# Foundation

## Advanced Python

## Weeks 2 & 3

## Module 3 Exercises

| Version | Date | Author | Comments |
|---|---|---|---|
| 1.0 | 26 / 11 / 21 | Nikola Ignjatovic | First draft |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Module 3A: NumPy

1. a) Create a 2-D array of 100 elements: 1 – 100 with 25 rows and 4 columns.

   b) Use two ways of displaying the dimensionality of the above array

   c) Write the statement that returns number 77 from the above 2-D array

   d) Reshape the above 2-D array to a 3-D array with 10 rows and 5 columns, where each 1-D array contains 2 elements

   e) Write the statement that returns number 77 from the above 3-D array

   f) Reshape the above array to a 3-D array with 5 rows and 2 columns, where each 1-D array contains 10 elements

   g) Write the statement that returns number 77 from the above 3-D array

   h) Reshape the above array to a 3-D array with 2 rows and 10 columns, where each 1-D array contains 5 elements

   i) Write the statement that returns number 77 from the above 3-D array

   j) Reshape the above array to a 3-D array with 20 rows and 5 columns, where each 1-D array contains 1 element

   k) Write the statement that returns number 77 from the above 3-D array

   l) Flatten the last 3-D array

   m) Write the statement that returns number 77 from the above 1-D array

2. Write a Python function called `generate_2D_arr`, that accepts 2 parameters:

   - a positive integer (r), representing the number of rows
   - a positive integer (c), representing the number of columns

The function does not return any value but instead creates and prints a 2-D array of elements 0, 1, 2, ..., r*c-1, and prints out all rows and all columns of the array.

3. Write a Python function to:

   a) square numbers stored in a 1-D array.

   b) square root non-negative numbers stored in a 1-D array.

4. Write a Python function to find invalid marks stored in a 1-D array. The mark is invalid if it is less than 0 and if it is greater than 100.

5. Write a Python function to find valid marks stored in a 1-D array. The mark is valid if it is between 0 and 100.

6. Write a Python function to find all

   a) even

   b) odd

   integers stored in a 1-D array.

7. Write a Python function to test whether each element of a 1-D array is also present in a second array. The function should return common values between the two arrays.

8. Write a Python function that accepts 3 arguments: the 2-D array of strings, a string containing the value looked up for in the first column, and an integer representing the column where the corresponding value is to be found (1

represents 1st column, 2 represents 2nd column etc.). The function looks for the lookup value in the first column of the given 2-D array and returns the corresponding value in the given column (implementing the Excel VLOOKUP function with the exact match). The function returns the value '-1' (as string) if lookup value is not found in the first column, and 'None' (as string) if the given column does not exist within the data set. Note that the function always returns a string: it contains either the found value, '-1' or 'None'. The function accepts an array but the data set is given in the file data_set_1, with TAB separating the data.

9. Write a Python function that accepts 4 arguments:

   1) the array that includes headings

   2) the heading name where the value to be looked for will be found

   3) the value to be looked for

   4) the heading where the corresponding value to be returned will be found (implement the Excel INDEX function with 2 nested MATCH functions). The function returns None if the 2nd or 4th argument does not exist in the first row of the data set and -1 if the value looked for is not found in the column headed by 2nd argument. The function always returns a string: it contains either the found value, None or -1.
   **Note:** The function accepts an array but the data set is given in the file data_set_1.

10. A company assembled its products sale over 4 quarters of a year, stored in the flat file data_set_2. It wants to calculate:

    a) the total of sales for the whole year, for each product. The expected results are in format of 2-D array of 8 rows and 2 columns with the following headings: Products; Total Sales

b) the total of sales for each quarter. The expected results are in format of 2-D array of 2 rows and 4 columns with the following headings:

Total Sales: Q1;Total Sales: Q2;Total Sales: Q3;Total Sales: Q4

c) the overall total sales for that year. The expected result is in format of one int value.

