Python scientific packages

- Scientific packages are usually not installed by default.
- Installation procedures are several and may depend on the O.S. Here follow a few suggestions:

Python scientific packages

- Scientific packages are usually not installed by default.
- Installation procedures are several and may depend on the O.S. Here follow a few suggestions:
  - `ipython`, `numpy`, `scipy`, `matplotlib`
    - **Windows**: Can be installed from PyPI repository
    - **MacOS**: You may either use *Homebrew* or install from PyPI repository
    - **Linux**: All main distributions include scientific packages. If you want to have the latest version you may install from the PyPI repository.

Python scientific packages

Installation - 1

- Scientific packages are usually not installed by default.
- Installation procedures are several and may depend on the O.S. Here follow a few suggestions:
  - `ipython`, `numpy`, `scipy`, `matplotlib`
    - **Windows**: Can be installed from PyPI repository
    - **MacOS**: You may either use *Homebrew* or install from PyPI repository
    - **Linux**: All main distributions include scientific packages. If you want to have the latest version you may install from the PyPI repository.
  - `astropy`:
    - **Windows**: Can be installed from PyPI repository
    - **MacOS**: You may either use *Homebrew* or install from PyPI repository
    - **Linux**: You may install from the distribution repository or from PyPI.

Python scientific packages

- Scientific packages are usually not installed by default.
- Installation procedures are several and may depend on the O.S. Here follow a few suggestions:
  - `ipython`, `numpy`, `scipy`, `matplotlib`
    - **Windows**: Can be installed from PyPI repository
    - **MacOS**: You may either use *Homebrew* or install from PyPI repository
    - **Linux**: All main distributions include scientific packages. If you want to have the latest version you may install from the PyPI repository.
  - `astropy`:
    - **Windows**: Can be installed from PyPI repository
    - **MacOS**: You may either use *Homebrew* or install from PyPI repository
    - **Linux**: You may install from the distribution repository or from PyPI.
  - `astroquery`:
    - **Windows**: Can be installed from PyPI repository
    - **MacOS**: You may either use *Homebrew* or install from PyPI repository
    - **Linux**: You may install from the distribution repository (but not all of them provide this package) or from PyPI.

Python scientific packages

- Scientific packages are usually not installed by default.
- Installation procedures are several and may depend on the O.S. Here follow a few suggestions:
  - `ipython`, `numpy`, `scipy`, `matplotlib`
    - **Windows**: Can be installed from PyPI repository
    - **MacOS**: You may either use *Homebrew* or install from PyPI repository
    - **Linux**: All main distributions include scientific packages. If you want to have the latest version you may install from the PyPI repository.
  - `astropy`:
    - **Windows**: Can be installed from PyPI repository
    - **MacOS**: You may either use *Homebrew* or install from PyPI repository
    - **Linux**: You may install from the distribution repository or from PyPI.
  - `astroquery`:
    - **Windows**: Can be installed from PyPI repository
    - **MacOS**: You may either use *Homebrew* or install from PyPI repository
    - **Linux**: You may install from the distribution repository (but not all of them provide this package) or from PyPI.

the Python Package Index

- PyPI is the main repository for Python packages and applications

  http://pypi.python.org

- The command to be used is **pip**. E.g.:
  - **pip install astropy**
  - **pip list**
  - **pip uninstall astropy**

the Python Package Index

- PyPI is the main repository for Python packages and applications

http://pypi.python.org

- The command to be used is **pip**. E.g.:
  - **pip install astropy**
  - **pip list**
  - **pip uninstall astropy**
- Sometimes the installation will require the C/C++ compiler

the Python Package Index

- PyPI is the main repository for Python packages and applications

http://pypi.python.org

- The command to be used is **pip**. E.g.:
  - **pip install astropy**
  - **pip list**
  - **pip uninstall astropy**
- Sometimes the installation will require the C/C++ compiler
- Some suggestions:

the Python Package Index

- PyPI is the main repository for Python packages and applications
  http://pypi.python.org
- The command to be used is **pip**. E.g.:
  - **pip install astropy**
  - **pip list**
  - **pip uninstall astropy**
- Sometimes the installation will require the C/C++ compiler
- Some suggestions:
  - If you find a standard package for your O.S. (e.g.: `msi` [Windows], `pkg` [MacOS], `rpm/deb` [Linux]) it may be better use it.

the Python Package Index

- PyPI is the main repository for Python packages and applications
  http://pypi.python.org
- The command to be used is **pip**. E.g.:
  - **pip install astropy**
  - **pip list**
  - **pip uninstall astropy**
- Sometimes the installation will require the C/C++ compiler
- Some suggestions:
  - If you find a standard package for your O.S. (e.g.: `msi` [Windows], `pkg` [MacOS], `rpm/deb` [Linux]) it may be better use it.
  - Otherwise use **pip**.
  - **pip** is also recommended if you need the latest version of the package

- PyPI is the main repository for Python packages and applications
  http://pypi.python.org
- The command to be used is **pip**. E.g.:
  - **pip install astropy**
  - **pip list**
  - **pip uninstall astropy**
- Sometimes the installation will require the C/C++ compiler
- Some suggestions:
  - If you find a standard package for your O.S. (e.g.: `msi` [Windows], `pkg` [MacOS], `rpm/deb` [Linux]) it may be better use it.
  - Otherwise use **pip**.
  - **pip** is also recommended if you need the latest version of the package
  - Avoid to use both installation methods (maybe at different times)

the Python Package Index

- PyPI is the main repository for Python packages and applications
http://pypi.python.org
- The command to be used is **pip**. E.g.:
  - **pip install astropy**
  - **pip list**
  - **pip uninstall astropy**
- Sometimes the installation will require the C/C++ compiler
- Some suggestions:
  - If you find a standard package for your O.S. (e.g.: `msi` [Windows], `pkg` [MacOS], `rpm/deb` [Linux]) it may be better use it.
  - Otherwise use **pip**.
  - **pip** is also recommended if you need the latest version of the package
  - Avoid to use both installation methods (maybe at different times)

the Python Package Index

Installation - 2

- PyPI is the main repository for Python packages and applications
  http://pypi.python.org
- The command to be used is **pip**. E.g.:
  - **pip instal**
  - **pip list**
  - **pip unins**

> When both python 2.x and python 3.x are installed you may want to specify the command as: **pip3**

- Sometimes the installation will require the C/C++ compiler
- Some suggestions:
  - If you find a standard package for your O.S. (e.g.: `msi` [Windows], `pkg` [MacOS], `rpm/deb` [Linux]) it may be better use it.
  - Otherwise use **pip**.
  - **pip** is also recommended if you need the latest version of the package
  - Avoid to use both installation methods (maybe at different times)

- PyPI is the main repository for Python packages and applications
  http://pypi.python.org
- The command to be used is **pip**. E.g.:
  - **pip instal**
  - **pip list**
  - **pip unins**

> When both python 2.x and python 3.x are installed you may want to specify the command as: **pip3**

- Sometimes the installation will require the C/C++ compiler
- Some suggestions:
  - If you find a standard package for your O.S. (e.g.: `msi` [Windows], `pkg` [MacOS], `rpm/deb` [Linux]) it may be better use it.
  - Otherwise use **pip**.
  - **pip** is also recommended if you need the latest version of the package
  - Avoid to use both installation methods (maybe at different times)

→

`ipython` is an enhanced python environment well suited for interactive use.

Let's see an example:

```
$ ipython
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
Type "copyright", "credits" or "license" for more information.

IPython 5.2.2 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.
...
```

Interactive python: ipython

`ipython` is an enhanced python environment well suited for interactive use.

Let's see an example:

```
$ ipython
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
Type "copyright", "credits" or "license" for more information.

IPython 5.2.2 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.
...
```

- The `ipython` environment:
  - Line completion and history
  - Interaction with O.S.
  - Enhanced introspection tools
  - Simpler graphics
  - "magic" commands

Interactive python: ipython

`ipython` is an enhanced python environment well suited for interactive use.

Let's see an example:

```
$ ipython
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
Type "copyright", "credits" or "license" for more information.

IPython 5.2.2 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.
...
```

- The `ipython` environment:
  - Line completion and history
  - Interaction with O.S.
  - Enhanced introspection tools
  - Simpler graphics
  - "magic" commands
- `ipython --pylab`
  - includes numpy as **np**
  - includes simplified **matplotlib** commands

Interactive python: ipython

`ipython` is an enhanced python environment well suited for interactive use.

Let's see an example:

```
$ ipython
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
Type "copyright", "credits" or "license" for more information.

IPython 5.2.2 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.
...
```

- The `ipython` environment:
  - Line completion and history
  - Interaction with O.S.
  - Enhanced introspection tools
  - Simpler graphics
  - “magic” commands
- `ipython --pylab`
  - includes `numpy` as **np**
  - includes simplified **matplotlib** commands

`ipython` can be used with other programming languages and provides support for parallel computing applications, but these aspects will not be covered here.

Interactive python: ipython

`ipython` is an enhanced python environment well suited for interactive use.

Let's see an example:

```
$ ipython
Python 3.5.2 (default, Nov 23 2017, 16:37:01)
Type "copyright", "credits" or "license" for more information.

IPython 5.2.2 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.
...
```

- The `ipython` environment:
  - Line completion and history
  - Interaction with O.S.
  - Enhanced introspection tools
  - Simpler graphics
  - “magic” commands
- `ipython --pylab`
  - includes `numpy` as **np**
  - includes simplified **matplotlib** commands

`ipython` can be used with other programming languages and provides support for parallel computing applications, but these aspects will not be covered here.

Numerical Python

The `numpy` (Numerical Python) module is usually imported as **np**.

It provides:

1. An **ndarray** object:
    - collection of homogeneous items
    - organized as an N-dimensional array
    - accessed by indexes

Numerical Python

The `numpy` (Numerical Python) module is usually imported as **np**.

It provides:

1. An **ndarray** object:
   - collection of homogeneous items
   - organized as an N-dimensional array
   - accessed by indexes
2. Fast operations on `ndarray` objects

The `numpy` (Numerical Python) module is usually imported as **np**.

It provides:

1. An **ndarray** object:
   - collection of homogeneous items
   - organized as an N-dimensional array
   - accessed by indexes
2. Fast operations on `ndarray` objects
3. Linear algebra functions

The `numpy` (Numerical Python) module is usually imported as **np**.

It provides:

1. An **ndarray** object:
   - collection of homogeneous items
   - organized as an N-dimensional array
   - accessed by indexes
2. Fast operations on `ndarray` objects
3. Linear algebra functions
4. Fourier transforms

The `numpy` (Numerical Python) module is usually imported as **np**.

It provides:

1. An **ndarray** object:
   - collection of homogeneous items
   - organized as an N-dimensional array
   - accessed by indexes
2. Fast operations on `ndarray` objects
3. Linear algebra functions
4. Fourier transforms
5. Random numbers generators

The `numpy` (Numerical Python) module is usually imported as **np**.

It provides:

1. An **ndarray** object:
   - collection of homogeneous items
   - organized as an N-dimensional array
   - accessed by indexes
2. Fast operations on `ndarray` objects
3. Linear algebra functions
4. Fourier transforms
5. Random numbers generators

→

Numerical Python

The `numpy` (Numerical Python) module is usually imported as **np**.

It provides:

1. An **ndarray** object:
   - collection of homogeneous items
   - organized as an N-dimensional array
   - accessed by indexes
2. Fast operations on `ndarray` objects
3. Linear algebra functions
4. Fourier transforms
5. Random numbers generators

→

The array class - 1

array creation:

```
>>> import numpy as np

>>> ar1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar2 = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])

>>> ar1.size, ar2.size
(8, 8)
>>> ar1.shape, ar2.shape
((8,), (2, 4))
>>> ar1.dtype, ar2.dtype
(dtype('int64'), dtype('int64'))

>>> ar5 = np.zeros((2,4))
>>> ar6 = np.ones((2,4,3))

>>> ar7 = np.identity(10)

>>> ar8 = np.linspace(0, np.pi, 5)

>>> # ar9 = np.loadtxt("datafile.txt")
>>> # ar9 = np.fromfile("datafile.txt")
```

The `array` class - 1

`array` creation:

```
>>> import numpy as np

>>> ar1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar2 = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])

>>> ar1.size, ar2.size
(8, 8)
>>> ar1.shape, ar2.shape
((8,), (2, 4))
>>> ar1.dtype, ar2.dtype
(dtype('int64'), dtype('int64'))

>>> ar5 = np.zeros((2,4))
>>> ar6 = np.ones((2,4,3))

>>> ar7 = np.identity(10)

>>> ar8 = np.linspace(0, np.pi, 5)

>>> # ar9 = np.loadtxt("datafile.txt")
>>> # ar9 = np.fromfile("datafile.txt")
```

The `array` class - 1

`array` creation:

```
>>> import numpy as np

>>> ar1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar2 = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])

>>> ar1.size, ar2.size
(8, 8)
>>> ar1.shape, ar2.shape
((8,), (2, 4))
>>> ar1.dtype, ar2.dtype
(dtype('int64'), dtype('int64'))

>>> ar5 = np.zeros((2,4))
>>> ar6 = np.ones((2,4,3))

>>> ar7 = np.identity(10)

>>> ar8 = np.linspace(0, np.pi, 5)

>>> # ar9 = np.loadtxt("datafile.txt")
>>> # ar9 = np.fromfile("datafile.txt")
```

Creating an 8 elements `array` arranged as a 1-D vector from a python list

The `array` class - 1

`array` creation:

```
>>> import numpy as np

>>> ar1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar2 = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])

>>> ar1.size, ar2.size
(8, 8)
>>> ar1.shape, ar2.shape
((8,), (2, 4))
>>> ar1.dtype, ar2.dtype
(dtype('int64'), dtype('int64'))

>>> ar5 = np.zeros((2,4))
>>> ar6 = np.ones((2,4,3))

>>> ar7 = np.identity(10)

>>> ar8 = np.linspace(0, np.pi, 5)

>>> # ar9 = np.loadtxt("datafile.txt")
>>> # ar9 = np.fromfile("datafile.txt")
```

Creating an 8 elements

`ar2` has 8 elements too, but is arranged as a 2 by 4 matrix

The `array` class - 1

`array` creation:

```
>>> import numpy as np

>>> ar1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar2 = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])

>>> ar1.size, ar2.size
(8, 8)
>>> ar1.shape, ar2.shape
((8,), (2, 4))
>>> ar1.dtype, ar2.dtype
(dtype('int64'), dtype('int64'))

>>> ar5 = np.zeros((2,4))
>>> ar6 = np.ones((2,4,3))

>>> ar7 = np.identity(10)

>>> ar8 = np.linspace(0, np.pi, 5)

>>> # ar9 = np.loadtxt("datafile.txt")
>>> # ar9 = np.fromfile("datafile.txt")
```

Creating an 8 elements

`ar2` has 8 elements too, but is arranged as a 2 by 4 matrix

`shape` shows the arrangement of arrays

array creation:

```
>>> import numpy as np

>>> ar1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar2 = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])

>>> ar1.size, ar2.size
(8, 8)
>>> ar1.shape, ar2.shape
((8,), (2, 4))
>>> ar1.dtype, ar2.dtype
(dtype('int64'), dtype('int64'))

>>> ar5 = np.zeros((2,4))
>>> ar6 = np.ones((2,4,3))

>>> ar7 = np.identity(10)

>>> ar8 = np.linspace(0, np.pi, 5)

>>> # ar9 = np.loadtxt("datafile.txt")
>>> # ar9 = np.fromfile("datafile.txt")
```

