Solutions - Data Handling with Python

Equifax cohort

**NOTE: Please include code solutions to share with group, if you have an alternative solution, please ‘insert row below’ and include**

Introduction to Data Handling with Python

# Store this two-dimensional array

y = np.array([[12, 5, 2, 4],

[ 7, 6, 8, 8],

[ 1, 6, 7, 7]])

1. Display the first 2 rows and the first 3 columns.

|  |
| --- |

2. Display the first column of y.

|  |
| --- |

3. Display the first row of y.

|  |
| --- |

Read the contents of file *cdc\_1.csv*, containing heights, weights and

ages, into array data. To do this, you can use the below code:

data = np.genfromtxt('data/cdc\_1.csv', delimiter=',', skip\_header=1)

4. Calculate the standard deviation and the mean for

* height
* weight
* age

Output the results in a printed summary

|  |
| --- |

**Extension -** Next, read the contents of file cdc\_nan.csv, containing heights, weights and ages, into array data.

1. Separate the heights (column 0) and the weights (column 1)

|  |
| --- |

1. Calculate the median for the heights and the weights and assign the

values to variables.

|  |
| --- |

1. What has happened? Check if the arrays contain missing values.

| np.isnan(weights) |
| --- |

1. Array weights contain three nan values. Find their positions.

|  |
| --- |

1. Now calculate the median for the weights ignoring the nan values.

|  |
| --- |

1. 3.Data Structures, Functions, & Basic Types

### Objective

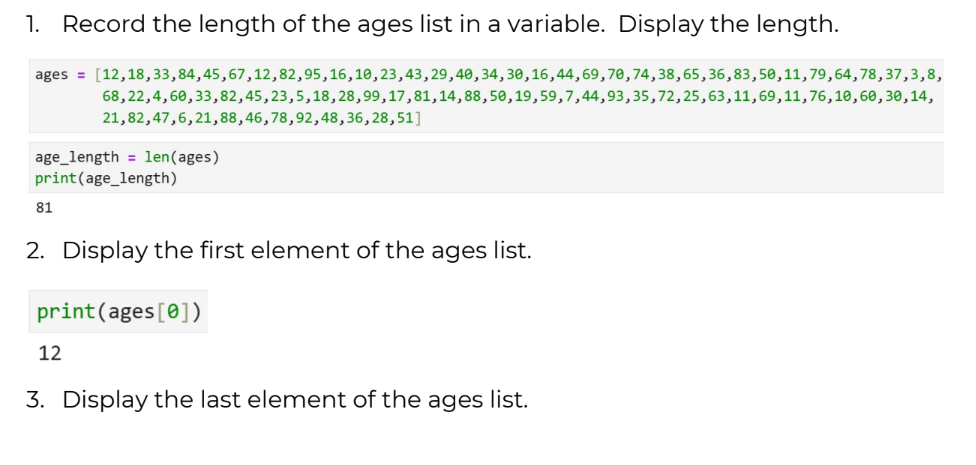
In this guided task, you will:

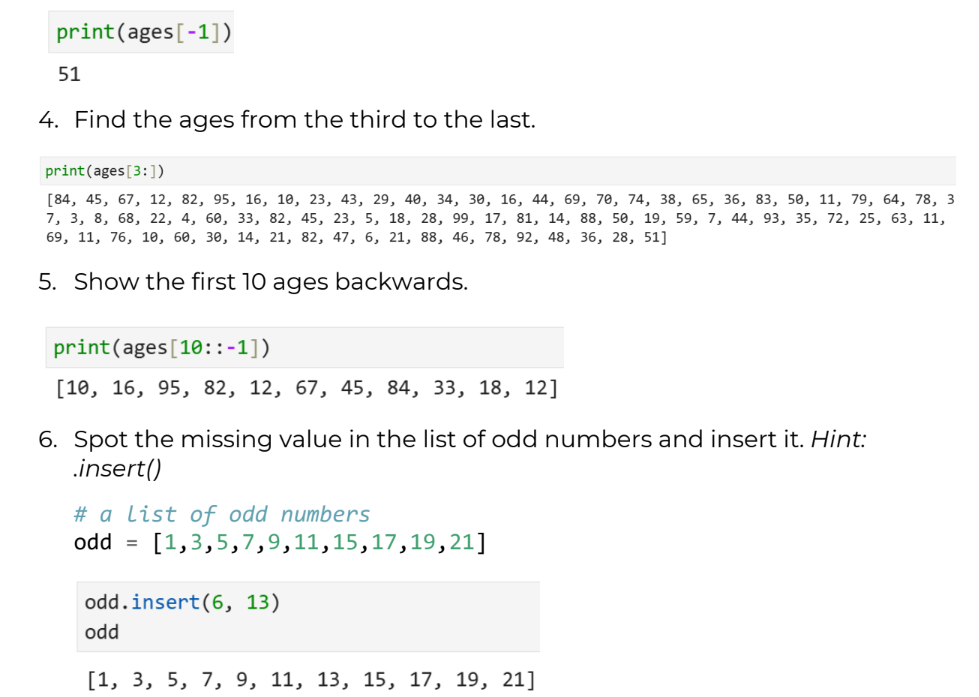
* Perform various operations on lists and strings.
* Perform operations related to tuples, dictionaries, and sets.
* Define and call functions.

