

PYTHON - ITERATORS



Ph.D. / Golden Gate Ave, San Francisco / Seoul National Univ / Carnegie Mellon / UC Berkeley / DevOps / Deep Learning / Visualization

Sponsor Open Source development activities and free contents for everyone.

Donate with **PayPal**

Thank you.

- K Hong (http://bogotobogo.com/about_us.php)

(<http://www.addthis.com/bookmark.php?v=250&username=khhong7>)



bogotobogo.com site search:

Iterators

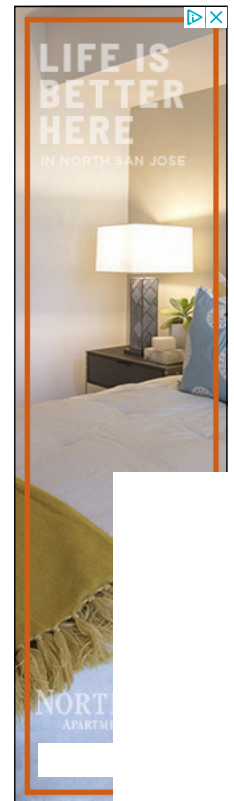
In computer science, an **iterator** is an object that allows a programmer to traverse through all the elements of a collection regardless of its specific implementation.

1. **iterable** produces **iterator** via **__iter__()**

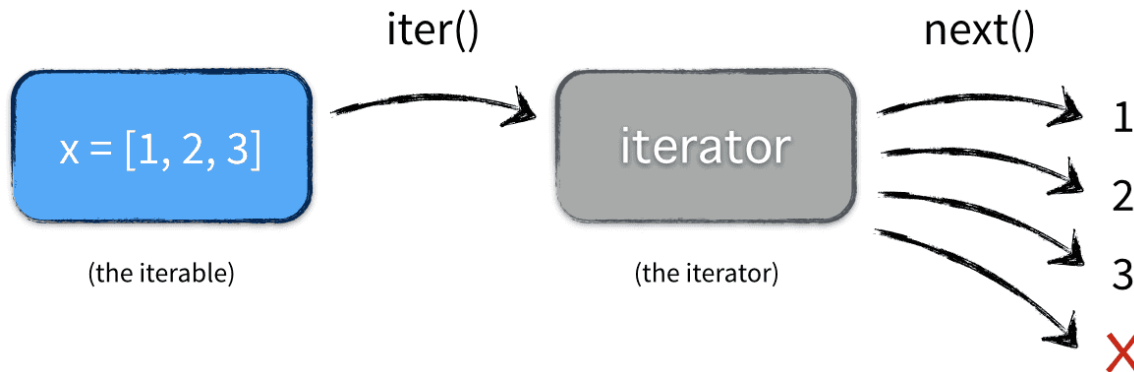
```
iterator = iterable.__iter__()
```

2. **iterator** produces a stream of values via **next()**

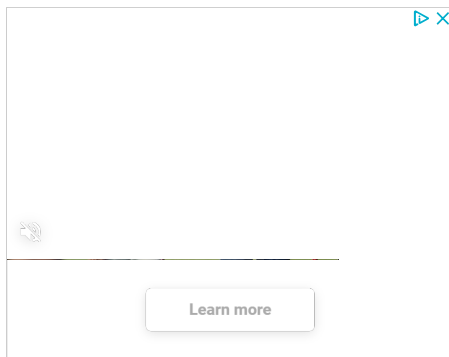
```
value = iterator.next()
value = iterator.next()
...
```



To be more specific, we can start from an iterable, and we know a list (e.g. [1,2,3]) is iterable. **iter(iterable)** produces an **iterator**, and from this we can get stream of values.



```
>>> # Python 3
>>> iterable = [1,2,3]
>>> iterator = iterable.__iter__()    # or iterator = iter(iterable)
>>> type(iterator)
<type 'listiterator'>
>>> value = iterator.__next__()    # or value = next(iterator)
>>> print(value)
1
>>> value = next(iterator)
>>> print(value)
2
>>>
>>> # Python 2
>>> iterable = [1,2,3]
>>> iterator = iterable.__iter__()
>>> type(iterator)
<type 'listiterator'>
>>> value = iterator.next()
>>> value
1
>>> value = next(iterator)
>>> value
2
```



Python tutorial

Python Home
(/python/pytut.php)

Introduction
(/python/python_introduction.ph

Running Python Programs (os, sys, import)
(/python/python_running.php)

