Numerical Processing and Basic Data Visualization

01204111 Computers and Programming

Chaiporn Jaikaeo
Department of Computer Engineering
Kasetsart University

Cliparts are taken from http://openclipart.org
Revised 2017-10-23
Outline

• Numerical processing using NumPy library
  ◦ Arrays vs. lists
  ◦ One-dimensional (1D) arrays
  ◦ Two-dimensional (2D) arrays

• Basic data visualization using Matplotlib library
  ◦ Line plots
  ◦ Scatter plots
  ◦ Heat maps
NumPy Library

• NumPy library provides
  ◦ Data types such as array and matrix specifically designed for processing large amount of numerical data
  ◦ A large collection of mathematical operations and functions, especially for linear algebra
  ◦ A foundation to many other scientific libraries for Python

• NumPy is not part of standard Python
  ◦ But is included in scientific Python distributions such as Anaconda
Using NumPy

• NumPy library is named `numpy` and can be imported using the `import` keyword, e.g.,

```python
import numpy
a = numpy.array([1,2,3])
```

• By convention, the name `numpy` is renamed to `np` for convenience using the `as` keyword, e.g.,

```python
import numpy as np
a = np.array([1,2,3])
```

• From now on we will simply refer to `numpy` module as `np`
Arrays vs. Lists – *Similarities*

- NumPy's **arrays** and regular Python's **lists** share many similarities.
  - Array members are accessed using `[]` operator.
  - Arrays are **mutable**.
  - Arrays can be used as a sequence for a **list comprehensions** or a **for** loop.

```python
>>> import numpy as np
>>> a = np.array([1,2,3,4,5])
>>> a
array([1, 2, 3, 4, 5])
>>> a[2]
3
>>> a[3] = 8
>>> a
array([1, 2, 3, 8, 5])
>>> for x in a:
...     print(x)
1
2
3
8
5
```
Arrays vs. Lists — *Similarities*

- Arrays can be two-dimensional, similar to nested lists

```python
>>> import numpy as np
>>> table = np.array([[1,2,3],[4,5,6]])
>>> table
array([[1, 2, 3],
       [4, 5, 6]])
>>> table[0]   # one-index access gives a single row
array([1, 2, 3])
>>> table[1]   # two-index access gives a single element
array([4, 5, 6])
>>> table[0][1]
2
>>> table[1][2]
6
```
Arrays vs. Lists – **Differences**

- An array can be used directly in a mathematical expression, resulting in another array
  - They work like *vectors* in mathematics
  - Math operators such as +, -, *, /, ** work with arrays right away
  - Arrays in the same expression must have the same size

```python
>>> import numpy as np
>>> a = np.array([1,2,3,4,5])
>>> b = np.array([6,7,8,9,10])
>>> a-3
array([-2, -1,  0,  1,  2])
>>> a+b
array([ 7,  9, 11, 13, 15])
>>> (2*a + 3*b)/10
array([ 2. ,  2.5,  3. ,  3.5,  4. ])
```
Arrays vs. Lists – Differences

- Math functions can be performed over arrays
  - However, the functions must be *vectorized*
  - NumPy provides vectorized versions of all functions in the `math` module

```python
>>> import math
>>> import numpy as np
>>> a = np.array([1,2,3,4,5])
>>> math.sqrt(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: only length-1 arrays can be converted to Python scalars
```

NumPy provides a vectorized version of `sqrt`:

```python
>>> np.sqrt(a)
array([ 1.        ,  1.41421356,  1.73205081,  2.        ,  2.23606798])
```
Task: **Degree Conversion**

- Read a file containing a list of temperature values in degrees Celsius
- Print out all corresponding values in degrees Fahrenheit

Enter file name: degrees.txt

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>32.0</td>
</tr>
<tr>
<td>50.0</td>
</tr>
<tr>
<td>98.6</td>
</tr>
<tr>
<td>122.0</td>
</tr>
<tr>
<td>154.4</td>
</tr>
<tr>
<td>212.0</td>
</tr>
</tbody>
</table>

degrees.txt

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>37</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>68</td>
</tr>
<tr>
<td>100</td>
</tr>
</tbody>
</table>
Degree Conversion – Ideas

• Although techniques from previous chapters could be used, we will solve this problem using arrays

• Steps
  ◦ **Step 1:** read all values in the input file into an array
  ◦ **Step 2:** apply the conversion formula directly to the array
    \[ F = \frac{9}{5}C + 32 \]
  ◦ **Step 3:** print out the resulting array
Reading Data File using NumPy

