Workshop 10: Functions as arguments of functions & Multi-dimensional arrays

January 11, 2021

You should ensure that you have completed all exercises before the next workshop.

Passing functions as arguments to functions

We have already discussed functions in the last few workshops, but today we consider passing functions as arguments to other functions.

Look at Figure 1.These concepts apply even if the formal arguments are other functions! Consider the following example. Function `f1` and function `f2` are defined with an argument `x`. This argument can be any type of data which can be passed to the numpy functions such as `sin, exp, cos`. Two examples of allowable data are a single floating point number, or an array of numbers. All of the functions evaluate mathematical functions at each point in the x-array and returns another array that contains their value. The 3rd function defined in the code adds two functions together.

Code Example 1 Add functions

```
import numpy as np

def f1(x):
    return np.exp(-x) * np.cos(2*np.pi*x)

def f2(x):
    return np.cos(x)

def f3(x):
    return np.sin(x)

def addfunctions(fa, fb, x):
"""This function adds function b to function a and returns their sum.
"""
    return fa(x) + fb(x)

n = 12
x = np.linspace(0, 1, n)
# This prints out a test of our addfunctions against adding the functions without
# passing them to another function (it should print the same number twice!)

print(f'Add functions {addfunctions(f1, f2, x[4])} {f1(x[4]) + f2(x[4])}')
```

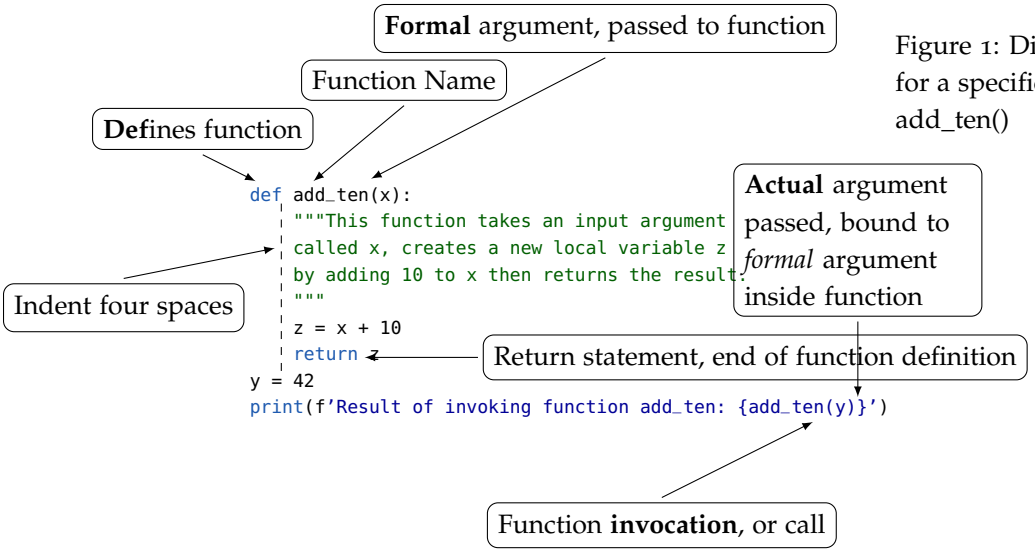

Figure 1: Diagram of the syntax for a specific function called add_ten()

Multidimensional arrays:

Multidimensional arrays are a convenient way to store data when there's more than one parameter associated with the data. As an example, you have already been working with multidimensional arrays when you used the `DataFrame` data type used by Pandas (e.g. the data that contained the population of US states over a number of different years). Before we get started on multidimensional arrays we'll have another look at the one-dimensional arrays we've been working with for the last few weeks.

Code Example 2 numpy array, x of length 5

```
import numpy as np
n = 5
x = np.zeros(5, dtype = 'float')
print('Initial x', x)
# Set the individual values of array
x[0] = 1.5
x[1] = 1.6
x[2] = 1.7
x[3] = 1.8
x[4] = 1.9
print('New x', x)
```

Which will print out

```
Initial x [ 0. 0. 0. 0. 0.]
New x [ 1.5  1.6  1.7  1.8  1.9]
```

You can visualise what the final array looks like to Python in Figure 2. As you already know, by referencing `x[i]`, Python will access

the *i*th element of the array. You can use the Python function `len()` to compute the length of this array. The length of an is equal to the number elements the array contains. In this case, the length of the array is 5.