### Lists

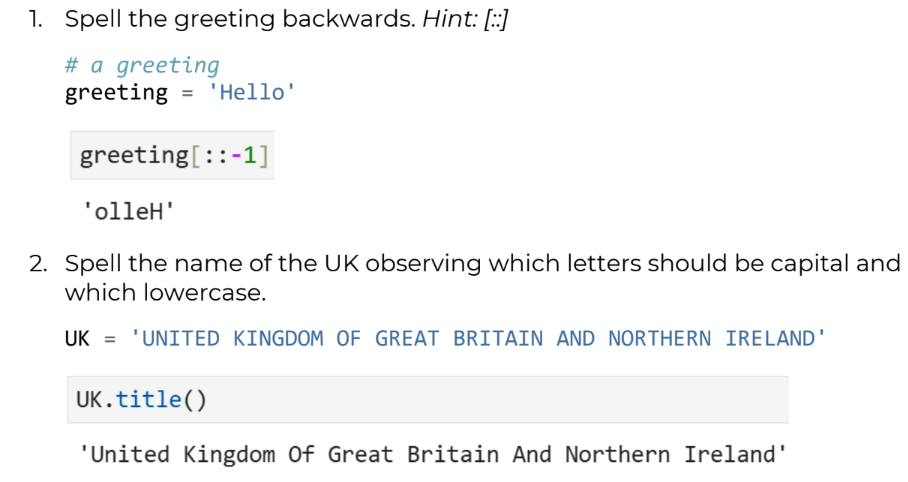
ages =

[12,18,33,84,45,67,12,82,95,16,10,23,43,29,40,34,30,16,44,69,70,74,38, 65, 36, 83, 50, 11, 79, 64, 78, 37, 3, 8,68, 22, 4, 60, 33, 82, 45, 23, 5, 18, 28, 99, 17, 81,14, 88, 50, 19, 59, 7, 44, 93,35,72,25,63,11,69,11,76,10,60,30,14,21,82,47,6 , 21, 88,46,78,92,48,36,28,51]





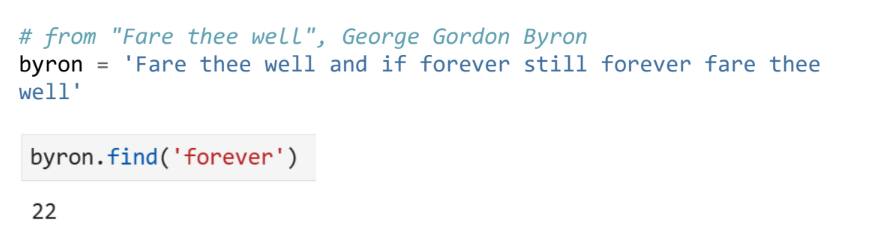
### Strings



3. Find the position of the first occurrence of the word ‘forever’.

# from "Fare thee well", George Gordon Byron

byron = 'Fare thee well and if forever still forever fare thee well'



### Functions

Having created a function that calculate the area of a circle with:

def AoC(r):

pi= 3.14

return pi\* (r \*\* 2)