• NumPy provides `loadtxt()` function that
  ◦ Reads a text file containing a list of numbers
  ◦ Converts number-like strings to floats by default
  ◦ Skips all empty lines automatically
  ◦ Returns all values as an array

```python
>>> import numpy as np
>>> c_array = np.loadtxt("degrees.txt")
>>> c_array
array([[ 0.,  10.,  37.,  50.,  68., 100.]])
```

• All the above are done within one function call
  ◦ No more puzzling list comprehension!
**Degree Conversion – Program**

```python
import numpy as np

filename = input("Enter file name: ")
c_array = np.loadtxt(filename)
f_array = 9/5*c_array + 32
for f in f_array:
    print(f)
```

Enter file name: **degrees.txt**

<table>
<thead>
<tr>
<th>32.0</th>
<th>50.0</th>
<th>98.6</th>
<th>122.0</th>
<th>154.4</th>
<th>212.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td>37</td>
<td>50</td>
<td>68</td>
<td>100</td>
</tr>
</tbody>
</table>
Task: **Data Set Statistics**

- Read a specified data set file containing a list of values
- Compute and report their *mean* and *standard deviation*

Enter file name: `values.txt`
Mean of the values is 39.47
Standard deviation of the values is 22.29

```plaintext
tabular data from values.txt:
68.70
31.53
16.94
9.95
52.55
29.65
64.01
69.52
30.08
21.77
```
Data Set Statistics – Ideas

• From statistics, the mean of the data set \((x_1, x_2, \ldots, x_n)\) is

\[
\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i
\]

\[
X = \text{<data set in NumPy array>}
\]

\[
n = \text{len}(X)
\]

\[
\text{mean} = \text{sum}(X)/n
\]

• And its standard deviation is

\[
\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2}
\]

\[
\text{stdev} = \text{np.sqrt(}
\text{sum((X-mean)**2)) / (n-1)}
\]

)
import numpy as np

filename = input("Enter file name: ")
X = np.loadtxt(filename)
n = len(X)
mean = sum(X)/n
stdev = np.sqrt( sum((X-mean)**2) / (n-1) )
print(f"Mean of the values is {mean:.2f}")
print(f"Standard deviation of the values is {stdev:.2f}")
Enter file name: values.txt
Mean of the values is 39.47
Standard deviation of the values is 22.29
Computing with 2D Arrays

• Processing **numerical tabular data** using 2D arrays offers several benefits over regular Python nested lists

• Some benefits are:
  ◦ Convenient text file reading and writing, including CSV files
  ◦ Math operations/functions are done in a vectorized style
  ◦ Much faster speed with large data sets
Task: **Score Query**

- Read a score table from the CSV file, named `scores.txt`, then
  - Show the numbers of students and subjects found in the input file
  - Ask user to query for a specified student's score in a specified subject

<table>
<thead>
<tr>
<th>Student</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>#1</td>
</tr>
<tr>
<td></td>
<td>75</td>
</tr>
<tr>
<td>#2</td>
<td>67</td>
</tr>
<tr>
<td>#3</td>
<td>58</td>
</tr>
</tbody>
</table>

Reading data from scores.txt
Found scores of 3 student(s) on 4 subject(s)
Enter student no.: 2
Enter subject no.: 1
Student #2's score on subject #1 is 67.0
The `loadtxt()` function also works with CSV files

- The parameter `delimiter=","` must be given

```python
>>> import numpy as np
>>> table = np.loadtxt("scores.txt",delimiter=",")
>>> table
array([[  75.,   34.,   64.,   82.],
       [  67.,   79.,   45.,   71.],
       [  58.,   74.,   79.,   63.]])
```
Arrays have several properties to describe their sizes, shapes, and arrangements.

