Importing Data: Python Data Playbook

IMPORTING TEXT DATA INTO PYTHON USING NUMPY

Xavier Morera
BIG DATA INC.

@xmorera www.xaviermorera.com

```
which_file = "Creative Commons Attribution-ShareAlike 3.0"
license_file = open(which_file, mode='r')
license_name = license_file.readline()
license = license_file.read()
license_file.close()
print(license_name)
print(license)
```

Reading Text Files

Import plain text files using open() with mode='r' ***(for read-only)***

Now you can read() or readline() and show contents using print()

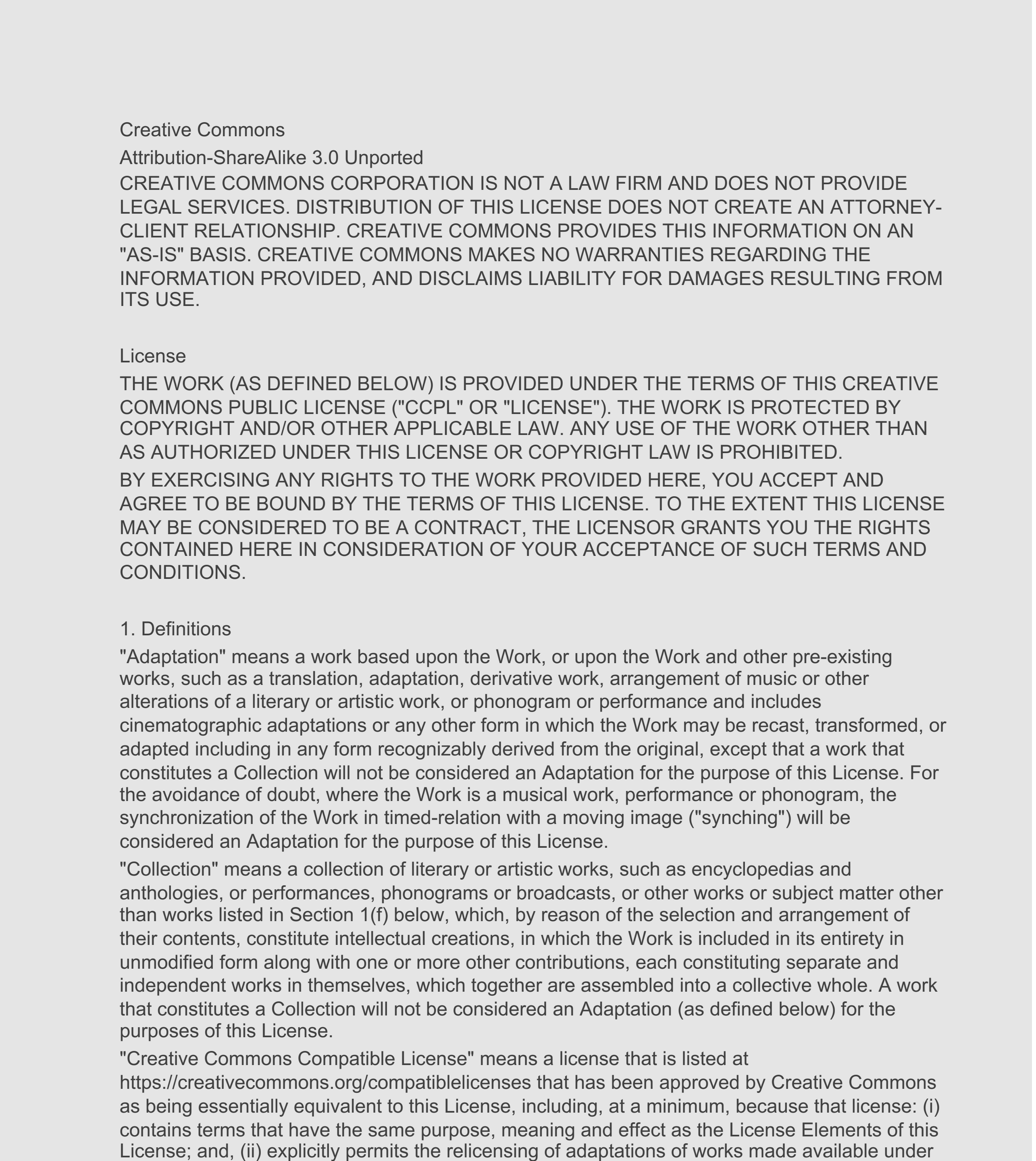
But don't forget to close()

```
with open (which_file, 'r') as file:
    print(file.read())
```