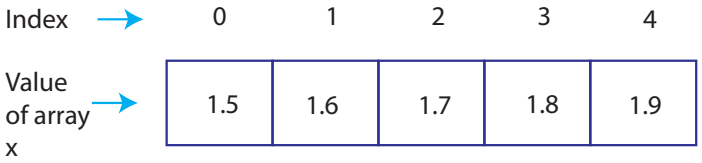

Figure 2: Visual of a 1D array defined in Code Example 2. Visual of a 1D (two-dimensional) array. Printing `x[2]` in Python would yield the value 1.7.

Multi-dimensional arrays are arrays that have more than one dimension. Consider the Figure 3, where there are now two indices: one is associated with the number of rows, and the other is associated with the number of columns. The following code is used to define this array:

Code Example 3 Two-dimensional array

```
import numpy as np
Ncolumns = 5
Nrows = 3

#
# Notice that we have dimensioned the array with the first index assigned
# to the number of rows, and the second index assigned to the number
# of columns.
#
xGrid = np.zeros((Nrows, Ncolumns), dtype = 'float')
# Set the individual values of array
xGrid[0, 0] = 1.0
xGrid[0, 1] = 3.6
xGrid[2, 2] = 52.0
```

Which will print out:

```
[[  1.    3.6   0.    0.    0. ]
 [  0.    0.    0.    0.    0. ]
 [  0.    0.   52.    0.    0. ]]
```

Compare this output with the diagram of Figure 3. If you look at the square brackets in Python's output you will see that what is really going on is that a 2D array is really an array of arrays. This is a 1D array with three elements: each element is an array with 5 elements. A 3D array can be visualized as a cube.

This concept works for any number of array shapes, which take

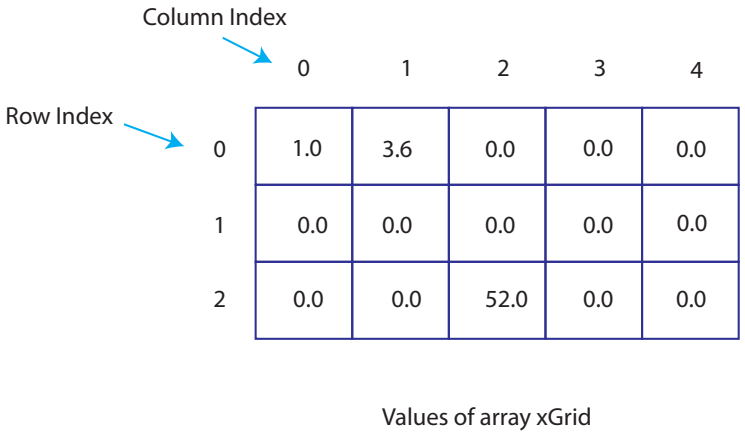

Figure 3: If we define the array `xGrid` as in Code Example 3, then printing `xGrid[0, 1]` in Python would yield the value 3.6. Likewise, printing `xGrid[2, 2]` in Python would yield 52.0.

the name *n-dimensional* arrays. Consider an example with a 3 dimensional array as in Code Example 4. The most interior index (the right-most array index) is called the 'fastest running index'.

Consider the example of a 3-dimensional array for $humidity(t, \theta, \lambda)$ where θ is the longitude and λ is the latitude and t is the time:

Code Example 4 3-D Array

```
import numpy as np

nx = 10
ny = 3
nz = 2

mult_d_array = np.zeros((nx, ny, nz)) # initialize the 3d array
print(mult_d_array)
for i in range(nx):
    for j in range(ny):
        for k in range(nz):
            if i == j:
                mult_d_array[i, j, k] = 1.
            elif j == k:
                mult_d_array[i, j, k] = 1.
            elif i == k:
                mult_d_array[i, j, k] = 1.

print(mult_d_array.shape)

for i in range(nx):
    for j in range(ny):
        for k in range(nz):
            print(i, j, k, mult_d_array[i, j, k])
```

You can access one entire axis by using the :. This is useful for 'slicing' an array while making plots and graphs. Also, rather than find-

ing the length of a multidimensional array, it is more useful to find its `shape`. The `shape()` function will show you the dimension of each dimension of a multidimensional array.

