Introduction to Python Programming II

Theo Narh (Ph.D)

Graduate School of Nuclear & Allied Sciences (SNAS)
University of Ghana

Outline

Functions

Defining Your Own Functions I

- We can define our own functions in Python and reuse them throughout the program. The syntax for defining a function is as follows:

```
def functionName(parameters):
  code detailing what the function should do
  return [expression]
```

- There are two keywords here, def and return.
- **def**

  tells the program that the indented code from the next line onwards is part of the function.
- **return**

  is the keyword that we use to return an answer from the function. There can be more than one return statements in a function.

Defining Your Own Functions II

- However, once the function executes a return statement, the function will exit. If your function does not need to return any value, you can omit the return statement. Alternatively, you can write return or return None.
- Suppose we want to determine if a given number is a prime number. Here's how we can define the function using the modulus (%) operator

```
def checkIfPrime(numberToCheck):
  for x in range(2, numberToCheck):
    if (numberToCheck%x == 0):
      return False
  return True
```

- To use this function, we type

```
answer = checkIfPrime(13)
```

Variable Scope I

- An important concept to understand when defining a function is the concept of variable scope. Variables defined inside a function are treated differently from variables defined outside. There are two main differences.
- Firstly, any variable declared inside a function is only accessible within the function. These are known as local variables. Any variable declared outside a function is known as a global variable and is accessible anywhere in the program.

Variable Scope II

- To understand this, try the code below:

```
def foo():
    y = "local"
```

- then RUN this command::

```
foo()
print(y)
```

- Output::

```
NameError: name 'y' is not defined
```

Variable Scope III

- The output shows an error because we are trying to access a local variable y in a global scope whereas the local variable only works inside foo() or local scope.
- Create a variable outside of a function, and use it inside the function

```
x = "awesome"
def myfunc():
  print("Python is " + x)

myfunc()
```

- Output::

```
Python is awesome
```

Variable Scope IV

- Create a variable inside a function, with the same name as the global variable

```
x = "awesome"
def myfunc():
  x = "fantastic"
  print("Python is " + x)

myfunc()
print("Python is " + x)
```

- Output::

```
Python is fantastic
Python is awesome
```

Python modules

SymPy I

1. Basic operations
   - Evaluate $\cos(x) + 1$ at $x = 0$.

```python
from sympy import *
x = symbols("x")

expr = cos(x) + 1
print(expr.subs(x, 0))
```

   - Ans: 2

SymPy II

2. Sums and Products:: $\sum_{i=1}^{4} x + iy$, $\prod_{i=0}^{5} x + iy$

```
>>> from sympy import *
>>> init_printing(use_latex=True)

>>> x, y, i = symbols('x, y, i')
>>> summation(x + i*y, (i, 1, 4))  # Sum over i=1,2,3,4.
4*x + 10*y
>>> product(x + i*y, (i, 0, 5))  # Multiply over i=0,1,2,3
x*(x + y)*(x + 2*y)*(x + 3*y)*(x + 4*y)*(x + 5*y)
```

SymPy III

3. Linear Algebra

$$
\begin{aligned}
x + y + z &= 5\\
2x + 4y + 3z &= 2\\
5x + 10y + 2z &= 4
\end{aligned}
$$

SymPy IV

```
>>> x, y, z = symbols('x y z')

>>> # Define the augmented matrix M = [A|b].
>>> M = Matrix([ [1, 1, 1, 5],
... [2, 4, 3, 2],
... [5, 10, 2, 4] ])

>>> # Solve the system by::
>>> solve_linear_system(M, x, y, z)
    98      -45
{x: --, y: ----, z: 2/11}
    11       11
```

4. Trigonometry

```
>>> x = Symbol('x')

>>> expr = sin(2*x) + cos(2*x)

>>> expand_trig(expr)
                          2
2*sin(x)*cos(x) + 2*cos (x) - 1

>>> trigsimp(cos(x)**2 + sin(x)**2)
1
```

SymPy VI

5. Factorise

```
>>> x, y = symbols('x y')
>>> factor(x**2 -y**2)
(x - y)*(x + y)
```

SymPy VII

6. simplify

```
>>> x, y = symbols('x y')
>>> simplify((x**3 + x**2 - x -1) / (x**2 + 2*x +1))
x - 1
>>> simplify((x**4 -1) / (x-1))
 4
x  - 1
------
x - 1
```

SymPy VIII

7. expand

```
>>> x, y = symbols('x y')
>>> expand((x + 2*y)**3)
 3      2           2      3
x  + 6*x *y + 12*x*y  + 8*y
```