Using a Context Manager

Control when to allocate and release resources

- Using with
- No need to use close()



Creative Commons

Attribution-ShareAlike 3.0 Unported

CREATIVE COMMONS CORPORATION IS NOT A LAW FIRM AND DOES NOT PROVIDE LEGAL SERVICES. DISTRIBUTION OF THIS LICENSE DOES NOT CREATE AN ATTORNEY-CLIENT RELATIONSHIP. CREATIVE COMMONS PROVIDES THIS INFORMATION ON AN "AS-IS" BASIS. CREATIVE COMMONS MAKES NO WARRANTIES REGARDING THE INFORMATION PROVIDED, AND DISCLAIMS LIABILITY FOR DAMAGES RESULTING FROM ITS USE.

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

"Adaptation" means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.

"Collection" means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined below) for the purposes of this License.

"Creative Commons Compatible License" means a license that is listed at https://creativecommons.org/compatiblelicenses that has been approved by Creative Commons as being essentially equivalent to this License, including, at a minimum, because that license: (i) contains terms that have the same purpose, meaning and effect as the License Elements of this License; and, (ii) explicitly permits the relicensing of adaptations of works made available under

Plain text

Useful for certain scenarios

Not so much for others

Better alternatives

Flat files and numeric data

```
Id,Rep,Address,Name
0,100,"on the server farm","Community"
1,101,"New York, NY","Xavier Morera"
2,101,"Alpharetta, GA","Irene Gurdian"
3,101,"","Juli Luci"
```

Text Files: Delimited Files and CSV

Data stored in a text file, in rows with columns

Each row is an entry or a record, and each column is a field

- Separated by a delimiter and potentially with a header

RFC 4180

Id	Rep	Address	Name
0	100	on the server farm	Community
1	101	New York, NY	Xavier Morera
2	101	Alpharetta, GA	Irene Gurdian
3	101		Juli Luci

Fixed-width Text File

Not covered in this training

```
Id,Name,Address

1,Xavier Morera,600 meters north, 75 east from a landmark

1,Xavier Morera,"600 meters north, 75 east from a landmark"
```

Text Files: Delimited Files and CSV

Delimited files separate fields using a specific character

- Comma, tab, pipe…

CSV also have enclosing character

```
1.000000000000000000e+00 1.000000000000000000e+00
2.000000000000000000e+00 2.000000000000000000e+00
3.000000000000000000e+00 4.000000000000000000e+00
4.000000000000000000e+00 5.000000000000000000e+00
5.000000000000000000e+00 8.000000000000000000e+00
```

Text Files: NumPy

We may run into some files that look like this

Created using NumPy

- Learn how to load them shortly

```
[[ 0., 0., 7.],
 [ 3., 1., 3.],
 [ 3., 0., 5.]]
```