Modules and IDLE (Import, Reload, exec)
(/python/python_modules_idle.p

Object Types - Numbers, Strings, and None
(/python/python_numbers_string

Strings - Escape Sequence, Raw String, and Slicing
(/python/python_strings.php)

Strings - Methods
(/python/python_strings_method

Formatting Strings - expressions and method calls
(/python/python_string_formattir

Files and os.path
(/python/python_files.php)

Traversing directories recursively
(/python/python_traversing_direr

Subprocess Module
(/python/python_subprocess_mc

Regular Expressions with Python
(/python/python_regularExpressi

Object Types - Lists
(/python/python_lists.php)

Object Types - Dictionaries and Tuples

Iterators in Python are a fundamental part of the language and in many cases go unseen as they are implicitly used in the **for** (foreach) statement, in **list comprehensions**, and in **generator expressions**.

"Iterators are the **secret sauce** of Python 3. They're everywhere, underlying everything, always just out of sight. Comprehensions are just a simple form of iterators. Generators are just a simple form of iterators. A function that yields values is a nice, compact way of building an iterator without building an iterator." - Mark Pilgrim

All of Python's standard built-in sequence types support iteration, as well as many classes which are part of the standard library.

The **for** loop can work on any sequence including lists, tuples, and strings:

```
>>> for x in [1, 2, 3, 4, 5]:
    print(x ** 3, end=' ')

1 8 27 64 125
>>>
>>> for x in (1, 2, 3, 4, 5):
    print(x ** 3, end=' ')

1 8 27 64 125
>>>
>>> for x in 'Beethoven':
    print(x * 3, end=' ')

BBB eee eee ttt hhh ooo vvv eee nnn
>>>
```

Learn About Twilio's Voice

Ad Build like a disruptor on an e-grade platform. 55,000+ busines

Twilio

[Learn More](#)

The **for** loop works on any **iterable object**. Actually, this is true of all iteration tools that scan objects from left to right in Python including **for** loops, the **list comprehensions**, and the **map** built-in function, etc.

Though the concept of **iterable objects** is relatively recent in Python, it has come to permeate the language's design. Actually, it is a generalization of the sequences. An object is **iterable** if it is either a physically stored sequence or an object that produces one result at a time in the context of an iteration tool like a **for** loop.

Note: For Python 2.x, the **print** is not a function but a statement. So, the right statement for the print is:

(/python/python_dictionaries_tup

Functions def, *args, **kwargs
(/python/python_functions_def.p

Functions lambda
(/python/python_functions_lamb

Built-in Functions
(/python/python_functions_built_

map, filter, and reduce
(/python/python_fncls_map_filter.

Decorators
(/python/python_decorators.php

List Comprehension
(/python/python_list_comprehen

Sets (union/intersection) and
itertools - Jaccard coefficient
and shingling to check
plagiarism
(/python/python_sets_union_inte

Hashing (Hash tables and
hashlib)
(/python/python_hash_tables_ha

Dictionary Comprehension with
zip
(/python/python_dictionary_com

The yield keyword
(/python/python_function_with_y

Generator Functions and
Expressions
(/python/python_generators.php

generator.send() method
(/python/python_function_with_g

Iterators
(/python/python_iterators.php)

Classes and Instances (__init__,
__call__, etc.)
(/python/python_classes_instanc

if __name__ == '__main__':
(/python/python_if_name_equa

```
>>> for x in [1, 2, 3, 4, 5]:
...     print x ** 3,
...
1 8 27 64 125
```

File Iterators

As a way of understanding the **file iterator**, we'll look at how it works with a file. Open file objects have **readline()** method. This reads one line each time we call **readline()**, we advance to the next line. At the end of the file, an empty string is returned. We detect it to get out of the loop.

```
>>> f = open('C:\\workspace\\Masters.txt')
>>> f.readline()
'Michelangelo Buonarroti \n'
>>> f.readline()
'Pablo Picasso\n'
>>> f.readline()
'Rembrandt van Rijn \n'
>>> f.readline()
'Leonardo Da Vinci \n'
>>> f.readline()
'Claude Monet \n'
>>> f.readline()
'\n'
# Returns an empty string at end-of-file
>>> f.readline()
''
>>>
```

Request a Free Demo

Ad Easily Connect With Custom
Global Scale.

Twilio

[Learn More](#)

But files also have a **__next__()** method that has an identical effect. It returns the next line from a file each time it is called. The only difference is that **__next__()** method raises a built-in **StopIteration** exception (http://www.bogotobogo.com/python/python_modules_idle.php#python_exceptions) at end-of-file instead of returning an empty string.