- Observe no use of () because they are not functions.

```python
>>> table
array([[ 75.,  34.,  64.,  82.],
       [ 67.,  79.,  45.,  71.],
       [ 58.,  74.,  79.,  63.]]

>>> table.ndim  # give the number of array's dimension
2

>>> table.shape  # give the lengths in all dimensions
(3, 4)

>>> table.size  # give the total size
12
```
Caveats – One-Row Data File

- If input file contains only one row of data, `loadtxt()` will return a 1D array.
- To force 2D array reading, call `loadtxt()` with the parameter `ndmin=2`.

```python
>>> import numpy as np
>>> table = np.loadtxt("1row.txt",delimiter="\"\")
>>> table
array([ 75.,  34.,  64.,  82.])
>>> table.ndim 1
>>> table.shape (4,)
```

- Force minimum number of dimensions to 2.

```python
>>> table = np.loadtxt("1row.txt",delimiter="\",ndmin=2)
>>> table
array([[ 75.,  34.,  64.,  82.]])
>>> table.ndim 2
>>> table.shape (1, 4)
```

One dimension
4 members

Two dimensions
1x4 members

Force minimum number of dimensions to 2
import numpy as np

FILENAME = "scores.txt"
print(f"Reading data from {FILENAME}"

table = np.loadtxt(FILENAME, delimiter="", ndmin=2)
nrows, ncols = table.shape
print(f"Found scores of {nrows} student(s) on {ncols} subject(s)"

student_no = int(input("Enter student no.: "))
subject_no = int(input("Enter subject no.: "))
score = table[student_no-1][subject_no-1]
print(f"Student #{student_no}'s score on subject #{subject_no} is {score}"

Reading data from scores.txt
Found scores of 3 student(s) in 4 subject(s)
Enter student no.: 3
Enter subject no.: 4
Student #3's score on subject #4 is 63.0
Task: **Who Fails**

- Read a score table from the CSV file `scores.txt`, then report who fails which subject

<table>
<thead>
<tr>
<th>Student</th>
<th>Subject</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>75</td>
<td>34</td>
<td>64</td>
<td>82</td>
<td></td>
</tr>
<tr>
<td>#2</td>
<td>67</td>
<td>79</td>
<td>45</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>#3</td>
<td>58</td>
<td>74</td>
<td>79</td>
<td>63</td>
<td></td>
</tr>
</tbody>
</table>

Reading data from scores.txt
Found scores of 3 student(s) on 4 subject(s)
Student #1 fails subject #2 with score 34.0
Student #2 fails subject #3 with score 45.0
### Who Fails – Ideas

- Find student who fails the exam (score < 50)
- Scan through all scores and check one by one

<table>
<thead>
<tr>
<th></th>
<th>Subject 1</th>
<th>Subject 2</th>
<th>Subject 3</th>
<th>Subject 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student 1</td>
<td>75</td>
<td>34</td>
<td>64</td>
<td>82</td>
</tr>
<tr>
<td>Student 2</td>
<td>67</td>
<td>79</td>
<td>45</td>
<td>71</td>
</tr>
<tr>
<td>Student 3</td>
<td>58</td>
<td>74</td>
<td>79</td>
<td>63</td>
</tr>
</tbody>
</table>
Who Fails – Steps

• For each student (each row),
  ◦ Go from the first to the last subject (columns)
  ◦ Check whether the score is below 50 or not
  ◦ If so, print out the student #, subject #, and the score
import numpy as np

FILENAME = "scores.txt"
print(f"Reading data from {FILENAME}"
table = np.loadtxt(FILENAME, delimiter="", ndmin=2)
nrows, ncols = table.shape
print(f"Found scores of {nrows} student(s) on {ncols} subject(s)"

for r in range(nrows):
    for c in range(ncols):
        score = table[r][c]
        if score < 50:
            print(f"Student #{r+1} fails subject #{c+1} with score {score}"

Reading data from scores.txt
Found scores of 3 student(s) on 4 subject(s)
Student #1 fails subject #2 with score 34.0
Student #2 fails subject #3 with score 45.0

scores.txt
75,34,64,82
67,79,45,71
58,74,79,63
Task: Score Sheet Summary

- Read a score table from the CSV file `scores.txt`, then
  - Display the total score for each student
  - Display the average for each subject

Reading data from scores.txt
Found scores of 3 student(s) on 4 subject(s)

Student Summary
===============
Total score for student #1: 255
Total score for student #2: 262
Total score for student #3: 274

Subject Summary
===============
Average score for subject #1: 66.67
Average score for subject #2: 62.33
Average score for subject #3: 62.67
Average score for subject #4: 72.00

<table>
<thead>
<tr>
<th>Student</th>
<th>Subject</th>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td></td>
<td>75</td>
<td>34</td>
<td>64</td>
<td>82</td>
</tr>
<tr>
<td>#2</td>
<td></td>
<td>67</td>
<td>79</td>
<td>45</td>
<td>71</td>
</tr>
<tr>
<td>#3</td>
<td></td>
<td>58</td>
<td>74</td>
<td>79</td>
<td>63</td>
</tr>
</tbody>
</table>