Write a Python function for each of the three calculations. From the calling environment (the main() function), save the outputs of functions for questions 10a and 10b to a text file.
**Note:** The sample flat file data_set_2 has 8 rows and 5 columns. The solution must work for any number of rows and any number of columns.

11. A company is looking at its products sale over 4 years. It wants to calculate the total income from sales for each year as follows:

unit price x number sold per year.

The sample flat file data_set_3 contains the list of products with their price, as well as the number of sales per each product for each of the 4 years.
Write a Python function that calculates the total income from sales per each product for each of the 4 years and returns the 2-D array with 8 rows and 5 columns, where the 1st row contains the following headings:

Product;Total Income: Yr1;Total Income: Yr2;Total Income: Yr3;Total Income: Yr4

The returned array then needs to be written to a file in form of a data set from the calling environment - the main() function.

**Note:** The sample flat file data_set_3 has 8 rows and 6 columns. The solution must work for any number of rows and any number of columns.

12. A company specialising in maritime transport material has business across 20 countries in 6 regions, as shown in the table below:

| Region | Countries |
|---|---|
| Africa | Angola, Cameroon, Tanzania, South Africa |
| Asia | India, Japan, Singapore, South Korea |
| Europe | France, Greece, Italy, Spain, Portugal |
| North America | U.S.A. |
| Central America | Cuba, Mexico, Panama |
| South America | Argentina, Brazil, Venezuela |

The file "sales.csv" stores customers' details who traded with the company, listing their country of residence, date of birth and sales amount. Write the function `number_of_sales_per_region_and_age()` that accepts the array obtained from loading the dataset present in sales.csv file, and returns the dictionary of dictionaries, showing the number of sales within each region by age range. To work out the age range of each customer extract the year, month and day of the current date through the following code:

```
import datetime
current_datetime = datetime.datetime.now()
current_year = current_datetime.year
current_month = current_datetime.month
current_day = current_datetime.day
```

Then use slicing to extract the year, month & day from the date_of_birth (a string) and work out the age following this algorithm:
if the current month is less than the birth month, or they are equal but the current day is less than the birth day, then the age is equal to the current year minus the birth year minus 1; otherwise the age is the current year minus the birth year.

Customer's age ranges are: adult, if 18 <= age < 40; middle-aged, if 40 <= age < 65, and elderly, if age >= 65.

For example, for the data set in sales.csv file, the function should return the dictionary:

{'Africa': {'adult': 67, 'middle-aged': 86, 'elderly': 67},
'Asia': {'adult': 79, 'middle-aged': 83, 'elderly': 81},
'Europe': {'adult': 95, 'middle-aged': 106, 'elderly': 80},
'North America': {'adult': 20, 'middle-aged': 16, 'elderly': 17},
'Central America': {'adult': 50, 'middle-aged': 53, 'elderly': 62},
'South America': {'adult': 53, 'middle-aged': 48, 'elderly': 48}
}

13. Write a Python function that returns the last two elements from each Nth dimension in a N-D multidimensional array ( N >= 1). If the Nth dimension has less than 2 elements the function returns `None`. For example, for the 3-D array shaped as (2, 2, 3):

```
[[[ 1,  2,  3], [ 4,  5,  6]],
 [[ 7,  8,  9], [10, 11, 12]]]
```

the function should return the last 2 elements from each 3rd dimension:

```
[[[ 2,  3], [ 5,  6]],
 [[ 8,  9], [11, 12]]]
```

Another example: for the 3-D array shaped as (2, 3, 1):

```
[[[1], [2], [3]],
 [[4], [5], [6]]]
```

the function should return `None`, as the last (3rd) dimension has 1 element.

**Tip:** Compose the expression to be returned within a string using slicing. For example, the expression that needs to be returned in case of the first example – the 3D array named arr_2_2_3 and shaped as (2, 2, 3), is <u>one</u> of the following:
arr_2_2_3[:, :, 1:]
arr_2_2_3[:, :, 1:3]
arr_2_2_3[:, :, [1, 2]]

(all from the 1st and 2nd dimensions and the last 2 columns, indexed as 1 and 2, from the 3rd dimension).