SymPy IX

8. Change of subject: $a = \frac{b+cd}{e}$

```
>>> a, b, c, d, e  = symbols('a b c d e')
>>> expr1 = Eq(a, (b + c*d)/e)
>>> expr1
    b + c*d
a = -------
       e

>>> # make d the subject
>>> expr2 = Eq(d, solve(expr1, d)[0])
>>> expr2
    a*e - b
d = -------
       c
```

Numpy I

- This package provides basic routines for manipulating large arrays and matrices of numeric data.
- There are several ways to import NumPy. The standard approach is to use a simple import statement:

```
import numpy
import numpy as np
from numpy import *
```

1. **Arrays**: Arrays are similar to lists in Python, except that every element of an array must be of the same type, typically a numeric type like float or int. Arrays make operations with large amounts of numeric data very fast and are generally much more efficient than lists.

1. Create an array $A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$.

```
>>> import numpy as np
>>> A = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> A
[[1. 2. 3.]
 [4. 5. 6.]]
```

2. Select the **first** element in A.

```
>>> A[0,0]
1.0
```

3. Select the **last** row in A.

```
>>> A[1,:]
[4. 5. 6.]
```

Numpy III

4. Select the **second** column in A.

```
>>> A[:,1]
[2. 5.]
```

5. Select the **first two** columns in A.

```
>>> A[:,:2]
 [[1. 2.]
 [4. 5.]]
```

6. Transpose A.

```
>>> A.transpose()
 [[1. 4.]
  [2. 5.]
 [3. 6.]]
```

7. Convert A into binary string (i.e., not in human-readable form) and revert.

```
>>> s = A.tostring()
>>> s
b'\x00\x00\x00\x00\x00\x00\xf0?\x00\x00\x00\x00\x00\x00\x00
00\x00\x08@\x00\x00\x00\x00\x00\x00\x10@\x00\x00\x00\x00\x0
x00\x00\x00\x00\x18@'

>>> np.fromstring(s)
[1. 2. 3. 4. 5. 6.]
```

2. Concatenate $\boldsymbol{B}$ and $\boldsymbol{C}$:

$$B = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad C = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

Numpy V

```
>>> B = np.array([[1, 2], [3, 4]], float)
>>> C = np.array([[5, 6], [7,8]], float)

>>> np.concatenate((B,C))
 [[1. 2.]
  [3. 4.]
  [5. 6.]
 [7. 8.]]
>>>
>>> np.concatenate((B,C), axis=0)
 [[1. 2.]
  [3. 4.]
  [5. 6.]
 [7. 8.]]
```

```
>>> np.concatenate((B,C), axis=1)
 [[1. 2. 5. 6.]
 [3. 4. 7. 8.]]
```

3. Other ways to create arrays
   1. `arange` function

      ```
      >>> np.arange(5, dtype=float)
      [0. 1. 2. 3. 4.]

      >>> np.arange(1, 6, 2, dtype=int)
      [1 3 5]
      ```

Numpy VII

2. `newaxis` function

```
>>> a = np.array([1, 2, 3], float)
>>> a
[1. 2. 3.]
>>> a[:,np.newaxis]
[[1.]
 [2.]
 [3.]]
>>> a[:,np.newaxis].shape
(3, 1)
```

3. `zeros` `and` `ones` `like` functions

```
>>> a = np.array([[1, 2, 3], [4, 5, 6]], float)
>>> a
[[1. 2. 3.]
 [4. 5. 6.]]
>>> np.zeros_like(a)
[[0. 0. 0.]
 [0. 0. 0.]]
>>> np.ones_like(a)
[[1. 1. 1.]
 [1. 1. 1.]]
```

Numpy IX

4. `array mathematics` function

```
>>> a = np.array([1,2,3], float)
>>> b = np.array([5,2,6], float)
>>> a + b
[6. 4. 9.]
>>> a - b
[-4.  0. -3.]
>>> a % b
[1. 0. 3.]
>>> b**a
[  5.   4. 216.]
```

Numpy X

5. array iteration

```
>>> a = np.array([1, 4, 5], int)
>>> for x in a:
...     print(x)
... pass
  File "<stdin>", line 3
    pass
    ^
SyntaxError: invalid syntax
```