Code Example 5 Program 4

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

nx = 10
ny = 8

cos_times_sin = np.zeros((nx, ny))  # initialize the 2d array
print(np.shape(cos_times_sin))

x = np.linspace(0., 2. * np.pi, nx)
y = np.linspace(0., 2. * np.pi, ny)

for i in range(nx):
    for j in range(ny):
        cos_times_sin[i, j] = np.cos(x[i]) * np.sin(y[j])

plt.plot(x, cos_times_sin[:, 5])  # Plot all x at y[5]
plt.show()
```

In the above example, the fastest running index is j, and the slowest running index is i.

Exercises:

The following sequence of exercises will help you prepare for your assessment. You are required to experiment with multidimensional arrays and pass a function as an argument to a new function.

1. Working with functions passed as arguments to functions.

   (a) In Code Example 1 we pass `x[4]` as an argument to each function. Print out the value of `x[4]`. Then print out the entire array `x`. Print out `f2([x4])` and see if that makes sense. Next print out `f2(x)`. How is this different than passing the single value `x[4]`? The difference between `f2[x4]` and `f2(x)` is that in the first instance, f2 returns a single value. In the second instance you have passed the entire array, x, so the function f2 returns an evaluation of sin for every value in x. Now, pass the entire array x to the function `addfunctions`. What is the result? What is the length of the array returned by addfunction? Using matplotlib, make a plot of the result.

   (b) Define a new function `f4(x, y)` that returns the value of `sin(x)cos(y)`. Define a new array of points, `y` that has values from 0 to 2π. Add code to your program that will make a plot

the value of `f4` at x = .2 and all the values of y. Make another plot of `f4` at y = y[4] and all values of x.

2. Multidimensional arrays

   (a) Write a program that defines a 2-dimensional array called `almost_ones` that is 5x6 and filled with ones. Add a loop so that whenever i=j, it sets that element to 0. Add a loop that prints out i, j, and `almost_ones[i, j]`. Which index is the fastest running index? Can you see as the indices are printed when you should expect to see a zero?

   (b) You are given the function $f(x,y) = \sin(xy^2\pi)$ where $0 \leq x \leq 4$ and $0 \leq y \leq 2$. Write a program to make a plot of this function when $x = 3.$ for all y. Make another plot of this function when $y = 1.$ for all x.

   (c) As an example consider that you want to represent the wavy surface of the ocean at a snapshot in time. That is, you want to see what the initial wave field looks like ($t = 0$) and then what it looks like after .8 seconds. Consider the following code:

Code Example 6 Program 2

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

def oceanWaves(t, x):
    return np.exp(-t/4.) * np.cos(10*2*np.pi*x-t)

n = 100
x = np.linspace(0, 1, n)

plt.plot(x, oceanWaves(0., x))
plt.plot(x, oceanWaves(.8, x))
plt.show()
```

If you'd like to see how the wave field evolves over more than a few instances of time, Code Example 6 is a slow method of visualising your data. Instead, you can use multidimensional arrays!

Code Example 7 Program 3

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

def oceanWaves(t, x):
    return np.exp(-t/4.) * np.cos(10*2*np.pi*x-t)

nx = 100  # The number of points in the x direction
x = np.linspace(0, 1, nx)
nt = 5 # The number of time slices
```

```
t = np.linspace(0, 5, nt)

# Initialize a 2D array
#   The first index of the array holds the data for the time slice
#   The second index of the array holds the data for its location in x

WavesInTime = np.zeros((nt, nx)) # Initialize a 2D array.

for iT in range(nt):
    for iX in range(nx):
        WavesInTime[iT, iX] = oceanWaves(t[iT], x[iX])

# Make a plot with all the time slices by using a loop.
# Notice that the plot is being created in a loop!
for iT in range(nt):
    plt.plot(x, WavesInTime[iT, :])
plt.show()
```

(d) Type in the code, above, and see what it plots up. Add labels to your figure. Decrease the number of time slices to 3. Increase it to 10. How effective are the plots in understanding the ocean waves when there are fewer versus more curves?

(e) Add a new function to represent the contribution to the total wave field from a passing ship to Code Example 7, as in the code example below. Make a new plot that includes the effects of both the ocean waves and the ship waves. (**Hint:** Consider adding the two functions together.

(f) Make another plot that compares the height of the sea surface in time with and without ship waves. How do the ship waves effect the behavior of the sea surface?

Code Example 8 Program 4

```
def shipWaves(t, x):
    return .4*np.exp(-t/6.) * np.sin(3*2*np.pi*x-t)
```

3. Create a function that takes two integers x and y and does the following:

(a) Creates an array of zeros of size x by y

(b) Loops through each dimension of this new array, where each element is equal to the sin of the row index multiplied by the sin of the column index

(c) Return this array to the main program.

(d) Plot this function for x = 20, y = 20, using `plt.contourf()`. Use a search engine to look what `contourf` will do and the type

of arguments it takes. Is this pattern what you'd expect? Add a colorbar. What happens if you increase the number of x and y? ry using x = 40.5 and y = 40. What is the error? Modify the function by converting the arguemnts to an integer to prevent this error.