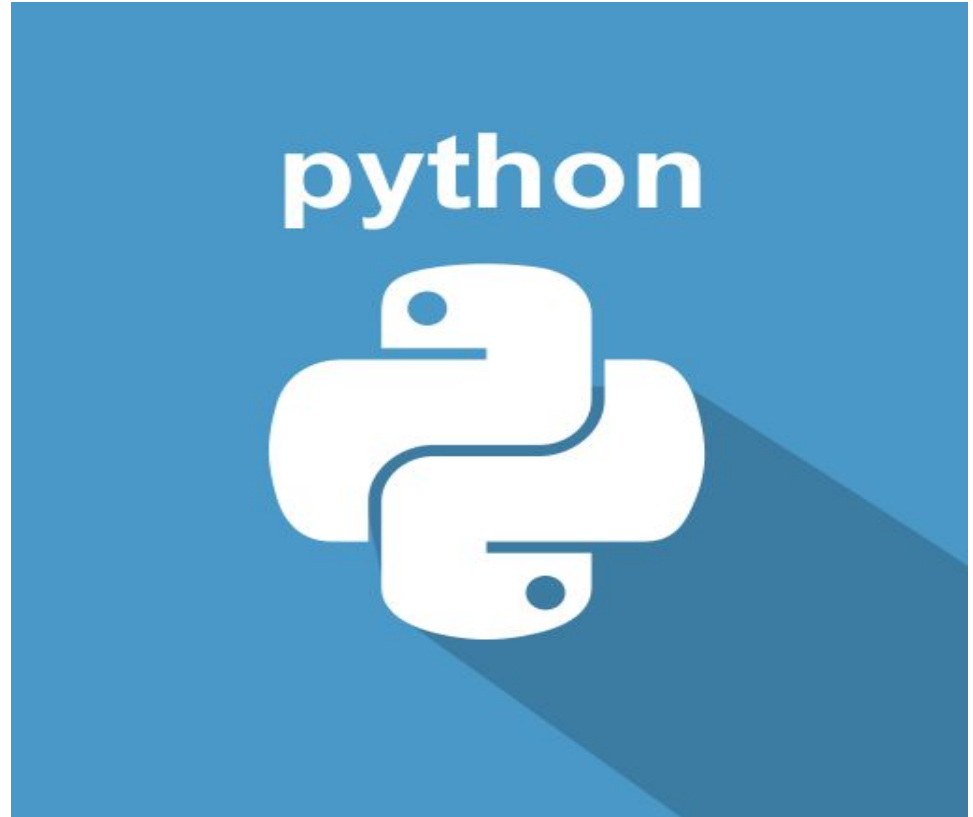


Lecture 2

Introduction to Computers and Python Programming

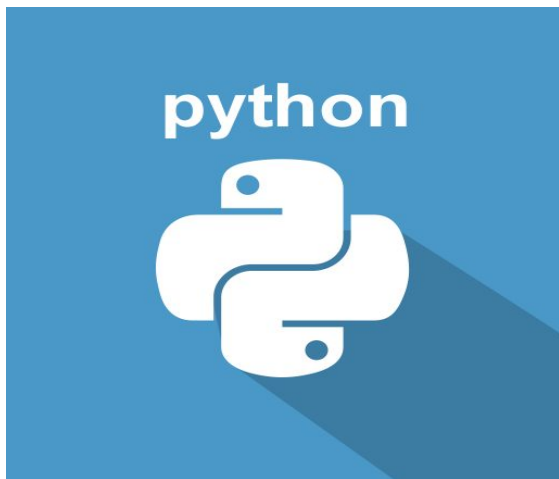


Last Time

- **Introduced Ourselves**
- **Discussed Hardware and Software**
- **Learned How Computers Store Data**
- **Discussed How Programs Work**
 - **Fetch-Decode-Execute Cycle**
- **Introduced Python**

Topics

- **Installing Python Stack with Anaconda**
- **Installing and Using Jupyter Notebooks**
- **Getting Started With Python**
- **First Python Examples -- Visualizing Data**
- **An Exercise -- Finding Nearest Neighbors**



Installing Python Stack with Anaconda

What is Anaconda?

