LECTURE 4 MULTI-DIMENSIONAL ARRAYS

Fundamentals of Programming

COMP1005/COMP5005

Discipline of Computing

Curtin University

Updated: 22 August 2019

Copyright Warning

Learning Outcomes

- Understand and use multi-dimensional arrays in Python using the Numpy library
- Have awareness of sub-modules available in the Scipy library and how to access them
- Define and use simple functions

- Apply multi-dimensional arrays to multi-dimensional science data
- Use matplotlib to plot multi-dimensional data

MULTI-DIMENSIONAL ARRAYS

Fundamentals of Programming

Lecture 4

Arrays are Awesome! (revision)

- They are fast
- They make sense
- They don't take any more space than they need
- They can store lots of useful data

BUT

- They are not part of "standard" Python
- We need to use a package…

NumPy (revision)

- Pronounced "num-pai"
- This is the core library for scientific computing in Python – everything else builds upon it
- Provides high-performance N-dimensional arrays
- Includes:
  - Operations and functions to manipulate arrays
  - Sophisticated (broadcasting) functions
  - Tools for integrating C/C++ and Fortran code
  - Useful linear algebra, Fourier transform, and random number capabilities

1-Dimensional Arrays (revision)

- Importing NumPy

```
import numpy as np
```

- Creating arrays

```
listarray = np.array([ 1, 2, 3, 4]) # From a list
emptyarray = np.empty(100)        # Empty array, or is it?
array([  6.01346953e-154, 8.38666105e+228, 5.81816236e+180, ...,
         1.21696631e-152, 7.20358919e+159, 6.01334435e-154])

zeroarray = np.zeros(100)         # Array of zeros
array([ 0.,  0.,  0., ...,  0.,  0.,  0.])

onesarray = np.ones(100)          # Array of ones
randomarray = np.random.rand(100) # Array of random numbers 0<=x<1

arangearray = np.arange(0,20,2)
                                   # 0, 2, 4..18
linarray = np.linspace(0, 1, 6)
                                   #0, 0.2, 0.4.. 1.0
linarray = np.linspace(0, 1, 5, endpoint=False)
                                   #0, 0.2, 0.4.. 0.8
```

1-Dimensional Arrays (revision)

- Accessing elements

```
listarray = np.array([ 1, 2, 3, 4])
print(listarray[0])          # 1
listarray[2] + listarray[3]  # 3+4=7
```

- Looping through elements

```
for index in range(len(listarray)):
    print(listarray[index])

for value in listarray:
    print(value)
```

- Slicing

```
print(listarray[1:3])   # [2, 3]
print(listarray[:3])    # [1, 2, 3]
print(listarray[1:])    # [2, 3, 4]
print(listarray[::-1]   # [4, 3, 2, 1]
```

2-D ARRAYS

Fundamentals of Programming

Lecture 4

2-Dimensional Arrays

- Common in the real world and particularly in science and engineering
- Might be:
  - Values at points on a grid
  - Brightness (or red/green/blue-ness) of pixels in an image
  - Equivalent to a spreadsheet of cell values
  - Matrices representing coefficients in calculations

Defining 2-D Arrays

- Building from lists:

```
listarray = np.array([[1, 2], [3, 4]])
[[1 2]
 [3 4]]
```

- Initialised values:

```
zeroarray = np.zeros( (10, 10) )
[[ 0.   0.   0. ...,  0.   0.   0.]
 [ 0.   0.   0. ...,  0.   0.   0.]
...,
 [ 0.   0.   0. ...,  0.   0.   0.]
 [ 0.   0.   0. ...,  0.   0.   0.]]

onesarray = np.ones( (10, 10) )
[[ 1.   1.   1. ...,  1.   1.   1.]
 [ 1.   1.   1. ...,  1.   1.   1.]
...,
 [ 1.   1.   1. ...,  1.   1.   1.]
 [ 1.   1.   1. ...,  1.   1.   1.]]
```

2-D Arrays with ranges of numbers

- Meshgrid functions are similar to using arange to create a 1-D array…
- Create an array of integers, going from 0 to 4 across each dimension

```
y,x = np.mgrid[0:5, 0:5]

y is …
array([[0, 0, 0, 0, 0],
       [1, 1, 1, 1, 1],
       [2, 2, 2, 2, 2],
       [3, 3, 3, 3, 3],
       [4, 4, 4, 4, 4]])

x is…
array([[0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4],
       [0, 1, 2, 3, 4]])
```

Included for completeness,
this will not be examined

Describing arrays

- Print the contents of an array

```
a = np.array([[1,2,3],[4,5,6]])
print(a)
```

- Get the total number of entries in an array

```
np.size(a)        # 6
```

- get the dimensions of an array

```
np.shape(a)       # (2,3)
```

- Get the length of the first dimension in an array

```
len(a)            #  2
```

Reshaping arrays

- Arrays are really all 1-D, but are interpreted as multi-dimensional

```
a = np.array([[1,2,3],[4,5,6]])
np.shape(a)      #  (2,3)
np.size(a)       #  6
```

- Change the shape of an array to another **of the same size**

```
d = a.reshape(3,2)
         # array([[1,2],[3,4],[5,6]])

d = a.reshape(1,6)
         # array([[1,2,3,4,5,6]])

d = a.reshape(6,1)
         # array([[1],[2],[3],[4],[5],[6]])
```

Accessing 2-D Arrays

- Now need two indexes – one for rows and one for columns

```
listarray = np.array([[1, 2, 3], [4, 5, 6],[7, 8, 9]])
```

row \ column	**0**	**1**	**2**
0	1	2	3
1	4	5	6
2	7	8	9

```
listarray[0, 0] is 1
listarray[2, 0] is 7
listarray[1, 2] is 6
```

Looping through arrays

- 2-D looping requires two nested loops…

```
numarray = np.array([[1, 2, 3, 4], [5, 6, 7, 8],
                     [9, 10, 11, 12]])

print('Shape is: ', np.shape(numarray))               # (3,4)
print('Number of rows: ', len(numarray[:,0]))         # 3
print('Number of cols: ', len(numarray[0,:]))         # 4

row, col = 0, 0

while row < len(numarray[:,0]):
    while col < len(numarray[0,:]):
        print('Element [', row, ',', col, '] is: ',
               numarray[row, col])
        col = col + 1
    row = row + 1
    col = 0
```

Looping through arrays

- 2-D looping requires two nested loops…

```
numarray = np.array([[1, 2, 3, 4], [5, 6, 7, 8
                    [9, 10, 11, 12]])

print('Shape is: ', np.shape(numarray))
print('Number of rows: ', len(numarray[:,0]))
print('Number of cols: ', len(numarray[0,:]))

row, col = 0, 0

while row < len(numarray[:,0]):
    while col < len(numarray[0,:]):
        print('Element [', row, ',', col, '] i
               numarray[row, col])
        col = col + 1
    row = row + 1
    col = 0
```

```
Shape is:  (3, 4)
Number of rows:  3
Number of cols:  4
Element [ 0 , 0 ] is:  1
Element [ 0 , 1 ] is:  2
Element [ 0 , 2 ] is:  3
Element [ 0 , 3 ] is:  4
Element [ 1 , 0 ] is:  5
Element [ 1 , 1 ] is:  6
Element [ 1 , 2 ] is:  7
Element [ 1 , 3 ] is:  8
Element [ 2 , 0 ] is:  9
Element [ 2 , 1 ] is:  10
Element [ 2 , 2 ] is:  11
Element [ 2 , 3 ] is:  12
```

Looping in a Pythonic way…

- Arrays are sequences, so we can simplify…

```
for row in numarray:
    for element in row:
        print('Element:', element)
```

- To access the indexes, use enumerate…

```
for rindex, row in enumerate(numarray):
    for cindex, element in enumerate(row):
        print('Element: [', rindex, ',',
              cindex,'] is :', element)
```

Looping in a Pythonic way…

- Arrays are sequences, so we can simplify…

```
for row in numarray:
    for element in row:
        print('Element:', element)
```

```
Element: 1
Element: 2
Element: 3
Element: 4
Element: 5
Element: 6
Element: 7
Element: 8
Element: 9
Element: 10
Element: 11
Element: 12
```