6. array basic operation

```
>>> import numpy as np
>>> v = np.array([2, 4, 3], float)
>>> v.sum()
9.0
>>> v.prod()
24.0
>>> v.mean()
3.0
>>> v.var()
0.6666666666666666
>>> v.min()
2.0
>>> v.max()
4.0
>>> v = np.array([[0, 2], [3, -1], [3, 5]], float)
>>> v.mean(axis=0)
```

```
array([2., 2.])
>>> v.mean(axis=1)
array([1., 1., 4.])
>>> v.min(axis=1)
array([ 0., -1.,  3.])
>>> v.max(axis=0)
array([3., 5.])
>>> v = np.array([6, 2, 5, -1, 0], float)
>>> sorted(v)
[-1.0, 0.0, 2.0, 5.0, 6.0]
>>> v.sort()
>>> v = np.array([1, 1, 4, 5, 5, 5, 7], float)
>>> np.unique(v)
array([1., 4., 5., 7.])
>>> v = np.array([[1, 2], [3, 4]], float)
>>> v.diagonal()
array([1., 4.])
```

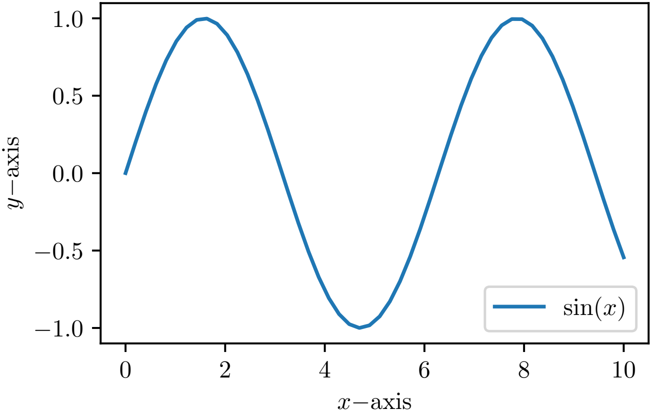
- Plot $\sin(x)$, $x \in [0, 10]$

```python
import numpy as np
from matplotlib.pylab import plt

plt.rc('text', usetex=True)
plt.rc('font', family='serif')
plt.rc('font', size=10.0)
plt.rc('legend', fontsize=10.0)
plt.rc('font', weight='normal')
x = np.linspace(0, 10)
plt.figure(figsize=(4, 2.5))
plt.plot(x, np.sin(x), label='$\sin(x)$')
plt.xlabel(r'$x\mathrm{-axis}$')
plt.ylabel(r'$y\mathrm{-axis}$')
plt.legend(loc='lower right')
```

Visualization with Matplotlib II

```python
plt.savefig('myplot1.pdf', bbox_inches='tight')

# Include the plot in the current LaTeX document
print(r"\begin{center}")
print(r"\includegraphics[width=0.75\textwidth]{myplot1.pdf
print(r"\end{center}")
```

Visualization with Matplotlib III



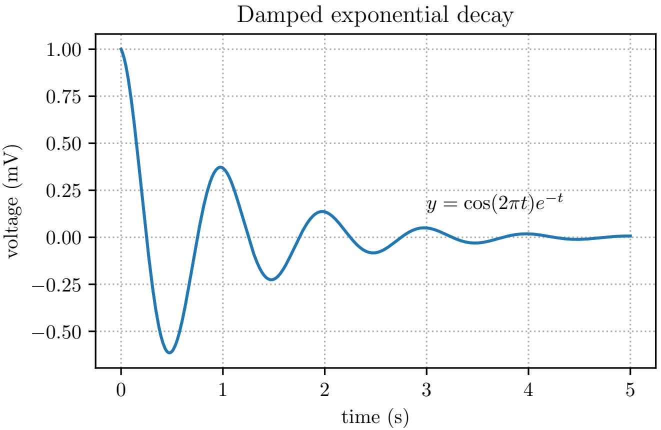
- plot $\cos(2\pi t)e^{-t}$, $t \in [0, 5]$

```python
import numpy as np
from matplotlib.pylab import plt
# Define f(t), the desired function to plot
def f(t):
    return np.cos(2 * np.pi * t) * np.exp(-t)
# Generate the points (t_i, y_i) to plot
t = np.linspace(0, 5, 500)
y = f(t)
# Begin with an empty plot, 5 x 3 inches
plt.clf()
plt.figure(figsize=(5, 3))
# Use TeX fonts
plt.rc("text", usetex=True)

```

Visualization with Matplotlib V

```python
# Generate the plot with annotations
plt.plot(t, y)
plt.title("Damped exponential decay")
plt.text(3, 0.15, r"$y = \cos(2 \pi t) e^{-t}$")
plt.xlabel("time (s)")
plt.ylabel("voltage (mV)")
plt.grid(linestyle='dotted')
# Save the plot as a PDF file
plt.savefig("myplot.pdf", bbox_inches="tight")
# Include the plot in the current LaTeX document
print(r"\begin{center}")
print(r"\includegraphics[width=0.75\textwidth]{myplot.pdf}
print(r"\end{center}")
```

