TØ 5. Extra

Table of contents

This note describes arrays and DataFrames - both of which are collection types that offer additional features compared to lists that are very useful for scientific data.

1 Data collections: Arrays

Another very common collection is an *array*. Arrays are typically used for numerical data as they make computations simple as they enable *element-wise calculations*. This means that we can perform the same calculation for every element in an array.

Consider for example converting distances in kilometers to distances in meters.

```
import numpy as np

distance_in_km = np.array([1.43, 75.12, 9.042, 1.337])
distance_in_m = distance_in_km * 1000
print(distance_in_m)
```

```
[ 1430. 75120.  9042.  1337.]
```

This works with all common arithmetic operations

- Addition: `my_array + 10`
- Subtraction: `my_array - 10`
- Multiplication: `my_array * 10`
- Division: `my_array / 10`

1.1 Exercise: Concentration conversion

In an experiment you have measured the concentration a solution in units of μM (micromolar) but you need to work with them in millimolar. Use the cell below to convert the concentrations.

```
import numpy as np

conc_uM = np.array([137.03, 205.1, 56.4, 732.2, 1563.3])

conc_mM = _______
```

```
print("Concentrations in millimolar: ", conc_mM)
```

2 Data collections: DataFrames

Lists lets us avoid having a huge amount of variables, arrays extend that to also let us make calculations across the entire dataset - however we are still left with only an implicit connection between different arrays.

We can go one step further and work with `DataFrames` which are kind of an extension of an excel-document in Python.

```
import pandas as pd

df = pd.DataFrame({'wavelengths': [400, 401], 'adsorptions': [2.451, 2.532]})
print(df)
```

```
   wavelengths  adsorptions
0          400        2.451
1          401        2.532
```

Now the full dataset is contained in one variable (here called `df`) - this makes working with a dataset much easier. A `DataFrame` can also be indexed, for example we extract the `wavelengths` like so

```
df['wavelengths']
```

```
0    400
1    401
Name: wavelengths, dtype: int64
```

So with a `DataFrame` indexing uses the name of the row or column rather than the numeric index we've seen for lists and arrays.

2.1 Exercise: Calculating cell sizes

The code below creates a dataset using a `DataFrame` with cell types and the upper and lower limits of their radii.

```
import pandas as pd

cell_data = pd.DataFrame({
    'cell_type': ["Mycoplasma", "Typical bacterium", "Yeast cell", "Red blood cell",
    "Lymphocyte", "Typical animal cell", "Typical plant cell", "Human egg cell (oocyte)",
    "Neuron (cell body)"],
'radius_lower_um': [0.2, 1, 5, 7, 8, 10, 20, 120, 10],
'radius_upper_um': [0.3, 2, 7, 8, 10, 20, 100, 140, 100]}).set_index('cell_type')

print(cell_data)
```

We would like to calculate the volume of these cells, assuming they are spherical

$$V(r) = \frac{4}{3}\pi r^3$$

2.1.a Calculate the volume of a myocoplasma

We will start by making the calculation as if Python was almost just a normal calculator.

Put in the correct lower limit of the radius and calculate the volume of the cell in the interactive cell below.

Here you are not expected to make use the `cell_data` variable.

```
pi = 3.1415
mycoplasma_radius = ______ # Write the correct radius
mycoplasma_volume = ______ # Calculate the volume

print("The volume is", mycoplasma_volume)
```

2.1.b Calculate the volume of mycoplasma - but smarter.

It's kind of silly to create the `cell_data`-table (or rather `DataFrame`) and then not use it. So now we want to extract the lower limit of the radius of the mycoplasma cell from the `DataFrame` rather than writing it manually.

To do so we need to use *indexing* to extract the entry we are interested in.

```
pi = 3.1415
mycoplasma_radius = cell_data['radius_lower_um'][______]
mycoplasma_volume = ______

print("The volume is", mycoplasma_volume)
```

2.1.c Calculate the volume of all the cells

Each column in `cell_data` is actually an array, so we can easily calculate volumes of every cell at once.

```
pi = 3.1415

# This extracts the array of lower limit cell sizes.
cell_volumes = ______ * cell_data['radius_lower_um']**3

print(cell_volumes)
```