- To access the indexes, use enumerate…

```
for rindex, row in enumerate(numarray):
    for cindex, element in enumerate(row):
        print('Element: [', rindex, ',',\
              cindex,'] is :', element)
```

Looping in a Pythonic way

- Arrays are sequences, so we can simplify…

```
for row in numarray:
    for element in row:
        print('Element:', element)
```

- To use the indexes, enumerate – rindex = 0,

```
for rindex, row in enumerate(numarray):
    for cindex, element in enumerate(row):
        print('Element: [', rindex, ',',\
              cindex,'] is :', element)
```

```
Element: [ 0 , 0 ] is : 1
Element: [ 0 , 1 ] is : 2
Element: [ 0 , 2 ] is : 3
Element: [ 0 , 3 ] is : 4
Element: [ 1 , 0 ] is : 5
Element: [ 1 , 1 ] is : 6
Element: [ 1 , 2 ] is : 7
Element: [ 1 , 3 ] is : 8
Element: [ 2 , 0 ] is : 9
Element: [ 2 , 1 ] is : 10
Element: [ 2 , 2 ] is : 11
Element: [ 2 , 3 ] is : 12
```

Slicing 2-D Arrays

- Consider each dimension/axis in turn…

```
listarray = np.array([[1, 2, 3, 4], [5, 6, 7, 8],
                      [9, 10, 11, 12]])
```

row \ col	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

```
listarray[1:, 0]   # is array([5, 9])
listarray[2, 1:]   # is array([10, 11, 12])
listarray[2, :]    # is array([9, 10, 11, 12])
listarray[:, ::2]  # is array([[ 1,  3],[ 5,  7],
                              [ 9, 11]])
```

Slicing 2-D Arrays

- Consider each dimension/axis in turn…

```
listarray = np.array([[1, 2, 3, 4], [5, 6, 7, 8],
                     [9, 10, 11, 12]])
```

row \ col	**0**	**1**	**2**	**3**
0	1	2	3	4
1	**5**	6	7	8
2	**9**	10	11	12

```
listarray[1:, 0]   # is array([5, 9])
listarray[2, 1:]   # is array([10, 11, 12])
listarray[2, :]    # is array([9, 10, 11, 12])
listarray[:, ::2]  # is array([[ 1,  3],[ 5,  7],
                              [ 9, 11]])
```

Slicing 2-D Arrays

- Consider each dimension/axis in turn…

```
listarray = np.array([[1, 2, 3, 4], [5, 6, 7, 8],
                      [9, 10, 11, 12]])
```

row \ col	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

```
listarray[1:, 0]    # is array([5, 9])
listarray[2, 1:]    # is array([10, 11, 12])
listarray[2, :]     # is array([9, 10, 11, 12])
listarray[:, ::2]   # is array([[ 1,  3],[ 5,  7],
                                [ 9, 11]])
```

Slicing 2-D Arrays

- Consider each dimension/axis in turn…

```
listarray = np.array([[1, 2, 3, 4], [5, 6, 7, 8],
                      [9, 10, 11, 12]])
```

row \ col	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

```
listarray[1:, 0]   # is array([5, 9])
listarray[2, 1:]   # is array([10, 11, 12])
listarray[2, :]    # is array([9, 10, 11, 12])
listarray[:, ::2]  # is array([[ 1,  3],[ 5,  7],
                                [ 9, 11]])
```

Slicing 2-D Arrays

- Consider each dimension/axis in turn…

```
listarray = np.array([[1, 2, 3, 4], [5, 6, 7, 8],
                      [9, 10, 11, 12]])
```

row \ col	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

```
listarray[1:, 0]   # is array([5, 9])
listarray[2, 1:]   # is array([10, 11, 12])
listarray[2, :]    # is array([9, 10, 11, 12])
listarray[:, ::2]  # is array([[ 1,  3],[ 5,  7],
                               [ 9, 11]])
```

- Slicing an array results in a new array

WORKING WITH 2-D ARRAYS

Fundamentals of Programming

Lecture 4

Operations

- As with 1D arrays, arithmetic operations are carried out on **each element** of an array

```
a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([[7, 8, 9], [10, 11, 12]])

c = a + b   # array([[ 8,10,12],[14,16,18]])
c = a + 1   # array([[2,3,4],[5,6,7]])
c = a - b   # array([[-6,-6,-6],[-6,-6,-6]])
c = a * b   # array([[7,16,27],[40,55,72]])
c = a / b   # array([[0.142,0.25,0.333],[0.4,0.454,0.5]])
```

Comparisons

- Can do element-wise comparisons of values using <, <=, >, >=, ==, !=

- Result is an array

```
a = np.array([[1, 2, 3], [4, 5, 6]])
b = np.array([[7, 8, 9], [10, 11, 12]])

d = a < b
     array([[ True,  True,  True],
            [ True,  True,  True]],
           dtype=bool)
```

Element-wise Functions

- These functions are carried out on **each element** of an array

```
a = np.array([[1, 2, 3], [4, 5, 6]])

c = np.sqrt(a)
 array([[ 1., 1.41421356, 1.73205081],
        [2. , 2.23606798, 2.44948974]])

c = np.sin(a)
 array([[0.8414, 0.9092,  0.1411],
        [-0.756 , -0.9589, -0.2794]])
```

Also… exp(), cos(), log(), add(), multiply() etc.

Array-wise Functions

- These functions return a single result across the array (or a dimension of the array)

```
a = np.array([[1,2,3],[4, 5, 6]])
b = np.array([[7,8,9],[10,11,12]])

a.sum()        21
b.min()        7
b.max()        12
a.mean()       3.5

a[:,0].sum()   5     # all rows, col=0
a[1,:].sum()   15    # row 1, all cols
```

MATRICES

Fundamentals of Programming

Lecture 4

Matrices in NumPy

- Matrices are set up in a similar way to arrays, using the matrix() function

```
a = matrix([[1, 2], [2, 3]])
b = matrix('1 -2; 2 3')
```

- Can then use matrix multiplication, and generate transpose, determinants and inverses

```
a + b            # matrix([[2, 0],[4, 6]])
c = 5 * a        # matrix([[ 5, 10],[10, 15]])
a * b            # matrix([[5, 4],[8, 5]])
a**-1            # matrix([[-3.,2.],[2.,-1.]])

np.linalg.det(a)   # -1
b.T              # matrix([[ 1,2],[-2,3]])
e_val, e_vect = np.linalg.eig(a)
e_val            # array([-0.23606798,  4.23606798])
e_vect           # matrix([[-0.85065081, -0.52573111],
                           [ 0.52573111, -0.85065081]])
```

SCIPY

Fundamentals of Programming

Lecture 4

Scipy: high level scientific computing

- A package containing toolboxes for scientific computing
  - interpolation
  - integration
  - optimisation
  - image processing
  - statistics
- Uses NumPy arrays
- Optimised and tested – do not reinvent the wheel!

Scipy

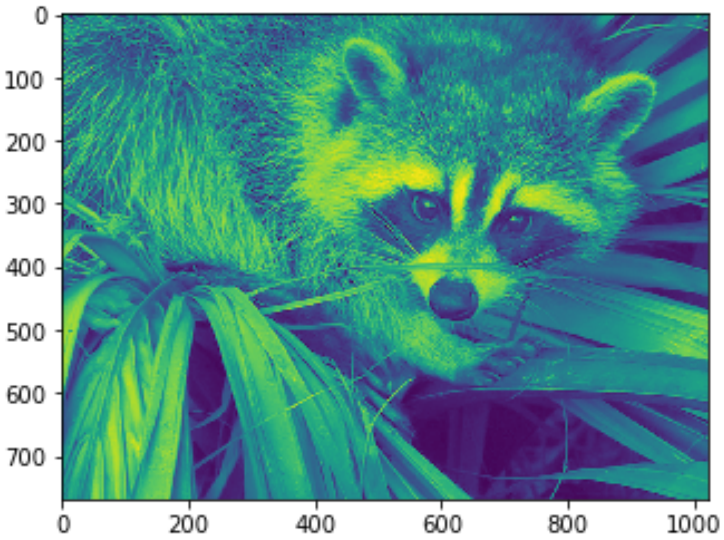
- scipy is composed of task-specific sub-modules:

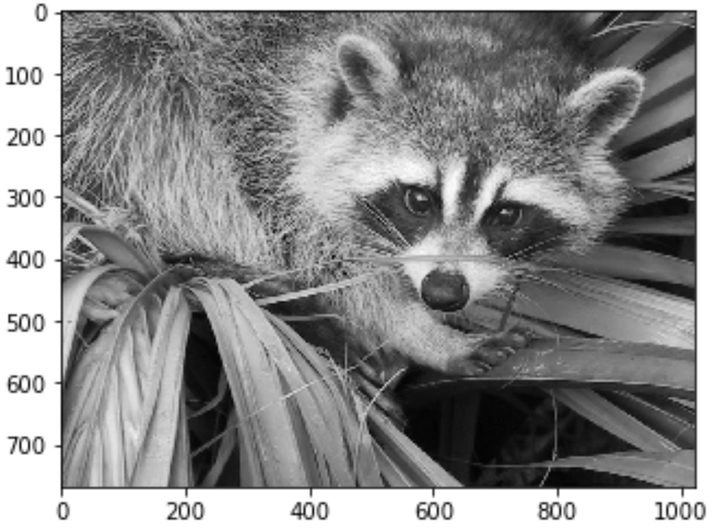
scipy.cluster	Vector quantization / Kmeans
scipy.constants	Physical and mathematical constants
scipy.fftpack	Fourier transform
scipy.integrate	Integration routines
scipy.interpolate	Interpolation
scipy.io	Data input and output
scipy.linalg	Linear algebra routines
scipy.ndimage	**n-dimensional image package**
scipy.odr	Orthogonal distance regression
scipy.optimize	Optimization
scipy.signal	Signal processing
scipy.sparse	Sparse matrices
scipy.spatial	Spatial data structures and algorithms
scipy.special	Any special mathematical functions
scipy.stats	Statistics

Example: Scipy.ndimage

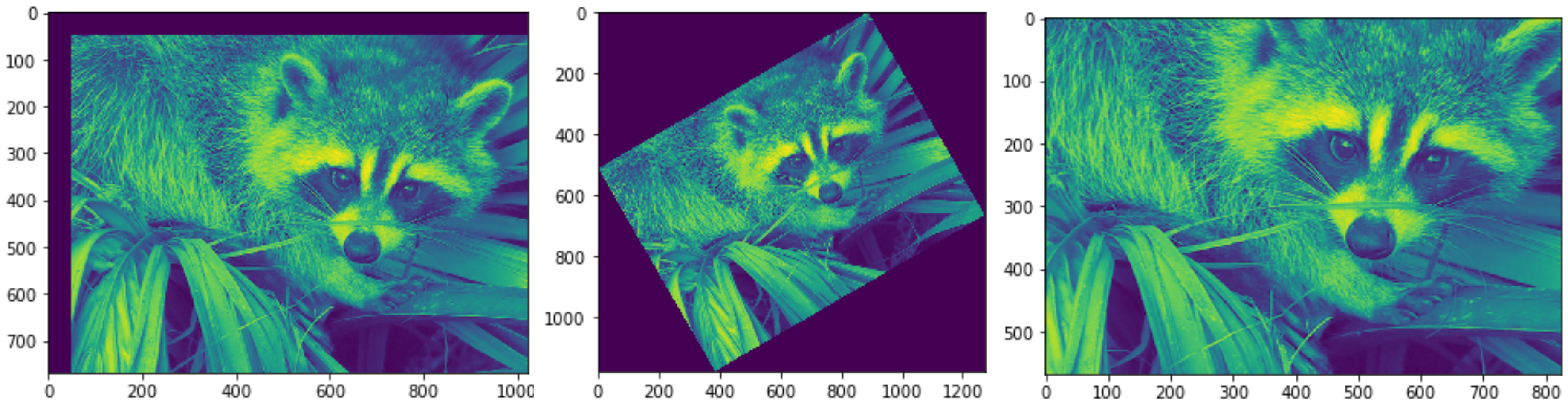
- This submodule provides image processing routines
- An image is a 2-D array

```
from scipy import ndimage
from scipy import misc
face = misc.face(gray=True)
plt.imshow(face)
plt.imshow(face,cmap=plt.cm.gray)
```



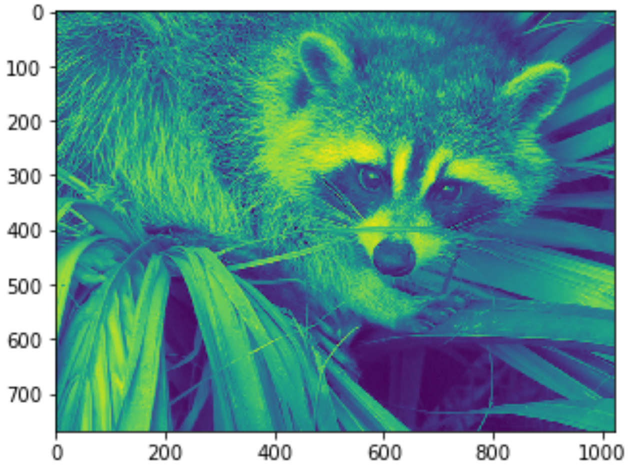


Example: Scipy.ndimage

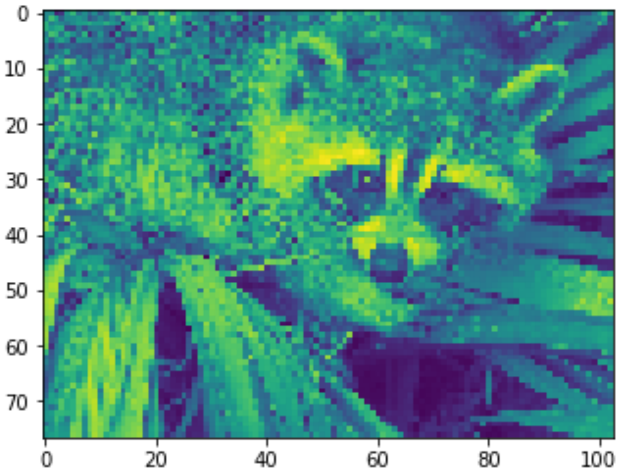


```
shifted_face = ndimage.shift(face, (50,50))
plt.imshow(shifted_face)

rotated_face = ndimage.rotate(face, 30)
plt.imshow(rotated_face)

cropped_face = face[100:-100,100:-100]
plt.imshow(cropped_face)
```

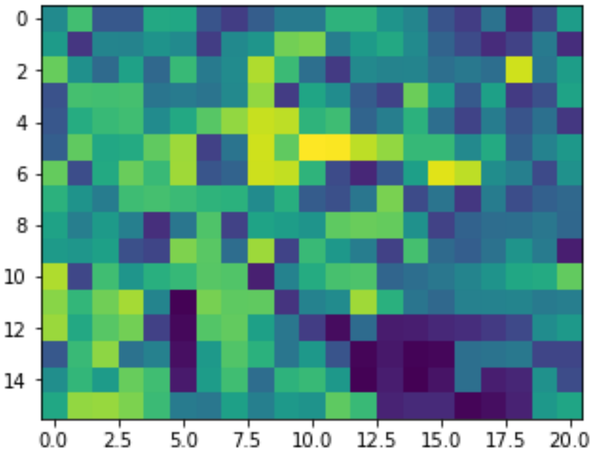
Example: Scipy.ndimage



face



pixel_face



pixel_face2

```
np.shape(face)  # (768, 1024)

plt.imshow(face)
pixel_face = face[::10,::10]
plt.imshow(pixel_face)

pixel_face2 = face[::50,::50]
plt.imshow(pixel_face2)
```

Scipy…

- We wont go deeply into Scipy in this unit
- Depending on your research area, it may be something you revisited later in your studies
- You shouldn't need to implement common routines for yourself
- Find trusted, tested packages like Scipy and get on with the real research!