`scores.txt`
75,34,64,82
67,79,45,71
58,74,79,63
Score Sheet Summary – Ideas

• The array read from the file can be seen as a list of rows

• We can traverse all the rows (i.e., students) with a for loop
  ◦ Then compute summation on each row

```python
>>> import numpy as np
>>> table = np.loadtxt("scores.txt",delimiter="",ndmin=2)
>>> table
array([[ 75.,  34.,  64.,  82.],
       [ 67.,  79.,  45.,  71.],
       [ 58.,  74.,  79.,  63.]])
>>> table[0]
array([ 75.,  34.,  64.,  82.])
```

• How to do the same by columns (i.e., subjects)
Array Transpose

- NumPy's arrays have the `array.T` property for viewing the transposed version of the arrays
  - This allows 2D array traversal by columns
  - The transpose provides a different view to an array, not a copy of it

```python
>>> table
array([[ 75.,  34.,  64.,  82.],
       [ 67.,  79.,  45.,  71.],
       [ 58.,  74.,  79.,  63.]])

>>> table.T
array([[ 75.,  67.,  58.],
       [ 34.,  79.,  74.],
       [ 64.,  45.,  79.],
       [ 82.,  71.,  63.]]))

>>> table.T[0]
array([ 75.,  67.,  58.])```
Score Sheet Summary – Program

```python
import numpy as np

FILENAME = "scores.txt"
print(f"Reading data from {FILENAME}")
table = np.loadtxt(FILENAME, delimiter="", ndmin=2)
nrows, ncols = table.shape
print(f"Found scores of {nrows} student(s) on {ncols} subject(s)"
print()
print("Student Summary")
print("=" * 60"
for r in range(nrows):
    print(f"Total score for student #{r+1}: {sum(table[r]):.0f}"
print()
print("Subject Summary")
print("=" * 60"
for c in range(ncols):
    avg = sum(table.T[c])/len(table.T[c])
    print(f"Average score for subject #{c+1}: {avg:.2f}"
```

Access the entire row `r`

Access the entire column `c`
Trivia: 2D Array Member Access

To access the element at row#i, column#j in 2D array, NumPy supports both $[i][j]$ and $[i,j]$ forms

- The $[i,j]$ form does not work with nested lists
- The $[i,j]$ form is often found in other programming languages such as MATLAB

```python
>>> import numpy as np
>>> a = np.array([[1,2,3],[4,5,6]])
>>> a
array([[1, 2, 3],
       [4, 5, 6]])
>>> a[0][2]
3
>>> a[0,2]
3
```
Bonus: Matrix Processing

- NumPy provides `matrix()` function for matrix creation
- Certain operations, such as *multiplication*, have a different meaning with matrices

```python
>>> import numpy as np
>>> a = np.array([[1, 2],[3, 4]])
>>> b = np.array([[5, 6],[7, 8]])
>>> a
array([[1, 2],
       [3, 4]])
>>> b
array([[5, 6],
       [7, 8]])
>>> a*b
array([[ 5, 12],
       [21, 32]])
>>> b*a
array([[ 5, 12],
       [21, 32]])
```

```
matrix-style multiplication
```

```
>>> a = np.matrix([[1, 2],[3, 4]])
>>> b = np.matrix([[5, 6],[7, 8]])
>>> a
matrix([[1, 2],
         [3, 4]])
>>> b
matrix([[5, 6],
         [7, 8]])
>>> a*b
matrix([[19, 22],
         [43, 50]])
>>> b*a
matrix([[23, 34],
         [31, 46]])
```

```
element-wise multiplication
```
Numbers have an important story to tell. They rely on you to give them a clear and convincing voice.

— Stephen Few —
Author of “Now You See It”
Matplotlib Library

- Matplotlib provides a rich set of data visualization tools
- Like NumPy, Matplotlib is not part of standard Python
  - Anaconda and other scientific Python distributions come with it

Images from http://matplotlib.org
Using Matplotlib

- Matplotlib library has many submodules, but most commonly used submodule is **pyplot**
  - It provides functions similar to that of MATLAB

- Pyplot can be imported using the **import** keyword, e.g.,

