Contents

Python

Python is a high-level programming language. It has libraries such as NumPy, SciPy, SymPy and Matplotlib which are useful for scientific computing. Its popularity is also attributed to its readability.

Python

Installation : Python

Installation

A quick (and recommended for windows/mac) way to install python along with other important packages is to install Anaconda. Anaconda is a software distribution which is pre-built and pre-configured collection of packages. Ancaonda could be downloaded and installed from: `https://www.anaconda.com/download/`

You can also install just python from `https://www.python.org/downloads/` but would need to install additional packages manually.

If you are using a Linux distribution python would be preinstalled.

A detailed installation instruction is available on Blackboard and Piazza.

Installation

A quick (and recommended for windows/mac) way to install python along with other important packages is to install Anaconda. Anaconda is a software distribution which is pre-built and pre-configured collection of packages. Ancaonda could be downloaded and installed from: `https://www.anaconda.com/download/`

You can also install just python from `https://www.python.org/downloads/` but would need to install additional packages manually.

If you are using a Linux distribution python would be preinstalled.

A detailed installation instruction is available on Blackboard and Piazza.

Installation

A quick (and recommended for windows/mac) way to install python along with other important packages is to install Anaconda. Anaconda is a software distribution which is pre-built and pre-configured collection of packages. Ancaonda could be downloaded and installed from: `https://www.anaconda.com/download/`

You can also install just python from `https://www.python.org/downloads/` but would need to install additional packages manually.

If you are using a Linux distribution python would be preinstalled.

A detailed installation instruction is available on Blackboard and Piazza.

Installation

A quick (and recommended for windows/mac) way to install python along with other important packages is to install Anaconda. Anaconda is a software distribution which is pre-built and pre-configured collection of packages. Ancaonda could be downloaded and installed from: `https://www.anaconda.com/download/`

You can also install just python from `https://www.python.org/downloads/` but would need to install additional packages manually.

If you are using a Linux distribution python would be preinstalled.

A detailed installation instruction is available on Blackboard and Piazza.

Installation

A quick (and recommended for windows/mac) way to install python along with other important packages is to install Anaconda. Anaconda is a software distribution which is pre-built and pre-configured collection of packages. Ancaonda could be downloaded and installed from: `https://www.anaconda.com/download/`

You can also install just python from `https://www.python.org/downloads/` but would need to install additional packages manually.

If you are using a Linux distribution python would be preinstalled.

A detailed installation instruction is available on Blackboard and Piazza.

Python

Testing Installation

Testing

Linux : Simply type python in terminal and you should be in python environment.
Mac/Windows : If you correctly installed Anaconda you should be able to open Anaconda Navigator and IPython.

Python

Basics

Structure of the code

The structure of a standard python code (Note, the indentation):

```
from module import *

def function(argument(s)):
  "Function documentation"
  return something

if __name__=="__main__":
  do_something
```

The indentations are used to indentify the content instead of brackets, {}. This adds to the readability of the code.
Unlike MATLAB, improper indentations result in (at best) syntax errors [**IndentationError: unexpected indent**] or (at worst) unexpected behavior of the program.

Structure of the code

The structure of a standard python code (Note, the indentation):

```
from module import *

def function(argument(s)):
  "Function documentation"
  return something

if __name__=="__main__":
  do_something
```

The indentations are used to indentify the content instead of brackets, {}. This adds to the readability of the code.

Unlike MATLAB, improper indentations result in (at best) syntax errors [**IndentationError: unexpected indent**] or (at worst) unexpected behavior of the program.

Structure of the code

The structure of a standard python code (Note, the indentation):

```
from module import *

def function(argument(s)):
  "Function documentation"
  return something

if __name__=="__main__":
  do_something
```

The indentations are used to indentify the content instead of brackets, $\{\}$. This adds to the readability of the code.
Unlike MATLAB, improper indentations result in (at best) syntax errors [**IndentationError: unexpected indent**] or (at worst) unexpected behavior of the program.

