TØ 5. Data types

Table of contents

One of the most useful aspects of Python is the ability to work with complex datasets. However, to do so it is useful to understand a little bit about **data types**.

In the end a computer stores everything as a binary sequence e.g. `0101001` but Python (and many other programming langauges) add abstractions that makes it easier to reason about our programs. We will start by exploring some primary data types and then look at a simple collection type, the list.

In the extra material we explore other types of collections, namely arrays and dataframes.

1 The primary data types.

We will consider three primary data types

- **Integers** (*whole numbers*): Used to represent whole numbers, for example the number of bases or codons.
- **Floats** (*decimal numbers*): Used to represent decimal numbers, for example a concentration in $\frac{\text{ng}}{\mu\text{L}}$.
- **Strings** (*text*): Used to represent text, for example mRNA-sequences or names.

The code block below shows a variable of each of these types

```
an_integer = 1
a_float = 3.1415
a_string = "Eukaryote"

print('Integer: ', an_integer, type(an_integer))
print('Float: ', a_float, type(a_float))
print('a_string: ', a_string, type(a_string))
```

Tip

Here the function `type` was used to obtain the type that Python uses for each of our variables. You don't need to understand the details of this - just notice that these three pieces of data were automatically understood to be one of the above types.

1.1 Exercise: mRNA and datatypes.

Consider working with an mRNA molecule with sequence `GCAUGCCGUUUGGAUGACCG`.

1.1.a Sequence data type

What Python data type would it be natural to use for an mRNA sequence?

1.1.b Sequence length

What Python data type would it be natural to use for the length of an mRNA sequence?

1.1.c In Python

Using the cell below, how many nucleotides are in the sequence?

```
mrna = "GCAUGCCGUUUGGAUGACCG"
sequence_length = len(mrna)

print("The sequence has length:", sequence_length)
```

> Tip
>
> The `len` function is incredibly useful - here we used it to return the number of characters in a string.

1.1.d mRNA codons

Use the cell the below to calculate the number of codons in the mRNA

```
mrna = "GCAUGCCGUUUGGAUGACCG"
sequence_length = len(mrna)
number_of_codons = _______

print("Number of codons:", number_of_codons)
```

Division and floats

You may have notice that the number printed above is a float not an integer.

This is because Python converts division of integers into floats, for example

```
numerator = 10
denominator = 5
print(numerator / denominator)
```

```
2.0
```

If we know that we want an integer we can either convert back to int or use a different division operation

```
numerator = 10
denominator = 5
print(numerator // denominator)
```

```
2
```

2 Data collections: Lists

The three primary types we saw above can describe a lot of things, however an actual dataset is typically a collection of several measurements/labels/etc. Imagine for example an experiment where the absorption coefficient is measured as a function of wavelength, we could represent that as

```
wavelength_1 = 400
absorption_1 = 2.451
wavelength_2 = 401
absorption_2 = 2.532
...
```

However, its pretty clear that this becomes really annoying to deal with very quickly - so we need something smarter. The simplest collection is a list

```
wavelengths = [400, 401, 402, 403, 404, 405]
absorptions = [2.451, 2.532, 2.567, 2.621, 2.584]
```

This is not too bad, it keeps the wavelengths grouped together and similarly for the absorptions.

Using lists for plotting

Lists are also the type of input that is usually used for plotting, consider for example the cell below that makes a very basic plot.

```
import matplotlib.pyplot as plt

x = [0, 1, 2, 3, 4]
y = [0, 1, 4, 9, 16]

fig, ax = plt.subplots()

ax.plot(x, y, '-o')
```

Line 3

Make a list of x-coordinates.