```python
import matplotlib.pyplot
matplotlib.pyplot.plot([1,2,3],[4,5,6])
matplotlib.pyplot.show()
```

- By convention, the lengthy module name is renamed to **plt**, e.g.,

```python
import matplotlib.pyplot as plt
plt.plot([1,2,3],[4,5,6])
plt.show()
```
Task: **Function Grapher**

- Create a chart with two line plots representing two relations, 
  \[ y_1 = \sin \left( \frac{\pi x}{2} \right) \text{ and } y_2 = \frac{1}{2} \cos \left( \frac{\pi x}{3} \right), \] for \( 0 \leq x \leq 5 \)
Creating Line Plots

• Pyplot provides `plot()` function to create a line plot
  ◦ `plot()` is very flexible for taking parameters
  ◦ We will use the form `plot(x, y)` where `x` and `y` are sequences (lists or arrays) for the x-axis and y-axis coordinates, respectively

• Call the `show()` function to show all the plot(s) created

```python
import matplotlib.pyplot as plt
plt.plot([1, 2, 3], [4, 3, 6])
plt.plot([1, 2, 3], [2, 4, 5])
plt.show()
```
Function Grapher – Ideas

- Create an array for the values of \( x \), \( 0 \leq x \leq 5 \)
- Compute two other arrays for \( y_1 \) and \( y_2 \)
- Create plots for \( x \) vs. \( y_1 \) and \( x \) vs. \( y_2 \)

```
import numpy as np
import matplotlib.pyplot as plt
x = np.array([0,1,2,3,4,5])
y1 = np.sin(np.pi*x/2)
y2 = 1/2*np.cos(np.pi*x/3)
plt.plot(x,y1)
plt.plot(x,y2)
plt.show()
```
Refining the Range

- Previous plots look very rough because using only \([0,1,2,3,4,5]\) for the values of \(x\) is too coarse.

- NumPy provides two functions to make finer sequences:
  - \texttt{arange\(start, stop, step)\) - like \texttt{range()\) but accepts fractional step size and returns an array (\texttt{stop} is excluded).
  - \texttt{linspace\(start, stop, length\) makes an array of length values from \texttt{start} to \texttt{stop} (including \texttt{stop} itself).

```python
>>> import numpy as np
>>> np.arange(0,5,0.5)
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5])
>>> np.linspace(0,5,11)
array([ 0. ,  0.5,  1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5,  5. ])
```
Function Grapher – Ideas

• We will use the `linspace()` function to create a finer sequence of $x$
  ◦ Create 100 points from 0 to 5 to make smoother curves

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

x = np.linspace(0, 5, 100)
y1 = np.sin(np.pi*x/2)
y2 = 1/2*np.cos(np.pi*x/3)

plt.plot(x, y1)
plt.plot(x, y2)
plt.show()
```
Decorating the Chart

- Call `plot(…, label=s)` to assign a legend label for the plot
- Call `legend()` to collect all plots' labels and create a legend box
- Call `xlabel(s)` and `ylabel(s)` to create labels for x-axis and y-axis, respectively
- Call `grid(True)` to display grid lines
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 5, 100)
y1 = np.sin(np.pi*x/2)
y2 = 1/2*np.cos(np.pi*x/3)
plt.plot(x, y1, label="y1")
plt.plot(x, y2, label="y2")

# decorate the figure
plt.grid(True)
plt.xlabel("x")
plt.ylabel("y")
plt.legend()

plt.show()
Task: Cannon Ball

• Plot the trajectory of a cannon ball when it's fired at four different angles: $30^\circ$, $45^\circ$, $60^\circ$, and $75^\circ$
  ◦ Only show the trajectory in the first 10 seconds only
  ◦ Suppose the initial speed of the cannon ball is 100 m/s
Cannon Ball – Ideas

- We simply use the projectile-motion formulas from high-school physics:

\[
x(t) = u \cos(\theta) \times t, \\
y(t) = u \sin(\theta) \times t - \frac{1}{2} g t^2
\]

where \( u \) is the initial speed (in m/s) and \( g \) is the Earth's gravitational acceleration, which is 9.81 m/s\(^2\)
Cannon Ball – Steps

• The program will roughly work in the following steps

• **Step 1:** Prepare an array of time values, \( t \), from 0 to 10 using `linspace()` function

