Lesson Python

Slides Intro

Open Jupyter lab and move to the swc-python folder

- Through Anaconda navigator
- JupyterLab

Lesson start

Variables

- Any Python interpreter can be used as a calculator

```
3+5*4
```

- This is great but more useful is to assign this value to a variable that can be used later. For example, we can track the weight of a patient who weighs 60 kilograms by assigning the value 60 to a variable weight_kg:

```
weight_kg=60
```

In Python, variable names:

- can include letters, digits, and underscores
- cannot start with a digit
- are case sensitive.

For example `0weight` is not a valid variable name, show case it

```
0weight
```

`weight0` is a valid name and `weight` is not the same as `Weight`.

Types of variables

Python knows various types of data. Three common ones are:

- integer numbers (like `weight_kg`)
- floating point numbers, and
- strings

floating numbers

```
weight_kg=60.3
```

strings

```
patient_id='001'
```

Using variables

We may want to store our patient's weight in pounds as well as kilograms:

```
weight_lb=2.2*weight_kg
```

We might decide to add a prefix to our patient identifier:

```
patient_id = 'inflam_' + patient_id
```

Built in Python functions

To carry out common tasks with data and variables in Python, the language provides us with several built-in functions. To display information to the screen, we use the `print` function:

```
print(weight_lb)
print(patient_id)
```

We can display multiple things at once using only one `print` call:

```
print(patient_id, 'weight in kilograms:', weight_kg)
```

We can also call a function inside of another function call. For example, Python has a built-in function called `type` that tells you a value's data type:

```
print(type(60.3))
print(type(patient_id))
```

We can use `#` for comments in python

```
patient_id #this variable describes the id of each patient
```

- go to slides and mention the key points of this part ### Key points
- Basic data types in Python include integers, strings, and floating-point numbers.
- Use `variable = value` to assign a value to a variable in order to record it in memory.
- Use `print(something)` to display the value of something.
- Use # some kind of explanation to add comments to programs.
- Built-in functions are always available to use.

Python Lists

Main question: How can I store many values together?

- Unlike NumPy arrays, lists are built into the language so we do not have to load a library to use them. We create a list by putting values inside square brackets and separating the values with commas:

```
odds = [1, 3, 5, 7]
```

- We can access elements of a list using indices: numbered positions of elements in the list. These positions are numbered starting at 0, so the first element has an index of 0.

```
print('first element:', odds[0])
print('last element:', odds[3])
print('last element:', odds[-1])
```

- Since a list can contain any Python variables, it can even contain other lists.

```
veg = [['lettuce', 'tomato', 'peppers' ],
     ['cucumber'],
     [ 'cilantro', 'zucchini']]
```

- Play with the indexes to access different items of the list of lists

```
veg[0], veg[-2:]
```

- There is one important difference between lists and strings: we can change the values in a list, but we cannot change individual characters in a string. For example:

```
names = ['Curie', 'Darwing', 'Turing']  # typo in Darwin's name
print('names are originally:', names)
names[1] = 'Darwin'  # correct the name
print('final value of names:', names)
```

but:

```
name = 'Darwin'
name[0] = 'd'
```

does not work .

- There are many ways to change the contents of lists besides assigning new values to individual elements:

```
odds.append(9)
print('odds after adding a value:', odds)
```

```
removed_element = odds.pop(0)
print('odds after removing the first element:', odds)
print('removed_element:', removed_element)
```

- go to slides to show the first exercise and keypoints

Keypoints

- [value1, value2, value3, …] creates a list.
- Lists can contain any Python object, including lists (i.e., list of lists).
- Lists are indexed and sliced with square brackets (e.g., list[0] and list[2:9]), in the same way as strings and arrays.
- Lists are mutable (i.e., their values can be changed in place).
- Strings are immutable (i.e., the characters in them cannot be changed).

Analyze Patient Data

- How can I process tabular data files in Python?

Loading data

- Go to slides to present the case of today

Importing useful libraries

```
import numpy
```

```
numpy.loadtxt(fname='data/inflammation-01.csv', delimiter=',')
```

- assign the data to a variable `data`

```
data = numpy.loadtxt(fname='inflammation-01.csv', delimiter=',')
```

- explore the data

```
print(data)
```

- print the data dimensions

```
print(data.shape)
```

- extracting a single number from the data array
- go to slides to see how python arrays are handled

```
print('first value in data:', data[0, 0])
```

Slicing data

An index like `[30, 20]` selects a single element of an array, but we can select whole sections as well. For example, we can select the first ten days (columns) of values for the first four patients (rows) like this:

```
data[0:4, 0:10]
```

- We also don't have to include the upper and lower bound on the slice. If we don't include the lower bound, Python uses 0 by default; if we don't include the upper, the slice runs to the end of the axis, and if we don't include either (i.e., if we use ':' on its own), the slice includes everything:

```
data[:3, 36:]
```

Analyzing data

- Finding the mean value of inflammation for all patients for all days :

```
print(numpy.mean(data))
```

- `numpy.mean` as buit-in function from the numpy library , that takes an array as an argument
- Let's use three other NumPy functions to get some descriptive values about the dataset. We'll also use multiple assignment, a convenient Python feature that will enable us to do this all in one line.

```
maxval, minval, stdval = numpy.amax(data), numpy.amin(data), numpy.std(data)
```

```
print('maximum inflammation:', maxval)
print('minimum inflammation:', minval)
print('standard deviation:', stdval)
```

- When analyzing data, though, we often want to look at variations in statistical values, such as the maximum inflammation per patient or the average inflammation per day. One way to do this is to create a new temporary array of the data we want, then ask it to do the calculation:

```
patient_0 = data[0, :] # 0 on the first axis (rows), everything on the second (columns)
print('maximum inflammation for patient 0:', numpy.amax(patient_0))
```

- What if we need the maximum inflammation for each patient over all days (as in the next diagram on the left) or the average for each day (as in the diagram on the right)? As the diagram below shows, we want to perform the operation across an axis:

- go to figure from the slides presentation

- To support this functionality, most array functions allow us to specify the axis we want to work on. If we ask for the average across axis 0 (rows in our 2D example), we get:

```
print(numpy.mean(data, axis=0))
```

- so you will get one value per day across all patients , which should be 40

```
print(numpy.mean(data, axis=0).shape)
```

- To have an average value per patient across all days :

```
print(numpy.mean(data, axis=1))
```

- go to slides for the exercise and key points of this part

Key points

- Import a library into a program using import libraryname.
- Use the numpy library to work with arrays in Python.
- The expression `array.shape` gives the shape of an array.
- Use array[x, y] to select a single element from a 2D array.
- Array indices start at 0, not 1.
- Use low:high to specify a slice that includes the indices from low to high-1.
- Use # some kind of explanation to add comments to programs.
- Use numpy.mean(array), numpy.amax(array), and numpy.amin(array) to calculate simple statistics.
- Use numpy.mean(array, axis=0) or numpy.mean(array, axis=1) to calculate statistics across the specified axis.

Visualizing Tabular Data

- How can I visualize tabular data in Python?

```
import matplotlib.pyplot as plt
image = plt.imshow(data)
plt.show()
```

- make sure `data = numpy.loadtxt(fname='inflammation-01.csv', delimiter=',')`

- Visualizing the average values across all patients per day

- go to slides to explain the axis in np.mean

```

ave_inflammation = numpy.mean(data, axis=0)
ave_plot = plt.plot(ave_inflammation)
```

Key points

- Use the pyplot module from the matplotlib library for creating simple visualizations.