Importing Text Files Using NumPy

Homogeneous multidimensional array, called ndarray

Table of elements, usually numbers, of the same type

- Dimensions are called axes

Several functions available for loading data

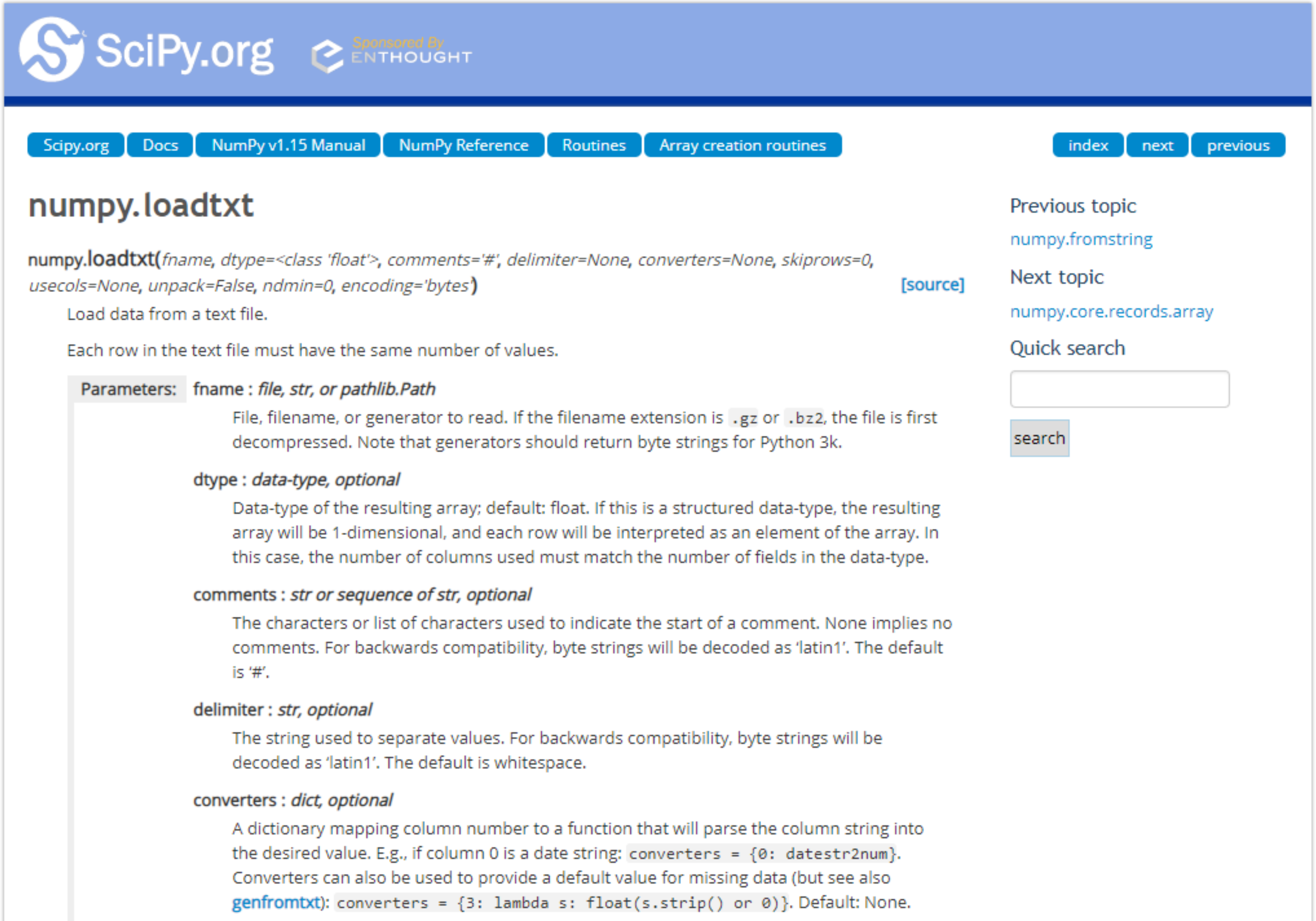
```
import numpy as np

sample_array = np.array([0,0,7])

type(sample_array)

dir(sample_array)
```

Importing Text Files Using NumPy

Start by importing NumPy

Create arrays using array()

- Create an object of type numpy.ndarray



numpy.loadtxt

numpy.loadtxt(*fname, dtype=<class 'float'>, comments='#', delimiter=None, converters=None, skiprows=0, usecols=None, unpack=False, ndmin=0, encoding='bytes'*) [source]

Load data from a text file.

Each row in the text file must have the same number of values.