According to [Wikipedia](#):

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, etc.), that aims to simplify package management and deployment.

Why Anaconda?

- **Makes Installing and accessing the Python (and R) stack easy and consistent across multiple platforms**
- **Package Management for important data science tools is simplified**
- **Comes with over 1400 packages**

Installing Anaconda

<https://docs.anaconda.com/anaconda/install/>



Home

Anaconda Enterprise 5

Anaconda Enterprise 4

Anaconda Distribution

Installation

Installing on Windows

Installing on macOS

Installing on Linux

Installing on Linux POWER

Installing in silent mode

Verifying your installation

Anaconda installer file hashes

Updating from older versions

Uninstalling Anaconda

User guide

Reference



Installation

Review the system requirements listed below before installing Anaconda Distribution. If you don't want the hundreds of packages included with Anaconda, you can [install Miniconda](#), a mini version of Anaconda that includes just conda, its dependencies, and Python.



Looking for Python 3.5 or 3.6? See our [FAQ](#).

System requirements

- License: Free use and redistribution under the terms of the [End User License Agreement](#).
- Operating system: Windows 7 or newer, 64-bit macOS 10.10+, or Linux, including Ubuntu, RedHat, CentOS 6+, and others.
- If your operating system is older than what is currently supported, you can find older versions of the Anaconda installers in our [archive](#) that might work for you. Check our [FAQ](#) for version recommendations.
- System architecture: Windows- 64-bit x86, 32-bit x86; MacOS- 64-bit x86; Linux- 64-bit x86, 64-bit Power8/Power9.
- Minimum 5 GB disk space to download and install.

On Windows, macOS, and Linux, it is best to install Anaconda for the local user, which does not require administrator permissions and is the most robust type of installation. However, if you need to, you can install Anaconda system wide, which does require administrator permissions.

- [Installing on Windows](#)

v. latest

Getting Started with Anaconda

<https://docs.anaconda.com/anaconda/install/>



Home

Anaconda Enterprise 5

Anaconda Enterprise 4

Anaconda Distribution

Installation

Installing on Windows

Installing on macOS

Installing on Linux

Installing on Linux POWER

Installing in silent mode

Verifying your installation

Anaconda installer file hashes

Updating from older versions

Uninstalling Anaconda

User guide

Reference



Installation

Review the system requirements listed below before installing Anaconda Distribution. If you don't want the hundreds of packages included with Anaconda, you can [install Miniconda](#), a mini version of Anaconda that includes just conda, its dependencies, and Python.



Tip

Looking for Python 3.5 or 3.6? See our [FAQ](#).

System requirements

- License: Free use and redistribution under the terms of the [End User License Agreement](#).
- Operating system: Windows 7 or newer, 64-bit macOS 10.10+, or Linux, including Ubuntu, RedHat, CentOS 6+, and others.
- If your operating system is older than what is currently supported, you can find older versions of the Anaconda installers in our [archive](#) that might work for you. Check our [FAQ](#) for version recommendations.
- System architecture: Windows- 64-bit x86, 32-bit x86; MacOS- 64-bit x86; Linux- 64-bit x86, 64-bit Power8/Power9.
- Minimum 5 GB disk space to download and install.

