# Python Cheat Sheet

Functions and Description

## 1 Basic Python Functions

Importing the libraries. The "as np" part lets you run a function from numpy by writing for example `np.mean()` instead of `numpy.mean()`

```
import numpy as np
import matplotlib.pyplot as plt
```

### Python Lists

```
myList = [] # Defines an empty List
myList = [1,2,3] # Defines a 1D list with numbers 1, 2 and 3
myList[0] # Accesses the first element of the list, this will return 1
myList.append(4) # Adds the new element 4 to the list, the list hence becomes
    [1,2,3,4]
myList.extend([5,6,7]) # Extends the  myList with the elements of the list [5,6,7],
    so myList then becomes [1,2,3,4,5,6,7]
len(myList) # Returns the length of your list.
newList = myList + [8,9] # another way to extend a list. newList will be
    [1,2,3,4,5,6,7,8,9]. Note that myList + 8 does NOT work (you would have to write
    newList + [8]
```

### Python Functions

```
def functionName(param1, param2=defaultValue):
    # Define what function does
    # By default, param2 is equal to the defaultValue
    return param1**2+param2 # Return the result, this is an example
```

You can then call the function from elsewhere in the code. For example, if you later write

`y = functionName(4,3)`

then y will be set to $4^2 + 3$ as `**` is the exponent/power operator in Python.

### Python Loops

An iterable can be a list, a numpy array, and many other things. You can use the iterable `range(N)` to make item go from 0 to N-1 (i.e. N loops). For the while loop, the condition is a logic statement for example `myVariable < 5`. The loop will run as long as this statement is true (which can be for ever).

```
# For loop
for item in iterable:
    # Code block

# While loop
while condition:
    # Code block
```

### If-else Statements

You can use only if and else (but having an else is mandatory) and add as many `elif` as you want.

```
if condition:
    # Code block
elif condition:
    # Code block
else:
    # Code block
```

A compact notation that is sometimes useful is for example

`y = 2*x if x>0 else 0`

which sets y to 2*x for positive x and to 0 for negative x.

### Common Built-in Functions You Should Know

- `len()`, `range()`,
- `print()`
- `sum()`, `min()`, `max()`

### Important Variable and Data Types

- Numbers: `int`, `float`
- Text: `str`
- Boolean: `bool` (`True`/`False`)
- Containers: `list`, `dict`

## 2 Numpy Library

### Creating Arrays

```
myArr = np.array([1, 2, 3]) # Creates a 1D NumPy array called myArr with values 1,2,3
zeros = np.zeros((2, 2)) # Creates the 2D 2x2 NumPy array filled with zeros
ones = np.ones((3, 3)) # Creates the 2D 3x3 NumPy array filled with ones
linspace = np.linspace(0, 10, 5) # Creates 5 equally spaced points between 0 and 10 (
    including 0 and 10)
arange = np.arange(0, 10, 2) # Creates points in the range between 0 and 10 with  the
     spacing of 2 (including 0 but not 10)
```

### Selecting Elements

```
myArr[0] # Selects the first element
myArr[1:5] # Selects elements from second to fifth.
myArr[[2,4,5]] # Selects 3rd,5th and 6th element
myArr[-1] # Selects last element
myArr[index] # Selects elements from ‘myArr‘, which are listed in array ‘index‘ if
    index contains only integers.
myArr[condition] #Selects only elements that fulfil a condition. For example myArr[
    myArr>4] will select only elements that are >4
```

## NumPy Array Operations

Note that these are **different from the list operations intorduced above.**

```
arr1 + arr2 # Element-wise summation
arr1 - arr2  # Element-wise subtraction
arr1 * arr2 # Element-wise multiplication
np.dot(arr1, arr2) # Matrix multiplication
```

## Common NumPy Functions

```
np.sum(myArr) # Sums all the elements
np.max(myArr) # Returns the max element
np.min(myArr) # Returns the min element
np.mean(myArr) # Returns the mean value of all elements
np.std(myArr) # Returns the biased standard deviation of elements
np.std(myArr, ddof=1) # Returns the unbiased standard deviation of elements
np.histogram(myArr, bins) # Returns the frequencies of the histogram defined by the
    bins (bins can be a number or an array, see the documentation of this function
    for details.
np.count_nonzero(myArr) # Returns the number of non zero elements in the array
```

```
# All functions can be applied along a certain axis, e.g.
np.sum(myArr, axis=0) # Sums all the elements in each column - axis 0
np.sum(myArr, axis=1) # Sums all the elements in each row - axis 1
# If an axis is given (e.g. axis=1), the resulting array will have the shape of the
    remaining axis.
```

## Generating Random Numbers

```
np.random.randint(low, high, shape) # Returns a np array of a certain shape, filled
    with random integers between low and high.
np.random.normal(mean, std, shape) # Returns a np array of a certain shape, filled
    with numbers following a normal distribution N(mean, std**2)
np.random.uniform(low, high, shape ) # Returns a np array of a certain shape, filled
    with numbers following a uniform distribution between low and high
```

## Shape Manipulations

```
np.shape(myArr) # Returns the shape of your array, for example (4,2) if you have 4
    rows and 2 columns
np.reshape(myArr, newShape) # Reshapes the array to a newShape while saving the total
     number of elements in the original myArr. I.e. if myArr.shape = (10, ), thennp.
    reshape(myArr, (5,2)) will give the new array with the shape of (5,2).
```

## Loading Data

```
myArr = np.loadtxt(path, delimiter, skiprows, usecols, dtype) # Loads the data from a
     text file in ‘path‘, using the symbol in ‘delimiter‘ to separate columns,
    skipping first ‘skiprows‘ rows, reading only ‘usecols‘ columns (i.e. usecols=(1,
    3) - will read only columns 2 and 4), and coverting data to ‘dtype‘ in the end.
```

# 3 Matplotlib Library

## Basic Plotting

This code will create a line connecting the points (1,4), (2,5), and (3,6), with round markers at each point (if you do not specify "marker" it will just be a line). You can use lists or numpy arrays for the data.

```
x = [1, 2, 3]
y = [4, 5, 6]
plt.plot(x, y, marker="o")
plt.title('Title')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.show()
```

If you want to plot several data sets together, you can write for example

`plt.plot(x,y2)`

after the first plot command. Python will automatically choose different colours.

## Other Plots

```
plt.scatter(x, y) # Creates points instead of a line
plt.bar(x, y) # Creates vertical bars located at positions in x array with heights
    defined in y array.
plt.hist(data, bins=10) # Creates a histogram of points in data array. The number of
    bins can be set by ‘bins‘ argument. Instead of a number, you can also provide an
    array of edges.
plt.errorbar(x, y, yerr=yerr) # Creates the scatter-plot of ‘x and ‘y‘ arrays with
    errorbars defined in ‘yerr‘ array.
```

# 4 Other

If you write a function that makes sense for a single number, but not for an array (for example if it contains "if a>0" somewhere but it is not clear if this should hold for each item in the array separately, or for the array as a whole), you will get an error message. To avoid this, the function np.vectorize exists:

```
np.vectorize(functionName) # Returns a version of your function that can be run with
    NumPy arrays.
```

Which you can then run for example like this:

`y = np.vectorize(functionName)(4,2)`

(note the positions of the brackets).