[argparse](#)
(/python/python_argparse.php)

[Exceptions](#)
(/python/python_try_except_final

[@static method vs class
method](#)
(/python/python_differences_bet

[Private attributes and private
methods](#)
(/python/python_private_attribut

[bits, bytes, bitstring, and
constBitStream](#)
(/python/python_bits_bytes_bitst

[json.dump\(s\) and json.load\(s\)](#)
(/python/python-json-dumps-
loads-file-read-write.php)

[Python Object Serialization -
pickle and json](#)
(/python/python_serialization_pic

[Python Object Serialization -
yaml and json](#)
(/python/python_yaml_json_conv

[Priority queue and heap queue
data structure](#)
(/python/python_PriorityQueue_I

[Graph data structure](#)
(/python/python_graph_data_stru

[Dijkstra's shortest path
algorithm](#)
(/python/python_Dijkstras_Short

[Prim's spanning tree algorithm](#)
(/python/python_Prims_Spanning

[Closure](#)
(/python/python_closure.php)

[Functional programming in
Python](#)
(/python/python_functional_prog

[Remote running a local file
using ssh](#)
(/python/python_ssh_remote_rur

```
>>> f = open('C:\\workspace\\Masters.txt')
>>> f.__next__()
'Michelangelo Buonarroti \n'
>>> f.__next__()
'Pablo Picasso\n'
>>> f.__next__()
'Rembrandt van Rijn \n'
>>> f.__next__()
'Leonardo Da Vinci \n'
>>> f.__next__()
'Claude Monet \n'
>>> f.__next__()
'\n'
>>> f.__next__()
Traceback (most recent call last):
  File "<pyshell#33>", line 1, in <module>
    f.__next__()
StopIteration
>>>
```

This interface is what we call the **iteration protocol** in Python. Any object with a **__next__()** method to advance to a next result is considered **iterable**. Any such object can also be stepped through with a **for** loop because all iteration tools work internally by calling **__next__()** method on each iteration.

So, the best way to read a text file line by line is not reading it at all. Instead let the **for** loop to call **__next__()** method to advance to the next line. The file object's iterator will do the work of loading lines as we go.

The example below prints each line without reading from the file at all:

```
>>> for line in open('C:\\workspace\\Masters.txt'):
    print(line.upper(), end=' ')

MICHELANGELO BUONARROTI
PABLO PICASSO
REMBRANDT VAN RIJN
LEONARDO DA VINCI
CLAUDE MONET
```

Nylas APIs - Book a Demo

Ad Free demos to help you get started with Nylas APIs. Book a demo today.

Nylas APIs

Book Now

Here, we use **end=' '** in **print** to suppress adding a **\n** because line strings already have one. This is considered the best way to read text files line by line today. The reasons are:

1. It's the simplest to code.
2. It might be the quickest to run.
3. It is the best in terms of memory usage.

SQLite 3 - A. Connecting to DB, create/drop table, and insert data into a table
(/python/python_sqlite_connect_

SQLite 3 - B. Selecting, updating and deleting data
(/python/python_sqlite_select_up

MongoDB with PyMongo I - Installing MongoDB ...
(/python/MongoDB_PyMongo/py

Python HTTP Web Services - urllib, httplib2
(/python/python_http_web_servi

Web scraping with Selenium for checking domain availability
(/python/python_Web_scraping_

REST API : Http Requests for Humans with Flask
(/python/python-REST-API-Http-Requests-for-Humans-with-Flask.php)

Blog app with Tornado
(/python/Tornado/Python_Torna

Multithreading ...
(/python/Multithread/python_mu

Python Network Programming I - Basic Server / Client : A Basics
(/python/python_network_progra

Python Network Programming I - Basic Server / Client : B File Transfer
(/python/python_network_progra

Python Network Programming II - Chat Server / Client
(/python/python_network_progra

Python Network Programming III - Echo Server using socketserver network framework
(/python/python_network_progra

Python Network Programming

The older way is to call the file **readlines()** method to load the file's content into memory as a list of line strings:

```
>>> for line in open('C:\\workspace\\Masters.txt').readlines():
    print(line.upper(), end=' ')
```

```
MICHELANGELO BUONARROTI
PABLO PICASSO
REMBRANDT VAN RIJN
LEONARDO DA VINCI
CLAUDE MONET
```

This **readlines()** loads the entire file into memory all at once. So, it will not work for files too big to fit into the memory. On the contrary, the iterator-based version is immune to such memory-explosion issues and it might run quicker, too.