Test indentation

- Open Python.
- Type a = 1
- Add a space and type print(a)

You should have received an error.

```
>>> a = 1
>>>  print(a)
  File "<stdin>", line 1
    print(a)
    ^
IndentationError: unexpected indent
>>> print(a)
1
```

Test indentation

- Open Python.
- Type a = 1
- Add a space and type print(a)

You should have received an error.

```
>>> a = 1
>>>  print(a)
  File "<stdin>", line 1
    print(a)
    ^
IndentationError: unexpected indent
>>> print(a)
1
```

Text Editor

You can use any text editor you want. You can also write in the terminal itself. Some popular ones are:

1. SublimeText : `https://www.sublimetext.com/`
2. Atom : `https://atom.io/`
3. Vim : `https://vim.sourceforge.io/`

Sample code

Let us look at a sample code to compare the sine of two numbers to discuss the various elements of a python code.

```
import numpy as np
# You could also call it as from numpy import *
# Or as import numpy. We would discuss the two uses.
def compare(x,y):
  "Compares the sine of two numbers"
  if np.sin(x)==np.sin(y) :
    return ('Sine of numbers are equal')
  elif np.sin(x)>np.sin(y) :
    return("Sine of {0} greater than sine of {1}".format(x,y))
  else :
    return ("Sine of {0} greater than sine of {1}".format(y,x))

if __name__=="__main__":
  num1 = 0.4
  num2 = 0.5
  result = compare(num1,num2)
  print(result)
```

Variables

The various elements are :

1. Declaring variable. In the previous example we have declared two variables

```
num1 = 0.4
num2 = 0.5
# Other variable declarations:
name = 'Vivek' # A string variable
check = True # A boolean variable
```

There are certain illegal variables:

`and, del, from, not, while, as, elif, global, or, with, assert, else, if, pass, yield, break, except, import, print, class, exec, in, raise, continue, finally, is, return, def, for, lambda, try`

Why is **lambda** not allowed? We would come back to it.

Variables

The various elements are :

1. Declaring variable. In the previous example we have declared two variables

```
num1 = 0.4
num2 = 0.5
# Other variable declarations:
name = 'Vivek' # A string variable
check = True # A boolean variable
```

There are certain illegal variables:

`and, del, from, not, while, as, elif, global, or, with, assert, else, if, pass, yield, break, except, import, print, class, exec, in, raise, continue, finally, is, return, def, for, lambda, try`

Why is **lambda** not allowed? We would come back to it.

Functions

2. Defining and calling a function.

```
def compare(x,y):
  "Compares the sine of two numbers"
  if np.sin(x)==np.sin(y) :
    return ('The sine of numbers are equal')
  elif np.sin(x) > np.sin(y) :
    return("Sine of {0} greater than sine of {1}".format(x,y))
  else :
    return ("Sine of {0} greater than sine of {1}".format(y,x))
  # If you don't return anything it returns 'None'
#--Calling a function--#
result = compare(num1,num2)
```

Note:

1. We can define the function within the **main** section of the code too, but some prefer to declare it separately.
2. While calling the function, it is important to provide the correct number and data type of arguments.
3. There are certain special functions such as **lambda** function.

Functions

2. Defining and calling a function.

```
def compare(x,y):
  "Compares the sine of two numbers"
  if np.sin(x)==np.sin(y) :
    return ('The sine of numbers are equal')
  elif np.sin(x) > np.sin(y) :
    return("Sine of {0} greater than sine of {1}".format(x,y))
  else :
    return ("Sine of {0} greater than sine of {1}".format(y,x))
  # If you don't return anything it returns 'None'
#--Calling a function--#
result = compare(num1,num2)
```

Note:

1. We can define the function within the **main** section of the code too, but some prefer to declare it separately.
2. While calling the function, it is important to provide the correct number and data type of arguments.
3. There are certain special functions such as **lambda** function.

Functions

2. Defining and calling a function.

```
def compare(x,y):
  "Compares the sine of two numbers"
  if np.sin(x)==np.sin(y) :
    return ('The sine of numbers are equal')
  elif np.sin(x) > np.sin(y) :
    return("Sine of {0} greater than sine of {1}".format(x,y))
  else :
    return ("Sine of {0} greater than sine of {1}".format(y,x))
  # If you don't return anything it returns 'None'
#--Calling a function--#
result = compare(num1,num2)
```

Note:

1. We can define the function within the **main** section of the code too, but some prefer to declare it separately.
2. While calling the function, it is important to provide the correct number and data type of arguments.
3. There are certain special functions such as **lambda** function.

Functions

2. Defining and calling a function.

```
def compare(x,y):
  "Compares the sine of two numbers"
  if np.sin(x)==np.sin(y) :
    return ('The sine of numbers are equal')
  elif np.sin(x) > np.sin(y) :
    return("Sine of {0} greater than sine of {1}".format(x,y))
  else :
    return ("Sine of {0} greater than sine of {1}".format(y,x))
  # If you don't return anything it returns 'None'
#--Calling a function--#
result = compare(num1,num2)
```

Note:

1. We can define the function within the **main** section of the code too, but some prefer to declare it separately.
2. While calling the function, it is important to provide the correct number and data type of arguments.
3. There are certain special functions such as **lambda** function.