Parameters:

fname : *file, str, or pathlib.Path*

File, filename, or generator to read. If the filename extension is `.gz` or `.bz2`, the file is first decompressed. Note that generators should return byte strings for Python 3k.

dtype : *data-type, optional*

Data-type of the resulting array; default: float. If this is a structured data-type, the resulting array will be 1-dimensional, and each row will be interpreted as an element of the array. In this case, the number of columns used must match the number of fields in the data-type.

comments : *str or sequence of str, optional*

The characters or list of characters used to indicate the start of a comment. None implies no comments. For backwards compatibility, byte strings will be decoded as 'latin1'. The default is '#'.

delimiter : *str, optional*

The string used to separate values. For backwards compatibility, byte strings will be decoded as 'latin1'. The default is whitespace.

converters : *dict, optional*

A dictionary mapping column number to a function that will parse the column string into the desired value. E.g., if column 0 is a date string: `converters = {0: datestr2num}`. Converters can also be used to provide a default value for missing data (but see also genfromtxt): `converters = {3: lambda s: float(s.strip() or 0)}`. Default: None.

```
# cat badges-five-numpy.txt

badges_saved_np = np.loadtxt('badges-five-numpy.txt')

badges_saved_np

badges_saved_np.size

badges_saved_np.shape
```

Importing Data Using Numpy: loadtxt()

Load a file that was created with NumPy using loadtxt

Into ndarray

- Items of same type and same number of values

```
# cat badges-five.txt

badges_comma = np.loadtxt('badges-five.txt') # error

badges_comma = np.loadtxt('badges-five.txt', delimiter=',’)

badges_comma = np.loadtxt('badges-five.txt', delimiter=‘,’, usecols=0)

badges_comma
```

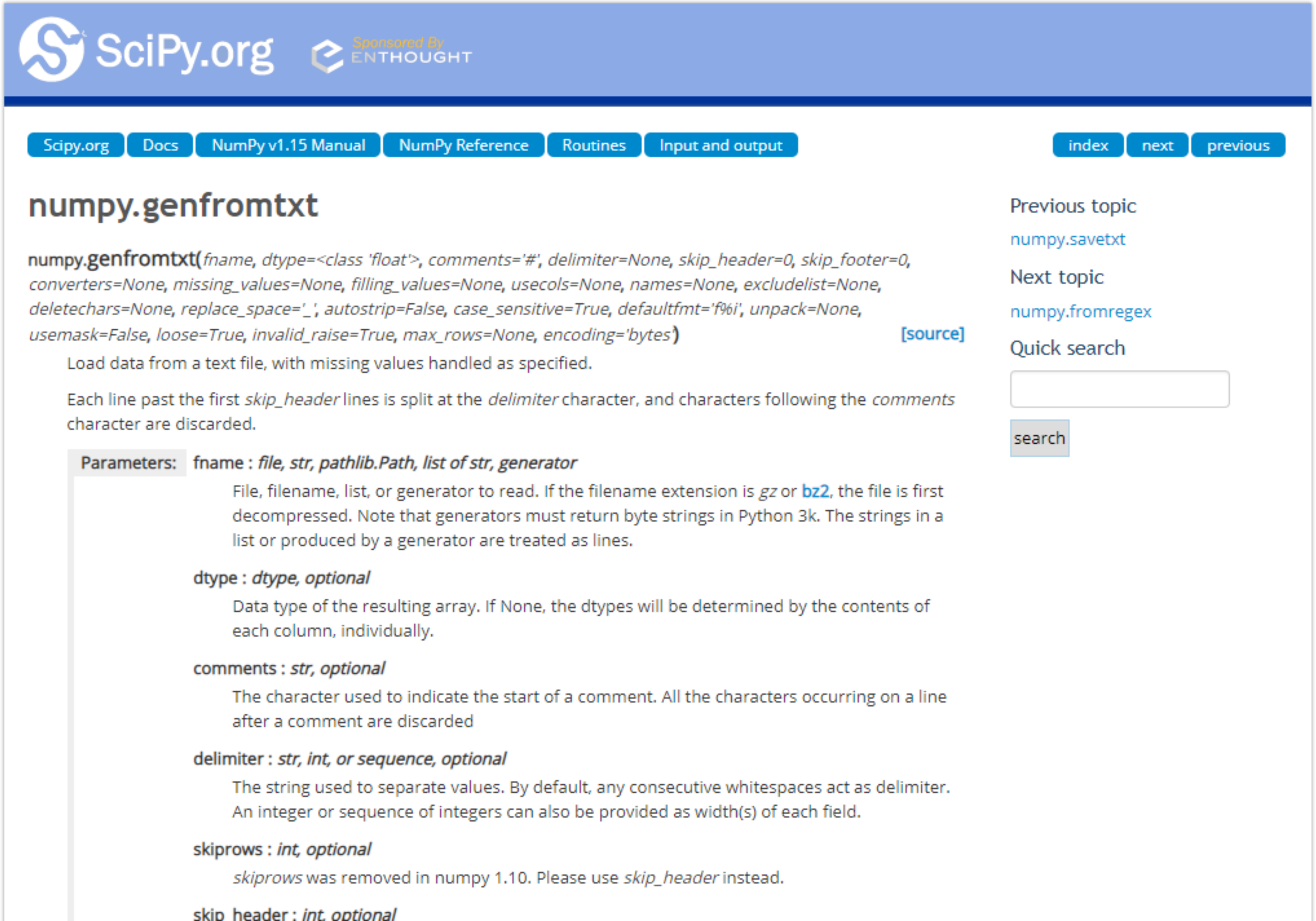
Importing Data Using Numpy: loadtxt()