iter and next

We have a built-in **next()** function for manual iteration. The **next()** function automatically calls an object's **__next__()** method. For an object **X**, the call **next(X)** is the same as **X.__next__()** but simpler.

```
>>> f = open('C:\\workspace\\Masters.txt')
>>> f.__next__()
'Michelangelo Buonarroti \n'
>>> f.__next__()
'Pablo Picasso\n'
>>>
>>> f = open('C:\\workspace\\Masters.txt')
>>> next(f)
'Michelangelo Buonarroti \n'
>>> next(f)
'Pablo Picasso\n'
>>>
```

Nylas APIs - Book a Demo

Ad Book a demo with Nylas to l
connect your app to 100% of err

Nylas APIs

Book Now

When the **for** loop begins, it obtains an iterator from the iterable object by passing it to the **iter** built-in function. This object returned by **iter** has the required the **__next__()** method. Let's look at the internals of this through **for** loop with lists. For Python versions < 3, we may want to use **next(iterObj)** instead of **iterobj.__next__()**.

IV - Asynchronous Request
Handling : ThreadingMixIn and
ForkingMixIn
(/python/python_network_progra

Python Interview Questions I
(/python/python_interview_quest

Python Interview Questions II
(/python/python_interview_quest

Python Interview Questions III
(/python/python_interview_quest

Python Interview Questions IV
(/python/python_interview_quest

Python Interview Questions V
(/python/python_interview_quest

Python Interview Questions VI
(/python/python_interview_quest

Python Interview Questions VII
(/python/python_interview_quest

Image processing with Python
image library Pillow
(/python/python_image_processi

Python and C++ with SIP
(/python/python_cpp_sip.php)

PyDev with Eclipse
(/python/pydev_eclipse_plugin_ir

Matplotlib
(/python/python_matplotlib.php)

Redis with Python
(/python/python_redis_with_pyth

NumPy array basics A
(/python/python_numpy_array_tu

NumPy Matrix and Linear
Algebra
(/python/python_numpy_matrix_

Pandas with NumPy and
Matplotlib
(/python/python_Pandas_NumPy

Celluar Automata

```
>>> # works only for v.3+

>>> L = [1, 2, 3]

>>> iter(L) is L      # L itself is not an iterator object
False

>>> iterObj = iter(L)
>>> iterObj.__next__ ()
1
>>> iterObj.__next__ ()
2
>>> iterObj.__next__ ()
3
>>> iterObj.__next__ ()
Traceback (most recent call last):
  File .....,
    iterObj.__next__ ()
StopIteration
>>>
```

The step:

```
iterObj = iter(L)
```

is not required for files because a file object is its own iterator. In other words, files have their own **`__next__()`** method. So, for file object, we do not need to get a returned iterator.

```
>>> f = open('C:\\workspace\\Masters.txt')
>>> iter(f) is f
True
>>> f.__next__ ()
'Michelangelo Buonarroti \n'
>>>
```

But list and other built-in object are not their own iterators because they support multiple open iterations. For an object like that, we must call **`iter`** to start iteration:

```
>>> L = [1, 2, 3]
>>> iter(L) is L
False
>>> L.__next__ ()
Traceback (most recent call last):
  File ...
    L.__next__ ()
AttributeError: 'list' object has no attribute '__next__'
>>>
>>>
>>> iterObj = iter(L)
>>> iterObj.__next__ ()
1
>>> next(iterObj)
2
>>>
```

(/python/python_cellular_automata)

Batch gradient descent
algorithm
(/python/python_numpy_batch_g)

Longest Common Substring
Algorithm
(/python/python_longest_comme)

Python Unit Test - TDD using
unittest.TestCase class
(/python/python_unit_testing.ph)

Simple tool - Google page
ranking by keywords
(/python/python_site_page_ranki)

Google App Hello World
(/python/GoogleApp/python_Goc)

Google App webapp2 and WSGI
(/python/GoogleApp/python_Goc)

Uploading Google App Hello
World
(/python/GoogleApp/python_Goc)

Python 2 vs Python 3
(/python/python_differences_Pyt)

virtualenv and
virtualenvwrapper
(/python/python_virtualenv_virtu)

Uploading a big file to AWS S3
using boto module
(/DevOps/AWS/aws_S3_uploading)

Scheduled stopping and
starting an AWS instance
(/DevOps/AWS/aws_stopping_sta)