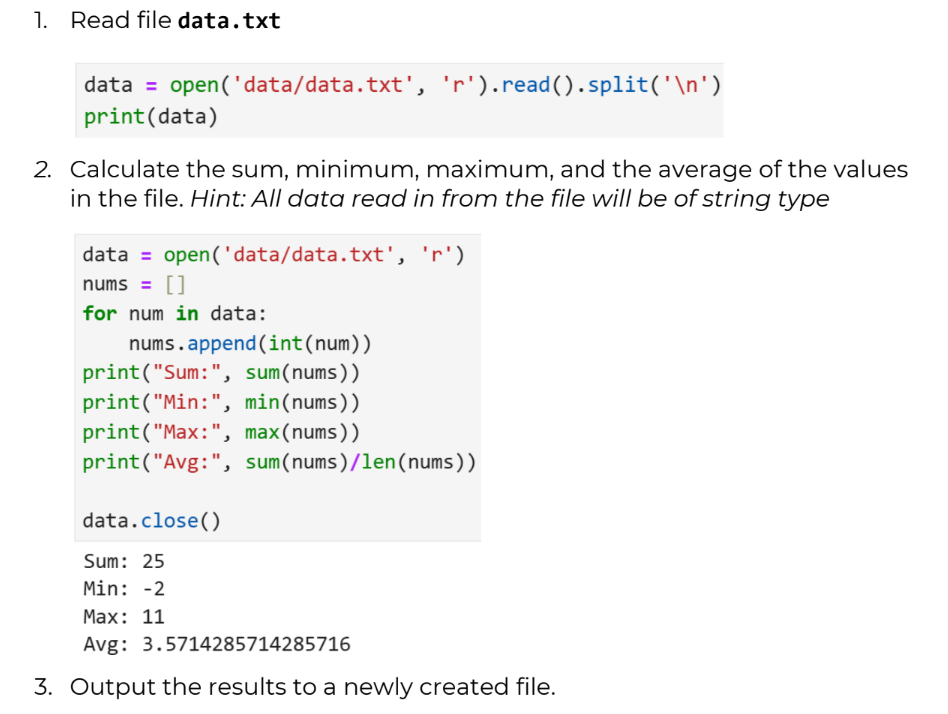
1. Create function that converts Fahrenheit to Celsius

| def f2c(f):  return (f - 32) \* 5/9 |
| --- |

1. Validate the datatype of the input for both functions.

| include int() or float() or use type() in (int, float, money) to check within the function |
| --- |

### File handling





## Introduction to Pandas

### Reading in data

1. Read the file mortgage\_applicants.csv, which sits in the data folder,

into a variable called mortgage.

Hint: Your path will need to reflect the location of the file.

| mortgages = pd.read\_csv('data/mortgage\_applicants.csv')  mortgages |
| --- |

2. The same data sits in an excel file called mortgage\_applicants.xlsx, in a

sheet called PrevYear. Read that into another DataFrame called

mortgage\_excel.

| mort\_xls = pd.read\_excel('data/mortgage\_applicants.xlsx')  mort\_xls |
| --- |

3. Some weather data is held in a file called weather\_data.json. Read it

into a dataframe called weather.

| weather = pd.read\_json('data/weather\_data.json')  weather |
| --- |

### Changing types and parsing times

1. What data type has each column in mortgage been read in as? Use a

method to find out.

| mortgages.dtypes |
| --- |

2. Convert the ID column so that it is instead represented as a Unicode

string. Hint: the type is denoted ‘string’

| mortgages['ID'] = mortgages['ID'].astype(str)  mortgages.dtypes |
| --- |

3. Convert the day column of the weather Dataframe to an appropriate

Type.

| weather['day'] = pd.to\_datetime(weather['day']) |
| --- |

### Querying DataFrames

1. Compute the monthly earnings of mortgage applicants, just using

Income.

| mortgages['monthly\_income'] = round(mortgages['Income']/12,2)  mortgages |
| --- |

1. Compute the ratio of debts to assets for each mortgage applicant.

| mortgages['debt\_to\_asset'] = mortgages['Debt'] / mortgages['Income']  mortgages |
| --- |

1. Display all mortgage applicants who have a Balance greater than

£1000.

| mortgages[mortgages['Balance']>100] |
| --- |

1. Display all mortgage applicants who have a Balance greater than

£1000 and a Debt below £50.

| mortgages[(mortgages['Balance']>1000) & (mortgages['Debt']>50)] |
| --- |

1. Display all loan applicants who have an Income greater than 30,000

who have a 10-year loan, and those with an Income greater than 20,000

who have a 20-year loan (together!)

| mortgages[((mortgages['Income']>30000) & (mortgages['Term'] == '10 Years')) | ((mortgages['Income']>20000) & (mortgages['Term'] == '20 Years'))] |
| --- |

### Aggregating DataFrames

1. Compute the average Balance of mortgage applicants depending on

the Term of their loan.

| mortgages.groupby("Term")['Balance'].transform(np.mean) |
| --- |

2. Compute the average Income of mortgage applicants depending on

whether they defaulted or not.

| mortgages.groupby("Default")[‘Income’].transform(np.mean) |
| --- |

3. Compute the average Debt, Income, and Balance of mortgage

applicants based on whether they defaulted or not, and the term of

their loan.

|  |
| --- |

4. Add a column to the DataFrame called MeanTermIncome, which

contains the mean Income of mortgage applicants based on their Term.

|  |
| --- |

### Data Cleaning with Pandas

Load in the dataset renfe\_trains.csv.