Files not generated with NumPy

- With delimiter, useful for files created by other means
- Options like usecols, skiprows, dtype, converter, and comments

Certain limitations, i.e. missing values

```
# cat badges-five-header.txt
badges_header = np.loadtxt('badges-five-header.txt', delimiter=',') # error
badges_header = np.loadtxt('badges-five-header.txt', delimiter=',', skiprows=1)
badges_header
badges_header.dtype
badges_str = np.loadtxt('badges-five-header.txt', delimiter=',', skiprows=1, dtype=np.uint)
badges_header.dtype
def Increase(the_id):
    return int(the_id) + 1000
badges_increased = np.loadtxt('badges-five.txt', delimiter=',', dtype=int, converters={0:
Increase})
badges_increased
```



numpy.genfromtxt

numpy.genfromtxt(*fname, dtype=<class 'float'>, comments='#', delimiter=None, skip_header=0, skip_footer=0, converters=None, missing_values=None, filling_values=None, usecols=None, names=None, excludelist=None, deletechars=None, replace_space='_', autostrip=False, case_sensitive=True, defaultfmt='f%i', unpack=None, usemask=False, loose=True, invalid_raise=True, max_rows=None, encoding='bytes'*) [source]

Load data from a text file, with missing values handled as specified.

Each line past the first *skip_header* lines is split at the *delimiter* character, and characters following the *comments* character are discarded.

Parameters:

fname : *file, str, pathlib.Path, list of str, generator*

File, filename, list, or generator to read. If the filename extension is *gz* or bz2, the file is first decompressed. Note that generators must return byte strings in Python 3k. The strings in a list or produced by a generator are treated as lines.

dtype : *dtype, optional*

Data type of the resulting array. If None, the dtypes will be determined by the contents of each column, individually.

comments : *str, optional*

The character used to indicate the start of a comment. All the characters occurring on a line after a comment are discarded

delimiter : *str, int, or sequence, optional*

The string used to separate values. By default, any consecutive whitespaces act as delimiter. An integer or sequence of integers can also be provided as width(s) of each field.

skiprows : *int, optional*

skiprows was removed in numpy 1.10. Please use *skip_header* instead.

skip_header : *int, optional*

```
# cat badges-five-missing-value.txt
badges_missing_value = np.genfromtxt('badges-five-missing-value.txt', delimiter=',')
badges_missing_value = np.genfromtxt('badges-five-missing-value.txt', delimiter=',', skip_header=1)
```

Importing Data Using Numpy: genfromtext()

Data with missing values can be imported with genfromtext

We have:

- First, missing_values **to specify what is considered a missing value**
- And filling_values to specify how to handle