FUNCTIONS

Fundamentals of Programming

Lecture 4

Functions

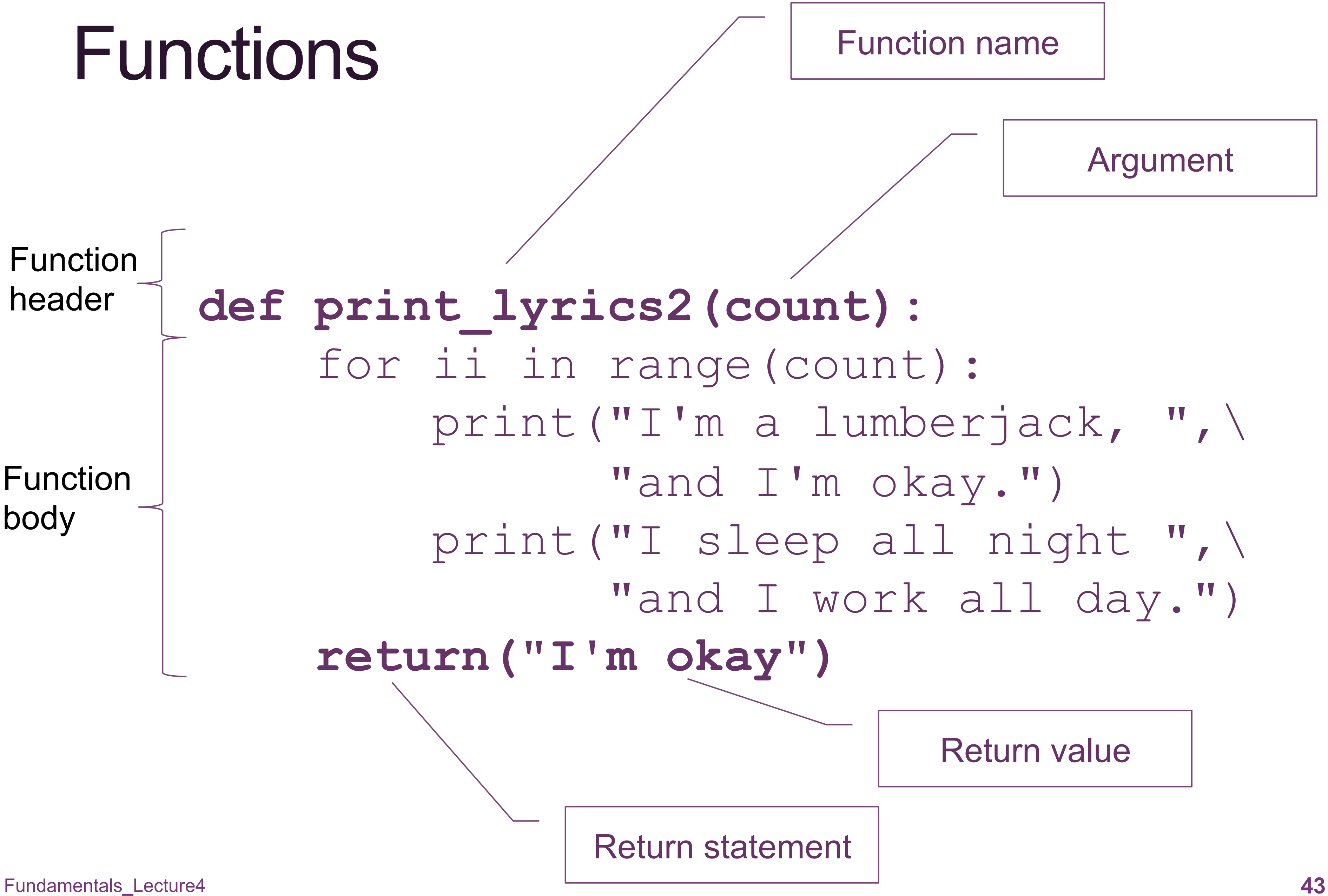
- We've been using lots of functions and methods – where do they come from?
- We can use a special keyword "def" to define a function…
  … then indent the block by 4 spaces

```
def print_lyrics():
    print("I'm a lumberjack, ",\
          "and I'm okay.")
    print("I sleep all night ",\
          "and I work all day.")
```

Functions - definition

- A function is a named sequence of statements
- A function can take arguments – parameters to process in the function
- A function may produce a return value – the result of the function's processing
- Functions we've used – int(), print()
- Need brackets even if there are no arguments

Functions



```
def print_lyrics2(count):
    for ii in range(count):
        print("I'm a lumberjack, ",\
              "and I'm okay.")
        print("I sleep all night ",\
              "and I work all day.")
    return("I'm okay")
```

Why Functions?

- Makes your program easier to read and debug.
- Functions can make a program smaller by eliminating repetitive code. Later, if you make a change, you only have to make it in one place.
- Dividing a long program into functions allows you to debug the parts one at a time and then assemble them into a working whole.
- Well-designed functions are often useful for many programs. Once you write and debug one, you can reuse it

https://en.wikibooks.org/wiki/Think_Python/Functions

Flow of Execution

- Functions need to be defined before their first use
- Execution always begins at the first statement of the program.
- Statements are executed one at a time, in order from top to bottom.
- This is called the **flow of execution**.

Functions and Flow

- Function **definitions** do not alter the flow of execution of the program
- Statements inside the function are not executed until the function is called.
- A function call is like a detour in the flow of execution.
- Instead of going to the next statement, the flow jumps to the body of the function, executes all the statements there, and then comes back to pick up where it left off.

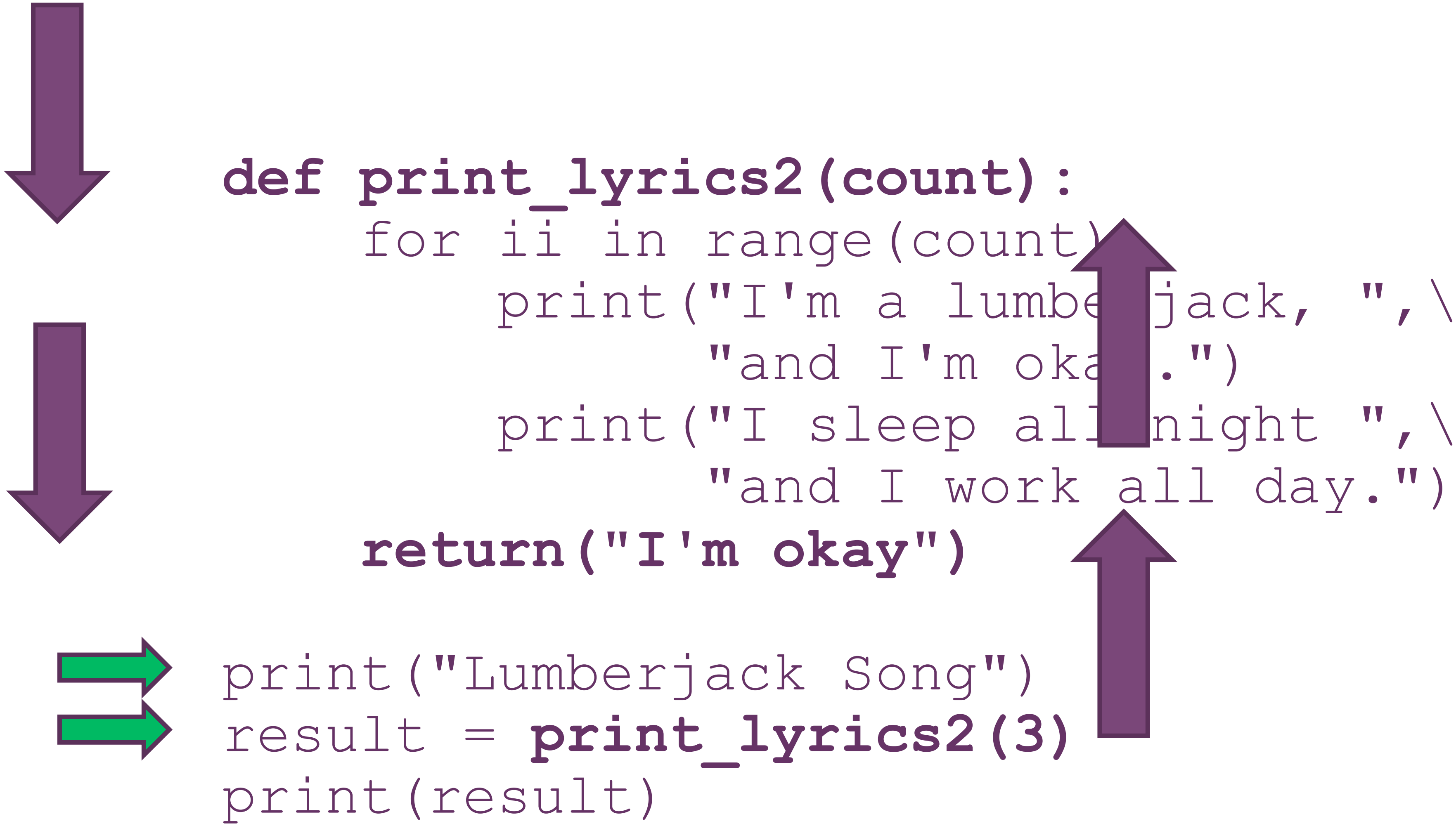
Functions and Flow

```
def print_lyrics2(count):
    for ii in range(count):
        print("I'm a lumberjack, ",\
              "and I'm okay.")
        print("I sleep all night ",\
              "and I work all day.")
    return("I'm okay")

print("Lumberjack Song")
result = print_lyrics2(3)
print(result)
```

