited to individuals or groups other than the club IDX, nor should this document be distributed, sold, or modified for personal use in any way.**

**IDX G9 CS H STUDY GUIDE ISSUE 5**

**By Iris and Gavin**

Computer Science Representation

* colors are represented using hexadecimal notation.
* # followed by six digits, where each pair represents the red, green, and blue (RGB) components.

Decimal Number System

* a base-2 numeral system that uses only two digits: 0 and 1.
* Each digit is referred to as a bit.
* The value of each bit is determined by its position, which represents a power of 2.
* For example, the binary number 1011 can be calculated as:
* (1 × 2³) + (0 × 2²) + (1 × 2¹) + (1 × 2⁰)

= 8 + 0 + 2 + 1 = 11 (in decimal)

Computer Storage Systems

* There are two systems for measuring storage:
* Binary Byte System: Uses powers of 2. For example, 1 KB = 1024 bytes.
* Decimal Byte System: Uses powers of 10. For example, 1 KB = 1000 bytes.

Hexadecimal Notation

* a base-16 system, using digits 0-9 and letters A-F (where A=10, B=11, ..., F=15).
* Compact and often used in memory addresses and color codes.
* For example, Decimal 255 is Hexadecimal FF. (uses less digits)

Binary to Hexadecimal

* To convert binary to hexadecimal:
  + Group the binary digits into sets of 4 (starting from the right).
  + Convert each group to its hexadecimal equivalent.
* Example:
  + Binary 11010111 → Grouped as 1101 0111 → Hexadecimal D7.

Binary Arithmetic

* Addition: Add binary digits like decimal, carrying over when the sum is 2 or more.
  + Example: 101 + 11 = 1000.
* Subtraction: Use the borrow method, similar to decimal subtraction.
  + Example: 1010 - 11 = 111.

Bitwise Operations

* Bitwise operations manipulate individual bits:
* AND (&): Returns 1 if both bits are 1. Example: 1010 & 1100 = 1000.
* OR (|): Returns 1 if at least one bit is 1. Example: 1010 | 1100 = 1110.
* XOR (^): Returns 1 if bits are different. Example: 1010 ^ 1100 = 0110.
* Left Shift (<<): Shifts bits to the left, adding 0s on the right. Example: 1010 << 1 = 10100.
* Right Shift (>>): Shifts bits to the right, discarding bits on the right. Example: 1010 >> 1 = 0101.

Series

* a one-dimensional array that consists of a set of index and a set of data
* Series data can be any list, dictionary, or scalar value

Creating a Series

* **List**: s1 = pd.Series(["12","34","56","78"])
  + **Set specific index**: s2 = pd.Series([1,2,3,4], index=["line1","line2","line3","line4"])
* **Dictionary**: s3 = pd.Series({"a":11, "b":22, "c":33, "d":44})

DataFrame

* a two-dimensional array that is capable of storing various data types with a set of index pairs (rows and columns) and a set of data (values)
* a spreadseet data representation

Create a DataFrame

* **Dictionary**
  + data = {'state': ['Ohio','Texas'], 'year': [2020,2021]} # create a dictionary, with column names as the key and list of datas to be put in the column
  + df = pd.DataFrame(data, columns=['state', 'year'], index=['line1','line2']) # use columns = [] can modify the order of the columns, if column name didn’t exist in the dictionary, it will become NaN
* **Nested List**
  + s1 = [‘Ohio’,2020]
  + s2 = [‘Texas’,2021]
  + data = [s1, s2] # each list in the list is a row in the df
  + df1 = pd.DataFrame(data, columns=[‘state’, ‘year’], index=['line1','line2'])
* **Nested Dictionary**
  + c1 = {‘state’:‘Ohio’, ‘year’:2020}
  + c2 = {‘state’:‘Texas’, ‘year’:2021}
  + df = pd.DataFrame({‘line1’: c1, ‘line2’:c2})
  + # notice that here, the line1 and line2 will be the column name while state and year will be the row name
* **List of Series**
  + columnList = ['state', 'year']
  + s1 = pd.Series(['Ohio', 2020], index=columnList) # one row in the df
  + s2 = pd.Series(['Texas', 2021], index=columnList) # one row in the df
  + df2 = pd.DataFrame([s1,s2], index=['line1', 'line2’])
* **Dictionary of Series**
  + columnList = ['line1', 'line2’] # name of rows into a list
  + s1 = pd.Series(['Ohio', 'Texas'], index=columnList) # Ohio and Texas into the first column
  + s2 = pd.Series([2020, 2021], index=columnList) # 2020 and 2021 into the second column
  + df2 = pd.DataFrame({'state':s1, 'year':s2}) # first column named state, second column named year
* **List of Dictionaries**
  + s1 = {‘state’: ‘Ohio’, ‘year’: 2020} # each into one row
  + s2 = {‘state’: ‘Texas’, ‘year’: 2021} # each into one row
  + data = [s1, s2] # data = [s1, s2, …]
  + df3 = pd.DataFrame(data, columns=[‘state’, ‘year’], index=['line1','line2'])