Creating an 8 elements

`ar2` has 8 elements too, but is arranged as a 2 by 4 matrix

`shape` shows the arrangement of arrays

`dtype` shows the type of array elements

array creation:

```
>>> import numpy as np

>>> ar1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar2 = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])

>>> ar1.size, ar2.size
(8, 8)
>>> ar1.shape, ar2.shape
((8,), (2, 4))
>>> ar1.dtype, ar2.dtype
(dtype('int64'), dtype('int64'))

>>> ar5 = np.zeros((2,4))
>>> ar6 = np.ones((2,4,3))

>>> ar7 = np.identity(10)

>>> ar8 = np.linspace(0, np.pi, 5)

>>> # ar9 = np.loadtxt("datafile.txt")
>>> # ar9 = np.fromfile("datafile.txt")
```

Creating an 8 elements

`ar2` has 8 elements too, but is arranged as a 2 by 4 matrix

`shape` shows the arrangement of arrays

`dtype` shows the type of array elements

A 2 by 4 array of zero's

The `array` class - 1

`array` creation:

```
>>> import numpy as np

>>> ar1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar2 = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])

>>> ar1.size, ar2.size
(8, 8)
>>> ar1.shape, ar2.shape
((8,), (2, 4))
>>> ar1.dtype, ar2.dtype
(dtype('int64'), dtype('int64'))

>>> ar5 = np.zeros((2,4))
>>> ar6 = np.ones((2,4,3))

>>> ar7 = np.identity(10)

>>> ar8 = np.linspace(0, np.pi, 5)

>>> # ar9 = np.loadtxt("datafile.txt")
>>> # ar9 = np.fromfile("datafile.txt")
```

Creating an 8 elements

`ar2` has 8 elements too, but is arranged as a 2 by 4 matrix

`shape` shows the arrangement of arrays

`dtype` shows the type of array elements

A 2 by 4 array of zero's

A 2 by 4 by 3 array of one's

The array class - 1

array creation:

```
>>> import numpy as np

>>> ar1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar2 = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])

>>> ar1.size, ar2.size
(8, 8)
>>> ar1.shape, ar2.shape
((8,), (2, 4))
>>> ar1.dtype, ar2.dtype
(dtype('int64'), dtype('int64'))

>>> ar5 = np.zeros((2,4))
>>> ar6 = np.ones((2,4,3))

>>> ar7 = np.identity(10)

>>> ar8 = np.linspace(0, np.pi, 5)

>>> # ar9 = np.loadtxt("datafile.txt")
>>> # ar9 = np.fromfile("datafile.txt")
```

Creating an 8 elements

`ar2` has 8 elements too, but is arranged as a 2 by 4 matrix

`shape` shows the arrangement of arrays

`dtype` shows the type of array elements

A 2 by 4 array of zero's

A 2 by 4 by 3 array of one's

The identity matrix

The `array` class - 1

`array` creation:

```
>>> import numpy as np

>>> ar1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar2 = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])

>>> ar1.size, ar2.size
(8, 8)
>>> ar1.shape, ar2.shape
((8,), (2, 4))
>>> ar1.dtype, ar2.dtype
(dtype('int64'), dtype('int64'))

>>> ar5 = np.zeros((2,4))
>>> ar6 = np.ones((2,4,3))

>>> ar7 = np.identity(10)

>>> ar8 = np.linspace(0, np.pi, 5)

>>> # ar9 = np.loadtxt("datafile.txt")
>>> # ar9 = np.fromfile("datafile.txt")
```

Creating an 8 elements

`ar2` has 8 elements too, but is arranged as a 2 by 4 matrix

`shape` shows the arrangement of arrays

`dtype` shows the type of `array` elements

A 2 by 4 array of zero's

A 2 by 4 by 3 array of one's

The i

A vector of 5 equally spaced values in [0, pi)

The `array` class - 1

`array` creation:

```
>>> import numpy as np

>>> ar1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar2 = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])

>>> ar1.size, ar2.size
(8, 8)
>>> ar1.shape, ar2.shape
((8,), (2, 4))
>>> ar1.dtype, ar2.dtype
(dtype('int64'), dtype('int64'))

>>> ar5 = np.zeros((2,4))
>>> ar6 = np.ones((2,4,3))

>>> ar7 = np.identity(10)

>>> ar8 = np.linspace(0, np.pi, 5)

>>> # ar9 = np.loadtxt("datafile.txt")
>>> # ar9 = np.fromfile("datafile.txt")
```

Creating an 8 elements

`ar2` has 8 elements too, but is arranged as a 2 by 4 matrix

`shape` shows the arrangement of arrays

`dtype` shows the type of array elements

A 2 by 4 array of zero's

A 2 by 4 by 3 array of one's

The i

A vector of 5 equally spa

Simple ways to read data from files into `arrays`

`array` creation:

```
>>> import numpy as np

>>> ar1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar2 = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])

>>> ar1.size, ar2.size
(8, 8)
>>> ar1.shape, ar2.shape
((8,), (2, 4))
>>> ar1.dtype, ar2.dtype
(dtype('int64'), dtype('int64'))

>>> ar5 = np.zeros((2,4))
>>> ar6 = np.ones((2,4,3))

>>> ar7 = np.identity(10)

>>> ar8 = np.linspace(0, np.pi, 5)

>>> # ar9 = np.loadtxt("datafile.txt")
>>> # ar9 = np.fromfile("datafile.txt")
```

Creating an 8 elements

`ar2` has 8 elements too, but is arranged as a 2 by 4 matrix

`shape` shows the arrangement of arrays

`dtype` shows the type of `array` elements

A 2 by 4 array of zero's

A 2 by 4 by 3 array of one's

The i

A vector of 5 equally spa

Simple ways to read data from files into `arrays`

"views" on arrays:

```
>>> ar3 = np.arange(1, 9)
>>> ar4 = ar3.reshape(2,4)
>>> ar3
array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar4
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
>>> ar3[3] = 0
>>> ar3
array([1, 2, 3, 0, 5, 6, 7, 8])
>>> ar4
array([[1, 2, 3, 0],
       [5, 6, 7, 8]])
```

The array class - 1

array creation:

```
>>> import numpy as np

>>> ar1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar2 = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])

>>> ar1.size, ar2.size
(8, 8)
>>> ar1.shape, ar2.shape
((8,), (2, 4))
>>> ar1.dtype, ar2.dtype
(dtype('int64'), dtype('int64'))

>>> ar5 = np.zeros((2,4))
>>> ar6 = np.ones((2,4,3))

>>> ar7 = np.identity(10)

>>> ar8 = np.linspace(0, np.pi, 5)

>>> # ar9 = np.loadtxt("datafile.txt")
>>> # ar9 = np.fromfile("datafile.txt")
```

Creating an 8 elements

`ar2` has 8 elements too, but is arranged as a 2 by 4 matrix

`shape` shows the arrangement of arrays

`dtype` shows the type of array elements

A 2 by 4 array of zero's

A 2 by 4 by 3 array of one's

The i

A vector of 5 equally spa

Simple ways to read data from files into `arrays`

"views" on arrays:

```
>>> ar3 = np.arange(1, 9)
>>> ar4 = ar3.reshape(2,4)
>>> ar3
array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar4
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
>>> ar3[3] = 0
>>> ar3
array([1, 2, 3, 0, 5, 6, 7, 8])
>>> ar4
array([[1, 2, 3, 0],
       [5, 6, 7, 8]])
```

A vector of values [1..8]

`array` creation:

```
>>> import numpy as np

>>> ar1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar2 = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])

>>> ar1.size, ar2.size
(8, 8)
>>> ar1.shape, ar2.shape
((8,), (2, 4))
>>> ar1.dtype, ar2.dtype
(dtype('int64'), dtype('int64'))

>>> ar5 = np.zeros((2,4))
>>> ar6 = np.ones((2,4,3))

>>> ar7 = np.identity(10)

>>> ar8 = np.linspace(0, np.pi, 5)

>>> # ar9 = np.loadtxt("datafile.txt")
>>> # ar9 = np.fromfile("datafile.txt")
```

Creating an 8 elements

`ar2` has 8 elements too, but is arranged as a 2 by 4 matrix

`shape` shows the arrangement of arrays

`dtype` shows the type of `array` elements

A 2 by 4 array of zero's

A 2 by 4 by 3 array of one's

The i

A vector of 5 equally spaced values in [0, pi)

Simple ways to read data from files into `arrays`

“views” on arrays:

```
>>> ar3 = np.arange(1, 9)
>>> ar4 = ar3.reshape(2,4)
>>> ar3
array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar4
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
>>> ar3[3] = 0
>>> ar3
array([1, 2, 3, 0, 5, 6, 7, 8])
>>> ar4
array([[1, 2, 3, 0],
       [5, 6, 7, 8]])
```

The very same `array`, but “seen” as a 2 by 4 array

The `array` class - 1

package: numpy - 5

`array` creation:

```
>>> import numpy as np

>>> ar1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar2 = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])
```

Creating an 8 elements

`ar2` has 8 elements too, but is arranged as a 2 by 4 matrix

```
>>> ar1.size, ar2.size
(8, 8)
>>> ar1.shape, ar2.shape
((8,), (2, 4))
>>> ar1.dtype, ar2.dtype
(dtype('int64'), dtype('int64'))
```

`shape` shows the arrangement of arrays

`dtype` shows the type of array elements

```
>>> ar5 = np.zeros((2,4))
>>> ar6 = np.ones((2,4,3))
```

A 2 by 4 array of zero's

A 2 by 4 by 3 array of one's

```
>>> ar7 = np.identity(10)
```

The i

```
>>> ar8 = np.linspace(0, np.pi, 5)
```

A vector of 5 equally spa

```
>>> # ar9 = np.loadtxt("datafile.txt")
>>> # ar9 = np.fromfile("datafile.txt")
```

Simple ways to read data from files into `arrays`

“views” on arrays:

```
>>> ar3 = np.arange(1, 9)
>>> ar4 = ar3.reshape(2,4)
>>> ar3
array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar4
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
>>> ar3[3] = 0
>>> ar3
array([1, 2, 3, 0, 5, 6, 7, 8])
>>> ar4
array([[1, 2, 3, 0],
       [5, 6, 7, 8]])
```

The very same `array`, but “seen” as a 2 by 4 array

If we change one element of `ar3` ...

The array class - 1

array creation:

```
>>> import numpy as np

>>> ar1 = np.array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar2 = np.array([[1, 2, 3, 4],[5, 6, 7, 8]])

>>> ar1.size, ar2.size
(8, 8)
>>> ar1.shape, ar2.shape
((8,), (2, 4))
>>> ar1.dtype, ar2.dtype
(dtype('int64'), dtype('int64'))

>>> ar5 = np.zeros((2,4))
>>> ar6 = np.ones((2,4,3))

>>> ar7 = np.identity(10)

>>> ar8 = np.linspace(0, np.pi, 5)

>>> # ar9 = np.loadtxt("datafile.txt")
>>> # ar9 = np.fromfile("datafile.txt")
```

Creating an 8 elements

`ar2` has 8 elements too, but is arranged as a 2 by 4 matrix

`shape` shows the arrangement of arrays

`dtype` shows the type of array elements

A 2 by 4 array of zero's

A 2 by 4 by 3 array of one's

The i

A vector of 5 equally spa

Simple ways to read data from files into `arrays`

"views" on arrays:

```
>>> ar3 = np.arange(1, 9)
>>> ar4 = ar3.reshape(2,4)
>>> ar3
array([1, 2, 3, 4, 5, 6, 7, 8])
>>> ar4
array([[1, 2, 3, 4],
       [5, 6, 7, 8]])
>>> ar3[3] = 0
>>> ar3
array([1, 2, 3, 0, 5, 6
>>> ar4
array([[1, 2, 3, 0],
       [5, 6, 7, 8]])
```

The very same `array`, but "seen" as a 2 by 4 array

If we change one element of `ar3` ...

... the same element is changed in `ar4`

Slices and selection of sub-arrays:

```
>>> a=np.arange(150).reshape(10,5,3)
>>> a
array([[[ 0,   1,   2],
        [  3,   4,   5],
        [  6,   7,   8],
        [  9,  10,  11],
        [ 12,  13,  14]],

       [[ 15,  16,  17],
        [ 18,  19,  20],
        [ 21,  22,  23],
        [ 24,  25,  26],
        [ 27,  28,  29]],
   ...

>>> a[1,:,:]
array([[15, 16, 17],
       [18, 19, 20],
       [21, 22, 23],
       [24, 25, 26],
       [27, 28, 29]])

>>> a[2:4,:,:]
array([[[30, 31, 32],
        [33, 34, 35],
        [36, 37, 38],
        [39, 40, 41],
        [42, 43, 44]],

       [[45, 46, 47],
        [48, 49, 50],
        [51, 52, 53],
        [54, 55, 56],
        [57, 58, 59]]])
>>> a[:,1,1]
array([ 4,  19,  34,  49,  64,  79,  94, 109, 124, 139])
```

Slices and selection of sub-arrays:

Make a 10 by 5 by 3 array ⟸

```
>>> a=np.arange(150).reshape(10,5,3)
>>> a
array([[[ 0,   1,   2],
        [ 3,   4,   5],
        [ 6,   7,   8],
        [ 9,  10,  11],
        [ 12,  13,  14]],

       [[ 15,  16,  17],
        [ 18,  19,  20],
        [ 21,  22,  23],
        [ 24,  25,  26],
        [ 27,  28,  29]],
   ...

>>> a[1,:,:]
array([[15, 16, 17],
       [18, 19, 20],
       [21, 22, 23],
       [24, 25, 26],
       [27, 28, 29]])

>>> a[2:4,:,:]
array([[[30, 31, 32],
        [33, 34, 35],
        [36, 37, 38],
        [39, 40, 41],
        [42, 43, 44]],

       [[45, 46, 47],
        [48, 49, 50],
        [51, 52, 53],
        [54, 55, 56],
        [57, 58, 59]]])
>>> a[:,1,1]
array([ 4, 19, 34, 49, 64, 79, 94, 109, 124, 139])
```

Slices and selection of sub-arrays:

Make a 10 by 5 by 3 array

You may think of it as a sequence of ten 5 by 3 arrays

```
>>> a=np.arange(150).reshap
>>> a
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8],
        [ 9, 10, 11],
        [ 12, 13, 14]],

       [[ 15, 16, 17],
        [ 18, 19, 20],
        [ 21, 22, 23],
        [ 24, 25, 26],
        [ 27, 28, 29]],
   ...

>>> a[1,:,:]
array([[15, 16, 17],
       [18, 19, 20],
       [21, 22, 23],
       [24, 25, 26],
       [27, 28, 29]])

>>> a[2:4,:,:]
array([[[30, 31, 32],
        [33, 34, 35],
        [36, 37, 38],
        [39, 40, 41],
        [42, 43, 44]],

       [[45, 46, 47],
        [48, 49, 50],
        [51, 52, 53],
        [54, 55, 56],
        [57, 58, 59]]])
>>> a[:,1,1]
array([ 4, 19, 34, 49, 64, 79, 94, 109, 124, 139])
```

Slices and selection of sub-arrays:

```
>>> a=np.arange(150).reshap
>>> a
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8],
        [ 9, 10, 11],
        [ 12, 13, 14]],

       [[ 15, 16, 17],
        [ 18, 19, 20],
        [ 21, 22, 23],
        [ 24, 25, 26],
        [ 27, 28,
  ...

>>> a[1,:,:]
array([[15, 16, 17],
       [18, 19, 20],
       [21, 22, 23],
       [24, 25, 26],
       [27, 28, 29]])

>>> a[2:4,:,:]
array([[[30, 31, 32],
        [33, 34, 35],
        [36, 37, 38],
        [39, 40, 41],
        [42, 43, 44]],

       [[45, 46, 47],
        [48, 49, 50],
        [51, 52, 53],
        [54, 55, 56],
        [57, 58, 59]]])
>>> a[:,1,1]
array([ 4, 19, 34, 49, 64, 79, 94, 109, 124, 139])
```