Functions and Flow

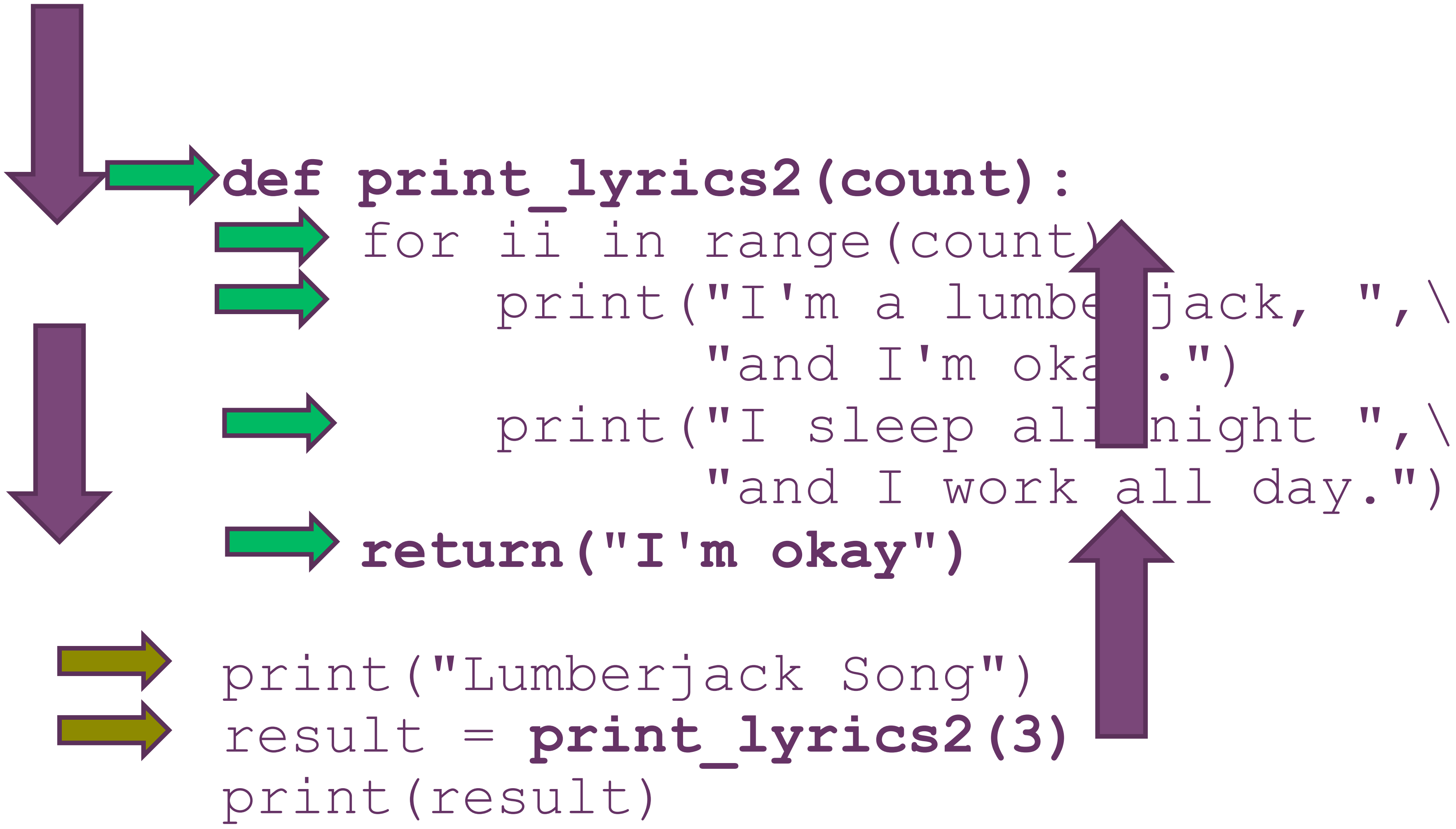
```
def print_lyrics2(count):
    for ii in range(count):
        print("I'm a lumberjack, ",\
              "and I'm okay.")
        print("I sleep all night ",\
              "and I work all day.")
    return("I'm okay")

print("Lumberjack Song")
result = print_lyrics2(3)
print(result)
```

Functions and Flow



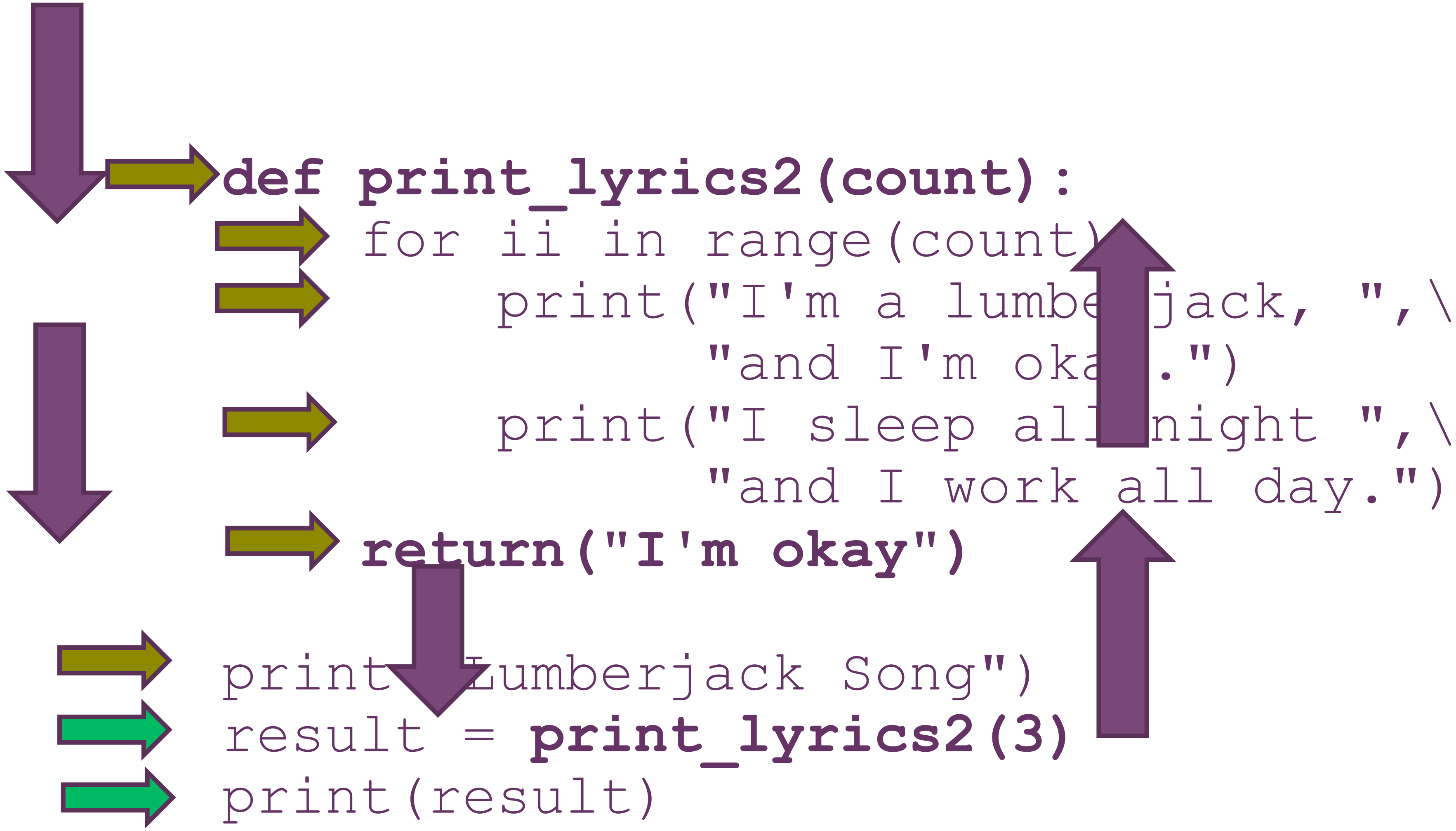
```
def print_lyrics2(count):
    for ii in range(count):
        print("I'm a lumberjack, ",\
              "and I'm okay.")
        print("I sleep all night ",\
              "and I work all day.")
    return("I'm okay")

print("Lumberjack Song")
result = print_lyrics2(3)
print(result)
```