Cloudera CDH5 - Scheduled
stopping and starting services
(/Hadoop/BigData_hadoop_CDH5)

Removing Cloud Files -
Rackspace API with curl and
subprocess
(/python/python_Rackspace_API_)

Checking if a process is
running/hanging and stop/run

Though Python iteration tools call these functions automatically, we can use them to apply the iteration protocol **manually**, too.

```
>>> L = [1, 2, 3]
>>>
>>> for x in L:
# Automatic iteration
    print(x ** 3, end=' ') # Obtains iter, calls __next__
# and catches exceptions

1 8 27
>>> iterObj = iter(L)      # Manual iteration which is
>>>                        # for loops usually do
```

Other Iterators

We've looked at the iterators for files and lists. How about the others such as dictionaries?

To step through the keys of a **dictionary** is to request its keys list explicitly:

```
>>> D = {'a':97, 'b':98, 'c':99}
>>> for k in D.keys():
    print(k, D[k])

a 97
c 99
b 98
```

Dictionaries have an iterator that automatically returns one key at a time in an iteration context:

```
>>> iterObj = iter(D)
>>> next(iterObj)
'a'
>>> next(iterObj)
'c'
>>> next(iterObj)
'b'
>>> next(iterObj)
Traceback (most recent call last):
  File ...
    next(iterObj)
StopIteration
>>>
```

a scheduled task on Windows
(/python/python-Windows-
Check-if-a-Process-is-Running-
Hanging-Schtasks-Run-
Stop.php)

Apache Spark 1.3 with PySpark
(Spark Python API) Shell
(/Hadoop/BigData_hadoop_Apac

Apache Spark 1.2 Streaming
(/Hadoop/BigData_hadoop_Apac

bottle 0.12.7 - Fast and simple
WSGI-micro framework for
small web-applications ...
(/python/Bottle/Python_Bottle_Fr

Flask app with Apache WSGI on
Ubuntu14/CentOS7 ...
(/python/Flask/Python_Flask_Blo

Selenium WebDriver
(/python/python_Selenium_WebI

Fabric - streamlining the use of
SSH for application deployment
(/python/Fabric/python_Fabric.ph

Ansible Quick Preview - Setting
up web servers with Nginx,
configure enviroments, and
deploy an App
(/DevOps/Ansible/Ansible_Setting

Neural Networks with
backpropagation for XOR using
one hidden layer
(/python/python_Neural_Networ

NLP - NLTK (Natural Language
Toolkit) ...
(/python/NLTK/NLTK_install.php)

RabbitMQ(Message broker
server) and Celery(Task queue)
...
(/python/RabbitMQ_Celery/pythc

OpenCV3 and Matplotlib ...
(/python/OpenCV_Python/pythor

Simple tool - Concatenating
slides using FFmpeg ...

So, we no longer need to call the **keys()** method to step through dictionary keys. The **for** loop will use the iteration protocol to grab one key at a time through:

```
>>> for k in D:
    print(k, D[k])

a 97
c 99
b 98
```

Other Python object types also support the iterator protocol and thus may be used in **for** loops too. For example, **shelves** which is an access-by-key file system for Python objects and the results from **os.popen** which is a tool for reading the output of shell commands are iterable as well:

```
>>> import os
>>> P = os.popen('dir')
>>> P.__next__()
' Volume in drive C has no label.\n'
>>> P.__next__()
' Volume Serial Number is 0C60-AED5\n'
>>> next(P)
Traceback (most recent call last):
  File ...
    next(P)
TypeError: _wrap_close object is not an iterator
>>>
```

Note that **popen** object support a **P.next()** method, they support the **P.__next__()** method, but not the **next(P)** built-in.

The iteration protocol also is the reason that we've had to wrap some results in a **list** call to see their values all at once (Python ver. 3.x). Object that are iterable returns results one at a time, not in a physical list:

```
>>> R = range(5)
>>> # Ranges are iterables in 3.0
>>> R
range(0, 5)
>>> # Use iteration protocol to produce results
>>> iterObj = iter(R)
>>> next(iterObj)
0
>>> next(iterObj)
1
>>> # Use list to collect all results at once
>>> list(range(5))
[0, 1, 2, 3, 4]
>>>
```

(/FFmpeg/ffmpeg_fade_in_fade_c

iPython - Signal Processing with NumPy
(/python/OpenCV_Python/pythor

iPython and Jupyter - Install Jupyter, iPython Notebook, drawing with Matplotlib, and publishing it to Github
(/python/IPython/IPython_Jupyte

iPython and Jupyter Notebook with Embedded D3.js
(/python/IPython/iPython_Jupyte

Downloading YouTube videos using youtube-dl embedded with Python
(/VideoStreaming/YouTube/youtu

Machine Learning : scikit-learn ...
(/python/scikit-learn/scikit_machine_learning_Su

Django 1.6/1.8 Web Framework ...
(/python/Django/Python_Django.