Lambda Function

Lambda functions are special anonymous functions. They can be called whenever function objects are required. They are restricted to a single expression.

```
f = lambda x : np.sin(x)+np.cos(x)
# define a function with 1 argument and return sin+cos of the argument
print(f(np.pi/4.))
```

If/else

3. If/ese conditional statements.

```
if np.sin(x)==np.sin(y) :
  return ('The sine of numbers are equal')
elif np.sin(x) > np.sin(y) : # Note the syntax.
  return("Sine of {0} greater than sine of {1}".format(x,y))
else :
  return ("Sine of {0} greater than sine of {1}".format(y,x))
```

Loops

Something that we did not touch in the sample code were the loops.

1. For loops: The code is iterated a fixed number of times.

```
for x in range(3): # Starts counting at 0 and ends at 2
    print "We're on time %d" % (x)
for x in range(1, 3): # Starts counting at the specified number
    print "We're on time %d" % (x)
for x in range(1, 10, 2): # Starts counting at the specified number and with step size specified as the last aargument.
    print "We're on time %d" % (x)
```

2. While loop. A condition is checked and the loop iterates as long as the condition is satisfied.

```
i = 1
while (i<10): # Change this to while(True) : and see what happens.
  print (i)
  i += 1
# Prints 1-9
```

Loops

Something that we did not touch in the sample code were the loops.

1. For loops: The code is iterated a fixed number of times.

```
for x in range(3): # Starts counting at 0 and ends at 2
    print "We're on time %d" % (x)
for x in range(1, 3): # Starts counting at the specified number
    print "We're on time %d" % (x)
for x in range(1, 10, 2): # Starts counting at the specified number and with
     step size specified as the last aargument.
    print "We're on time %d" % (x)
```

2. While loop. A condition is checked and the loop iterates as long as the condition is satisfied.

```
i = 1
while (i<10): # Change this to while(True) : and see what happens.
  print (i)
  i += 1
# Prints 1-9
```

Python

Some useful modules

Libraries for scientific computing

At the very start of the sample code, we called a library we would be using.

```
   import numpy as np
```

We can do it in two ways

```
>>> from numpy import *
>>> sin(pi)
1.2246467991473532e-16
>>> sin(pi/2.)
1.0
```

```
>>> import numpy as np
>>> sin(pi)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sin' is not defined
>>> np.sin(np.pi)
1.2246467991473532e-16
>>> np.sin(np.pi/2.)
1.0
```

Libraries for scientific computing

At the very start of the sample code, we called a library we would be using.

```
  import numpy as np
```

We can do it in two ways

```
>>> from numpy import *
>>> sin(pi)
1.2246467991473532e-16
>>> sin(pi/2.)
1.0
```

```
>>> import numpy as np
>>> sin(pi)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sin' is not defined
>>> np.sin(np.pi)
1.2246467991473532e-16
>>> np.sin(np.pi/2.)
1.0
```

- NumPy : It is one of the common libraries used. Useful for array manipulation and using special mathematical functions.

You can then define a simple numpy array.

```
b = np.array([(1.5,2,3), (4,5,6)])
print(b)
#------- Prints ----------#
#array([[ 1.5,  2. ,  3. ],
#    [ 4. ,  5. ,  6. ]])
a = np.array(1,2,3,4)    # WRONG
```

```
#--We can obtain information about an array's data--#
print(a.sum()) # Prints the sum of all elements
print(a.max()) # Prints the greatest element
print(a.min()) # Prints the smallest element
#----Data type of elements is important to know sometimes--#
print(b.dtype.name) #returns float64
```

- NumPy : It is one of the common libraries used. Useful for array manipulation and using special mathematical functions.

You can then define a simple numpy array.

```
b = np.array([(1.5,2,3), (4,5,6)])
print(b)
#------- Prints ----------#
#array([[ 1.5,  2. ,  3. ],
#    [ 4. ,  5. ,  6. ]])
a = np.array(1,2,3,4)    # WRONG
```

```
#--We can obtain information about an array's data--#
print(a.sum()) # Prints the sum of all elements
print(a.max()) # Prints the greatest element
print(a.min()) # Prints the smallest element
#----Data type of elements is important to know sometimes--#
print(b.dtype.name) #returns float64
```

- NumPy : Useful for array manipulation required for vector or tensor algebra.

You can then define a simple numpy array.

```
a = np.array([(1.5,2.3)])
b = np.array([(-2.3,1.5)])
print(np.dot(a,b.T))
#------- Prints ----------#
#[[0.]]
```

- NumPy : Useful for array manipulation required for vector or tensor algebra.