• **Step 2:** For each angle, \( \theta \)
  ◦ **2.1:** Compute an array of distances, \( x \), at time \( t \)
  ◦ **2.2:** Compute an array of heights, \( y \), at time \( t \)
  ◦ **2.3:** Create a line plot using the arrays \( x \) and \( y \), along with the label

• **Step 3:** Decorate and show the chart
import numpy as np
import matplotlib.pyplot as plt

g = 9.81  # Earth's gravity in m/s^2
u = 100   # initial speed in m/s
t = np.linspace(0,10,100)
angles = [30,45,60,75]
for theta in angles:
    x = u*np.cos(np.radians(theta))*t
    y = u*np.sin(np.radians(theta))*t - 1/2*g*(t**2)
    plt.plot(x,y,label=f"angle = {theta}")

# decorate the figure
plt.xlabel("distance (m)"")
plt.ylabel("height (m)"")
plt.legend()
plt.grid(True)

plt.show()
Task: *BMI Scatter Plot*

- Read pairs of weights (in kg) and heights (in cm) from a CSV file
- Compute the BMI value for each \((weight, height)\) pair
- Display a *scatter plot* using weights as the x values, and heights as the y values
- Use different color for each point, based on the BMI value
  - Also display a color bar
- Download full data file from
  - [https://elab.cpe.ku.ac.th/data/body.txt](https://elab.cpe.ku.ac.th/data/body.txt)

```plaintext
65.6, 174.0
71.8, 175.3
80.7, 193.5
72.6, 186.5
```
Creating a Scatter Plot

- Pyplot prepares the `scatter()` function for tasks like this:
  - At least two parameters, x and y, must be specified.

- The following example creates a scatter plot with points (1,4), (2,3), and (3,6):

```python
import matplotlib.pyplot as plt
x = [1, 2, 3]
y = [4, 3, 6]
plt.scatter(x, y)
plt.show()
```
Visualizing 3\textsuperscript{rd} Variable

- A basic scatter plot only displays two variables using coordinates \((x, y)\).
- A third variable can be \textbf{color-coded} into the plot by calling \texttt{scatter}(x, y, c=var3) where \texttt{var3} is a sequence for the third variable.

```python
import matplotlib.pyplot as plt
x = [1, 2, 3]
y = [4, 3, 6]
z = [10, 20, 30]
plt.scatter(x, y, c=z)
plt.show()
```
Adding Color Bar

- A **color bar** indicates the values of the color codes
  - Call `colorbar()` to add a color bar to the plot
  - Use the returned value to set the color bar's title via its `colorbar.ax.set_title()` method

```python
import matplotlib.pyplot as plt
x = [1, 2, 3]
y = [4, 3, 6]
z = [10, 20, 30]
plt.scatter(x, y, c=z)
cbar = plt.colorbar()
cbar.ax.set_title("z value")
plt.show()
```
Changing Color Map

- If you don't like the default color map, you can always choose a different one using the `set_cmap(name)` function, where name is the colormap's name.
  - Refer to [https://matplotlib.org/examples/color/colormaps_reference.html](https://matplotlib.org/examples/color/colormaps_reference.html) for a complete list.

```python
import matplotlib.pyplot as plt
x = [1, 2, 3]
y = [4, 3, 6]
z = [10, 20, 30]
plt.scatter(x, y, c=z)
cbar = plt.colorbar()
cbar.ax.set_title("z value")
plt.set_cmap("jet")
plt.show()
```
BMI Scatter Plot – Ideas

• We will create a scatter plot with three variables
  ◦ weights \(\rightarrow\) x values
  ◦ heights \(\rightarrow\) y values
  ◦ BMI \(\rightarrow\) color codes

• BMI are computed from weights and heights using the formula

\[
BMI = \frac{weight_{kg}}{(height_m)^2}
\]

• The calculation can be done directly over NumPy's arrays
import numpy as np
import matplotlib.pyplot as plt

table = np.loadtxt("body.txt",delimiter="",ndmin=2)
weight = table.T[0]  # extract weights from column#0
height = table.T[1]  # extract heights from column#1
bmi = weight/((height/100)**2)
plt.scatter(weight,height,c=bmi)

# decorating the chart
plt.xlabel("Weight (kg)")
plt.ylabel("Height (cm)")
plt.grid(True)
cbar = plt.colorbar()
cbar.ax.set_title("BMI")
plt.set_cmap("jet")
plt.show()
Bonus: Heat Map

- A heat map is a representation of 2D data in forms of color coding.
- Pyplot provides the `imshow()` function that conveniently generates a heat map from data in a 2D array (or a nested list).

