LEARNING PYTHON, PART 2: NUMPY

0. Import Numpy as a library

At the beginning, execute:

```
import numpy as np
```

Then, to execute a function in Numpy, we use the format `np.func()`

1. Numpy Arrays

Numpy has one-dimensional array (vector) and two-dimensional array (matrix).

In this note, `x` is a vector, and `y` is a matrix.

Notes:

- A numpy array can only hold a single type of data.
- Numpy codes are written in C.

Creating vectors

- Change a usual python list to a numpy array

```
np.array([0, 1, 2, 3])
## returns: array([0, 1, 2, 3])
```

- Create a numpy array, with a certain step size. The resulting array does *not include* `end`.

```
np.arange(start = 0, end, step = 1)
```

- Create a numpy array, with a certain length. The resulting array *includes* `end`.

```
np.linspace(start, end, length)
```

- Dot product of two vectors

```
np.dot(a, b)
```

Creating matrices

- Create a matrix, from a python (double) list

```
np.array([[0, 1, 2], [3, 4, 5]])
## returns: array([[0, 1, 2],
                   [3, 4, 5]])
```

- The `reshape` method: convert a vector to a matrix.

```
x = np.arange(6)
y = x.reshape(2, 3)
```

Note 1: elements are filled into the matrix *by row*.

Note 2: if the vector length is not equal to nrow * ncol, an error will occur.

- Create a numpy array (vector or matrix) of all zeros or ones.

```
## vectors
np.zeros(length)
np.ones(length)

## matrices: dimensions are input as a tuple
np.zeros((nrow, ncol))
np.ones((nrow, ncol))
```

- Create an identity matrix

```
np.eye(dim)
```

- Matrix multiplication

```
## Elementwise multiplication
n * m
np.multiply(m, n)  ## equivalent to n * m

## Matrix product
np.matmul(a, b)
```

Draw random samples

- Random samples from Uniform(0, 1) distribution:

```
np.random.rand(n) ## a vector of n samples
np.random.rand(nrow, ncol) ## a matrix of random samples
```

- Random samples from integer Uniform: low is inclusive, high is exclusive. Here, results may contain repeated numbers.

```
np.random.randint(low, high, n) ## vector
```

- Random samples from normal distribution:

```
## standard normal
np.random.randn(n) ## vector
np.random.randn(nrow, ncol) ## matrix
```

```
## normal with mean and sd
np.random.normal(mean, sd, n) ## vector
```

- Simplify function names:

```
from numpy.random import randn
```

- Set seed

```
np.random.seed(101)
```

Other useful functions and methods

- Max (and similarly, min)

```
x = np.arange(6)

## All of the following will reture 5
x.max()
max(x)
np.max(x)
```

- Argmax (and similarly, argmin)

```
## Only the method way works
x.argmax()
np.argmax(x)
```

- Check dimension of an array: no parentheses needed.

```
y.shape ## returns (nrow, ncol)
x.shape ## returns (n, )
```

- Check data type of an array: no parentheses needed.

```
x.dtype
```

2. Indexing and Selecting

For a vector of length n, the indexing starts from 0, and ends at (n-1)!!!

Selecting elements in a vector

- Visit elements in a vector: use bracket `[]`

```
x[start:end]
```

Note 1: As usual, index `start` is inclusive, and index `end` is exclusive.

Note 2: If `start` is omitted, the default value `0` is used. If `end` is omitted, the default value is the length of the array.

- Change elements in a vector

```
x[start:end] = new_values
```

Note: `new_values` can be a single number, or an array of length (`end-start`). If it is a number, then all affected elements are changed to this number.

- Slice of an array: not a copy, but a reference of the original array.

Changing the slice will change the original array, too!

```
x_slice = x[0:2]
x_slice[:] = 10
## the orginal array x becomes array([10, 10, 2, 3, 4, 5])
```

- Create a copy of an array

Changing the copy will NOT change the original array.

```
x_copy = x.copy()
```

Selecting elements in a matrix

- Visit a sub-matrix

```
## Both ways works
y[row1:row2][col1:col2]
y[row1:row2, col1:col2]
```

Note: if we want all rows, we still need to use a `:` before the comma, for example, `y[:, 0]`. Similar rules apply for columns.

Conditional selection

- Boolean array: contains only `True` or `False`; data type `dtype = bool`. For example:

```
x > 2
## returns: array([False,False,False,True,True,True], dtype=bool)
```

Note: applying a similar comparison statement on a matrix will return a boolean matrix.

- Conditional selection: only return results where `True`

```
x[x > 2]
## returns: array([3, 4, 5])
```

Note: applying a similar conditional selection on a matrix will return a boolean *vector*.

- In numpy, if we want to combine two conditional statements, use `&`, `|` instead of `and`, `or`. Also, use parentheses over both conditional statements.

3. Numpy Operations

When working with Numpy arrays, avoid looping over your data whenever possible.

Array operations

- Similar to R, arithmetic operations such as `+`, `-`, `*`, `/`, etc can be applied to each element of the array.
  Note: meaningless operation such as `0/0` will only lead to a warning, but not an error.

- Universal Array Functions:

```
np.sqrt(x)
np.exp(x)
np.log(x) ## note: log(0) will yield a warning
np.sin(x)
np.unique()
np.sort()
np.sum()
np.mean()
np.median()
np.std()
np.transpose() ## equivalent to .T method
```

See this link for more universal array functions.

See this link for statistics functions.

See this link for linear algebra functions.

3. Reading and Writing Data with Numpy

```
## Read from a CSV file
x = np.loadtxt(filename, delimiter = ',')
## Write to a CSV file
np.savetxt(filename, x, delimiter = ',')
```

Note: the `np.loadtxt` function load the entire file to the memory. So when the file is large, it is better to use the `open` function in Python.

In the following table (TO BE CONTINUED),

Function	**Python (Numpy)**	**R**
Create a vector/array	`x = np.array(range(1, 13))`	
Samples from uniform	np.random.rand(n)	runif(n)
Identity matrix	np.eye(dim)	diag(dim)