You can then define a simple numpy array.

```
a = np.array([(1.5,2.3)])
b = np.array([(-2.3,1.5)])
print(np.dot(a,b.T))
#------- Prints ----------#
#[[0.]]
```

Libraries for scientific computing

- Matplotlib : Used to create 2/3-D figures.

Let us say we want to plot a function and its derivative. The simplest we could try is sine or cosine function.
Do the following:

1. Import numpy
2. define a function which returns the sine of the argument (Try lambda function!)
3. define the range in which you want to evaluate the function (look up np.arange)

```
import numpy as np
# Define a dunction using lambda
f = lambda x : np.sin(x)
# Define the range of function
x = np.arange(-2*np.pi,2*np.pi,0.1)
```

Libraries for scientific computing

- Matplotlib : Used to create 2/3-D figures.

Let us say we want to plot a function and its derivative. The simplest we could try is sine or cosine function.
Do the following:

1. Import numpy
2. define a function which returns the sine of the argument (Try lambda function!)
3. define the range in which you want to evaluate the function (look up np.arange)

```
import numpy as np
# Define a dunction using lambda
f = lambda x : np.sin(x)
# Define the range of function
x = np.arange(-2*np.pi,2*np.pi,0.1)
```

Next we need to take derivatives. In python, the module that you could call is scipy. To plot we need matplotlib

```
from scipy.misc import derivative as deriv
import matplotlib.pyplot as plt
from matplotlib import mlab
d = deriv(f,x,0.01) #[function, derivative with respect to, steps]
# Look up scipy derivative to see the arguments you can define and how it
     takes derivatives.
```

Next we need to plot the function and its derivative.

```
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(x, f(x), 'g--', linewidth=1.5, label='Function')
ax.plot(x, d, 'k--', linewidth=1.5, label='Derivative')
# tidy up the figure
ax.grid(True)
ax.legend(loc='upper right')
ax.set_title('Graph of function and derivative')
ax.set_xlabel('x-values')
ax.set_ylabel('y-values')
plt.show()
```

Next we need to take derivatives. In python, the module that you could call is scipy. To plot we need matplotlib

```
from scipy.misc import derivative as deriv
import matplotlib.pyplot as plt
from matplotlib import mlab
d = deriv(f,x,0.01) #[function, derivative with respect to, steps]
# Look up scipy derivative to see the arguments you can define and how it
    takes derivatives.
```

Next we need to plot the function and its derivative.

```
fig, ax = plt.subplots(figsize=(8, 4))
ax.plot(x, f(x), 'g--', linewidth=1.5, label='Function')
ax.plot(x, d, 'k--', linewidth=1.5, label='Derivative')
# tidy up the figure
ax.grid(True)
ax.legend(loc='upper right')
ax.set_title('Graph of function and derivative')
ax.set_xlabel('x-values')
ax.set_ylabel('y-values')
plt.show()
```

Plot obtained:

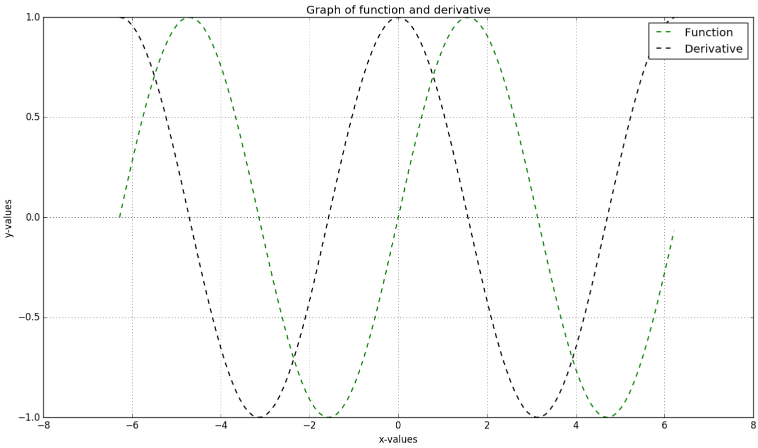

Figure: Example of use of matplot library

Libraries for scientific computing

- SymPy : It is python library for symbolic mathematics (similar to the sym/syms option in MATLAB)

You could manipulate the equations symbolically:

```
>> from sympy import *
>> x = Symbol('x')
>> f = x**2+2*x+1
>> factor(f)
(x+1)**2
```

Python

Additional Resources

Further References

You can refer to the following resources and books for learning / practising and also use these as references:

`http://docs.python-guide.org/en/latest/`

`https://developers.google.com/edu/python/?csw=1`

`https://docs.scipy.org/doc/numpy/reference/routines.html`

LaTeX

What is LaTeX?