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

data = np.loadtxt("temperature.txt", delimiter=",")
plt.imshow(data)

# decorate the figure
bar = plt.colorbar()
bar.ax.set_title("Degrees C")
plt.set_cmap("jet")
plt.show()
```

Download full `temperature.txt` file from https://elab.cpe.ku.ac.th/data/temperature.txt
Conclusion

• NumPy library offers the array datatype that allows processing lists of numbers all at once

• NumPy's arrays can be 1D arrays, 2D arrays, or even higher-dimensional arrays

• Matplotlib library provides many functions for creating various kinds of visualization from data stored in arrays
• Loading **numpy** and refer to it as np

```
import numpy as np
```

• Creating a 1D array of length $N$

```
A = np.array([val_0, val_1, ..., val_{N-1}])
```

• Creating a 2D array of $M$ rows and $N$ columns

```
A = np.array([
    [val_{0,0}, val_{0,1}, ..., val_{0,N-1}],
    [val_{1,0}, val_{1,1}, ..., val_{1,N-1}],
    ...
    [val_{M-1,0}, val_{M-1,1}, ..., val_{M-1,N-1}]
])
```
Syntax Summary (2)

- Accessing array's member at position \( i \) (starting at index 0)
  - Give a single value for 1D array, a row of values for 2D array
    \[ A[i] \]

- Accessing 2D array's member at row \( i \), column \( j \) (starting at 0,0)
  \[ A[i][j] \]

- Retrieving array's properties
  
  \[ A.ndim \]  # gives the number of array's dimensions
  \[ A.shape \]  # gives the lengths in all dimensions
  \[ A.size \]  # gives the total array size
  \[ A.T \]  # gives the transpose of the array
Syntax Summary (3)

• Reading a file, `filename`, containing a list of numbers as a 1D array

\[ A = \text{np.loadtxt}(\text{filename}) \]

• Reading a CSV file, `filename`, containing a table of numbers as a 2D array

\[ A = \text{np.loadtxt}(\text{filename}, \text{delimiter}=',', \text{ndmin}=2) \]

• Creating a 1D array of values from `start` to `stop` (excluding `stop`) with step

\[ A = \text{np.arange}(\text{start}, \text{stop}, \text{step}) \]

• Creating a 1D array of `nsteps` values from `start` to `stop` (including `stop`)

\[ A = \text{np.linspace}(\text{start}, \text{stop}, \text{nsteps}) \]
Syntax Summary (4)

- Loading `matplotlib.pyplot` module and refer to it as `plt`

```python
import matplotlib.pyplot as plt
```

- Creating a line plot of $X$ versus $Y$, where $X$ and $Y$ are 1D arrays, and give the label $s$ to the plot

```python
plt.plot(X, Y, label=s)
```

- Creating a scatter plot of $(x,y)$ points from 1D arrays $X$ and $Y$

```python
plt.scatter(X, Y)
```

- Creating a scatter plot of $(x,y)$ points from 1D arrays $X$ and $Y$, with color codes from 1D array $C$

```python
plt.scatter(X, Y, c=C)
```
Syntax Summary (5)

• Adding a legend box into the chart
  ```python
  plt.legend()
  ```

• Setting x-axis and y-axis labels
  ```python
  plt.xlabel(s)
  plt.ylabel(s)
  ```

• Displaying grid lines over the chart
  ```python
  plt.grid(True)
  ```

• Show the chart with all the created plots and settings
  ```python
  plt.show()
  ```
Syntax Summary (6)

• Adding a color bar with a title to the chart

```
bar = plt.colorbar()
bar.ax.set_title(s)
```

• Set the color map

```
plt.set_cmap(name)
```

• Creating a heat map from a 2D array

```
plt.imshow(A)
```
References

• NumPy User Guide
  ◦ https://docs.scipy.org/doc/numpy/user/index.html

• Matplotlib User's Guide
  ◦ https://matplotlib.org/users/index.html

• Introduction to Scientific Python: NumPy, SciPy, and Matplotlib by Sven Schmit at Stanford University
Revision History

- September 2016 – Supaporn Erjongmanee (supaporn.e@ku.ac.th)
  - Prepared slides for 2D arrays in C#

- October 2017 – Chaiporn Jaikaeo (chaiporn.j@ku.ac.th)
  - Revised for Python and added basic visualization