selecting

* **selection of one column**
  + df[‘colname’]
* **Selection of Several Columns**
  + df[[‘col1’,’col2’,’col3’,…]]
* **selection of one row**
  + df.loc[‘rowname’]
  + df.iloc[index]
* **Selection of Specific Row**
  + df.loc[df[‘col’]==value]
* **Selection of Rows and Columns**
  + iloc
    - df.iloc[[row index], [column index]]
    - df.iloc[starting index:ending index+1, starting index:ending index+1]
  + loc
    - df.loc[[row name], [column name]]
    - df.loc[starting name:ending name, starting name:ending name]

modifying

* **add/modify data in one column**
  + df[colname] = data # usually a list with elements’ number = row number in the df
* **add/modify data in one column**
  + df.loc[rowname] = data
  + df.iloc[rowindex] = data
* **Modifying Specific Cell**
  + df.loc[‘row’,’col’]
  + df.iloc[rowind,colind]
* **modifying one row of data**
  + df.loc[<condition>] = [val1, val2, …]
  + df.loc[<condition>, ['col1', 'col2', …]] = [val1, val2, …]
  + df.iloc[‘row\_idx’]= [val1,val2, …]
* **adding one row of data**
  + df.loc[str(len(df))] = [value1, value2, …]
  + df = df.\_append(pd.DataFrame([[value1, value2, …]], columns=df.columns))
* **Adding/Modifying One Column Data**
  + df[‘col’] = [value1, value2, …]
  + df.loc[:, ‘col’]= [value1, value2, …]

deleting

* **Deleting Entries from Row or Column Labels**
  + del df[col\_name] OR del df[df.columns[col\_index]]
  + df = df.drop([row\_label1,row\_label2])
  + df = df.drop([col\_name1, col\_name2], axis = 1)
    - if axis = 1, drops column
    - if axis = 0, drops row (no axis, assume axis = 0)
* **Deleting Entries from Row Label Number**
  + df = df.drop([row\_label\_num])
  + df = df.drop([row\_label\_num1,row\_label\_num2])

slicing

* **slicing with loc**
  + df.loc[rowStartLabel:rowEndLabel]
* **slicing with iloc**
  + df.iloc[rowStartId:rowEndId, colStartId:colEndId]
  + df.iloc[rowStartId:rowEndId, colStartId:colEndId]

Selection with Condition

* Selected part called: **Boolean Series**
* **Select elements with True at corresponding value**
  + Selected\_data = df[‘column name’] == values in the column
  + E.g. boolSeries = data[‘age’]==14
* **Multiple conditions**
  + Uses & (and) and |(or)
  + Selected\_data = (df[‘column name’] == values in the column) & (df[‘column name’] == values in the column)
  + E.g. boolSeries = (data[‘age’]>=14) & (data[‘country’]==‘USA’)
* **Put all in one line**
  + Data = df[df[‘column name’] == values in the column]
  + data = df[(df[‘column name’] == values in the column) & (df[‘column name’] == values in the column)]
* **Extracting Data**
  + Df1 = df.loc[df['col'] == 'condition', [‘col1,col2’]]
  + Df2 = df.loc[df[‘col’]==‘condition’] [[‘col1’,’col2’]]
* **Modifying Data**
  + df.loc[df['col'] == 'condition', 'col'] = value
  + df.loc[df['col'] == 'condition', [‘col1,col2’]] = [value1, value2]
* **Removing Data**
  + Df = df.loc[df[‘col’] != ‘condition’, :]

Data in different Formats

* **Excel**
  + Read – data = pd.read\_excel(‘filename.xlsx’)
  + Write – df.to\_excel(‘filename.xlsx’, ‘Sheet1’, index=False)
* **CSV**
  + Read – data = pd.read\_csv('filename.csv’)
  + Write – df.to\_csv('filename.csv', index=False)

other random programs

* df.dropna()
  + used to remove all the rows of which the content is NULL
* df.fillna(val)
  + used to replace the NULL values with a specified value
* **sort values**
  + df2 = df1.sort\_values (by='score', ascending=?)
    - if ascending == True, ascending order
    - if ascending == False, descending order
* **reset index**
  + df2 = df2.reset\_index(drop=True)
    - if drop == True, reset the indexes
    - if drop == False, index become a new role, adding new indexes
  + df2.index=range(len(df2))