Sponsor Open Source development activities and free contents for everyone.



Thank you.

- K Hong (http://bogotobogo.com/about_us.php)

Note on range():

For Python 3.x, **range** just returns an iterator, and generates items in the range **on demand** instead of building the list results in memory (Python 2.x does this). So, if we want to display result, we must use **list(range(...))**:

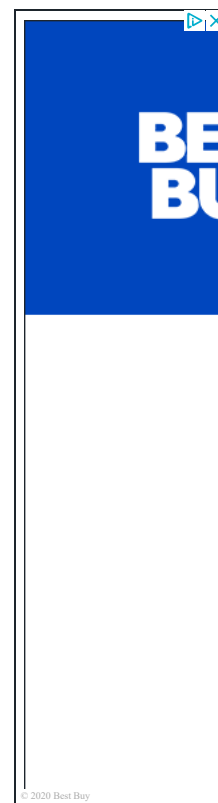
```
>>> L = range(5)

>>> L
range(0, 5)      # Python 3.x
>>> list(L)
[0, 1, 2, 3, 4]

>>> L
[0, 1, 2, 3, 4]  # Python 2.x
```

Now we should be able to see how it explains why the **enumerate** tool works the way it does:

```
>>> E = enumerate('Python')
>>> E
<enumerate object at 0x0000000003234678>
>>> iterObj = iter(E)
>>> # Generate results with iteration protocol
>>> next(iterObj)
(0, 'P')
>>> next(iterObj)
(1, 'y')
>>> list(enumerate('Python'))
[(0, 'P'), (1, 'y'), (2, 't'), (3, 'h'), (4, 'o'), (5, 'n')]
```



OpenCV 3 image and video processing with Python

OpenCV 3 with Python
(/python/OpenCV_Python/pythor)

Image - OpenCV BGR :
Matplotlib RGB
(/python/OpenCV_Python/pythor)

Basic image operations - pixel
access
(/python/OpenCV_Python/pythor)

iPython - Signal Processing with
NumPy
(/python/OpenCV_Python/pythor)

Signal Processing with NumPy I
- FFT and DFT for sine, square
waves, unitpulse, and random
signal
(/python/OpenCV_Python/pythor

Signal Processing with NumPy II
- Image Fourier Transform : FFT
& DFT
(/python/OpenCV_Python/pythor

Inverse Fourier Transform of an
Image with low pass filter:
cv2.idft()
(/python/OpenCV_Python/pythor

Image Histogram
(/python/OpenCV_Python/pythor

Video Capture and Switching
colorspaces - RGB / HSV
(/python/OpenCV_Python/pythor

Adaptive Thresholding - Otsu's
clustering-based image
thresholding
(/python/OpenCV_Python/pythor

Edge Detection - Sobel and
Laplacian Kernels
(/python/OpenCV_Python/pythor

Canny Edge Detection
(/python/OpenCV_Python/pythor

Hough Transform - Circles
(/python/OpenCV_Python/pythor

Watershed Algorithm : Marker-
based Segmentation I
(/python/OpenCV_Python/pythor

Watershed Algorithm : Marker-
based Segmentation II
(/python/OpenCV_Python/pythor

Image noise reduction : Non-
local Means denoising
algorithm
(/python/OpenCV_Python/pythor
local_Means_Denoising_Algorithr

Image object detection : Face

detection using Haar Cascade
Classifiers
(/python/OpenCV_Python/pythor

Image segmentation -
Foreground extraction Grabcut
algorithm based on graph cuts
(/python/OpenCV_Python/pythor

Image Reconstruction -
Inpainting (Interpolation) - Fast
Marching Methods
(/python/OpenCV_Python/pythor

Video : Mean shift object
tracking
(/python/OpenCV_Python/pythor

Machine Learning : Clustering -
K-Means clustering I
(/python/OpenCV_Python/pythor
Means_Clustering_Vector_Quanti

Machine Learning : Clustering -
K-Means clustering II
(/python/OpenCV_Python/pythor
Means_Clustering_Vector_Quanti