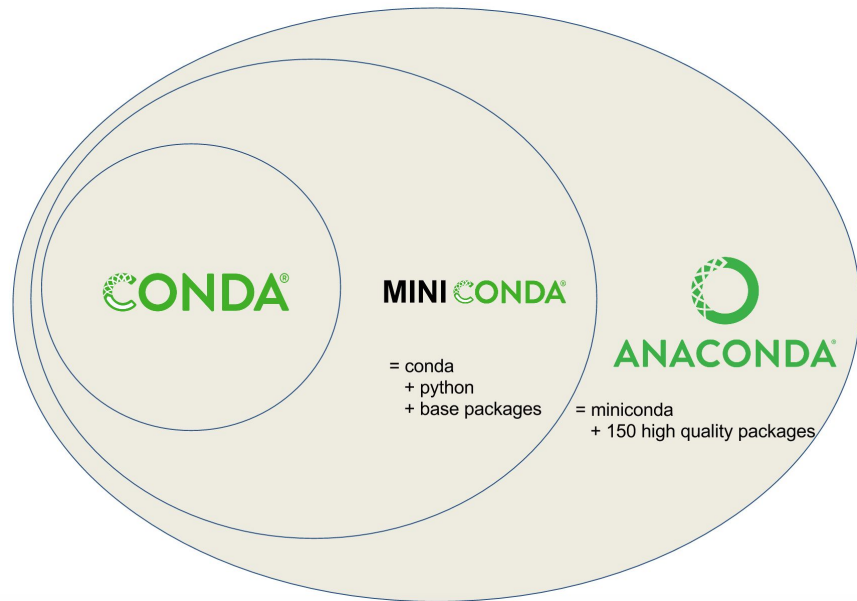
On Windows, macOS, and Linux, it is best to install Anaconda for the local user, which does not require administrator permissions and is the most robust type of installation. However, if you need to, you can install Anaconda system wide, which does require administrator permissions.

- [Installing on Windows](#)

v. latest

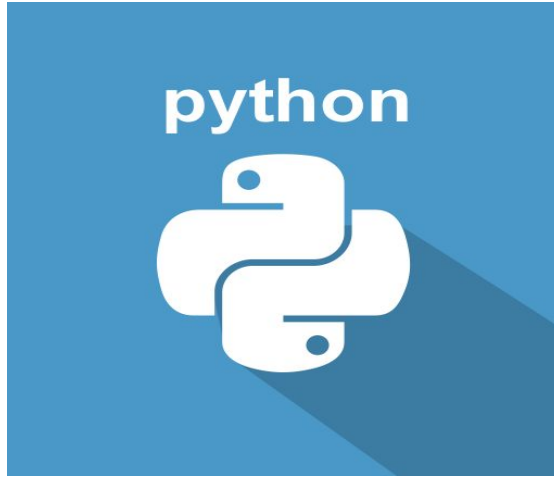
Getting Started with Anaconda

<https://docs.anaconda.com/anaconda/install/>



Getting Started with Anaconda

Launch Anaconda Navigator and you should be ready to go!!



Installing and Using Jupyter Notebooks

What are Jupyter Notebooks?

According to the [Jupyter Project](#):

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more.

Why Jupyter?

- **Powerful way to write and iterate on your Python code for data analysis**
- **Intuitive iteration -- write individual lines of code and run them one at a time**
- **Sharable documents with embedded code (via markdown)**
- **Accessible from almost any browser**

Jupyter Notebooks

- If you installed anaconda, just launch jupyter notebooks from anaconda navigator
- Pro-tip -- use Jupyter Lab
- Today we'll primarily run jupyter console, but I'll give a demonstration of full jupyter notebooks

Jupyter Notebooks

File Edit View Run Kernel Tabs Settings Help

Home > ... > Wednesday > auto_encoder

Name	Last Modified
celeba	4 months ago
models	4 months ago
VAE.ipynb	4 months ago
README.md	4 months ago
requirements.txt	4 months ago
utils.py	4 months ago

Console 1 VAE.ipynb

Markdown

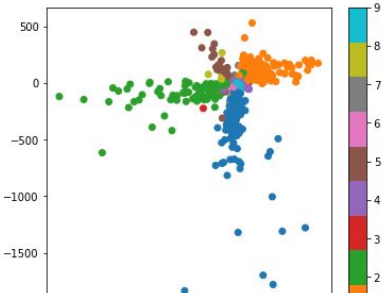
Python 3

Alternatively, we can load pre-trained weights.

```
[11]: # load_weights(folder='models/mnist_ae')
...
[12]: # save_weights(folder='models/mnist_ae')
...
```