Make a 10 by 5 by 3 array

You may think of it as a sequence of ten 5 by 3 arrays

This is the second element of the 10 elements sequence (remember: the first element has index 0)

Slices and selection of sub-arrays:

Make a 10 by 5 by 3 array

You may think of it as a sequence of ten 5 by 3 arrays

```
>>> a=np.arange(150).reshap
>>> a
array([[[ 0,  1,  2],
        [ 3,  4,  5],
        [ 6,  7,  8],
        [ 9, 10, 11],
        [ 12, 13, 14]],

       [[ 15, 16, 17],
        [ 18, 19, 20],
        [ 21, 22, 23],
        [ 24, 25, 26],
        [ 27, 28,
   ...
```

This is the second element of the 10 elements sequence (remember: the first element has index 0)

```
>>> a[1,:,:]
array([[15, 16, 17],
       [18, 19, 20],
       [21, 22, 23],
       [24, 25, 26],
       [27, 28, 29]
```

The third and fourth elements of the 10 elements sequence

```
>>> a[2:4,:,:]
array([[[30, 31, 32],
        [33, 34, 35],
        [36, 37, 38],
        [39, 40, 41],
        [42, 43, 44]],

       [[45, 46, 47],
        [48, 49, 50],
        [51, 52, 53],
        [54, 55, 56],
        [57, 58, 59]]])
>>> a[:,1,1]
array([ 4, 19, 34, 49, 64, 79, 94, 109, 124, 139])
```

Slices and selection of sub-arrays:

Make a 10 by 5 by 3 array

You may think of it as a sequence of ten 5 by 3 arrays

```
>>> a=np.arange(150).reshap
>>> a
array([[[ 0,   1,   2],
        [ 3,   4,   5],
        [ 6,   7,   8],
        [ 9,  10,  11],
        [ 12, 13,  14]],

       [[ 15, 16,  17],
        [ 18, 19,  20],
        [ 21, 22,  23],
        [ 24, 25,  26],
        [ 27, 28,
   ...
```

This is the second element of the 10 elements sequence (remember: the first element has index 0)

```
>>> a[1,:,:]
array([[15, 16, 17],
       [18, 19, 20],
       [21, 22, 23],
       [24, 25, 26],
       [27, 28, 29]
```

The third and fourth elements of the 10 elements sequence

```
>>> a[2:4,:,:]
array([[[30, 31, 32],
        [33, 34, 35],
        [36, 37, 38],
        [39, 40, 41],
        [42, 43, 44]],

       [[45, 46, 47],
        [48, 49, 50],
        [51, 52, 53],
        [54, 55, 5
        [57, 58, 5
```

A vector of all the elements of index [1,1] of the 10 elements sequence

```
>>> a[:,1,1]
array([ 4, 19, 34, 49, 64, 79, 94, 109, 124, 139])
```

Slices and selection of sub-arrays:

```
>>> a=np.arange(150).reshap
>>> a
array([[[ 0,   1,   2],
        [ 3,   4,   5],
        [ 6,   7,   8],
        [ 9,  10,  11],
        [ 12,  13,  14]],

       [[ 15,  16,  17],
        [ 18,  19,  20],
        [ 21,  22,  23],
        [ 24,  25,  26],
        [ 27,  28,
   ...

>>> a[1,:,:]
array([[15, 16, 17],
       [18, 19, 20],
       [21, 22, 23],
       [24, 25, 26],
       [27, 28, 29]

>>> a[2:4,:,:]
array([[[30, 31, 32],
        [33, 34, 35],
        [36, 37, 38],
        [39, 40, 41],
        [42, 43, 44]],

       [[45, 46, 47],
        [48, 49, 50],
        [51, 52, 53],
        [54, 55, 5
        [57, 58, 5
>>> a[:,1,1]
array([ 4, 19, 34, 49, 64, 79, 94, 109, 124, 139])
```

Make a 10 by 5 by 3 array

You may think of it as a sequence of ten 5 by 3 arrays

This is the second element of the 10 elements sequence (remember: the first element has index 0)

The third and fourth elements of the 10 elements sequence

A vector of all the elements of index [1,1] of the 10 elements sequence

→

Simple array operations

array <op> scalar:

```
>>> a = np.arange(0., 10, 0.3)
>>> a
array([ 0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1, 2.4, 2.7, 3. ,
        3.3, 3.6, 3.9, 4.2, 4.5, 4.8, 5.1, 5.4, 5.7, 6. , 6.3,
        6.6, 6.9, 7.2, 7.5, 7.8, 8.1, 8.4, 8.7, 9. , 9.3, 9.6,
        9.9])
>>> a*3
array([ 0. ,  0.9,  1.8,  2.7,  3.6,  4.5,  5.4,  6.3,  7.2,
         8.1,  9. ,  9.9, 10.8, 11.7, 12.6, 13.5, 14.4, 15.3,
        16.2, 17.1, 18. , 18.9, 19.8, 20.7, 21.6, 22.5, 23.4,
        24.3, 25.2, 26.1, 27. , 27.9, 28.8, 29.7])
```

array <op> scalar:

```
>>> a = np.arange(0., 10, 0.3)
>>> a
array([ 0. , 0.3, 0.6, 0.9,
       3.3, 3.6, 3.9, 4.2, 4
       6.6, 6.9, 7.2, 7.5, 7
       9.9])
>>> a*3
array([ 0. ,  0.9,  1.8,  2.7,  3.6,  4.5,  5.4,  6.3,  7.2,
        8.1,  9. ,  9.9, 10.8, 11.7, 12.6, 13.5, 14.4, 15.3,
       16.2, 17.1, 18. , 18.9, 19.8, 20.7, 21.6, 22.5, 23.4,
       24.3, 25.2, 26.1, 27. , 27.9, 28.8, 29.7])
```

The operation (**+**, **-**, *, /, ...) is performed on each element of the array

Simple array operations

array <op> scalar:

```
>>> a = np.arange(0., 10, 0.3)
>>> a
array([ 0. , 0.3, 0.6, 0.9,
        3.3, 3.6, 3.9, 4.2, 4
        6.6, 6.9, 7.2, 7.5, 7
        9.9])
>>> a*3
array([ 0. ,  0.9,  1.8,  2.7,  3.6,  4.5,  5.4,  6.3,  7.2,
         8.1,  9. ,  9.9, 10.8, 11.7, 12.6, 13.5, 14.4, 15.3,
        16.2, 17.1, 18. , 18.9, 19.8, 20.7, 21.6, 22.5, 23.4,
        24.3, 25.2, 26.1, 27. , 27.9, 28.8, 29.7])
```

The operation (**+**, **-**, *, /, ...) is performed on each element of the array

array <op> array:

```
>>> b = np.arange(10., 0, -0.3)
>>> b
array([ 10. , 9.7,  9.4,  9.1,  8.8,  8.5,  8.2,  7.9,  7.6,
         7.3,  7. ,  6.7,  6.4,  6.1,  5.8,  5.5,  5.2,  4.9,
         4.6,  4.3,  4. ,  3.7,  3.4,  3.1,  2.8,  2.5,  2.2,
         1.9,  1.6,  1.3,  1. ,  0.7,  0.4,  0.1])
>>> a+b
array([ 10., 10., 10., 10., 10., 10., 10., 10., 10., 10., 10.,
        10., 10., 10., 10., 10., 10., 10., 10., 10., 10., 10.,
        10., 10., 10., 10., 10., 10., 10., 10., 10., 10., 10.,
        10.])
```

Simple array operations

array <op> scalar:

```
>>> a = np.arange(0., 10, 0.3)
>>> a
array([ 0. , 0.3, 0.6, 0.9,
       3.3, 3.6, 3.9, 4.2, 4
       6.6, 6.9, 7.2, 7.5, 7
       9.9])
>>> a*3
array([ 0. ,  0.9,  1.8,  2.7,  3.6,  4.5,  5.4,  6.3,  7.2,
        8.1,  9. ,  9.9, 10.8, 11.7, 12.6, 13.5, 14.4, 15.3,
       16.2, 17.1, 18. , 18.9, 19.8, 20.7, 21.6, 22.5, 23.4,
       24.3, 25.2, 26.1, 27. , 27.9, 28.8, 29.7])
```

The operation (**+**, **-**, *, /, ...) is performed on each element of the array

array <op> array:

```
>>> b = np.arange(10., 0, -0.3)
>>> b
array([ 10. , 9.7,  9.4,  9.
        7.3,  7. ,  6.7,  6.
        4.6,  4.3,  4. ,  3.
        1.9,  1.6,  1.3,  1.
>>> a+b
array([ 10., 10., 10., 10., 10., 10., 10., 10., 10., 10., 10.,
       10., 10., 10., 10., 10., 10., 10., 10., 10., 10., 10.,
       10., 10., 10., 10., 10., 10., 10., 10., 10., 10., 10.,
       10.])
```

The operation is performed element by element. The operands must be of the same size and shape

Simple array operations

array <op> scalar:

```
>>> a = np.arange(0., 10, 0.3)
>>> a
array([ 0. , 0.3, 0.6, 0.9,
        3.3, 3.6, 3.9, 4.2, 4
        6.6, 6.9, 7.2, 7.5, 7
        9.9])
>>> a*3
array([ 0. ,  0.9,  1.8,  2.7,  3.6,  4.5,  5.4,  6.3,  7.2,
        8.1,  9. ,  9.9, 10.8, 11.7, 12.6, 13.5, 14.4, 15.3,
       16.2, 17.1, 18. , 18.9, 19.8, 20.7, 21.6, 22.5, 23.4,
       24.3, 25.2, 26.1, 27. , 27.9, 28.8, 29.7])
```

The operation (**+**, **-**, *, /, ...) is performed on each element of the array

array <op> array:

```
>>> b = np.arange(10., 0, -0.3)
>>> b
array([ 10. , 9.7,  9.4,  9.
        7.3,  7. ,  6.7,  6.4
        4.6,  4.3,  4. ,  3.7
        1.9,  1.6,  1.3,  1.
>>> a+b
array([ 10., 10., 10., 10., 10., 10., 10., 10., 10., 10., 10.,
       10., 10., 10., 10., 10., 10., 10., 10., 10., 10., 10.,
       10., 10., 10., 10., 10., 10., 10., 10., 10., 10., 10.,
       10.])
```

The operation is performed element by element. The operands must be of the same size and shape

Dot product ($\sum a_i b_i$):

```
>>> np.dot(a,b)
555.38999999999726
```

Vector product:

```
>>> prod = np.outer(a,b)
>>> prod.shape
(34, 34)
```

array <op> scalar:

```
>>> a = np.arange(0., 10, 0.3)
>>> a
array([ 0.  , 0.3, 0.6, 0.9,
        3.3, 3.6, 3.9, 4.2, 4
        6.6, 6.9, 7.2, 7.5, 7
        9.9])
>>> a*3
array([ 0.  ,  0.9,  1.8,  2.7,  3.6,  4.5,  5.4,  6.3,  7.2,
         8.1,  9.  ,  9.9, 10.8, 11.7, 12.6, 13.5, 14.4, 15.3,
        16.2, 17.1, 18.  , 18.9, 19.8, 20.7, 21.6, 22.5, 23.4,
        24.3, 25.2, 26.1, 27.  , 27.9, 28.8, 29.7])
```

The operation (**+**, **-**, *, /, ...) is performed on each element of the array

array <op> array:

```
>>> b = np.arange(10., 0, -0.3)
>>> b
array([ 10.  , 9.7,  9.4,  9.
         7.3,  7.  ,  6.7,  6.
         4.6,  4.3,  4.  ,  3.
         1.9,  1.6,  1.3,  1.
>>> a+b
array([ 10., 10., 10., 10., 10., 10., 10., 10., 10., 10., 10.,
        10., 10., 10., 10., 10., 10., 10., 10., 10., 10., 10.,
        10., 10., 10., 10., 10., 10., 10., 10., 10., 10., 10.,
        10.])
```

The operation is performed element by element. The operands must be of the same size and shape

Dot product ($\sum a_i b_i$):

```
>>> np.dot(a,b)
555.38999999999726
```

Vector product:

```
>>> prod = np.outer(a,b)
>>> prod.shape
(34, 34)
```

→

Function operating on arrays

All functions defined in `numpy` operate usually element by element

- Trigonometric functions

Function operating on arrays

All functions defined in `numpy` operate usually element by element

- Trigonometric functions
- Hyperbolic functions

Function operating on arrays

All functions defined in `numpy` operate usually element by element

- Trigonometric functions
- Hyperbolic functions
- Rounding (round(), floor(), ceil(), ...)

Function operating on arrays

All functions defined in `numpy` operate usually element by element

- Trigonometric functions
- Hyperbolic functions
- Rounding (round(), floor(), ceil(), ...)
- Sum, product, difference, division

Function operating on arrays

All functions defined in `numpy` operate usually element by element

- Trigonometric functions
- Hyperbolic functions
- Rounding (round(), floor(), ceil(), ...)
- Sum, product, difference, division
- Exponential, logarithms, bessel functions

Function operating on arrays

All functions defined in `numpy` operate usually element by element

- Trigonometric functions
- Hyperbolic functions
- Rounding (round(), floor(), ceil(), ...)
- Sum, product, difference, division
- Exponential, logarithms, bessel functions
- Floating point functions

Function operating on arrays

All functions defined in `numpy` operate usually element by element

- Trigonometric functions
- Hyperbolic functions
- Rounding (round(), floor(), ceil(), ...)
- Sum, product, difference, division
- Exponential, logarithms, bessel functions
- Floating point functions
- Arithmetic functions

Function operating on arrays

All functions defined in `numpy` operate usually element by element

- Trigonometric functions
- Hyperbolic functions
- Rounding (round(), floor(), ceil(), ...)
- Sum, product, difference, division
- Exponential, logarithms, bessel functions
- Floating point functions
- Arithmetic functions
- Complex functions

Beware of function name conflicts:

```
>>> import numpy as np
>>> a=np.array([[1,2,3],[4,5,6]])
>>> max(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: The truth value of an array with more than one element is ambiguous.
>>> np.max(a)
6
>>>
```

Function operating on arrays

All functions defined in `numpy` operate usually element by element

- Trigonometric functions
- Hyperbolic functions
- Rounding (round(), floor(), ceil(), ...)
- Sum, product, difference, division
- Exponential, logarithms, bessel functions
- Floating point functions
- Arithmetic functions
- Complex functions

Beware of function name conflicts:

```
>>> import numpy as np
>>> a=np.array([[1,2,3],[4,5,6]])
>>> max(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: The truth value of an array with more than one element is ambiguous.
>>> np.max(a)
6
>>>
```

Function operating on arrays

All functions defined in `numpy` operate usually element by element

- Trigonometric functions
- Hyperbolic functions
- Rounding (round(), floor(), ceil(), ...)
- Sum, product, difference, division
- Exponential, logarithms, bessel functions
- Floating point functions
- Arithmetic functions
- Complex functions

Beware of function name conflicts:

```
>>> import numpy as np
>>> a=np.array([[1,2,3],[4,5,6]
>>> max(a)
Traceback (most recent call las
  File "<stdin>", line 1, in <module>
ValueError: The truth value of an array with more than one element is ambiguous.
>>> np.max(a)
6
>>>
```

The error comes out because we tried to use the python standard function **max()** on an array object.