Machine Learning :
Classification - k-nearest
neighbors (k-NN) algorithm
(/python/OpenCV_Python/pythor
nearest_neighbors_k-NN.php)

Machine Learning with scikit-learn

scikit-learn installation
(/python/scikit-learn/scikit-
learn_install.php)

scikit-learn : Features and
feature extraction - iris dataset
(/python/scikit-
learn/scikit_machine_learning_fe

scikit-learn : Machine Learning
Quick Preview (/python/scikit-learn/scikit_machine_learning_qu

scikit-learn : Data
Preprocessing I - Missing /
Categorical data (/python/scikit-learn/scikit_machine_learning_Da
Missing-Data-Categorical-Data.php)

scikit-learn : Data
Preprocessing II - Partitioning a
dataset / Feature scaling /
Feature Selection /
Regularization (/python/scikit-learn/scikit_machine_learning_Da
II-Datasets-Partitioning-Feature-scaling-Feature-Selection-Regularization.php)

scikit-learn : Data
Preprocessing III -
Dimensionality reduction vis
Sequential feature selection /
Assessing feature importance
via random forests
(/python/scikit-learn/scikit_machine_learning_Da
III-Dimensionality-reduction-via-Sequential-feature-selection-Assessing-feature-importance-via-random-forests.php)

Data Compression via
Dimensionality Reduction I -
Principal component analysis
(PCA) (/python/scikit-learn/scikit_machine_learning_Da_PCA.php)

scikit-learn : Data Compression
via Dimensionality Reduction II
- Linear Discriminant Analysis
(LDA) (/python/scikit-learn/scikit_machine_learning_Da

scikit-learn : Data Compression
via Dimensionality Reduction III
- Nonlinear mappings via
kernel principal component
(KPCA) analysis (/python/scikit-

learn/scikit_machine_learning_Data
nonlinear-mappings-via-kernel-
principal-component-
analysis.php)

scikit-learn : Logistic
Regression, Overfitting &
regularization (/python/scikit-
learn/scikit-
learn_logistic_regression.php)

scikit-learn : Supervised
Learning & Unsupervised
Learning - e.g. Unsupervised
PCA dimensionality reduction
with iris dataset (/python/scikit-
learn/scikit_machine_learning_Su

scikit-learn :
Unsupervised_Learning -
KMeans clustering with iris
dataset (/python/scikit-
learn/scikit_machine_learning_Ur

scikit-learn : Linearly Separable
Data - Linear Model &
(Gaussian) radial basis function
kernel (RBF kernel)
(/python/scikit-
learn/scikit_machine_learning_Lir

scikit-learn : Decision Tree
Learning I - Entropy, Gini, and
Information Gain
(/python/scikit-
learn/scikt_machine_learning_De

scikit-learn : Decision Tree
Learning II - Constructing the
Decision Tree (/python/scikit-
learn/scikit_machine_learning_Cc

scikit-learn : Random Decision
Forests Classification
(/python/scikit-
learn/scikit_machine_learning_Ra

scikit-learn : Support Vector
Machines (SVM) (/python/scikit-
learn/scikit_machine_learning_Su

scikit-learn : Support Vector
Machines (SVM) II
(/python/scikit-

[learn/scikit_machine_learning_Su](#)

Flask with Embedded Machine
Learning I : Serializing with
pickle and DB setup
(/python/Flask/Python_Flask_Eml

Flask with Embedded Machine
Learning II : Basic Flask App
(/python/Flask/Python_Flask_Eml

Flask with Embedded Machine
Learning III : Embedding
Classifier
(/python/Flask/Python_Flask_Eml

Flask with Embedded Machine
Learning IV : Deploy
(/python/Flask/Python_Flask_Eml

Flask with Embedded Machine
Learning V : Updating the
classifier
(/python/Flask/Python_Flask_Eml

scikit-learn : Sample of a spam
comment filter using SVM -
classifying a good one or a bad
one (/python/scikit-
learn/scikit_learn_Support_Vecto

MACHINE LEARNING ALGORITHMS AND CONCEPTS

Batch gradient descent
algorithm
(/python/python_numpy_batch_g

Single Layer Neural Network -
Perceptron model on the Iris
dataset using Heaviside step
activation function
(/python/scikit-
learn/Perceptron_Model_with_Iri

Batch gradient descent versus
stochastic gradient descent
(/python/scikit-learn/scikit-

[learn_batch-gradient-descent-versus-stochastic-gradient-descent.php](#))

