Introduction to Programming W10

Packages, basics of scientific computing with NumPy

2022/06/10

1 Modules and packages

A **module** is a Python file (with the .py extension) including variable definitions, functions, statements, etc. Modules can be *imported* and reused in other code.

```
import math #import the math module
print(math.pi) #access the variable called pi from math
print(math.e) #access the variable called e from math
```

In the above code, the `import` keyword is used to import the math module and make it available to use. The math module contains a variable called pi (and a lot other like e for Euler's number), which can be accessed with `math.pi`. Here, math also acts as a *namespace* keeping the attributes, functions, variables together for the module, so using only `pi` would result in an error:

```
import math
print(pi) #incorrect usage, will result in an error
```

If we want to import only pi, we must use the `from` keyword:

```
from math import pi
print(pi) #correct, since we explicitly imported pi
```

However, with the above code other attributes, variables are not available in our code:

```
from math import pi
print(e) #would give error
print(math.pi) #would also give error because this is not the way we imported pi
```

All functions, attributed, variables of a module can be imported with `*`:

```
from math import *
print(pi) #correct, because everything was imported
print(e) #same
print(factorial(4)) #using the factorial function from the math module
```

Since everything was imported from math, we did not have to use `math.pi`, `math.e`, `math.factorial`. Usually, it s not a good practice to do this, because a module can have many variables and functions. Also, be careful with how you are naming your own functions because what you import can be overwritten:

```
from math import *
print(factorial(4))

def factorial(number): #using the same name for this function
    return number*2 #(not factorial...)

print(factorial(4))
```

Modules and their attributes can be renamed when imported:

```
import math as m #using the alias "m" for math
print(m.pi)

from math import e as euler_n #using the alias "euler_n" for e
print(euler_n)
```

A Python **package** is a directory (actual folder) of multiple modules and submodules. Sometimes, we call a collection of many packages a *library*, but usually these two words are interchangeable. Practically speaking, a package is still a module and it is not important for us that what we import: packages or modules. There are built-in modules in Python, such as the math module we just used, but other (not built-in) modules must be installed before we can use them. The Anaconda distribution of Python comes with a lot of data science, statistics, etc. packages/modules already installed. Some packages have requirements which mist be checked before installation (e.g., another package, or a specific version of a package). Your own Python files can be imported as well. Learn more about imports and package installation at the following links:
Imports:
https://realpython.com/python-import/
https://docs.python.org/3.9/reference/import.html
Built-in modules:
https://docs.python.org/3.9/py-modindex.html
Package installation:
Anaconda specific: https://docs.anaconda.com/anaconda/user-guide/tasks/install-packages/
Python package creation and installation (in general): https://packaging.python.org/en/latest/tutorials/installing-packages/
Side note: Knowing how to install packages will be important for your studies and graduate research.

2 NumPy

NumPy is a package for scientific computing in Python, widely used in both academic and industrial environments. It is also used in many other packages (i.e., a lot of packages require NumPy to be installed). It is not a built-in package, but comes already installed with the Anaconda distribution (if not specified otherwise at installation). It provides a multidimensional array object, an *ndarray* (short for n-dimensional array). While a *vector* is a one dimensional array (no difference between rows and columns), a *matrix* refers to 2-D arrays. Higher dimensional arrays are referred to as *tensors*. NumPy also includes various functions used in statistical operations, random simulation, and linear algebra.

It is important that unlike lists, NumPy arrays have a fixed size at creation. So arrays cannot grow dynamically as lists. Elements in an array must be from the same data type. Manipulating numerical data is always faster and consume less memory using arrays than using lists. Practically speaking, an array is a grid of values that can be indexed by integers, Booleans, or another array. While the *rank* of an array denotes its dimensions, the *shape* of the array is a tuple giving its size along each dimension. The easiest way to initialize an array is using lists:

```
import numpy as np #import the numpy package
arr_1 = np.array([1,10,15])
arr_2  = np.array([[1.3,2.5,3.5], [1.22, 1.2, 2.33]])
print("This is how our first array looks like:", arr_1)
print("The shape of the first array:", arr_1.shape)
print("This is how our second array looks like:", arr_2)
print("The shape of the second array:", arr_2.shape)
print("The number of dimensions of the second array:",  arr_2.ndim)
print("Python data type of the arrays:", type(arr_1), type(arr_2))
print("NumPy data type of the second array:", arr_2.dtype.name)
print("How many elements are in the second array?", arr_2.size)

arr = np.array([[1,2,3],[4,5]]) #this would be incorrect since dimensions doesn't match
```