Line 4

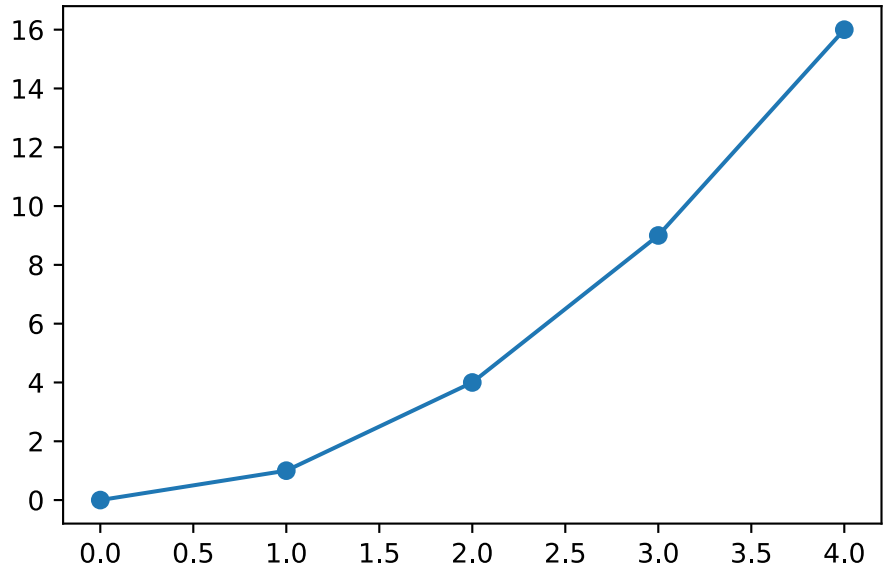
Make a list of y-coordinates.

Line 6

Make the figure (fig) and axis (ax) for plotting

Line 8

Plot the data in the axis.



This code produces the shown figure. You will learn much more about plotting through out your studies as it is an essential part of working with scientific data.

For plotting we use every entry in the lists, but there are many situations in which we want to select a specific element of a dataset - this is achieved through *indexing*.

For example with the `wavelengths`-list we made before, we can index elements like so

```
element_1 = wavelengths[0]
print("The first element is: ", element_1)
```

```
The first element is:  400
```

The indexing happened with the code `wavelengths[0]` where `[0]` specifies we want the first element. This may seem a little surprising, why do we use `[0]`? There is a simple reason for this, explained briefly in the footnote[1].

2.1 Exercise: Indexing a list of molecules

In an experiemnt, several molecules were identified in a sample, use the cell below to

1. Print the first molecule
2. Print the third molecule
3. Print the last molecule

```
molecules = ["glucose", "ATP", "cholesterol", "lactate", "glycine"]

first_index = ______
print("The first molecule is: ", molecules[first_index])

third_index = ______
print("The third molecule is: ", molecules[third_index])

last_index = ______
print("The last molecule is: ", molecules[last_index])
```

2.2 Exercise: Building a sentence

The following list contains parts of a sentence in the wrong order.

Fill in the indices so that the printed sentence is correct.

```
sentence_pieces = [
    "powerhouse",
    "are the",
```

[1] In many programming languages, counting starts at 0 instead of 1. This comes from how computers store lists in memory.

You can think of a list as starting at some location in memory, and each new element being a fixed step away.
The index then tells the computer how many steps to move from the start:

- index `0` → move 0 steps → first element
- index `1` → move 1 step → second element
- index `2` → move 2 steps → third element

This makes it simple and efficient for the computer to find elements. Most modern programming languages have kept this convention.

```
    "mitochondria",
    "of the cell"]

first_index = ______
second_index = ______
third_index = ______
fourth_index = ______


print(
    sentence_pieces[first_index],
    sentence_pieces[second_index],
    sentence_pieces[third_index],
    sentence_pieces[fourth_index]
)
```

3 Data collections: Revisiting strings

Earlier we introduced strings as a basic datatype, but in some ways they can actually also be considered a collection - and therefore strings can also be indexed.

For example, we can print the fourth element of an mRNA sequence like shown below

```
mrna = "GCAUGCCGUUUGGAUGACCG"
print(mrna[3])
```

```
U
```

This also lets us use another form of indexing a *slice*, for example

```
mrna = "GCAUGCCGUUUGGAUGACCG"
codon = mrna[0:3]
print(codon)
```

```
GCA
```

Here `[0:3]` means: start at index 0 and stop *before* index 3.

3.1 Exercise: Find the third codon.

For the `mrna` sequence use a slice to print the third codon

```
mrna = "GCAUGCCGUUUGGAUGACCG"
codon = mrna[______]
print(codon)
```