All functions defined in `numpy` operate usually element by element

- Trigonometric functions
- Hyperbolic functions
- Rounding (round(), floor(), ceil(), ...)
- Sum, product, difference, division
- Exponential, logarithms, bessel functions
- Floating point functions
- Arithmetic functions
- Complex functions

Beware of function name conflicts:

```
>>> import numpy as np
>>> a=np.array([[1,2,3],[4,5,6]
>>> max(a)
Traceback (most recent call las
  File "<stdin>", l
ValueError: The tr                                          is ambiguous.
>>> np.max(a)
6
>>>
```

The error comes out because we tried to use the python standard function **max()** on an array object.

The function to use to find the maximum value in an array is **np.max()**

Function operating on arrays

All functions defined in `numpy` operate usually element by element

- Trigonometric functions
- Hyperbolic functions
- Rounding (round(), floor(), ceil(), ...)
- Sum, product, difference, division
- Exponential, logarithms, bessel functions
- Floating point functions
- Arithmetic functions
- Complex functions

Beware of function name conflicts:

```
>>> import numpy as np
>>> a=np.array([[1,2,3],[4,5,6]
>>> max(a)
Traceback (most recent call las
  File "<stdin>", I
ValueError: The tr                                          is ambiguous.
>>> np.max(a)
6
>>>
```

The error comes out because we tried to use the python standard function **max()** on an array object.

The function to use to find the maximum value in an array is **np.max()**

numpy: sub-modules

Let's have a look to some `numpy` sub-modules by means of the manual:

- numpy.ma: Operations on "masked" arrays.

numpy: sub-modules

Let's have a look to some numpy sub-modules by means of the manual:

- numpy.ma: Operations on "masked" arrays.
- numpy.linalg: Linear algebra algorithms.

numpy: sub-modules

Let's have a look to some numpy sub-modules by means of the manual:

- numpy.ma: Operations on "masked" arrays.
- numpy.linalg: Linear algebra algorithms.
- numpy.matlib: Matrix operations

numpy: sub-modules

Let's have a look to some numpy sub-modules by means of the manual:

- numpy.ma: Operations on "masked" arrays.
- numpy.linalg: Linear algebra algorithms.
- numpy.matlib: Matrix operations
- numpy.random: Random numbers and distributions

numpy: sub-modules

Let's have a look to some numpy sub-modules by means of the manual:

- numpy.ma: Operations on “masked” arrays.
- numpy.linalg: Linear algebra algorithms.
- numpy.matlib: Matrix operations
- numpy.random: Random numbers and distributions
- numpy.fft: Discrete Fourier transforms

`numpy`: sub-modules

Let's have a look to some `numpy` sub-modules by means of the manual:

- numpy.ma: Operations on "masked" arrays.
- numpy.linalg: Linear algebra algorithms.
- numpy.matlib: Matrix operations
- numpy.random: Random numbers and distributions
- numpy.fft: Discrete Fourier transforms

→

numpy: sub-modules

Let's have a look to some numpy sub-modules by means of the manual:

- numpy.ma: Operations on "masked" arrays.
- numpy.linalg: Linear algebra algorithms.
- numpy.matlib: Matrix operations
- numpy.random: Random numbers and distributions
- numpy.fft: Discrete Fourier transforms

→

Using **numpy** - 1

Problem: Verify whether there is significant correlation between number of births and moon phase.

Input data: Number of births for each day from 6/1/1978 to 11/15/1986 (From the civil registry of Firenze).

file: `dati_nascite.dat`

```
791124 0 0 0
791125 0 1 1
791126 0 0 0
.....
```

Using **numpy** - 1

Problem: Verify whether there is significant correlation between number of births and moon phase.

Input data: Number of births for each day from 6/1/1978 to 11/15/1986 (From the civil registry of Firenze).

file: `dati_nascite.dat`

```
791124 0 0 0
791125 0 1 1
791126 0 0 0
.....
```

A file 3090 lines long, each line contains date, number of males, number of females, total number of births.

Using **numpy** - 1

Problem: Verify whether there is significant correlation between number of births and moon phase.

Input data: Number of births for each day from 6/1/1978 to 11/15/1986 (From the civil registry of Firenze).

file: `dati_nascite.dat`

```
791124 0 0 0
791125 0 1 1
791126 0 0 0
.....
```

A file 3090 lines long, each line contains date, number of males, number of females, total number of births.

In the following slide we'll see how to set up a few useful tools (functions) and then we'll try to provide an answer.

Using **numpy** - 1

Problem: Verify whether there is significant correlation between number of births and moon phase.

Input data: Number of births for each day from 6/1/1978 to 11/15/1986 (From the civil registry of Firenze).

file: `dati_nascite.dat`

```
791124 0 0 0
791125 0 1 1
791126 0 0 0
.....
```

A file 3090 lines long, each line contains date, number of males, number of females, total number of births.

In the following slide we'll see how to set up a few useful tools (functions) and then we'll try to provide an answer.

→

Using **numpy** - 1

Problem: Verify whether there is significant correlation between number of births and moon phase.

Input data: Number of births for each day from 6/1/1978 to 11/15/1986 (From the civil registry of Firenze).

file: `dati_nascite.dat`

```
791124 0 0 0
791125 0 1 1
791126 0 0 0
.....
```

A file 3090 lines long, each line contains date, number of males, number of females, total number of births.

In the following slide we'll see how to set up a few useful tools (functions) and then we'll try to provide an answer.

→

file: moon.py

```
import math

# Mean moon month: 29 days, 12 hours, 43 minutes, 11 seconds
MOON_MONTH = 29*86400 + 12*3600 + 43*60 + 11 # seconds
MOON_QUARTER = MOON_MONTH*0.25 # Duration of a moon quarter

# First new moon after 1/1/1970
NEW_MOON_1 = 660262.0 # seconds
MOON_ZERO = NEW_MOON_1 - MOON_QUARTER*0.5

def moon_phase(nsec): # Moon phase at given time
                    # 0: new, 1:first quarter, 2:full, 3:last quarter
    phase_sec = math.fmod(nsec-MOON_ZERO, MOON_MONTH)
    phase = int(phase_sec/MOON_QUARTER)
    return phase
```

Notes:

file: moon.py

```
import math

# Mean moon month: 29 days, 12 hours, 43 minutes, 11 seconds
MOON_MONTH = 29*86400 + 12*3600 + 43*60 + 11 # seconds
MOON_QUARTER = MOON_MONTH*0.25 # Duration of a moon quarter

# First new moon after 1/1/1970
NEW_MOON_1 = 660262.0 # seconds
MOON_ZERO = NEW_MOON_1 - MOON_QUARTER*0.5

def moon_phase(nsec): # Moon phase at given time
                    # 0: new, 1:first quarter, 2:full, 3:last quarter
    phase_sec = math.fmod(nsec-MOON_ZERO, MOON_MONTH)
    phase = int(phase_sec/MOON_QUARTER)
    return phase
```

Notes:

- For time related computations we use the standard `time` module. It's base time reference is the number of seconds since 1/1/1970 00:00 (`sec70`, in the following).

file: moon.py

```
import math

# Mean moon month: 29 days, 12 hours, 43 minutes, 11 seconds
MOON_MONTH = 29*86400 + 12*3600 + 43*60 + 11 # seconds
MOON_QUARTER = MOON_MONTH*0.25 # Duration of a moon quarter

# First new moon after 1/1/1970
NEW_MOON_1 = 660262.0 # seconds
MOON_ZERO = NEW_MOON_1 - MOON_QUARTER*0.5

def moon_phase(nsec): # Moon phase at given time
                      # 0: new, 1:first quarter, 2:full, 3:last quarter
    phase_sec = math.fmod(nsec-MOON_ZERO, MOON_MONTH)
    phase = int(phase_sec/MOON_QUARTER)
    return phase
```

Notes:

- For time related computations we use the standard `time` module. It's base time reference is the number of seconds since 1/1/1970 00:00 (`sec70`, in the following).
- The value `NEW_MOON_1` was found from a table selecting a “convenient” date and converting into `sec70`.

file: moon.py

```
import math

# Mean moon month: 29 days, 12 hours, 43 minutes, 11 seconds
MOON_MONTH = 29*86400 + 12*3600 + 43*60 + 11 # seconds
MOON_QUARTER = MOON_MONTH*0.25 # Duration of a moon quarter

# First new moon after 1/1/1970
NEW_MOON_1 = 660262.0 # seconds
MOON_ZERO = NEW_MOON_1 - MOON_QUARTER*0.5

def moon_phase(nsec): # Moon phase at given time
                    # 0: new, 1:first quarter, 2:full, 3:last quarter
    phase_sec = math.fmod(nsec-MOON_ZERO, MOON_MONTH)
    phase = int(phase_sec/MOON_QUARTER)
    return phase
```

Notes:

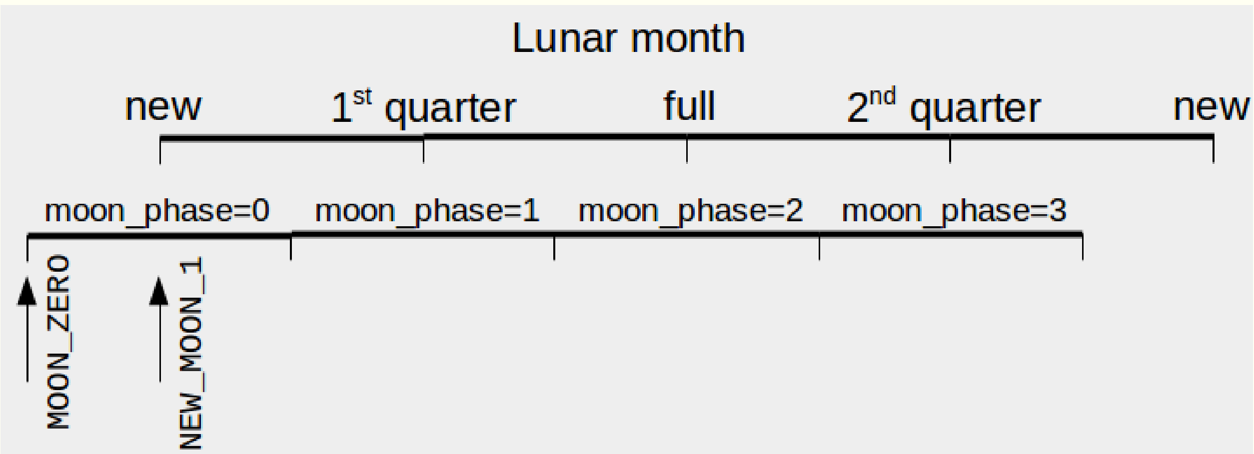
- For time related computations we use the standard `time` module. It's base time reference is the number of seconds since 1/1/1970 00:00 (`sec70`, in the following).
- The value `NEW_MOON_1` was found from a table selecting a "convenient" date and converting into `sec70`.
- The value `MOON_ZERO` is set at half a moon quarter before new moon so that we have four interval centered around the start of each quarter (see figure below).

file: moon.py

```
import math

# Mean moon month: 29 days, 12 hours, 43 minutes, 11 seconds
MOON_MONTH = 29*86400 + 12*3600 + 43*60 + 11 # seconds
MOON_QUARTER = MOON_MONTH*0.25 # Duration of a moon quarter

# First new moon after 1/1/1970
NEW_MOON_1 = 660262.0 # seconds
MOON_ZERO = NEW_MOON_1 - MOON_QUARTER*0.5

def moon_phase(nsec): # Moon phase at given time
                      # 0: new, 1:first quarter, 2:full, 3:last quarter
    phase_sec = math.fmod(nsec-MOON_ZERO, MOON_MONTH)
    phase = int(phase_sec/MOON_QUARTER)
    return phase
```

Notes:

- For time related computations we use the standard `time` module. It's base time reference is the number of seconds since 1/1/1970 00:00 (`sec70`, in the following).
- The value `NEW_MOON_1` was found from a table selecting a "convenient" date and converting into `sec70`.
- The value `MOON_ZERO` is set at half a moon quarter before new moon so that we have four interval centered around the start of each quarter (see figure below).

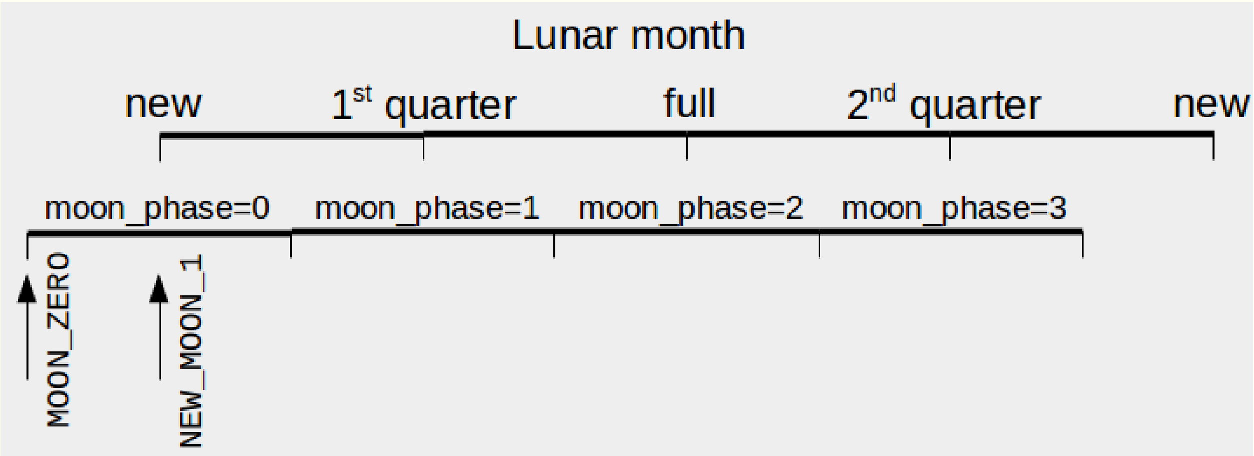


file: moon.py

```
import math

# Mean moon month: 29 days, 12 hours, 43 minutes, 11 seconds
MOON_MONTH = 29*86400 + 12*3600 + 43*60 + 11 # seconds
MOON_QUARTER = MOON_MONTH*0.25 # Duration of a moon quarter

# First new moon after 1/1/1970
NEW_MOON_1 = 660262.0 # seconds
MOON_ZERO = NEW_MOON_1 - MOON_QUARTER*0.5

def moon_phase(nsec): # Moon phase at given time
                    # 0: new, 1:first quarter, 2:full, 3:last quarter
    phase_sec = math.fmod(nsec-MOON_ZERO, MOON_MONTH)
    phase = int(phase_sec/MOON_QUARTER)
    return phase
```

Notes:

- For time related computations we use the standard `time` module. It's base time reference is the number of seconds since 1/1/1970 00:00 (`sec70`, in the following).
- The value `NEW_MOON_1` was found from a table selecting a "convenient" date and converting into `sec70`.
- The value `MOON_ZERO` is set at half a moon quarter before new moon so that we have four interval centered around the start of each quarter (see figure below).



file: tt.py

```
import time

def toseconds(aammgg): # Converts date "yymmdd" into sec70 (at noon)
    year = int(aammgg[:2])+1900
    month = int(aammgg[2:4])
    day = int(aammgg[4:6])
    tt = (year, month, day, 12, 0, 0, 0, 0, 0)
    return time.mktime(tt)
```

