Functions within Functions, Multidimensional Arrays

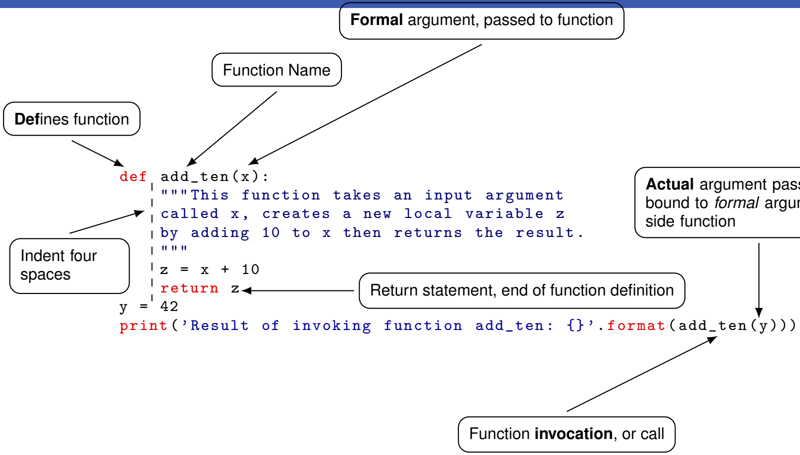
NSC 1002

11 Jan 2021

Schedule for Term 2

- Term 2 Workshops: Mondays 11:35-1:25, Fridays 10:35-12:35.
- Term 2 last workshop is Friday, 19 February, 2021 from 10:36-12:35
- Quizzes: Three Quizzes remaning: January 15th, February 22nd, and February 29th
- **Assessment Due: Friday, 19 February at noon.**
- Today: Multidimensional Arrays - you may need this for your assessment!

Functions revisited



```
def add_ten(x):
    """This function takes an input argument
    called x, creates a new local variable z
    by adding 10 to x then returns the result.
    """
    z = x + 10
    return z
y = 42
print('Result of invoking function add_ten: {}'.format(add_ten(y)))
```

Formal argument, passed to function
Function Name
Defines function
Indent four spaces
Actual argument passed, bound to *formal* argument inside function
Return statement, end of function definition
Function **invocation**, or call

Figure: Diagram of the syntax for a specific function called add_ten(). Don't forget to use the docstring in the triple quotes!

Functions as arguments of Functions

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 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('Separately: {}'.format(f1(x[4]) + f2(x[4])))
print('Using addfunctions: {} '.format(addfunctions(f1, f2, x[4])))
```

Functions as arguments of Functions

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 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('Separately: {}'.format(f1(x[4]) + f2(x[4])))
print('Using addfunctions: {} '.format(addfunctions(f1, f2, x[4])))
```

```
Separately:  0.4793871783994834
Using addfunctions:  0.4793871783994834
```

Example for 1D array

Code Example 2 numpy array, x of length 5

```
import numpy as np
n = 5
x = np.zeros(n, 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)
```

Example for 1D array

Code Example 2 numpy array, x of length 5

```
import numpy as np
n = 5
x = np.zeros(n, 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)
```

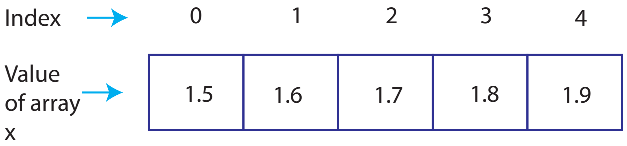


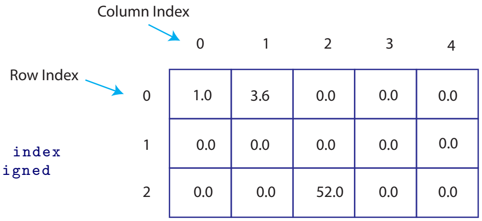
Figure: Visual of a 1D (one-dimensional) array defined in Code Example 2. Printing `x[2]` in Python would yield the value 1.7.

Example for 2D 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
```



	0	1	2	3	4
0	1.0	3.6	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	52.0	0.0	0.0

Values of array xGrid

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.

Plenty of chance to practise in the workshop...