Visualization with Matplotlib VI



Visualization with Matplotlib VII

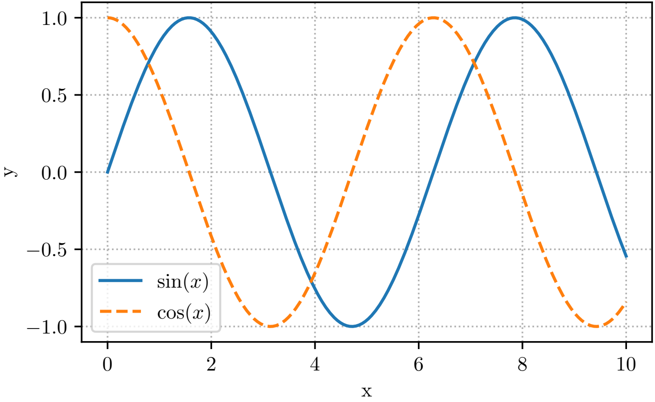
- Making multiple plots on a single figure.

```
import numpy as np
from matplotlib.pylab import plt

x = np.linspace(0, 10, 500)
plt.clf()
plt.figure(figsize=(5, 3))
# Use TeX fonts
plt.rc("text", usetex=True)

plt.plot(x, np.sin(x), '-', label=r'$\sin(x)$')
plt.plot(x, np.cos(x), '--', label=r'$\cos(x)$');
plt.xlabel("x")
plt.ylabel("y")
plt.grid(linestyle='dotted')
```

Visualization with Matplotlib VIII

```python
plt.legend(loc='best')

# Save the plot as a PDF file
plt.savefig("myplot2.pdf", bbox_inches="tight")
# Include the plot in the current LaTeX document
print(r"\begin{center}")
print(r"\includegraphics[width=0.75\textwidth]{myplot2.pdf
print(r"\end{center}")
```

Visualization with Matplotlib IX

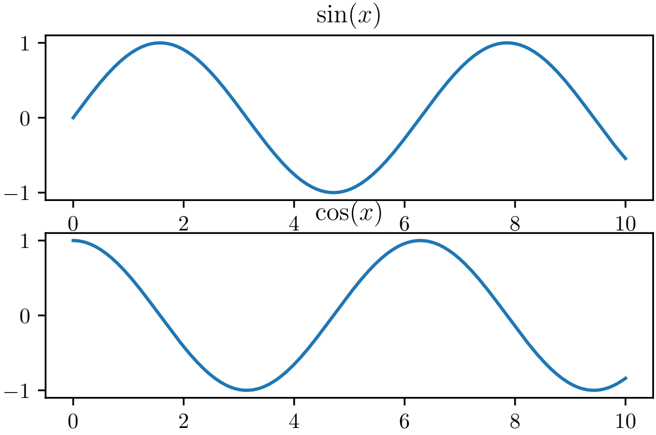


- Making multiple plots on different figures.

```python
import numpy as np
from matplotlib.pylab import plt

x = np.linspace(0, 10, 500)
plt.clf()
plt.figure(figsize=(5, 3))
# Use TeX fonts
plt.rc("text", usetex=True)

# create the first of two panels and set current axis
plt.subplot(2, 1, 1) # (rows, columns, panel number)
plt.plot(x, np.sin(x))
plt.title(r"$\sin (x)$")

```

Visualization with Matplotlib XI

```python
# create the second panel and set current axis
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x));
plt.title(r"$\cos (x)$")

# Save the plot as a PDF file
plt.savefig("myplot3.pdf", bbox_inches="tight")
# Include the plot in the current LaTeX document
print(r"\begin{center}")
print(r"\includegraphics[width=0.75\textwidth]{myplot3.pdf
print(r"\end{center}")

```


sin(x)
1
0
−1
0
2
4
6
8
10
cos(x)
1
0
−1
0
2
4
6
8
10

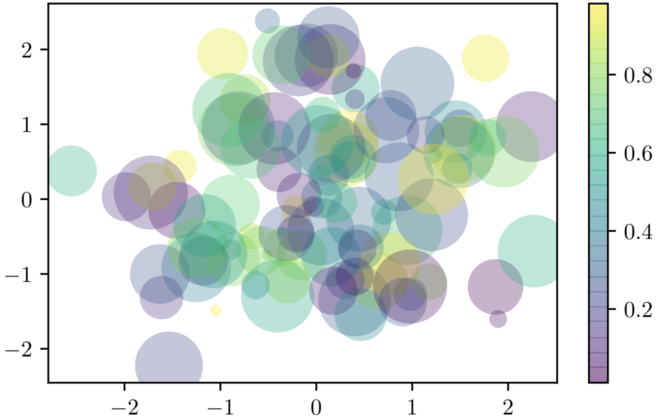
- Create a scatter plot.

```
import numpy as np
from matplotlib.pylab import plt

plt.clf()
plt.figure(figsize=(5, 3))
# Use TeX fonts
plt.rc("text", usetex=True)

# create the scatter plot

rng = np.random.RandomState(0)
x = rng.randn(100)
y = rng.randn(100)
colors = rng.rand(100)
```