**Initial data inspection**

1. Inspect the columns of the DataFrame. Specifically, consider the type of

each column and whether it seems reasonable. If not, investigate why.

| trains = pd.read\_csv("data/renfe\_trains.csv")  trains  trains.dtypes |
| --- |

2. It seems like we have some bad values in the price column with the

value ‘price’.

You can see them by using the method .value\_counts().

Inspect the specific rows where this is the case.

| trains['price'].value\_counts()  trains[trains['price'] =='price'] |
| --- |

3. It looks like some sort of error has meant the column names have been

fed into the data in intervals. Let’s drop these rows as they are clearly an

accident.

| trains\_clean = trains[trains['price'] !='price']  trains\_clean |
| --- |

4. We can now represent price using the appropriate type. Convert it to

the appropriate data type.

| trains\_clean['price'] = trains\_clean['price'].astype(float)  trains\_clean.dtypes |
| --- |

**Missing values**

1. Identify whether there are missing values in the DataFrame.

| trains\_clean.isnull().any() |
| --- |

2. Which columns are they in?

|  |
| --- |

3. Inspect some rows which contain them.

| trains\_clean[trains\_clean['vehicle\_class'].isnull()] |
| --- |

4. Drop all rows which have missing `vehicle\_class` and `price` and

`fare`. Hint: how=’all’

| trains\_clean2 = trains\_clean.dropna(subset=['vehicle\_class','price','fare'], how='all')  trains\_clean2 |
| --- |

5. Run the below code. What does it suggest about ticket price with

respect to vehicle\_class and fare?

df[['vehicle\_class', 'fare', 'price']].groupby(['vehicle\_class',

'fare']).mean()

|  |
| --- |

6. Fill the remaining missing price values with the mean of all the prices.

| trains\_clean2 = trains\_clean2.fillna({"price": np.mean(trains\_clean2['price']).round(2)})  trains\_clean2.iloc[11] |
| --- |

7. Check you have gotten rid of all NaN values in df.

| trains\_clean2.isnull().any() |
| --- |

**Deduplication**

1. Use duplicated to see whether the dataset contains any duplicated

rows.

|  |
| --- |

2. As the dataset constitutes ticket price search results, there’s a good

chance duplication has come about due to the data collection method.

E.g., there are many tickets available on each train.

We would typically investigate this further, but in order to see the

functionality pandas offers, just remove the duplicate rows

|  |
| --- |

**Extension:**

**Additional string data manipulation**

1. Capitalise the word renfe in the company column.

|  |
| --- |

2. Generate a statistical summary of trips whose vehicle\_class contains

Turista.

|  |
| --- |

1. Display all rows whose fare ends with a +

|  |
| --- |

## Data Manipulation with Pandas

Load in the dataset renfe\_trains\_cleaned.csv.

**Pivot tables**

1. Use a pivot table to explore how price differs with respect to the type of

fare for each destination.

**Working with time series**

1. Convert departure & arrival to a more appropriate datatype.

|  |
| --- |

1. Calculate the duration of each train journey and add it as a column

called duration.

|  |
| --- |

1. Make departure the index of the DataFrame.

|  |
| --- |

1. Sort the index low to high (earlier to later). This will make slicing

possible later.

|  |
| --- |

1. Create a subset of the DataFrame called madrid\_to\_barca which

contains only journeys with origin as MADRID and destination as

BARCELONA.

|  |
| --- |

1. Select only those tickets in madrid\_to\_barca which are in the Promo

category for fare and Turista for vehicle\_class. Update

madrid\_to\_barca to only contain these.

|  |
| --- |

1. Compute a seven day rolling average for price for the madrid\_to\_barca

DataFrame. Add it as a column called rolling.

(To try after completing the combining tables section) Plot the rolling

average vs. the actual values of price.

|  |
| --- |

**Combining tables**

1. Read in the fare\_conditions.csv file. It contains the conditions for

the type of ticket that has been purchased.

|  |
| --- |

2. Add the fare conditions to the original df DataFrame.

|  |
| --- |