Functions and Flow



```
def print_lyrics2(count):
    for ii in range(count):
        print("I'm a lumberjack, ",\
              "and I'm okay.")
        print("I sleep all night ",\
              "and I work all day.")
    return("I'm okay")

print("Lumberjack Song")
result = print_lyrics2(3)
print(result)
```

Functions and Flow



```
def print_lyrics2(count):
    for ii in range(count):
        print("I'm a lumberjack, ",\
              "and I'm okay.")
        print("I sleep all night ",\
              "and I work all day.")
    return("I'm okay")

print("Lumberjack Song")
result = print_lyrics2(3)
print(result)
```

General Code Structure

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

def calc_heat(row,col):
    subgrid = b[row-1:row+2,col-1:col+2]
    result = 0.1 * (subgrid.sum()+ b[row,col])
    return result

size = 10
b = np.zeros((size,size))
b2 = np.zeros((size,size))

for i in range(size):
    b[i,0] = 10

for timestep in range(5):
    for r in range(1, size-1):
        for c in range (1, size-1 ):
            b2[r,c] = calc_heat(r,c)
    for i in range(size):
        b2[i,0] = 10
    b = b2.copy()

plt.title('Heat Diffusion Simulation')
plt.imshow(b2, cmap=plt.cm.hot)
plt.show()
```

import statements

function definitions

set up variables

input data

process data

output data

Functions and Methods

- You may read about **methods**…
- They are a special type of function associated with a class/object
- Objects are datatypes that have associated data and operations
- Our plot figures are objects:
  - data - the settings and values they are given
    - e.g. color='red', data to plot is x2
  - operations - the methods for working with the data
    - e.g. setlabel(), plot()
- We will look at object-orientation later on

Writing programs

- Ultimately, you should be able to write programs from a specification (description) and identify when to include functions.
- Look for:
    - decisions (if-elif-else)
    - iteration (while or for loops)
    - repeated code / tasks (functions)
    - data to store (variables – and what types of data)
    - data to read in / print out (input and print calls)

An exam question

A dancing competition is being held. Each competitor in this competition is judged by seven judges. Each judge submits a score (an integer between 0 and 10, including 0 and 10) and the competitor's score is the average of the seven judge's scores.

An exam question – continued…

Design a short Python program which will:

*Input the number of competitors. Your algorithm should **repeat the input** until the number of competitors input is between 3 and 16 (including 3 and 16).*

For each competitor, your program should:

- *Input the judges' scores. If an input score is invalid, then your algorithm should **repeat the input** until the score is valid.*
- *Output the score for the competitor. Note the average is calculated as a real number.*

An exam question – continued…

Note you must make good use of sub modules (functions) and control structures (if/while/for) as shown in the lectures and practical exercises.

- As a guide to what is expected, have a look at the following **pseudocode** and convert it into Python.
- You should use at least one function to validate the values that are entered and make the code much cleaner.

Pseudocode (v1)

```
MAIN
   numJudges = 7
   numCompetitors = input "Enter number of competitors
                          (between 3 and 16 inc)"

   FOR comp = 0 TO numCompetitors-1 CHANGEBY 1
      totalC = 0
      OUTPUT “input scores between 0 and 10 for each Judge”
      FOR j = 0 TO numJudges-1 CHANGEBY 1
          scoreJ = input "Score for judge "
          totalC = totalC + scoreJ
      ENDFOR
      scoreC = totalC / numJudges
      OUTPUT "Score for competitor , , is", scoreC
   ENDFOR

END
```

Pseudocode (v2)

```
MAIN
   numJudges = 7
   numCompetitors = input "Enter number of competitors
                          (between 3 and 16 inc)"
   while numCompetitors < 3 AND numCompetitors > 16 DO
      numCompetitors = input "Error - Re-enter number of competitors
                             (between 3 and 16 inc)"

   FOR comp = 0 TO numCompetitors-1 CHANGEBY 1
      totalC = 0
      OUTPUT “input scores between 0 and 10 for each Judge”
      FOR j = 0 TO numJudges-1 CHANGEBY 1
          scoreJ = input "Score for judge "
          while numCompetitors < 3 AND numCompetitors > 16 DO
             numCompetitors = input "Error - Re-enter score (0-10)"
          totalC = totalC + scoreJ
      ENDFOR
      scoreC = totalC / numJudges
      OUTPUT "Score for competitor , , is", scoreC
   ENDFOR

END
```

Pseudocode (v2)

```
MAIN
   numJudges = 7
   numCompetitors = input "Enter number of competitors (3-16 inc)"
   while numCompetitors < 3 AND numCompetitors > 16 DO
      numCompetitors = input "Error - Re-enter number of competitors
                             (3-16 inc)"

   FOR comp = 0 TO numCompetitors-1 CHANGEBY 1
      totalC = 0
      OUTPUT “input scores between 0 and 10 for each Judge”
      FOR j = 0 TO numJudges-1 CHANGEBY 1
          scoreJ = input "Score for judge (0-10)"
          while numCompetitors < 3 AND numCompetitors > 16 DO
             numCompetitors = input "Error - Re-enter score (0-10)"
          totalC = totalC + scoreJ
      ENDFOR
      scoreC = totalC / numJudges
      OUTPUT "Score for competitor , , is", scoreC
   ENDFOR

END
```

Pseudocode (v3 1/2)

```
SUBMODULE inputValue
IMPORT lower, upper, prompt
EXPORT value

INPUT value
WHILE value < lower OR > upper DO
   OUTPUT “Error – re-enter number (lower-upper)”
   OUTPUT prompt
   INPUT value
ENDWHILE
```

- Putting this code into a submodule lets us reuse it multiple times for reading in and validating input

Pseudocode (v3 2/2)

```
MAIN
   numJudges = 7
   numCompetitors = inputValue <-- 3, 16, "Enter number of
                    competitors (between 3 and 16 inc)"
   FOR comp = 0 TO numCompetitors-1 CHANGEBY 1
      totalC = 0
      OUTPUT “input scores between 0 and 10 for each Judge”
      FOR j = 0 TO numJudges-1 CHANGEBY 1
          scoreJ = inputValue <-- 0, 10, "Score for judge "
          totalC = totalC + scoreJ
      ENDFOR
      scoreC = totalC / numJudges
      OUTPUT "Score for competitor , , is", scoreC
   ENDFOR
END
```

PYTHON MODULES

Modules

- We've imported modules and packages to gain access to functions written by others
- We can create functions inside our programs to reuse throughout the programs

- If we want to reuse the functions in many programs…
  - Create our own module
  - Import the module into our programs

Module textfun.py

- We can create a module with some text-related function – textfun.py
- In it we will create some methods:
  - novowels(inString)
  - reverseupper(inString)
  - upperskip2(inString)
- To use our functions, we can add:
  - import textfun

    …at the start of out programs

Module textfun.py

```
#
# textfun.py – module of text-related functions
#
vowels = 'aeiouAEIOU'

def novowels(inString):
    outString=''
    for i in inString:
        if not i in vowels:
            outString = outString + i
    return outString

def reverseupper(inString):
    return(inString[::-1].upper())
```

Test Program – testing.py

testing.py:

```
#
# testing.py – test program for textfun.py
#
import textfun

testString = 'helloHELLO'

print(textfun.novowels(testString))
print(textfun.reverseupper(testString))
```

```
> python testing.py
hllHLL
OLLEHOLLEH
```

Packages

- If we had a group of related modules, we could group them in a package
- The module files are placed in a directory together
  - A special file, __init__.py indicates the directory is a package

- So, if we had modules **textfun.py** and **numfun.py**. we might group them in a package fun
- Then we could import them using:

```
import fun.textfun as tfun
import fun.numfun as nfun
```

Scipy.ndimage package

```
$ ls anaconda3/pkgs/scipy-0.18.1-np111py35_0/lib/python3.5/site-packages/
scipy

BENTO_BUILD.txt   _build_utils     linalg          sparse
HACKING.rst.txt   _lib             linalg.pxd      spatial
INSTALL.rst.txt   cluster          misc            special
LICENSE.txt       constants        ndimage         stats
THANKS.txt        fftpack          odr             version.py
__config__.py     integrate        optimize
__init__.py       interpolate      setup.py
__pycache__       io               signal
```

```
$ ls anaconda3/pkgs/scipy-0.18.1-np111py35_0/lib/python3.5/site-packages/
scipy/ndimage/

__init__.py             filters.py              morphology.py
__pycache__             fourier.py              setup.py
_nd_image.so            interpolation.py        tests
_ni_label.so            io.py
_ni_support.py          measurements.py
```

Paths

- A filesystem is big, really big.
- Modules and Packages need to be located quickly by Python
- We *could* have lots of modules in the local directory '.' – which would get messy
- The operating system has a "PATH" variable to give it a list of directories to search through
- Part of the installation of a program **updates the path** to include the new program
- **Anaconda** looks after this for us.

```
[12345678@saeshell01p ~]$ echo $PATH
/usr/local/bin:/bin:/usr/bin:/usr/local/sbin:/usr/
sbin:/sbin:/opt/anaconda/bin:/home/12345678/.local/
bin:/home/12345678/bin
```

__main__

- Python programs and python modules are just code
- Python provides a way to tell if you are running code directly (python3) or just using the functions (import)

- If the code is called directly, the variable `__name__` will equal `"__main__"`
- Otherwise `__name__` will refer to the calling code
- We can include an if statement to check for this and run specific code
- Any other code that's not in a function definition will run at import time/run time
  - e.g. `vowels = 'aeiouAEIOU'`

textfun.py

```
def reverseupper(inString):
    return(inString[::-1].upper())

def main():
    print('\nTesting textfun.py')
    testString = 'helloHELLO'
    print('\nTest string is: ', testString)
    print('novowels: ', novowels(testString))
    print('reverseupper: ', reverseupper(testString))
    print('Testing complete')

if __name__ == '__main__':
    main()
```

Main as test code for textfun.py

```
$ python textfun.py

Testing textfun.py

Test string is:  helloHELLO
novowels:  hllHLL
reverseupper:  OLLEHOLLEH
Testing complete
```

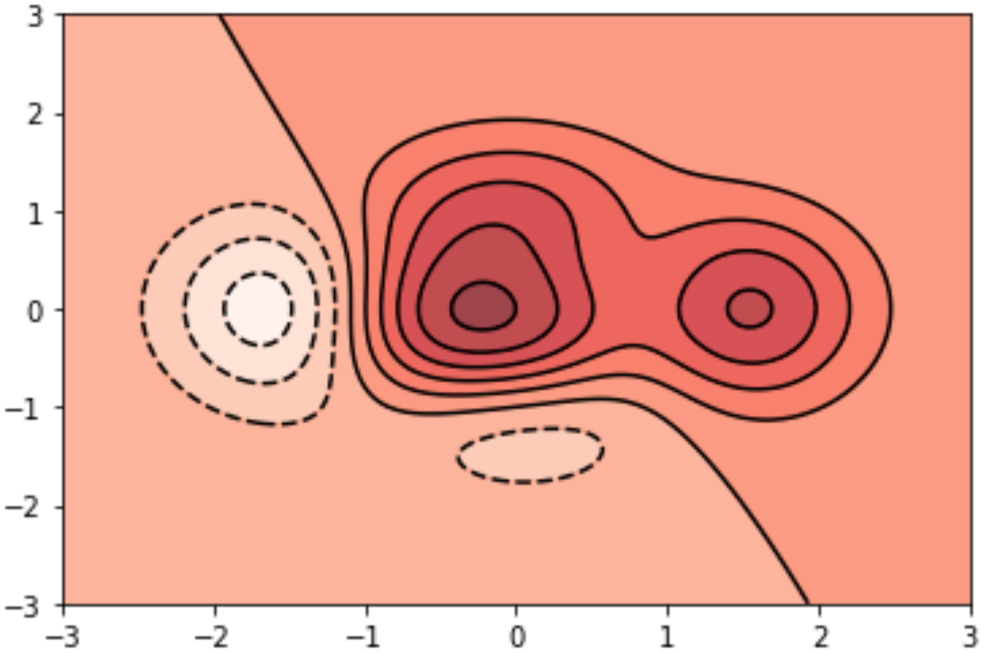
A good reference:
http://www.scipy-lectures.org/intro/language/reusing_code.html

PLOTTING MULTI-DIMENSIONAL DATA

Fundamentals of Programming

Lecture 4

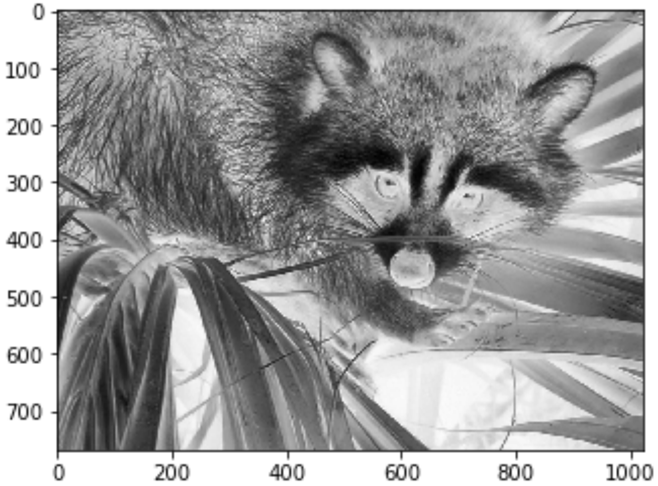
Contour plot

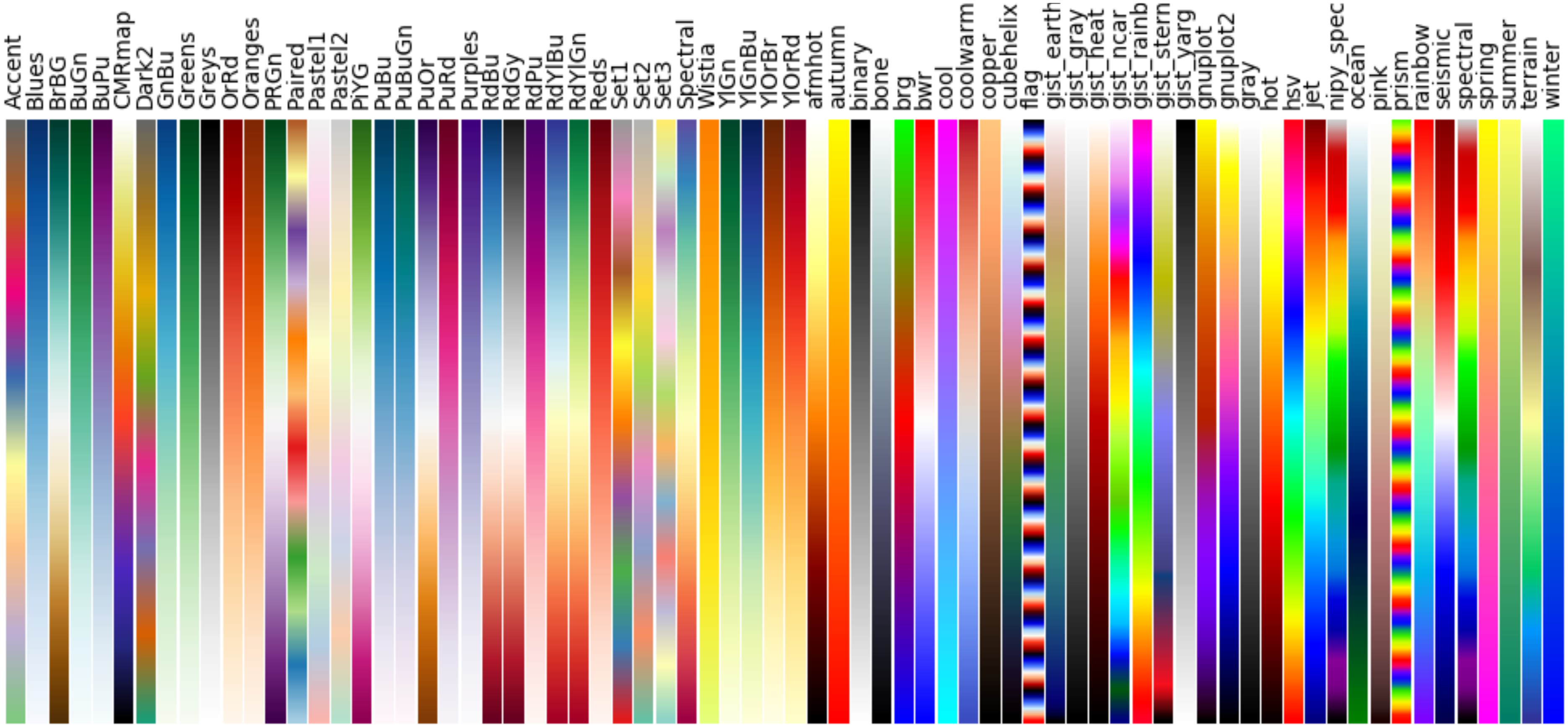