Below, we plot the latent representation of all the MNIST test images and their groundtruth label. As we can see, the clusters are already grouped together quite nicely.

```
[15]: x_test_encoded = encoder.predict(x_test, batch_size=8)
plt.figure(figsize=(6, 6))
plt.scatter(x_test_encoded[:, 0], x_test_encoded[:, 1], c=y_test, cmap='tab10')
plt.colorbar()
plt.show()
```





Getting Started with Python

A simple example

```
x = 34 - 23      # A comment.  
y = "Hello"      # Another one.  
z = 3.45  
if z == 3.45 or y == "Hello":  
    x = x + 1  
    y = y + " World" # String concat.  
print(x)  
print(y)
```

A simple example

```
x = 34 - 23      # A comment.  
y = "Hello"      # Another one.  
z = 3.45  
if z == 3.45 or y == "Hello":  
    x = x + 1  
    y = y + " World" # String concat.  
print(x)  
print(y)
```

Data Types

Data types: categorize value in memory

- e.g., int for integer, float for real number, str used for storing strings in memory

Standard Built-in Data Types

- **Numbers**
- **String**
- **List**
- **Tuple**
- **Dictionary**

Literal

- A literal is something that the parser recognizes as syntax for writing an object directly.

Examples of Literals

-3, -2, -1, 0, 1, 2, 3 (int literals)

2+3j, 0+5j, 2j, -3-5j (complex literals)

3.5, -2.7 (float literals)

"" , "hello" (str literals)

[], [1,2] (list literals)

(), (1,), (1,2) (tuple literals)

{}, {'a': 2} (dict literals)

Numeric Data Types

- **Numeric: number written in a program**
 - No decimal point considered int, otherwise, considered float

Strings and String Literals

- **String**: sequence of characters that is used as data
- **String literal**: string that appears in actual code of a program
 - Must be enclosed in single (') or double (") quote marks
 - String literal can be enclosed in triple quotes (''' or ''')
 - Enclosed string can contain both single and double quotes and can have multiple lines

Strings and String Literals

"hello"+"world" "helloworld" # concatenation

"hello"*3 "hellohellohello" # repetition

"hello"[0] "h" # indexing

"hello"[-1] "o" # (from end)

"hello"[1:4] "ell" # slicing

len("hello") 5 # size

"hello" < "jello" 1 # comparison

"e" in "hello" 1 # search

"escapes: \n etc, \033 etc, \if etc"

'single quotes' ""triple quotes"" r"raw strings"

Performing Calculations

Math expression: performs calculation and gives a value

- Math operator: tool for performing calculation
- Operands: values surrounding operator
 - Variables can be used as operands

Performing Calculations

expression: A data value or set of operations to compute a value.

Examples: $1 + 4 * 3$

42

Arithmetic operators we will use:

+ - * / addition, subtraction/negation, multiplication, division

% modulus, a.k.a. remainder

****** exponentiation

Performing Calculations

Two types of division:

- `/` operator performs floating point division
- `//` operator performs integer division
 - Positive results truncated, negative rounded away from zero

Performing Calculations

precedence: Order in which operations are computed.

*** / % ** have a higher precedence than + -**

1 + 3 * 4 is 13

Parentheses can be used to force a certain order of evaluation.

(1 + 3) * 4 is 16

Operator Precedence

Python operator precedence:

1. Operations enclosed in parentheses
 - Forces operations to be performed before others
2. Exponentiation (**)
3. Multiplication (*), division (/ and //), and remainder (%)
4. Addition (+) and subtraction (-)

Operator Precedence

Higher precedence performed first

- Same precedence operators execute from left to right

Comments

- **Comments**: notes of explanation within a program
 - Ignored by Python interpreter
 - Intended for a person reading the program's code
 - Begin with a # character
- **End-line comment**: appears at the end of a line of code
 - Typically explains the purpose of that line

Displaying Output with the `print` Function

- **Function**: piece of prewritten code that performs an operation
- **Argument**: data given to a function
 - Example: data that is printed to screen
- **print function**: displays output on the screen

More About Data Output

- **print function displays line of output**
 - Newline character at end of printed data
 - Special argument `end='delimiter'` causes print to place *delimiter* at end of data instead of newline character