Using **numpy** - 3

file: `tt.py`

```
import time

def toseconds(aammgg): # Converts date "yymmdd" into sec70 (at noon)
    year = int(aammgg[:2])+1900
    month = int(aammgg[2:4])
    day = int(aammgg[4:6])
    tt = (year, month, day, 12, 0
    return time.mktime(tt)
```

see: time.mktime

file: `tt.py`

```
import time

def toseconds(aammgg): # Converts date "yymmdd" into sec70 (at noon)
    year = int(aammgg[:2])+1900
    month = int(aammgg[2:4])
    day = int(aammgg[4:6])
    tt = (year, month, day, 12, 0
    return time.mktime(tt)
```

see: time.mktime

... let's go on with `ipython`:

```
In [1]: import tt, moon
In [2]: filename = "dati_nascite.dat"
In [3]: data = np.loadtxt(filename, dtype=np.int_,
   ...: converters={0: tt.toseconds}, unpack=True)
In [4]: cvt_moon = np.vectorize(moon.moon_phase)
In [5]: moonphase = cvt_moon(data[0])

In [6]: m0 = np.array([1 if x==0 else 0 for x in moonphase])
In [7]: m1 = np.array([1 if x==1 else 0 for x in moonphase])
In [8]: m2 = np.array([1 if x==2 else 0 for x in moonphase])
In [9]: m3 = np.array([1 if x==3 else 0 for x in moonphase])

In [10]: births0 = np.dot(m0,data[3])
In [11]: births1 = np.dot(m1,data[3])
In [12]: births2 = np.dot(m2,data[3])
In [13]: births3 = np.dot(m3,data[3])

In [14]: births = births0+births1+births2+births3
In [15]: births_dd = births/len(moonphase)

In [16]: n_births = [births0, births1, births2, births3]
In [17]: n_expect = [np.sum(m0)*births_dd, np.sum(m1)*births_dd,
    ...: np.sum(m2)*births_dd, np.sum(m3)*births_dd]
```

Using numpy - 3

file: tt.py

```
import time

def toseconds(aammgg): # Converts date "yymmdd" into sec70 (at noon)
    year = int(aammgg[:2])+1900
    month = int(aammgg[2:4])
    day = int(aammgg[4:6])
    tt = (year, month, day, 12, 0
    return time.mktime(tt)
```

see: time.mktime

... let's go on with ipython:

```
In [1]: import tt, moon
In [2]: filename = "dati_nascite.dat"
In [3]: data = np.loadtxt(filename, dtype=np.int_,
   ...: converters={0: tt.toseconds}, unpack=True)
In [4]: cvt_moon = np.vectorize(moon.moon_phase)
In [5]: moonphase = cvt_moon(data[0])

In [6]: m0 = np.array([1 if x==0 else 0 for x in moonphase])
In [7]: m1 = np.array([1 if x==1 else 0 for x in moonphase])
In [8]: m2 = np.array([1 if x==2 else 0 for x in moonphase])
In [9]: m3 = np.array([1 if x==3 else 0 for x in moonphase])

In [10]: births0 = np.dot(m0,data[3])
In [11]: births1 = np.dot(m1,data[3])
In [12]: births2 = np.dot(m2,data[3])
In [13]: births3 = np.dot(m3,data[3])

In [14]: births = births0+births1+births2+births3
In [15]: births_dd = births/len(moonphase)

In [16]: n_births = [births0, births1, births2, births3]
In [17]: n_expect = [np.sum(m0)*births_dd, np.sum(m1)*births_dd,
    ...: np.sum(m2)*births_dd, np.sum(m3)*births_dd]
```

see: np.loadtxt
→ converters
→ unpack

Using numpy - 3

file: tt.py

```python
import time

def toseconds(aammgg): # Converts date "yymmdd" into sec70 (at noon)
    year = int(aammgg[:2])+1900
    month = int(aammgg[2:4])
    day = int(aammgg[4:6])
    tt = (year, month, day, 12, 0
    return time.mktime(tt)
```

see: time.mktime

... let's go on with ipython:

```
In [1]: import tt, moon
In [2]: filename = "dati_nascite.dat"
In [3]: data = np.loadtxt(filename, dtype=np.int_,
   ...: converters={0: tt.toseconds}, unpack=True)
In [4]: cvt_moon = np.vectorize(moon.moon_phase)
In [5]: moonphase = cvt_moon(data[0])

In [6]: m0 = np.array([1 if x==0 else 0 for x in moonphase])
In [7]: m1 = np.array([1 if x==1 else 0 for x in moonphase])
In [8]: m2 = np.array([1 if x==2 else 0 for x in moonphase])
In [9]: m3 = np.array([1 if x==3 else 0 for x in moonphase])

In [10]: births0 = np.dot(m0,data[3])
In [11]: births1 = np.dot(m1,data[3])
In [12]: births2 = np.dot(m2,data[3])
In [13]: births3 = np.dot(m3,data[3])

In [14]: births = births0+births1+births2+births3
In [15]: births_dd = births/len(moonphase)

In [16]: n_births = [births0, births1, births2, births3]
In [17]: n_expect = [np.sum(m0)*births_dd, np.sum(m1)*births_dd,
    ...: np.sum(m2)*births_dd, np.sum(m3)*births_dd]
```

see: np.loadtxt
→ converters
→ unpack

see: np.vectorize

Using numpy - 3

file: tt.py

```
import time

def toseconds(aammgg): # Converts date "yymmdd" into sec70 (at noon)
    year = int(aammgg[:2])+1900
    month = int(aammgg[2:4])
    day = int(aammgg[4:6])
    tt = (year, month, day, 12, 0
    return time.mktime(tt)
```

see: time.mktime

... let's go on with ipython:

```
In [1]: import tt, moon
In [2]: filename = "dati_nascite.dat"
In [3]: data = np.loadtxt(filename, dtype=np.int_,
   ...: converters={0: tt.toseconds}, unpack=True)
In [4]: cvt_moon = np.vectorize(moon.moon_phase)
In [5]: moonphase = cvt_moon(data[0])

In [6]: m0 = np.array([1 if x==0 else
In [7]: m1 = np.array([1 if x==1 else
In [8]: m2 = np.array([1 if x==2 else
In [9]: m3 = np.array([1 if x==3 else

In [10]: births0 = np.dot(m0,data[3])
In [11]: births1 = np.dot(m1,data[3])
In [12]: births2 = np.dot(m2,data[3])
In [13]: births3 = np.dot(m3,data[3])

In [14]: births = births0+births1+births2+births3
In [15]: births_dd = births/len(moonphase)

In [16]: n_births = [births0, births1, births2, births3]
In [17]: n_expect = [np.sum(m0)*births_dd, np.sum(m1)*births_dd,
    ...: np.sum(m2)*births_dd, np.sum(m3)*births_dd]
```

see: np.loadtxt
→ converters
→ unpack

see: np.vectorize

m0 has ones in dates when moon_phase is 0, m1 in dates when moon_phase is 1, etc.

file: tt.py

```
import time

def toseconds(aammgg): # Converts date "yymmdd" into sec70 (at noon)
    year = int(aammgg[:2])+1900
    month = int(aammgg[2:4])
    day = int(aammgg[4:6])
    tt = (year, month, day, 12, 0
    return time.mktime(tt)
```

see: time.mktime

... let's go on with ipython:

```
In [1]: import tt, moon
In [2]: filename = "dati_nascite.dat"
In [3]: data = np.loadtxt(filename, dtype=np.int_,
   ...: converters={0: tt.toseconds}, unpack=True)
In [4]: cvt_moon = np.vectorize(moon.moon_phase)
In [5]: moonphase = cvt_moon(data[0])

In [6]: m0 = np.array([1 if x==0 else
In [7]: m1 = np.array([1 if x==1 else
In [8]: m2 = np.array([1 if x==2 else
In [9]: m3 = np.array([1 if x==3 else

In [10]: births0 = np.dot(m0,data[3])
In [11]: births1 = np.dot(m1,data[3
In [12]: births2 = np.dot(m2,data[3
In [13]: births3 = np.dot(m3,data[3

In [14]: births = births0+births1+births2+births3
In [15]: births_dd = births/len(moonphase)

In [16]: n_births = [births0, births1, births2, births3]
In [17]: n_expect = [np.sum(m0)*births_dd, np.sum(m1)*births_dd,
    ...: np.sum(m2)*births_dd, np.sum(m3)*births_dd]
```

see: np.loadtxt
→ converters
→ unpack

see: np.vectorize

m0 has ones in dates when moon_phase is 0, m1 in dates when moon_phase is 1, etc.

births0 is the total number of births in dates when moon_phase is 0, etc.

file: `tt.py`

```
import time

def toseconds(aammgg): # Converts date "yymmdd" into sec70 (at noon)
    year = int(aammgg[:2])+1900
    month = int(aammgg[2:4])
    day = int(aammgg[4:6])
    tt = (year, month, day, 12, 0
    return time.mktime(tt)
```

see: time.mktime

... let's go on with `ipython`:

```
In [1]: import tt, moon
In [2]: filename = "dati_nascite.dat"
In [3]: data = np.loadtxt(filename, dtype=np.int_,
   ...: converters={0: tt.toseconds}, unpack=True)
In [4]: cvt_moon = np.vectorize(moon.moon_phase)
In [5]: moonphase = cvt_moon(data[0])

In [6]: m0 = np.array([1 if x==0 else
In [7]: m1 = np.array([1 if x==1 else
In [8]: m2 = np.array([1 if x==2 else
In [9]: m3 = np.array([1 if x==3 else

In [10]: births0 = np.dot(m0,data[3])
In [11]: births1 = np.dot(m1,data[3
In [12]: births2 = np.dot(m2,data[3
In [13]: births3 = np.dot(m3,data[3

In [14]: births = births0+births1+births2+births3
In [15]: births_dd = births/len(moonphase)

In [16]: n_births = [births0, births1,
In [17]: n_expect = [np.sum(m0)*births
    ...: np.sum(m2)*births_dd, np.sum(n
```

see: np.loadtxt
→ `converters`
→ `unpack`

see: np.vectorize

`m0` has ones in dates when `moon_phase` is 0, `m1` in dates when `moon_phase` is 1, etc.

`births0` is the total number of births in dates when `moon_phase` is 0, etc.

Measured and expected number of births in the four moon phases

file: tt.py

```
import time

def toseconds(aammgg): # Converts date "yymmdd" into sec70 (at noon)
    year = int(aammgg[:2])+1900
    month = int(aammgg[2:4])
    day = int(aammgg[4:6])
    tt = (year, month, day, 12, 0
    return time.mktime(tt)
```

see: time.mktime

... let's go on with ipython:

```
In [1]: import tt, moon
In [2]: filename = "dati_nascite.dat"
In [3]: data = np.loadtxt(filename, dtype=np.int_,
   ...: converters={0: tt.toseconds}, unpack=True)
In [4]: cvt_moon = np.vectorize(moon.moon_phase)
In [5]: moonphase = cvt_moon(data[0])

In [6]: m0 = np.array([1 if x==0 else
In [7]: m1 = np.array([1 if x==1 else
In [8]: m2 = np.array([1 if x==2 else
In [9]: m3 = np.array([1 if x==3 else

In [10]: births0 = np.dot(m0,data[3])
In [11]: births1 = np.dot(m1,data[3
In [12]: births2 = np.dot(m2,data[3
In [13]: births3 = np.dot(m3,data[3

In [14]: births = births0+births1+births2+births3
In [15]: births_dd = births/len(moonphase)

In [16]: n_births = [births0, births1,
In [17]: n_expect = [np.sum(m0)*births
    ...: np.sum(m2)*births_dd, np.sum(n
```

see: np.loadtxt
→ converters
→ unpack

see: np.vectorize

m0 has ones in dates when moon_phase is 0, m1 in dates when moon_phase is 1, etc.

births0 is the total number of births in dates when moon_phase is 0, etc.

Measured and expected number of births in the four moon phases

→

Here are the instructions shown in the previous slide, gathered into a file.

file: births.py

```
import numpy as np
import tt, moon

filename = "dati_nascite.dat"
data = np.loadtxt(filename, dtype=np.int_,
                  converters={0: tt.toseconds}, unpack=True)

cvt_moon = np.vectorize(moon.moon_phase)
moonphase = cvt_moon(data[0])

m0 = np.array([1 if x==0 else 0 for x in moonphase])
m1 = np.array([1 if x==1 else 0 for x in moonphase])
m2 = np.array([1 if x==2 else 0 for x in moonphase])
m3 = np.array([1 if x==3 else 0 for x in moonphase])

births0 = np.dot(m0,data[3])
births1 = np.dot(m1,data[3])
births2 = np.dot(m2,data[3])
births3 = np.dot(m3,data[3])

births = births0+births1+births2+births3

f_births = np.array([births0,births1,births2,births3])/births
f_expect = np.array([np.sum(m0),np.sum(m1),np.sum(m2),np.sum(m3)])/len(moonphase

chi_sq = sum([(x-y)**2/y for (x,y) in zip(f_births, f_expect)])
```

Using **numpy** - 4

Here are the instructions shown in the previous slide, gathered into a file.

file: births.py

```
import numpy as np
import tt, moon

filename = "dati_nascite.dat"
data = np.loadtxt(filename, dtype=np.int_,
                  converters={0: tt.toseconds}, unpack=True)

cvt_moon = np.vectorize(moon.moon_phase)
moonphase = cvt_moon(data[0])

m0 = np.array([1 if x==0 else 0 for x in moonphase])
m1 = np.array([1 if x==1 else 0 for x in moonphase])
m2 = np.array([1 if x==2 else 0 for x in moonphase])
m3 = np.array([1 if x==3 else 0 for x in moonphase])

births0 = np.dot(m0,data[3])
births1 = np.dot(m1,data[3])
births2 = np.dot(m2,data[3])
births3 = np.dot(m3,data[3])

births = births0+births1+bir

f_births = np.array([births0
f_expect = np.array([np.sum(                                en(moonphase

chi_sq = sum([(x-y)**2/y for (x,y) in zip(f_births, f_expect)])
```

We also compute the expected and observed frequencies of births and the χ^2 statistics

Using **numpy** - 4

Here are the instructions shown in the previous slide, gathered into a file.

file: `births.py`

```
import numpy as np
import tt, moon

filename = "dati_nascite.dat"
data = np.loadtxt(filename, dtype=np.int_,
                  converters={0: tt.toseconds}, unpack=True)

cvt_moon = np.vectorize(moon.moon_phase)
moonphase = cvt_moon(data[0])

m0 = np.array([1 if x==0 else 0 for x in moonphase])
m1 = np.array([1 if x==1 else 0 for x in moonphase])
m2 = np.array([1 if x==2 else 0 for x in moonphase])
m3 = np.array([1 if x==3 else 0 for x in moonphase])

births0 = np.dot(m0,data[3])
births1 = np.dot(m1,data[3])
births2 = np.dot(m2,data[3])
births3 = np.dot(m3,data[3])

births = births0+births1+bir

f_births = np.array([births0
f_expect = np.array([np.sum(                                        en(moonphase

chi_sq = sum([(x-y)**2/y for (x,y) in zip(f_births, f_expect)])
```

We also compute the expected and observed frequencies of births and the χ^2 statistics

⇑

The `zip()` function converts two lists: $[a_0, a_1, a_2, ...]$, $[b_0, b_1, b_2, ...]$

into a list of two element tuples:

$[(a_0, b_0), (a_1, b_1), (a_2, b_2), ...]$

Let's proceed with `ipython`:

```
$ ipython --pylab
 ....
In [1]: %run births.py

In [2]: plt.bar(np.arange(0.6, 4.5, 1), f_births, width=0.4, color="blue")

In [3]: plt.bar(np.arange(1, 4.5, 1), f_expect, width=0.4, color="green")
```