Note: NumPy is conventionally imported as "np" to ensure better readability. You must follow this convention in your assignments.The bottom line although will not result in an error just a "warning" because the lists in the nested list do not have the same length, so there is a dimension mismatch.

Other useful NumPy methods:

```
import numpy as np
arr = np.zeros((2,4)) # array filled with zeros of the specified dimensions
print(arr)
arr = np.ones(5) #array filled with 1s of the specified dimensions
print(arr)
arr = np.arange(5) #creates an array with a range of elements
print(arr)
arr = np.arange(5,18,3) #you can specify the first number, last number, and step size
```

```
print(arr)
arr = np.linspace(0,20, num=7) #creates an array with values spaced linearly in an interval
print(arr)
```

Sometimes, it is convenient to directly convert a list to an array or the other way around:

```
my_list = [[1.23,5.44,5.3], [0.1, 5.0, 1.44]]
print(my_list,"\n")
my_array = np.array(my_list)
print(my_array, type(my_array))

arr = np.arange(2,11,2)
print(arr,"\n")
lst = list(arr)
print(lst, type(lst))
```

You can add, sort, reshape, index, etc. arrays:

```
arr = np.array([2,1,6,7,8,9,97,6,8])
print(np.sort(arr))

arr_1 = np.array([1,2,3,4])
arr_2 = np.ones(4)
print("First array:", arr_1)
print("Second array:", arr_2)
print("Added together:", arr_1+arr_2)

arr_1 = np.array([[1,2,3],[4,5,6]])
arr_2 = np.array([[2,3,4],[5,6,7]])
concated = np.concatenate((arr_1, arr_2)) #note that the arrays are passed as a tuple!
print(concated)

arr_1 = np.arange(8)
print(arr_1)
arr_2 = arr_1.reshape(2,4) #reshaping the array by specifying the desired dimensions
print(arr_2)

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print("command arr[1]:", arr[1])
print("command arr[1:3]:", arr[1:3]) #same as lists: start is inclusive, end is exclusive
print("command arr[1][2]:", arr[1][2])
```

You can create random arrays of any specified dimensionality:

```
from numpy import random
#generate a 4x5 2-D array with random integers from 0 to 50
arr_1 = random.randint(50, size=(4,5))
print(arr_1, "\n")

#generate a 1-D array with 4 random floating point numbers
arr_2 = random.rand(4)
print(arr_2, "\n")

#generate a 4x2x3 3-D array with random floating point numbers
arr_3 = random.rand(4,2,3)
print(arr_3) #check how the 3 dimensional structure looks like!
```

Packages have many functions and methods, and it does not make sense to remember them all. That is why packages have *documentations*. The full documentation for NumPy can be found at `https://numpy.org/doc/stable/index.html` . For example, the documentation for `.arange`:
`https://numpy.org/doc/stable/reference/generated/numpy.arange.html?highlight=arange#numpy.arange` .
It is good to remember the basic syntax and functions of useful packages, but when specific problems arise or you are

not sure how to use a method, just google the problem/name of the method and the answer will be in the documentation (same applied to built-in packages/modules, like `math`). Since nowadays we use many packages in computer science, it is not realistic to know everything by heart and never checking documentations.
Although it is outside of the scope of this class to go through all useful NumPy functions and methods, it is recommended that you go over this basic guide found in the documentation (parts involving other packages can be skipped for now):
`https://numpy.org/doc/stable/user/absolute_beginners.html#`