More About Data Output

- **print function uses space as item separator**
 - Special argument `sep='delimiter'` causes print to use *delimiter* as item separator

More About Data Output

- **Special characters appearing in string literal**
 - Preceded by backslash (\)
 - Examples: newline (\n), horizontal tab (\t)
 - Treated as commands embedded in string

More About Data Output

- **When + operator used on two strings in performs string concatenation**
 - Useful for breaking up a long string literal

Variables

Variable: name that represents a value stored in the computer memory

- Used to access and manipulate data stored in memory
- A variable references the value it represents

Variables

- **Assignment statement**: used to create a variable and make it reference data
 - General format is variable = expression
 - Example: age = 29
 - **Assignment operator**: the equal sign (=)

Variables

- In assignment statement, variable receiving value must be on left side
- A variable can be passed as an argument to a function
 - Variable name should not be enclosed in quote marks
- You can only use a variable if a value is assigned to it

Variable Naming Rules

- **Rules for naming variables in Python:**
 - Variable name cannot be a Python keyword
 - Variable name cannot contain spaces
 - First character must be a letter or an underscore
 - After first character may use letters, digits, or underscores
 - Variable names are case sensitive
- **A Variable name should reflect its use**

Variable Reassignment

- Variables can reference different values while program is running
- Garbage collection: removal of values that are no longer referenced by variables
 - Carried out by Python interpreter
- Variables that have been assigned to one type can be reassigned to another type

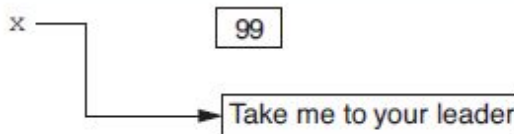
Reassigning a Variable to a Different Type

- A variable in Python can refer to items of any type

Figure 2-7 The variable `x` references an integer



Figure 2-8 The variable `x` references a string



Importing libraries

- Python organizes additional functionality into *libraries*
- Some libraries are built-in (e.g. math)
- Other libraries are installed or created by the user (e.g. numpy)
- You can add functionality from other libraries into your current context with the import statement

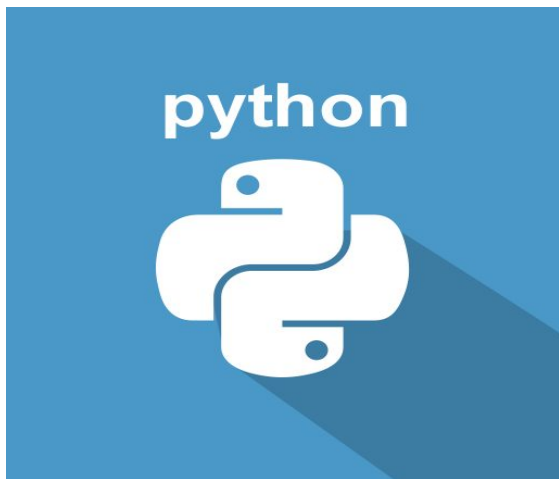
Importing libraries: example

```
import numpy
```

```
import numpy as np
```

```
from math import *
```

```
from math import log
```



matplotlib

matplotlib

- **Matplotlib is a python library that allows you to visualize data and mathematical functions**
- **It comes with tools for making plots and graphs**

matplotlib

- Installation would normally use one of the following commands

`pip install matplotlib`

`conda install matplotlib`

- You most likely installed the full version of anaconda in which case matplotlib is already installed

matplotlib

To use matplotlib we import the library as we learned earlier

```
import matplotlib.pyplot as plt  
%matplotlib inline
```

The second line is called a “magic” and is specific to jupyter notebooks. It tells the notebook to automatically display matplotlib plots.

matplotlib

Let's generate some data for us to plot

Generate points between -2 and 2 and store them in the variable x

x = np.linspace(-2,2, 200)