Let's proceed with

> The *magic* command `%run` executes the content of file `births.py` in the ipython environment, as if lines were written at the prompt

```
$ ipython --pylab
 ....
In [1]: %run births.py

In [2]: plt.bar(np.arange(0.6, 4.5, 1), f_births, width=0.4, color="blue")

In [3]: plt.bar(np.arange(1, 4.5, 1), f_expect, width=0.4, color="green")
```

Using **numpy** - 5

Let's proceed with

```
$ ipython --pylab
 ....
In [1]: %run births.py

In [2]: plt.bar(np.arange(0.                                    blue")

In [3]: plt.bar(np.arange(1,                                    een")
```

The *magic* command `%run` executes the content of file `births.py` in the ipython environment, as if lines were written at the prompt

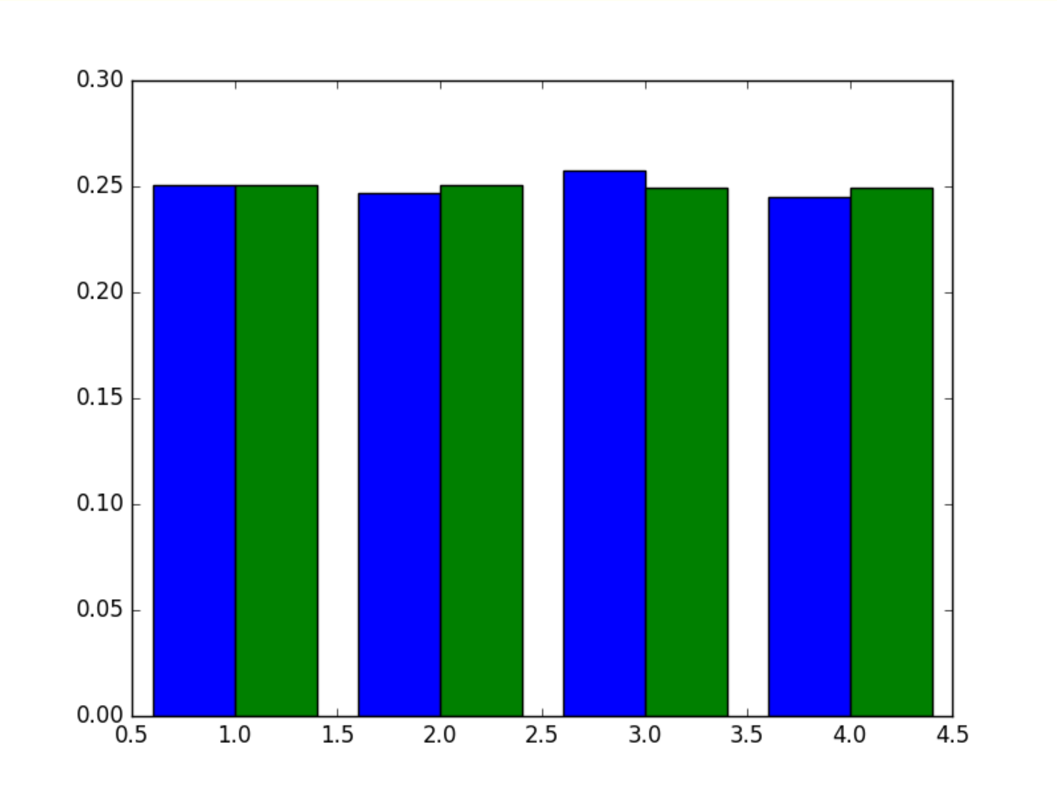
Let's plot observed (blue) and expected (green) frequency of births in each moon phase

Let's proceed with

```
$ ipython --pylab
 ....
In [1]: %run births.py

In [2]: plt.bar(np.arange(0.                blue")

In [3]: plt.bar(np.arange(1,                een")
```

The *magic* command `%run` executes the content of file `births.py` in the ipython environment, as if lines were written at the prompt

Let's plot observed (blue) and expected (green) frequency of births in each moon phase

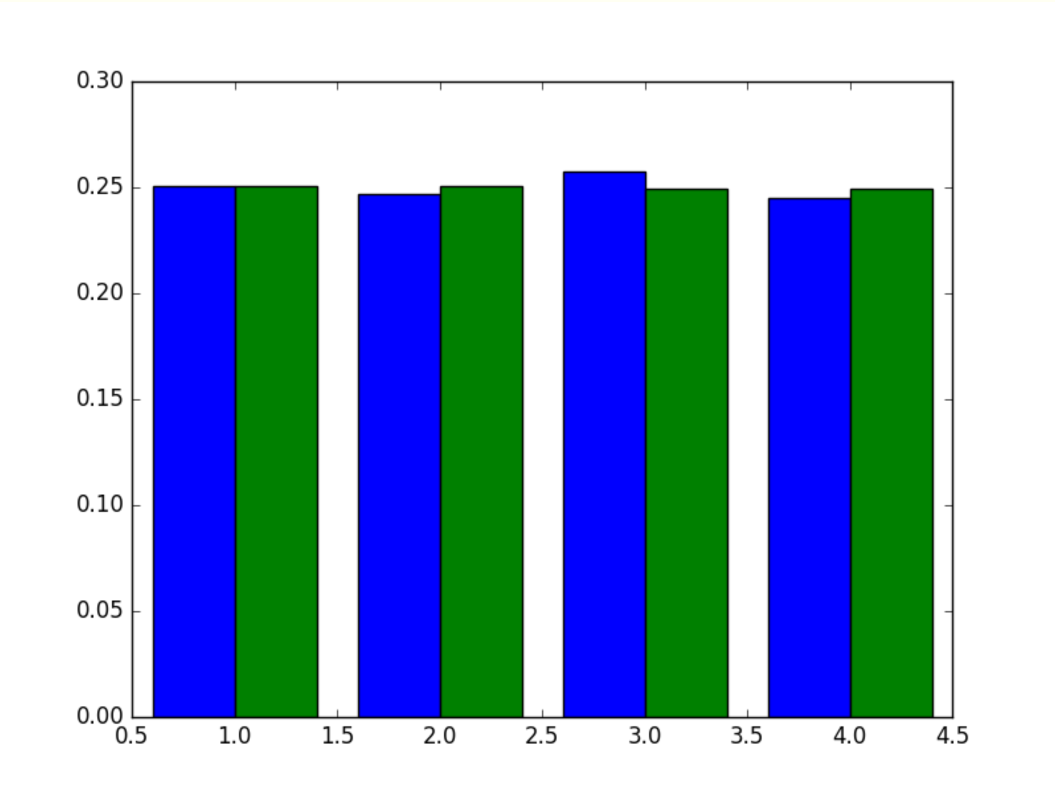


Let's proceed with

```
$ ipython --pylab
 ....
In [1]: %run births.py

In [2]: plt.bar(np.arange(0.                                    blue")

In [3]: plt.bar(np.arange(1,                                    een")
```

The *magic* command `%run` executes the content of file `births.py` in the ipython environment, as if lines were written at the prompt

Let's plot observed (blue) and expected (green) frequency of births in each moon phase



Are differences meaningful?

```
In [5]: chi_sq
Out[5]: 0.00038396010576387837
```

	P										
DF	**0.995**	**0.975**	**0.20**	**0.10**	**0.05**	**0.025**	**0.02**	**0.01**	**0.005**	**0.002**	**0.001**
1	0.0000393	0.000982	1.642	2.706	3.841	5.024	5.412	6.635	7.879	9.550	10.828
2	0.0100	0.0506	3.219	4.605	5.991	7.378	7.824	9.210	10.597	12.429	13.816
3	0.0717	0.216	4.642	6.251	7.815	9.348	9.837	11.345	12.838	14.796	16.266
4	0.207	0.484	5.989	7.779	9.488	11.143	11.668	13.277	14.860	16.924	18.467

matplotlib - 1

matplotlib is a graphic 2D package to be used in `python` programs and particularly suited for the `ipython` environment.

- Generates “publication quality” drawings

matplotlib - 1

matplotlib is a graphic 2D package to be used in `python` programs and particularly suited for the `ipython` environment.

- Generates "publication quality" drawings
- Can be used interactively

`matplotlib` - 1

matplotlib is a graphic 2D package to be used in `python` programs and particularly suited for the `ipython` environment.

- Generates "publication quality" drawings
- Can be used interactively
- Modeled on the well know *matlab* graphic commands

`matplotlib` - 1

matplotlib is a graphic 2D package to be used in `python` programs and particularly suited for the `ipython` environment.

- Generates “publication quality” drawings
- Can be used interactively
- Modeled on the well know *matlab* graphic commands
- Simple to use with default setup

matplotlib - 1

matplotlib is a graphic 2D package to be used in `python` programs and particularly suited for the `ipython` environment.

- Generates “publication quality” drawings
- Can be used interactively
- Modeled on the well know *matlab* graphic commands
- Simple to use with default setup
- Every graphic details can be individually controlled

`matplotlib` - 1

matplotlib is a graphic 2D package to be used in `python` programs and particularly suited for the `ipython` environment.

- Generates "publication quality" drawings
- Can be used interactively
- Modeled on the well know *matlab* graphic commands
- Simple to use with default setup
- Every graphic details can be individually controlled
- Advanced use via OO interface

matplotlib - 1

matplotlib is a graphic 2D package to be used in `python` programs and particularly suited for the `ipython` environment.

- Generates “publication quality” drawings
- Can be used interactively
- Modeled on the well know *matlab* graphic commands
- Simple to use with default setup
- Every graphic details can be individually controlled
- Advanced use via OO interface
- Provided with a 3d toolkit

matplotlib - 1

matplotlib is a graphic 2D package to be used in `python` programs and particularly suited for the `ipython` environment.

- Generates “publication quality” drawings
- Can be used interactively
- Modeled on the well know *matlab* graphic commands
- Simple to use with default setup
- Every graphic details can be individually controlled
- Advanced use via OO interface
- Provided with a 3d toolkit

→

`matplotlib` - 1

matplotlib is a graphic 2D package to be used in `python` programs and particularly suited for the `ipython` environment.

- Generates “publication quality” drawings
- Can be used interactively
- Modeled on the well know *matlab* graphic commands
- Simple to use with default setup
- Every graphic details can be individually controlled
- Advanced use via OO interface
- Provided with a 3d toolkit

→

matplotlib is well suited for ipython:

```
$ ipython --pylab
....
In [1]: %run births.py

In [2]: plt.plot(data[3])

In [3]: x = np.arange(len(data[3]))

In [4]: a,b = np.polyfit(x, data[3], 1)

In [5]: plt.plot((0, x[-1]),(b, a*x[-1]+b),linewidth=2,color="red")

In [6]: plt.title("Births trend")
```

matplotlib is well suited for ipython:

```
$ ipython --pylab
....
In [1]: %run births.py

In [2]: plt.plot(data[3])

In [3]: x = np.arange(len(data[3]))

In [4]: a,b = np.polyfit(x, data[3], 1)

In [5]: plt.plot((0, x[-1]),(b, a*x[-1]+b),linewidth=2,color="red")

In [6]: plt.title("Births trend")
```

Here we plot all the data (with blue line)

matplotlib is well suited for ipython:

```
$ ipython --pylab
....
In [1]: %run births.py

In [2]: plt.plot(data[3])

In [3]: x = np.arange(len(data[3]))

In [4]: a,b = np.polyfit(x, data[3], 1)

In [5]: plt.plot((0, x[-1]),(b, a*x[-1]+b),linewidth=2,color="red")

In [6]: plt.title("Births trend")
```

Here we plot all the data (with blue line)

See: np.polyfit

matplotlib is well suited for ipython:

```
$ ipython --pylab
....
In [1]: %run births.py

In [2]: plt.plot(data[3])

In [3]: x = np.arange(len(data[3]))

In [4]: a,b = np.polyfit(x, data[3], 1)

In [5]: plt.plot((0, x[-1]),(b, a*x[-1]+b),linewidth=2,color="red")

In [6]: plt.title("Births trend")
```

Here we plot all the data (with blue line)

See: np.polyfit

Here we plot the red line

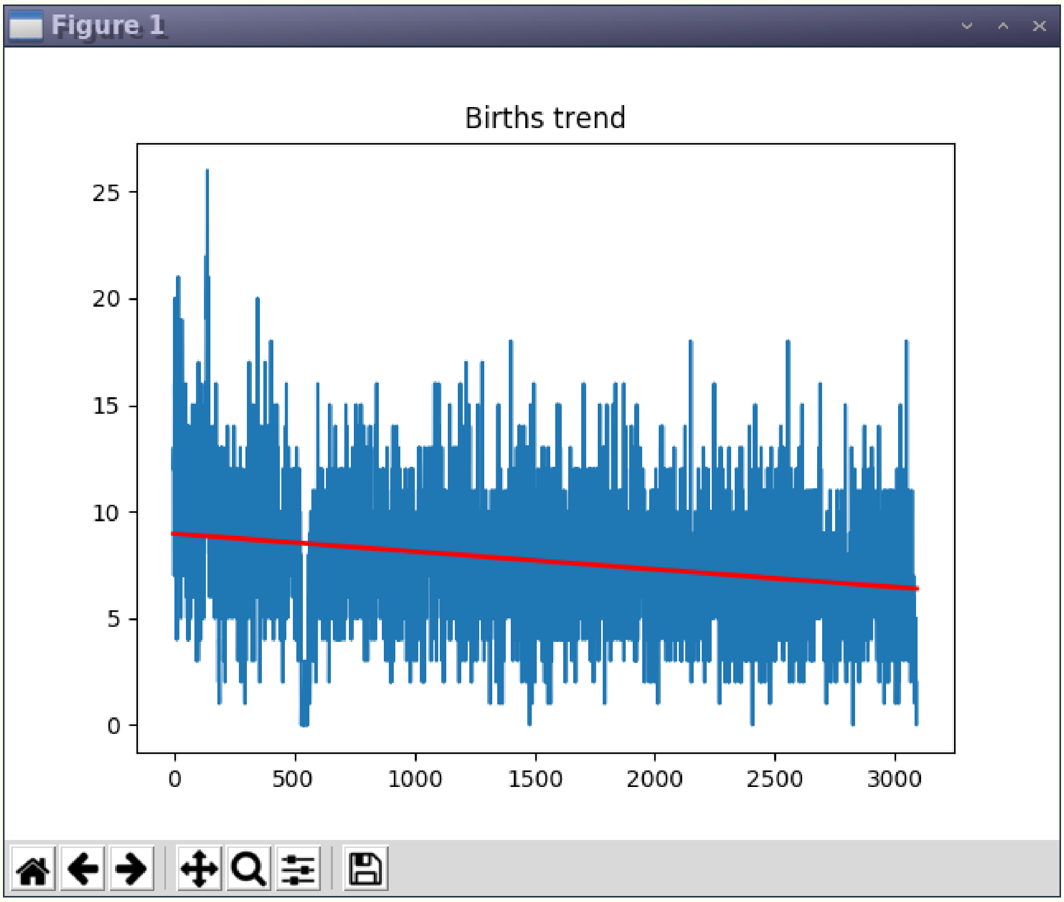
`matplotlib` is well suited for ipython:

```
$ ipython --pylab
....
In [1]: %run births.py

In [2]: plt.plot(data[3])

In [3]: x = np.arange(len(data[3]))

In [4]: a,b = np.polyfit(x, data[3], 1)

In [5]: plt.plot((0, x[-1]),(b, a*x[-1]+b),linewidth=2,color="red")

In [6]: plt.title("Births trend")
```

Here we plot all the data (with blue line)