[Single Layer Neural Network - Adaptive Linear Neuron using linear \(identity\) activation function with batch gradient descent method \(/python/scikit-learn/Single-Layer-Neural-Network-Adaptive-Linear-Neuron.php\)](#)

[Single Layer Neural Network : Adaptive Linear Neuron using linear \(identity\) activation function with stochastic gradient descent \(SGD\) \(/python/scikit-learn/Single-Layer-Neural-Network-Adaptive-Linear-Neuron-with-Stochastic-Gradient-Descent.php\)](#)

[Logistic Regression \(/python/scikit-learn/logistic_regression.php\)](#)

[VC \(Vapnik-Chervonenkis\) Dimension and Shatter \(/python/scikit-learn/scikit_machine_learning_VC\)](#)

[Bias-variance tradeoff \(/python/scikit-learn/scikit_machine_learning_Bias-variance-Tradeoff.php\)](#)

[Maximum Likelihood Estimation \(MLE\) \(/python/scikit-learn/Maximum-Likelihood-Estimation-MLE.php\)](#)

[Neural Networks with backpropagation for XOR using one hidden layer \(/python/python_Neural_Network\)](#)

[minHash \(/Algorithms/minHash_Jaccard_Similarity\)](#)

[tf-idf weight \(/Algorithms/tf_idf_term_frequency\)](#)

Natural Language Processing
(NLP): Sentiment Analysis I
(IMDb & bag-of-words)
(/Algorithms/Machine_Learning_I

Natural Language Processing
(NLP): Sentiment Analysis II
(tokenization, stemming, and
stop words)
(/Algorithms/Machine_Learning_I

Natural Language Processing
(NLP): Sentiment Analysis III
(training & cross validation)
(/Algorithms/Machine_Learning_I

Natural Language Processing
(NLP): Sentiment Analysis IV
(out-of-core)
(/Algorithms/Machine_Learning_I

Locality-Sensitive Hashing (LSH)
using Cosine Distance (Cosine
Similarity)
(/Algorithms/Locality_Sensitive_H

ARTIFICIAL NEURAL NETWORKS (ANN)

[Note] Sources are available at
Github - Jupyter notebook files
(<https://github.com/Einsteinish/Artificial-Neural-Networks-with-Jupyter.git>)

1. Introduction (/python/scikit-learn/Artificial-Neural-Network-ANN-1-Introduction.php)

2. Forward Propagation
(/python/scikit-learn/Artificial-Neural-Network-ANN-2-Forward-Propagation.php)

3. Gradient Descent
(/python/scikit-learn/Artificial-Neural-Network-ANN-3-Gradient-Descent.php)

4. Backpropagation of Errors
(/python/scikit-learn/Artificial-Neural-Network-ANN-4-Backpropagation.php)

5. Checking gradient
(/python/scikit-learn/Artificial-Neural-Network-ANN-5-Checking-Gradient.php)

6. Training via BFGS
(/python/scikit-learn/Artificial-Neural-Network-ANN-6-Training-via-BFGS-Broyden-Fletcher-Goldfarb-Shanno-algorithm-a-variant-of-gradient-descent.php)

7. Overfitting & Regularization
(/python/scikit-learn/Artificial-Neural-Network-ANN-7-Overfitting-Regularization.php)

8. Deep Learning I : Image Recognition (Image uploading)
(/python/scikit-learn/Artificial-Neural-Network-ANN-8-Deep-Learning-1-Image-Recognition-Image-Uploading.php)

9. Deep Learning II : Image Recognition (Image classification) (/python/scikit-learn/Artificial-Neural-Network-ANN-9-Deep-Learning-2-Image-Recognition-Image-Classification.php)

10 - Deep Learning III : Deep Learning III : Theano, TensorFlow, and Keras
(/python/scikit-learn/Artificial-Neural-Network-ANN-10-Deep-Learning-3-Theano-TensorFlow-Keras.php)

CONTACT

BogoToBogo
contactus@bogotobogo.com (mailto:#)

FOLLOW BOGOTOBOGO

f (<https://www.facebook.com/KHongSanFrancisco>) **🐦**
(<https://twitter.com/KHongTwit>) **g⁺**
(<https://plus.google.com/u/0/+KHongSanFrancisco/posts>)

ABOUT US (/ABOUT_US.PHP)

contactus@bogotobogo.com (mailto:contactus@bogotobogo.co)

Golden Gate Ave, San Francisco, CA 94115

Golden Gate Ave, San Francisco, CA 94115

Copyright © 2016, bogotobogo
Design: Web Master (<http://www.bogotobogo.com>)