Visualization with Matplotlib XIV

```
sizes = 1000 * rng.rand(100)

plt.scatter(x, y, c=colors, s=sizes, alpha=0.3,
cmap='viridis')
plt.colorbar();  # show color scale

# Save the plot as a PDF file
plt.savefig("myplot4.pdf", bbox_inches="tight")
# Include the plot in the current LaTeX document
print(r"\begin{center}")
print(r"\includegraphics[width=0.75\textwidth]{myplot4.pdf
print(r"\end{center}")

```

Visualization with Matplotlib XV



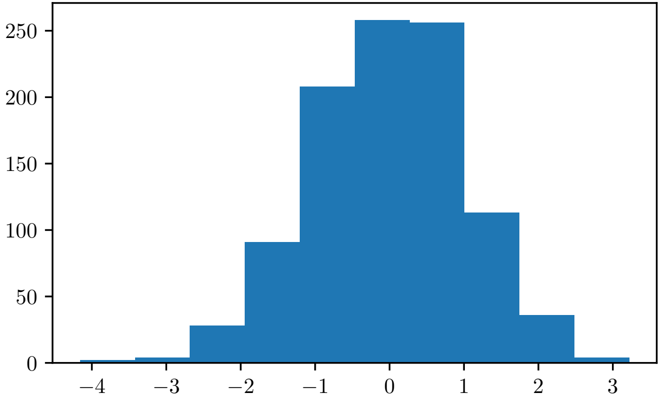
- Create histograms.

```
import numpy as np
from matplotlib.pylab import plt

plt.clf()
plt.figure(figsize=(5, 3))
# Use TeX fonts
plt.rc("text", usetex=True)

# create the histogram

data = np.random.randn(1000)
plt.hist(data)

# Save the plot as a PDF file
```

Visualization with Matplotlib XVII

```
plt.savefig("myplot5.pdf", bbox_inches="tight")
# Include the plot in the current LaTeX document
print(r"\begin{center}")
print(r"\includegraphics[width=0.75\textwidth]{myplot5.pdf
print(r"\end{center}")

```

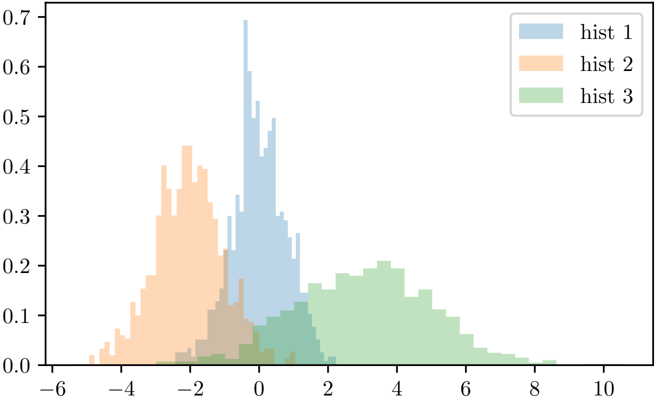
Visualization with Matplotlib XVIII



```python
import numpy as np
from matplotlib.pylab import plt

plt.clf()
plt.figure(figsize=(5, 3))
# Use TeX fonts
plt.rc("text", usetex=True)

# create the histogram
x1 = np.random.normal(0, 0.8, 1000)
x2 = np.random.normal(-2, 1, 1000)
x3 = np.random.normal(3, 2, 1000)

kwargs = dict(histtype='stepfilled',
 alpha=0.3, density=True, bins=40)
```

```
plt.hist(x1, **kwargs, label=r"hist 1")
plt.hist(x2, **kwargs, label=r"hist 2")
plt.hist(x3, **kwargs, label=r"hist 3");
plt.legend(loc=0)
# Save the plot as a PDF file
plt.savefig("myplot7.pdf", bbox_inches="tight")
# Include the plot in the current LaTeX document
print(r"\begin{center}")
print(r"\includegraphics[width=0.75\textwidth]{myplot7.pdf
print(r"\end{center}")
```

Visualization with Matplotlib XXI



la fin! la fin! la fin!