LaTeX is a typesetting/document-preparation system.

- Unlike Word/Page it is not based on what-you-see-is-what-you-get philosophy.
- Highly used in academia to produce manuscript quality documents.

This presentation itself was made in LaTeX!

What is LaTeX?

LaTeX is a typesetting/document-preparation system.

- Unlike Word/Page it is not based on what-you-see-is-what-you-get philosophy.
- Highly used in academia to produce manuscript quality documents.

This presentation itself was made in LaTeX!

LaTeX

Installation : LaTeX

Latex Editors

One of the cross-platform latex editors is Texmaker, which can be downloaded and installed from:
`http://www.xm1math.net/texmaker/download.html`
You can choose other latex editors:

- TeXstudio : `http://www.texstudio.org/`
- TeXworks : `https://www.tug.org/texworks/`
- TeXShop(only for Mac) : `http://pages.uoregon.edu/koch/texshop/`

LaTeX

Testing Installation

Testing

If you have successfully installed the software you should be able to : Open the software and Make the following simple document:

```
\documentclass{article}
\usepackage[utf8]{inputenc}

\title{First document}
\author{Your Name}
\date{\today}

\begin{document}

\begin{titlepage}
\maketitle
\end{titlepage}

First document to be tested.

\end{document}
```

You must have obtained a simple document in pdf format.

LaTeX

Some useful packages

Packages

- Packages are add-on features in LaTeXand many of them are pre-installed.
- If you need something that is not currently installed you can easily do that. Depending on your OS the procedure would vary.
- Some commonly used packages could be found at `https://en.wikibooks.org/wiki/LaTeX/Package_Reference`. This list is far from exhaustive.

You could find a file on course website called "useful_packages.tex" which consists of all the possible required packages. When you compile your document, if a certain package is not installed you can easily install it.

LaTeX

Examples

Working Example

```
Texts can very easily be inserted. A newline could begin with \\
Equations are simply introduced in the equation environment.
\begin{equation}
  \boldsymbol \sigma = \lambda \tr[\boldsymbol \epsilon]+ 2\mu\boldsymbol \epsilon
\end{equation}
```

Texts can very easily be inserted. A newline could begin with
Equations are simply introduced in the equation environment.

$$\boldsymbol{\sigma} = \lambda \mathbf{t}[\epsilon] + 2\mu\epsilon \tag{1}$$

You can note that I have used `$\boldsymbol$` far too many times. It becomes cumbersome to write. Hence you can define your own command and make short notations. Lets say you want those commands:

- `$\newcommand{\cs}{\boldsymbol \sigma}$`
- `$\newcommand{\epsb}{\boldsymbol \epsilon}$`

Working Example

```
Texts can very easily be inserted. A newline could begin with \\
Equations are simply introduced in the equation environment.
\begin{equation}
  \boldsymbol \sigma = \lambda \tr[\boldsymbol \epsilon]+ 2\mu\boldsymbol \epsilon
\end{equation}
```

Texts can very easily be inserted. A newline could begin with
Equations are simply introduced in the equation environment.

$$\boldsymbol{\sigma} = \lambda \mathbf{t}[\epsilon] + 2\mu\epsilon \tag{1}$$

You can note that I have used `$\boldsymbol$` far too many times. It becomes cumbersome to write. Hence you can define your own command and make short notations. Lets say you want those commands:

- `$\newcommand{\cs}{\boldsymbol \sigma}$`
- `$\newcommand{\epsb}{\boldsymbol \epsilon}$`

The result:

```
\begin{equation}
  \cs = \lambda \tr[\epsb]+ 2\mu\epsb
\end{equation}
```

$$\boldsymbol{\sigma} = \lambda \mathbf{t}[\epsilon] + 2\mu\epsilon \qquad (2)$$

You don’t have to worry about creating such short notations, you can find them on blackboard. It is ’fairly’ readable. The document is called "short_notations.tex"

LaTeX

Additional Resources

Additional Resources

To debug or to learn how to do something new, the quickest guide would be looking for a similar question on TeX stackexchange (`https://tex.stackexchange.com/`)

Additional learning material can be found at:

- The Not So Short Introduction to $\LaTeX_{2\epsilon}$: `http://ftp.math.purdue.edu/mirrors/ctan.org/info/lshort/english/lshort.pdf`
- The $\LaTeX$wikibook : `https://en.wikibooks.org/wiki/LaTeX`