```
def f(x, y):
    return (1 - x / 2 + x ** 5 + y ** 3) * \
                np.exp(-x ** 2 -y ** 2)

n = 256
x = np.linspace(-3, 3, n)
y = np.linspace(-3, 3, n)
X, Y = np.meshgrid(x, y)

plt.contourf(X, Y, f(X, Y), 8, alpha=.75, cmap='jet')
C = plt.contour(X, Y, f(X, Y), 8,colors='black',linewidth=.5)
```

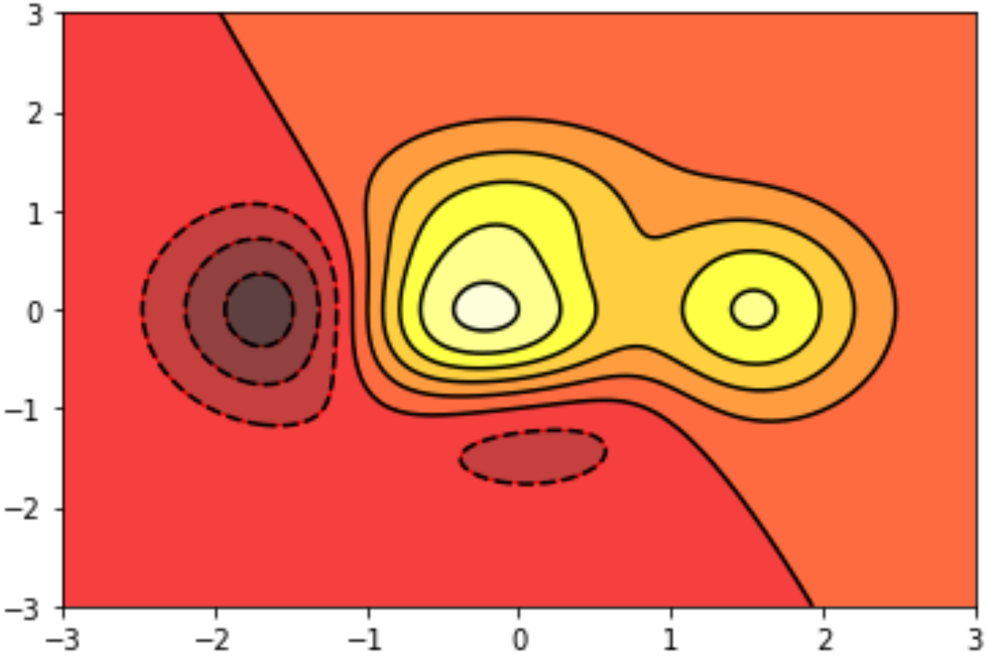
Colour maps

- Reverse map by appending _r (e.g. gray_r)



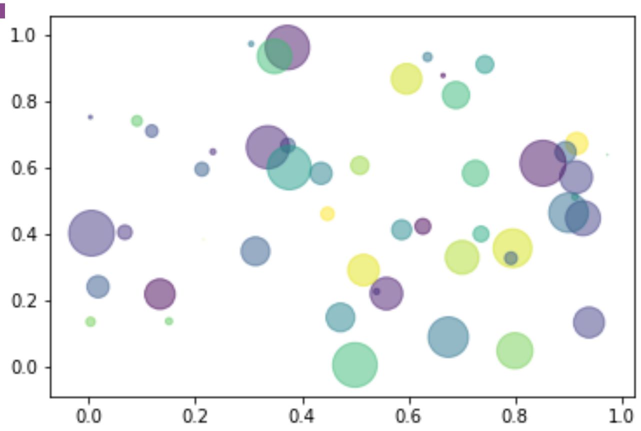


Contour plot

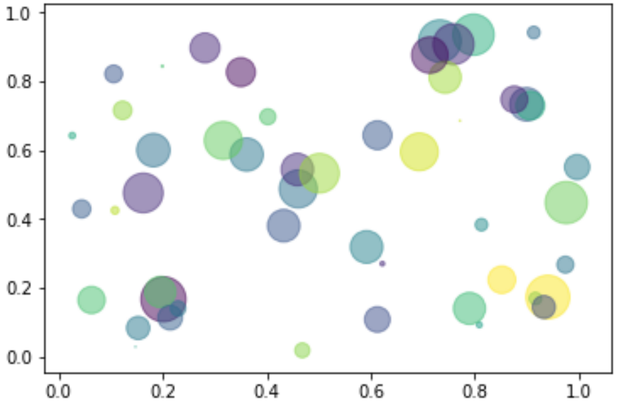


```
def f(x, y):
    return (1 - x / 2 + x ** 5 + y ** 3) * \
            np.exp(-x ** 2 -y ** 2)

n = 256
x = np.linspace(-3, 3, n)
y = np.linspace(-3, 3, n)
X, Y = np.meshgrid(x, y)

plt.contourf(X, Y, f(X, Y), 8, alpha=.75, cmap='hot')
C = plt.contour(X, Y, f(X, Y), 8,colors='black',linewidth=.5)
```

Scatter plot



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

N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colours = np.random.rand(N)
area = np.pi * (15 * np.random.rand(N)) ** 2

plt.scatter(x, y, s=area, c=colours, alpha = 0.5)
plt.show()
```



Summary

- Learnt about how to use multi-dimensional arrays in Python using the Numpy library
- Looked at the submodules available in the Scipy library, and how to access them
- Learnt how to define and use functions

- Apply multi-dimensional arrays to multi-dimensional science data (in pracs)
- Learnt for to use matplotlib to plot multi-dimensional data

Practical Sessions

- Time to catch up on previous pracs
- This week's prac (Prac 4) will be short
- Note that the modifications, reflections and extension questions in the pracs are likely sources of test and exam questions

Assessments

- No assessments this week
- The next assessment will be held during Practical 5
- It will be a short practical test looking at plotting – along with similar tasks as PracTest1

References

- Scpiy Lecture Notes
- Corey Schafer tutorials
- https://alexandria.astro.cf.ac.uk/Joomla-python/index.php/week-6-two-dimensional-arrays
- http://scikit-image.org/docs/dev/user_guide/numpy_images.html
- Think Python – How to Think Like a Computer Scientist
- https://en.wikibooks.org/wiki/Think_Python/Functions

Next week…

- Scripts:
  - Getting Python to interact with the operating system
- Automation:
  - How can we run 10's, 100's or 1000's of experiments by typing a single command?