Quizes, Functions, & Imports

NSC 1002 Workshop 5

17 Nov 2020

Outline

Quizes

- The first quiz (worth 2% of your overall mark) is next week.
- It will last for 15 minutes. You will do it online.
- It will cover concepts from previous workshops.

Functions

- Recall from last time: using computer programs to solve problems is great!
- The problems we encounter often involve repetitive tasks that are too complex or time-consuming to do by hand (e.g., loops)
- Functions allow us to repeat a set of tasks on different data **without** having to re-write code
- A function is a set of instructions that performs a task

Functions

- Python has built-in functions for a variety of tasks:
- E.g. `abs()`, `print()`, `str()`, ...
- Why functions?
  - No unnecessary repetition of code
  - Promotes “modular” behaviour of Python
  - Organization and readability of code

Lambda Function

Simple one-line functions. They have the form:

```
name = lambda arguments: function
```

Example:

```
g = lambda a, x: a*x**2
```

Lambda Function

Simple one-line functions. They have the form:

```
name = lambda arguments: function
```

Example:

```
g = lambda a, x: a*x**2
```

```
>>> g(3,2)
12

>>> g(-1,2.5)
-6.25
```

Function, more general

```
def function_name(arguments):
    """ This is a description of function."""
    statement 0
    statement 1
    return VALUE
pass    # or any  unindented statement. Indicates the end of the function.
```

Function example 0

```
def g(a,x):
    """ Calculate and return a*x**2."""
     y = a * x**2
    return y
#
```

Equivalent

```
def g(a,x):
    """ Calculate and return a*x**2."""
    return a * x**2
#
```

Function example 0

```
def g(a,x):
    """ Calculate and return a*x**2."""
     y = a * x**2
    return y
#
```

Equivalent

```
def g(a,x):
    """ Calculate and return a*x**2."""
    return a * x**2
#
```

```
>>> g(3,2)
12

>>> g(4,-1)
4
```

Function example 0.1

```
def say_hello(name):
    """ This function greets a user """
    return 'Hello {}'.format(name)
```

Function example 0.1

```
def say_hello(name):
    """ This function greets a user """
    return 'Hello {}'.format(name)
```

```
>>> s = say_hello('Norm')
>>> print(s)
Hello Norm
```

Function example 0.2

```
def say_hello(name):
    """ This function greets a user """
    print('Hello {}'.format(name))
```

Function example 0.2

```
def say_hello(name):
    """ This function greets a user """
    print('Hello {}'.format(name))
```

```
>>> say_hello('Norm')
Hello Norm
```

Function example 0.2

```
def say_hello(name):
    """ This function greets a user """
    print('Hello {}'.format(name))
```

```
>>> say_hello('Norm')
Hello Norm

>>> s = say_hello('Norm')
Hello Norm
>>> print(s)
None
```

This is because `say_hello` has no **return** statement! Most functions have a return statement, but it is not required.

Function example 1

```
def add_lists(x0, x1):
    """    Adds lists element-wise if
    len(x0) == len(x1)    """
    if len(x0) == len(x1):
        x_add = [0] * len(x0)   #  initialize list
        for i in range(len(x0)):
            x_add[i] = x0[i] + x1[i]
    return x_add

# Now exexcute it
l1 = [1, 2, 3]
l2 = [4, 5, 6]
xa = add_lists(l1, l2)
print(xa)       # [5, 7, 9] is only printed
```

Numpy and Scipy
Two key packages in Python!

- **NumPy**: **Num**eric **Py**thon
  - Contains routines common to numerics:
  - Defines numy *arrays*!
  - Trigonometric: sine, cosine, deg2rad, rad2deg
  - Matrix operations: eigenvalues, norm, condition
  - Many, many more `http://docs.scipy.org/doc/numpy/reference/`
- **SciPy**: **Sci**entific **Py**thon
  - Contains routines common to scientific computing:
  - Interpolation, signal processing, (Fast) Fourier Transforms (FFT), statistics
  - Also, many more `http://docs.scipy.org/doc/scipy/reference/`

Importing external modules

```
import numpy
```
or
```
import numpy as np
```
This is the recommended way!

or
```
from numpy import cos, sin, pi
```
or ...

~~`from numpy import *`~~

Importing external modules

```
import numpy
```
or
```
import numpy as np
```
This is the recommended way!

or
```
from numpy import cos, sin, pi
```
or ...

~~`from numpy import *`~~

This will import **all** functions from numpy: takes a long time, can conflict with other loaded functions.

Python uses **namespaces** to keep track of function names:
e.g., `np.cos(np.pi)`

Numpy

```
>>> import numpy
>>> x = numpy.array([1, 2, 3])
>>> y = numpy.array([4, 5, 6])
>>> print(x + y)
[5, 7, 9]
```

Numpy

```
>>> import numpy as np
>>> x = np.array([1, 2, 3])
>>> y = np.array([4, 5, 6])
>>> print(x + y)
[5, 7, 9]
```

Numpy

```
>>> import numpy as np
>>> x = np.array([-0.8, -0.6, -1.0, -0.6, -0.9])
>>> print(np.mean(x), np.std(x))
(-0.78, 0.16)
```

- NumPy has **functions** called “mean” and “std”, which return the arithmetic mean and standard deviation of a numpy array

Numpy

```
>>> import numpy as np
>>> x = np.array([-0.8, -0.6, -1.0, -0.6, -0.9])
>>> print(np.mean(x), np.std(x))
(-0.78, 0.16)
```

- NumPy has **functions** called “mean” and “std”, which return the arithmetic mean and standard deviation of a numpy array
- A `numpy array` is similar to a list, but has key differences
  - Holds *homogeneous* data (in this case it **must** be homogeneous)
  - Can be referenced and sliced (e.g. x[0:2] will give us [-0.8, -0.6])
  - If they are the same shape, you can perform mathematical operations (e.g. x + y)
  - Faster than lists (stored together in memory, rather than apart)

External modules

- One of Python's advantages is an extensive list of functions written by its community
  - numpy
  - scipy, includes statistics, e.g., `from scipy import stats`
  - matplotlib — for plotting
  - pandas – for data reading analysis
  - xarray – advanced data analysis
  - and many, many more!

External modules

A library of functions (like numpy) contains modules, which contain functions

`numpy`	
`numpy.random`	`numpy.linalg`
· `numpy.random.randn(N1, N2, ...)`	· `numpy.linalg.norm(A)`
· `numpy.random.rand(N1, N2, ...)`	· `numpy.linalg.svd(A)`

Questions?

Questions?

Now do the worksheet examples!