Find out how the `eval()` function works and use it to evaluate the expression in the return statement.

14. Write a Python function that returns the last two elements from each of the highest dimension that contains at least 2 elements in a N-D multidimensional array ( N >= 1). If there is no dimension with at least 2 elements, the function returns `None`.

For example: for the 3-D array shaped as (2, 3, 1):

```
[[[1], [2], [3]],
 [[4], [5], [6]]]
```

the function should return the last 2 elements of each 2nd dimension (as the 3rd dimension has only 1 element):

```
[[[2], [3]]
 [[5], [6]]]
```

Another example: for the 2-D array shaped as (1, 1): `[[5]]` the function should return `None`, as each of the two dimensions has 1 element.

**Note**: for arrays where the last dimension has at least 2 elements this function will return the same array as the solution in task 3. For arrays where the last dimension has less than 2 elements, this function looks at the highest dimension that has at least 2 elements. If there is such a dimension, it returns its last 2 elements; if there isn't, it returns `None`.

**Tip**: Here, the expression that needs to be returned in case of the first example – the 3D array named arr_2_3_1 and shaped as (2, 3, 1), is <u>one</u> of the following:
arr_2_3_1[:, 1:]
arr_2_3_1[:, 1:3]

arr_2_3_1[:, [1, 2]]
(all from the 1st dimension and the last 2 columns, indexed as 1 and 2, from the 2nd dimension, ignoring the 3rd dimension, as it has 1 element).

# Module 3B: Pandas

1. Think about correlations that might exists for the planets dataset.

   a) For example do you think there might a correlation between the mass of a planet and the force of its gravitational pull?

   b) How about the mass of the planet and the number of moons orbiting that planet?

   c) Or the mass of a planet and the distance of the planet from the sun?

   Run the corr() function; do you always get expected correlations or do some of the answers surprise you?

2. Use Pandas to list the first 3 closest planets to the sun, sorted in ascending order (the closest first).

3. You are provided with a file called planet_colours.csv. Examine its contents, you will see that the file contains a list of all the planets and their colours. Your job is to merge the initial DataFrame provided with another DataFrame based on this new file and output a list of planet name, mass, and the planets colours.
   You will need the following code in your script to prevent data truncation:

```
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', None)
```

4. Print a list including the name of the planets, their mean temperature and distance from the sun.

5. Sort the previous query by the mean temperature in ascending order.

6. Print a list of planet names and the length of the day (this will be in hours).

7. Now add a new column that shows the length of the day in terms of days.

8. Write a python script that uses Pandas to print a list of planet names and their mass, densities and volumes. Volume = Mass/Density.

9. List the planets that have the mean temperature above the average mean temperature of all planets.

10. List those planets which have more moons than Neptune. Sort the result by number of moons in descending order.

11. Using the titanic built-in dataset, show the number of man, women and children in each passengers class (the number of each passenger type split by class).

12. Add and populate two new columns to the above DataFrame: 'survived' and 'died' and place them to show the type of person, class, 'survived', 'died', 'total'.

13. Using the 'titanic' built-in dataset, calculate the number of passengers by class, for each value of survival (0, 1), split by each age group following the sinking of the titanic

a) creating a new data frame using queries, consisting of the following columns: class, age_range, survived and a column listing the number of passengers for each (class, survived, age_range) combination

b) creating and populating a new column within the existing data frame
step 1: define a function set_age_range() that will populate the new column 'age_range' with the different age ranges, as defined below
step 2: create a new column in the titanic data frame and populate it passing the custom-made set_age_range() function to the apply() method

c) creating and populating a new column within the existing data frame, but using the built-in Pandas function cut() instead of applying the custom-made function to populate the column. Use the where() NumPy function to replace NaN values with 'unknown' in the age_range column.

Categorise the age ranges using the following categories:
0-18
18-25
26-34
35-44
45-54
55-64
65+
unknown