Evaluate the squares of each x value and store them in the variable y2

y2 = x2**

Evaluate the cubes of each x value and store them in the variable y3

y3 = x3**

matplotlib

- `plt.plot(...)` will plot our data
 - `c = '...'` determines the color of the plot
 - `label = '...'` determines the label (used in the legend)
- `plt.xlabel(), plt.ylabel()` labels the x and y axes
- `plt.title('...')` titles the plot
- `plt.legend()` shows the legend

matplotlib

plot y2 vs x and give it the color blue by using the c parameter

```
plt.plot(x, y2, c='blue', label=r'y =  $x^2$ ')
```

```
plt.plot(x, y3, c='red', label=r'y =  $x^3$ ')
```

label the x-axis

```
plt.xlabel('x')
```

label the y-axis

```
plt.ylabel('y')
```

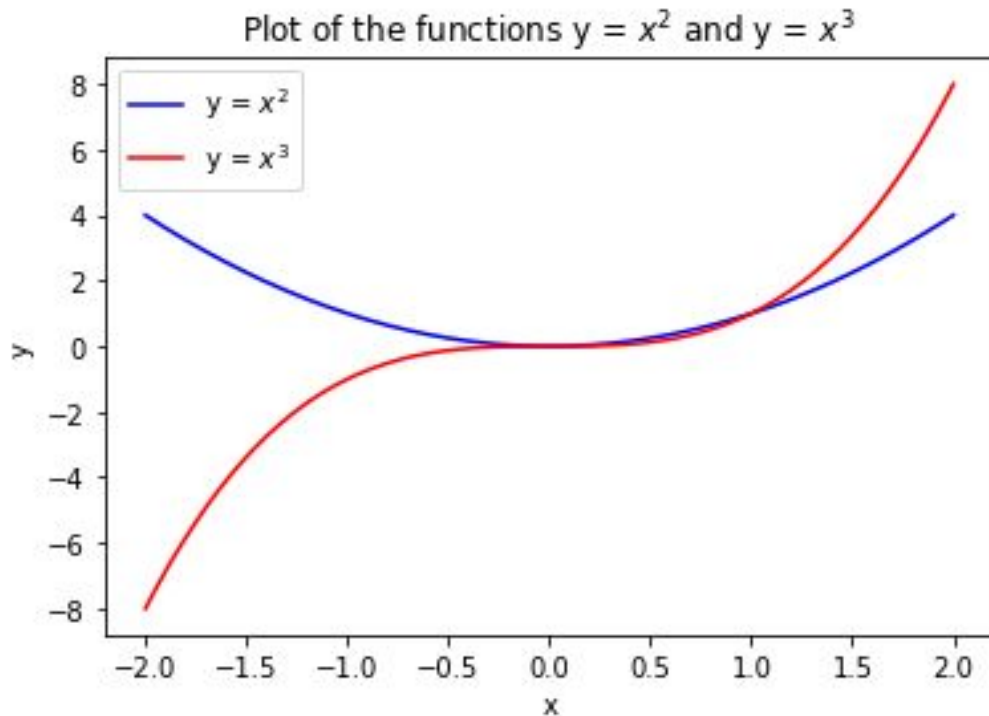
title the plot

```
plt.title(r'Plot of the functions y =  $x^2$  and y =  $x^3$ ')
```

show the legend

```
plt.legend()
```

matplotlib



matplotlib

```
plt.suptitle(r'Plot of the functions  $y = x^2$  and  $y = x^3$ ') # title the plot
```

```
plt.subplot(1, 2, 1) # create first subplot
```

```
plt.plot(x, y2, c='blue', label=r' $y = x^2$ ') # plot the first function
```

```
plt.xlabel('x') # label subplot x-axis
```

```
plt.ylabel('y') # label subplot y-axis
```

```
plt.legend() # show the subplot legend
```

```
plt.subplot(1, 2, 2) # create the second subplot
```

```
plt.plot(x, y3, c='red', label=r' $y = x^3$ ') #plot the second function
```

```
plt.xlabel('x') # label subplot x-axis
```

```
plt.ylabel('y') # label subplot y-axis
```

```
plt.legend() # show the subplot legend
```

matplotlib

Plot of the functions $y = x^2$ and $y = x^3$