See: `np.polyfit`

Here we plot the red line



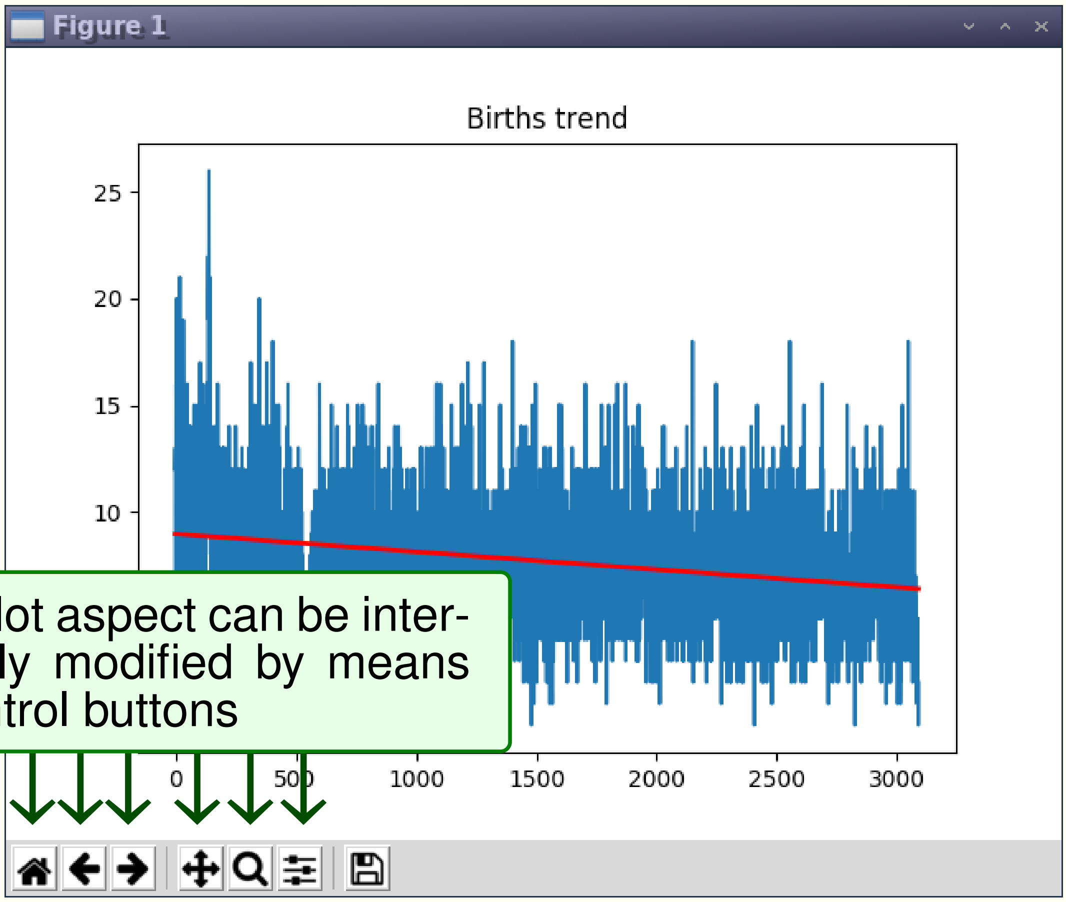
matplotlib is well suited for ipython:

```
$ ipython --pylab
....
In [1]: %run births.py

In [2]: plt.plot(data[3])

In [3]: x = np.arange(len(data[3]))

In [4]: a,b = np.polyfit(x, data[3], 1)

In [5]: plt.plot((0, x[-1]),(b, a*x[-1]+b),linewidth=2,color="red")

In [6]: plt.title("Births trend")
```

Here we plot all the data (with blue line)

See: `np.polyfit`

Here we plot the red line



The plot aspect can be interactively modified by means of control buttons

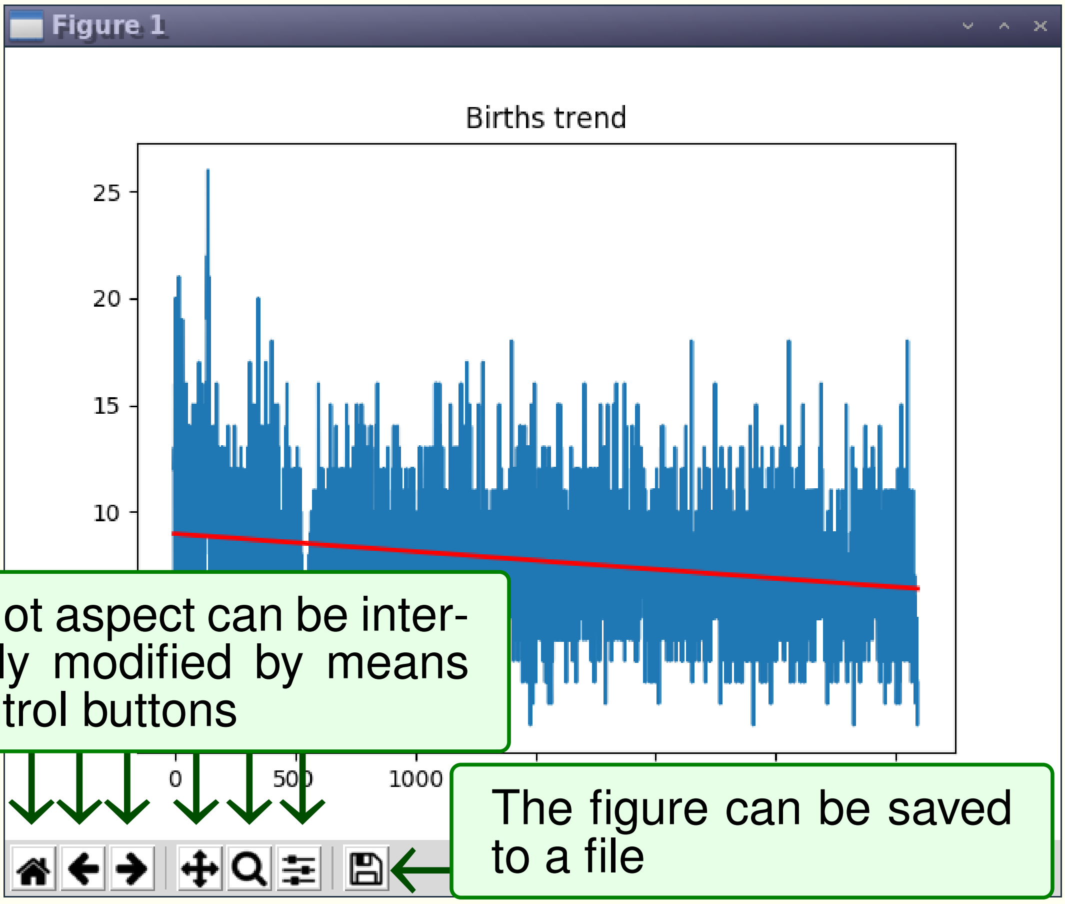
matplotlib is well suited for ipython:

```
$ ipython --pylab
....
In [1]: %run births.py

In [2]: plt.plot(data[3])

In [3]: x = np.arange(len(data[3]))

In [4]: a,b = np.polyfit(x, data[3], 1)

In [5]: plt.plot((0, x[-1]),(b, a*x[-1]+b),linewidth=2,color="red")

In [6]: plt.title("Births trend")
```

Here we plot all the data (with blue line)

See: np.polyfit

Here we plot the red line



The plot aspect can be interactively modified by means of control buttons

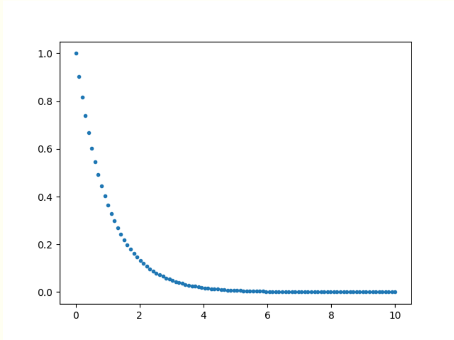
The figure can be saved to a file

Some examples:

Some examples:

Point plotting

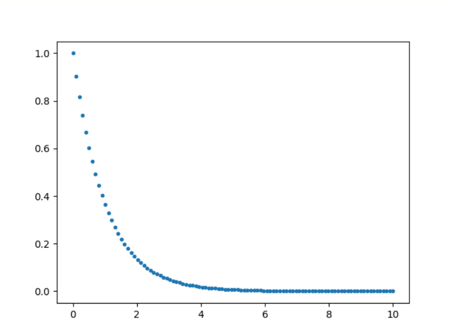
```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> a = np.linspace(0,10,100)
>>> b = np.exp(-a)
>>> plt.plot(a,b,".")
>>> plt.show()
```



Some examples:

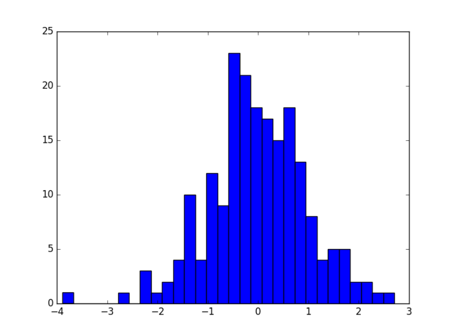
Point plotting

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> a = np.linspace(0,10,100)
>>> b = np.exp(-a)
>>> plt.plot(a,b,".")
>>> plt.show()
```



Histogram

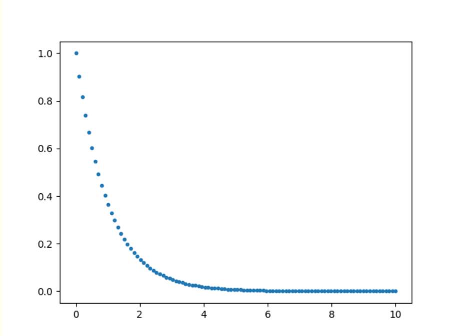
```
>>> from numpy.random import normal
>>> x = normal(size=200)
>>> plt.hist(x,bins=30)
>>> plt.show()
```



Some examples:

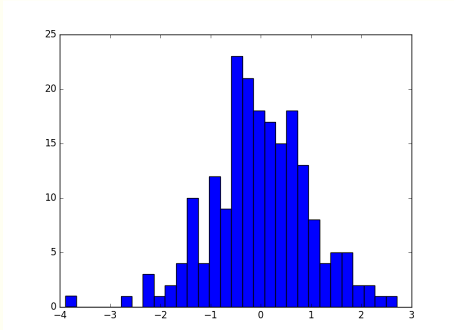
Point plotting

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> a = np.linspace(0,10,100)
>>> b = np.exp(-a)
>>> plt.plot(a,b,".")
>>> plt.show()
```



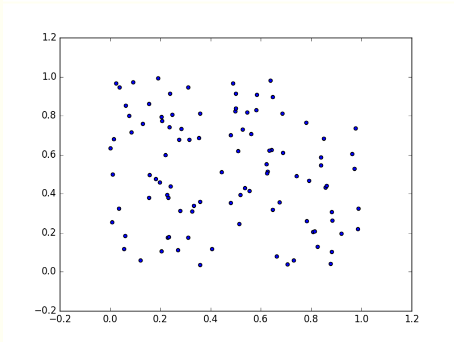
Histogram

```
>>> from numpy.random import normal
>>> x = normal(size=200)
>>> plt.hist(x,bins=30)
>>> plt.show()
```



Scatter plot

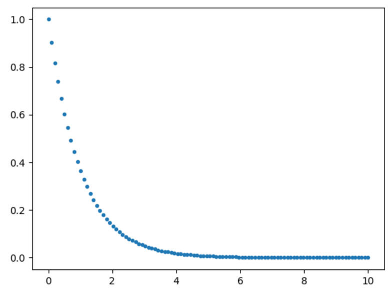
```
>>> from numpy.random import rand
>>> a = rand(100)
>>> b = rand(100)
>>> plt.scatter(a,b)
>>> plt.show()
```



Some examples:

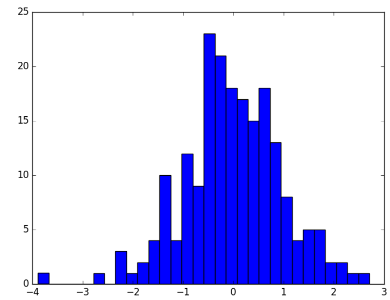
Point plotting

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> a = np.linspace(0,10,100)
>>> b = np.exp(-a)
>>> plt.plot(a,b,".")
>>> plt.show()
```



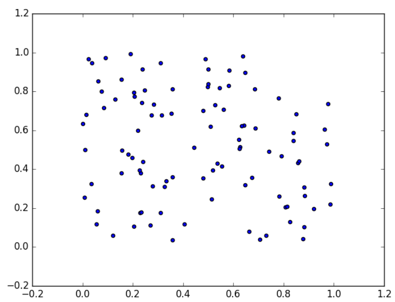
Histogram

```
>>> from numpy.random import normal
>>> x = normal(size=200)
>>> plt.hist(x,bins=30)
>>> plt.show()
```



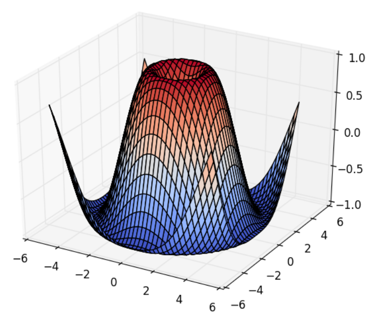
Scatter plot

```
>>> from numpy.random import rand
>>> a = rand(100)
>>> b = rand(100)
>>> plt.scatter(a,b)
>>> plt.show()
```



3D surface

```
>>> from matplotlib import cm
>>> from mpl_toolkits.mplot3d import Axes3D
>>> fig = plt.figure()
>>> ax = fig.gca(projection="3d")
>>> X = np.arange(-5, 5, 0.25)
>>> Y = np.arange(-5, 5, 0.25)
>>> X, Y = np.meshgrid(X, Y)
>>> R = np.sqrt(X**2 + Y**2)
>>> Z = np.sin(R)
>>> surf = ax.plot_surface(X, Y, Z, rstride=1,
                 cstride=1, cmap=cm.coolwarm)
>>> plt.show()
```



We have a map of wind data (speed and direction) around LBTO at Mt. Graham, to be properly plotted.

File: wind_2d.stat:

```
### GRID SIZE X, GRID SIZE Y, START EXIT, END EXIT, EXIT NUM, MIN WIND, MAX WIND
        100      100       4      10     7  1.4624E-02  5.78828
#  I   J    XLAT    XLON     ZS     W.MOD. W.ANG. W.X (UT) W.Y(VT)
   2   2   32.656 -109.941 1912.875 1.7029 62.383 -1.5089 -0.78941
   2   3   32.657 -109.941 1916.125 1.5125 71.936 -1.4380 -0.46900
   2   4   32.658 -109.941 1924.812 1.4617 76.891 -1.4236 -0.33153
   2   5   32.659 -109.941 1935.500 1.4756 85.017 -1.4700 -0.12814
  .......
```

File: plot_wind_2d.py:

```
import numpy as np
import matplotlib.cm as cm
import matplotlib.pyplot as plt

DELTAX=0.1               # Spatial resolution
SYMX=51-1                # Center position, X
SYMY=51-1                # Center position, Y
filename="wind_2d.stat"

data = np.loadtxt(filename, skiprows=3, unpack=True)

ipoints = int(np.max(data[0])-np.min(data[0])+1)
jpoints = int(np.max(data[1])-np.min(data[1])+1)

ZS = data[4].reshape(ipoints, -1).transpose() # Convert into maps
WM = data[5].reshape(ipoints, -1).transpose()
WUT = data[7].reshape(ipoints, -1).transpose()
WVT = data[8].reshape(ipoints, -1).transpose()

arrayI = np.arange(0, ipoints)*DELTAX # Spatial scales
arrayJ = np.arange(0, jpoints)*DELTAX
maxI = np.max(arrayI)
maxJ = np.max(arrayJ)
```

We have a map of wind data (speed and direction) around LBTO at Mt. Graham, to be properly plotted.

File: wind_2d.stat:

```
### GRID SIZE X, GRID SIZE Y, START EXIT, END EXIT, EXIT NUM, MIN WIND, MAX WIND
       100      100      4      10    7  1.4624E-02  5.78828
#  I   J    XLAT    XLON     ZS     W.MOD. W.ANG. W.X (UT) W.Y(VT)
   2   2   32.656 -109.941 1912.875 1.7029 62.383 -1.5089 -0.78941
   2   3   32.657 -109.941 1916.125 1.5125 71.936 -1.4380 -0.46900
   2   4   32.658 -109.941 1924.812 1.4617 76.891 -1.4236 -0.33153
   2   5   32.659 -109.941 1935.500 1.4756 85.017 -1.4700 -0.12814
  .......
```

File: plot_wind_2d.py:

```
import numpy as np
import matplotlib.cm as cm
import matplotlib.pyplot as plt

DELTAX=0.1             # Spatial resolution
SYMX=51-1              # Center position, X
SYMY=51-1              # Center position, Y
filename="wind_2d.stat"

data = np.loadtxt(filename, skiprows=3, unpack=True)

ipoints = int(np.max(data[0])-np.min(data[0])+1)
jpoints = int(np.max(data[1])-np.min(data[1])+1)

ZS = data[4].reshape(ipoints, -1).transpose() # Convert into maps
WM = data[5].reshape(ipoints, -1).transpose()
WUT = data[7].reshape(ipoints, -1).transpose()
WVT = data[8].reshape(ipoints, -1).transpose()

arrayI = np.arange(0, ipoints)*DELTAX # Spatial scales
arrayJ = np.arange(0, jpoints)*DELTAX
maxI = np.max(arrayI)
maxJ = np.max(arrayJ)
```

We have a map of wind data (speed and direction) around LBTO at Mt. Graham, to be properly plotted.

File: wind_2d.stat:

```
### GRID SIZE X, GRID SIZE Y, START EXIT, END EXIT, EXIT NUM, MIN WIND, MAX WIND
        100      100       4      10    7  1.4624E-02  5.78828
#  I   J    XLAT    XLON     ZS      W.MOD.  W.ANG.  W.X (UT) W.Y(VT)
   2   2   32.656 -109.941 1912.875 1.7029 62.383 -1.5089 -0.78941
   2   3   32.657 -109.941 1916.125 1.5125 71.936 -1.4380 -0.46900
   2   4   32.658 -109.941 1924.812 1.4617 76.891 -1.4236 -0.33153
   2   5   32.659 -109.941 1935.500 1.4756 85.017 -1.4700 -0.12814
  .......
```

File: plot_wind_2d.py:

```
import numpy as np
import matplotlib.cm as cm
import matplotlib.pyplot as plt

DELTAX=0.1            # Spatial resol
SYMX=51-1             # Center positi
SYMY=51-1             # Center positi
filename="wind_2d.stat"

data = np.loadtxt(filename, skiprows=3, unpack=True)

ipoints = int(np.max(data[0])-np.min(data[0])+1)
jpoints = int(np.max(data[1])-np.min(data[1])+1)

ZS = data[4].reshape(ipoints, -1).transpose() # Convert into maps
WM = data[5].reshape(ipoints, -1).transpose()
WUT = data[7].reshape(ipoints, -1).transpose()
WVT = data[8].reshape(ipoints, -1).transpose()

arrayI = np.arange(0, ipoints)*DELTAX # Spatial scales
arrayJ = np.arange(0, jpoints)*DELTAX
maxI = np.max(arrayI)
maxJ = np.max(arrayJ)
```

Read data from file. Note:
→ skiprows
→ unpack

We have a map of wind data (speed and direction) around LBTO at Mt. Graham, to be properly plotted.

File: wind_2d.stat:

```
### GRID SIZE X, GRID SIZE Y, START EXIT, END EXIT, EXIT NUM, MIN WIND, MAX WIND
        100      100       4       10     7  1.4624E-02  5.78828
#  I   J    XLAT     XLON      ZS      W.MOD.  W.ANG.  W.X (UT) W.Y(VT)
   2   2   32.656 -109.941 1912.875 1.7029 62.383 -1.5089 -0.78941
   2   3   32.657 -109.941 1916.125 1.5125 71.936 -1.4380 -0.46900
   2   4   32.658 -109.941 1924.812 1.4617 76.891 -1.4236 -0.33153
   2   5   32.659 -109.941 1935.500 1.4756 85.017 -1.4700 -0.12814
  .......
```

File: plot_wind_2d.py:

```
import numpy as np
import matplotlib.cm as cm
import matplotlib.pyplot as plt

DELTAX=0.1              # Spatial resol
SYMX=51-1               # Center positi
SYMY=51-1               # Center positi
filename="wind_2d.stat"

data = np.loadtxt(filename, skiprows=3, unpack=True)

ipoints = int(np.max(data[0])-np.min(data[0])+1)
jpoints = int(np.max(data[1])-np.min(data[1])+1)

ZS = data[4].reshape(ipoints, -1).transpose() # Cor
WM = data[5].reshape(ipoints, -1).transpose()
WUT = data[7].reshape(ipoints, -1).transpose()
WVT = data[8].reshape(ipoints, -1).transpose()

arrayI = np.arange(0, ipoints)*DELTAX # Spatial scales
arrayJ = np.arange(0, jpoints)*DELTAX
maxI = np.max(arrayI)
maxJ = np.max(arrayJ)
```

Read data from file. Note:
→ `skiprows`
→ `unpack`

Convert data columns into 2D arrays

We have a map of wind data (speed and direction) around LBTO at Mt. Graham, to be properly plotted.

File: wind_2d.stat:

```
### GRID SIZE X, GRID SIZE Y, START EXIT, END EXIT, EXIT NUM, MIN WIND, MAX WIND
       100      100       4      10    7  1.4624E-02  5.78828
#  I   J    XLAT    XLON     ZS      W.MOD. W.ANG. W.X (UT) W.Y(VT)
   2   2   32.656 -109.941 1912.875 1.7029 62.383 -1.5089 -0.78941
   2   3   32.657 -109.941 1916.125 1.5125 71.936 -1.4380 -0.46900
   2   4   32.658 -109.941 1924.812 1.4617 76.891 -1.4236 -0.33153
   2   5   32.659 -109.941 1935.500 1.4756 85.017 -1.4700 -0.12814
  .......
```

File: plot_wind_2d.py:

```
import numpy as np
import matplotlib.cm as cm
import matplotlib.pyplot as plt

DELTAX=0.1              # Spatial resol
SYMX=51-1               # Center positi
SYMY=51-1               # Center positi
filename="wind_2d.stat"

data = np.loadtxt(filename, skiprows=3, unpack=True)

ipoints = int(np.max(data[0])-np.min(data[0])+1)
jpoints = int(np.max(data[1])-np.min(data[1])+1)

ZS = data[4].reshape(ipoints, -1).transpose() # Cor
WM = data[5].reshape(ipoints, -1).transpose()
WUT = data[7].reshape(ipoints, -1).transpose()
WVT = data[8].reshape(ipoints, -1).transpose()

arrayI = np.arange(0, ipoints)*DELTAX # Sp
arrayJ = np.arange(0, jpoints)*DELTAX
maxI = np.max(arrayI)
maxJ = np.max(arrayJ)
```

Read data from file. Note:
→ skiprows
→ unpack

Convert data columns into 2D arrays

X and Y axes

File: plot_wind_2d.py - contd.:

```
im = plt.imshow(WM, interpolation='bilinear', origin='lower', # Show image
              extent=[0, maxI, 0, maxJ], cmap=cm.hot_r)

cbar = plt.colorbar(im, orientation='vertical')
cbar.set_label("Wind speed (m/s)")

maxzs = np.max(ZS)
minzs = np.min(ZS)
levels = np.linspace(minzs, maxzs, 25) # Set contour levels

plt.contour(arrayI,arrayJ,ZS,levels,colors='k',origin='lower',linewidths=0.5)

plt.plot([arrayI[SYMX]],[arrayJ[SYMY]],"o",color='black', ms=8)

plt.quiver(arrayI[::5],arrayJ[::5],WUT[::5,::5],WVT[::5,::5], # Arrows
         headwidth=6,headlength=6)

plt.title("Wind speed - hour 007 or hour 000 MST")
plt.xlabel('Km')
plt.ylabel('Km')
plt.tight_layout()
plt.show()
plotfile = 'wind_speed.png'
plt.savefig(plotfile, dpi=200)
print("Created file:", plotfile)
```

matplotlib - 5

Show the image. Note:
- → interpolation
- → origin
- → extent
- → cmap

File: plot_wind_2d.py - contd.:

```
im = plt.imshow(WM, interpolation='bilinear', or
              extent=[0, maxI, 0, maxJ], cmap=cm

cbar = plt.colorbar(im, orientation='vertical')
cbar.set_label("Wind speed (m/s)")

maxzs = np.max(ZS)
minzs = np.min(ZS)
levels = np.linspace(minzs, maxzs, 25) # Set contour levels

plt.contour(arrayI,arrayJ,ZS,levels,colors='k',origin='lower',linewidths=0.5)

plt.plot([arrayI[SYMX]],[arrayJ[SYMY]],"o",color='black', ms=8)

plt.quiver(arrayI[::5],arrayJ[::5],WUT[::5,::5],WVT[::5,::5], # Arrows
         headwidth=6,headlength=6)

plt.title("Wind speed - hour 007 or hour 000 MST")
plt.xlabel('Km')
plt.ylabel('Km')
plt.tight_layout()
plt.show()
plotfile = 'wind_speed.png'
plt.savefig(plotfile, dpi=200)
print("Created file:", plotfile)
```

File: plot_wind_2d.py - contd.:

Show the image. Note:
- → interpolation
- → origin
- → extent
- → cmap

```
im = plt.imshow(WM, interpolation='bilinear', or
              extent=[0, maxI, 0, maxJ], cmap=cm

cbar = plt.colorbar(im, orientation='vertica
cbar.set_label("Wind speed (m/s)")

maxzs = np.max(ZS)
minzs = np.min(ZS)
levels = np.linspace(minzs, maxzs, 25) # Set contour levels

plt.contour(arrayI,arrayJ,ZS,levels,colors='k',origin='lower',linewidths=0.5)

plt.plot([arrayI[SYMX]],[arrayJ[SYMY]],"o",color='black', ms=8)

plt.quiver(arrayI[::5],arrayJ[::5],WUT[::5,::5],WVT[::5,::5], # Arrows
         headwidth=6,headlength=6)

plt.title("Wind speed - hour 007 or hour 000 MST")
plt.xlabel('Km')
plt.ylabel('Km')
plt.tight_layout()
plt.show()
plotfile = 'wind_speed.png'
plt.savefig(plotfile, dpi=200)
print("Created file:", plotfile)
```

Add the colorbar at a side of image

File: plot_wind_2d.py - contd.:

```
im = plt.imshow(WM, interpolation='bilinear', or
              extent=[0, maxI, 0, maxJ], cmap=cm

cbar = plt.colorbar(im, orientation='vertica
cbar.set_label("Wind speed (m/s)")

maxzs = np.max(ZS)
minzs = np.min(ZS)
levels = np.linspace(minzs, maxzs, 25) # Set

plt.contour(arrayI,arrayJ,ZS,levels,colors='k',origin='lower',linewidths=0.5)

plt.plot([arrayI[SYMX]],[arrayJ[SYMY]],"o",color='black', ms=8)

plt.quiver(arrayI[::5],arrayJ[::5],WUT[::5,::5],WVT[::5,::5], # Arrows
         headwidth=6,headlength=6)

plt.title("Wind speed - hour 007 or hour 000 MST")
plt.xlabel('Km')
plt.ylabel('Km')
plt.tight_layout()
plt.show()
plotfile = 'wind_speed.png'
plt.savefig(plotfile, dpi=200)
print("Created file:", plotfile)
```

Show the image. Note:
- → interpolation
- → origin
- → extent
- → cmap

Add the colorbar at a side of image

Set the desired levels for contours

File: plot_wind_2d.py - contd.:

```
im = plt.imshow(WM, interpolation='bilinear', or
             extent=[0, maxI, 0, maxJ], cmap=cm

cbar = plt.colorbar(im, orientation='vertica
cbar.set_label("Wind speed (m/s)")

maxzs = np.max(ZS)
minzs = np.min(ZS)
levels = np.linspace(minzs, maxzs, 25) # Set

plt.contour(arrayI,arrayJ,ZS,levels,colors=                    s=0.5)

plt.plot([arrayI[SYMX]],[arrayJ[SYMY]],"o",color='black', ms=8)

plt.quiver(arrayI[::5],arrayJ[::5],WUT[::5,::5],WVT[::5,::5], # Arrows
         headwidth=6,headlength=6)

plt.title("Wind speed - hour 007 or hour 000 MST")
plt.xlabel('Km')
plt.ylabel('Km')
plt.tight_layout()
plt.show()
plotfile = 'wind_speed.png'
plt.savefig(plotfile, dpi=200)
print("Created file:", plotfile)
```

Show the image. Note:
- → `interpolation`
- → `origin`
- → `extent`
- → `cmap`

Add the colorbar at a side of image

Set the desired levels for contours

Plot the contours

File: plot_wind_2d.py - contd.:

```
im = plt.imshow(WM, interpolation='bilinear', or
             extent=[0, maxI, 0, maxJ], cmap=cm

cbar = plt.colorbar(im, orientation='vertica
cbar.set_label("Wind speed (m/s)")

maxzs = np.max(ZS)
minzs = np.min(ZS)
levels = np.linspace(minzs, maxzs, 25) # Set

plt.contour(arrayI,arrayJ,ZS,levels,colors=                    s=0.5)

plt.plot([arrayI[SYMX]],[arrayJ[SYMY]],"o",color='black', ms=8)

plt.quiver(arrayI[::5],arrayJ[::5],WUT[::5,                    ows
         headwidth=6,headlength=6)

plt.title("Wind speed - hour 007 or hour 000 MST")
plt.xlabel('Km')
plt.ylabel('Km')
plt.tight_layout()
plt.show()
plotfile = 'wind_speed.png'
plt.savefig(plotfile, dpi=200)
print("Created file:", plotfile)
```

Show the image. Note:
→ interpolation
→ origin
→ extent
→ cmap

Add the colorbar at a side of image

Set the desired levels for contours

Plot the contours

Plot the arrows

File: plot_wind_2d.py - contd.:

```
im = plt.imshow(WM, interpolation='bilinear', or
            extent=[0, maxI, 0, maxJ], cmap=cm

cbar = plt.colorbar(im, orientation='vertica
cbar.set_label("Wind speed (m/s)")

maxzs = np.max(ZS)
minzs = np.min(ZS)
levels = np.linspace(minzs, maxzs, 25) # Set

plt.contour(arrayI,arrayJ,ZS,levels,colors=                             s=0.5)

plt.plot([arrayI[SYMX]],[arrayJ[SYMY]],"o",color='black', ms=8)

plt.quiver(arrayI[::5],arrayJ[::5],WUT[::5,                 ws
         headwidth=6,headlength=6)

plt.title("Wind speed - hour 007 or hour 000 MST")
plt.xlabel('Km')
plt.ylabel('Km')
plt.tight_layout()
plt.show()
plotfile = 'wind_speed.png'
plt.savefig(plotfile, dpi=200)
print("Created file:", plotfile)
```

Show the image. Note:
- → interpolation
- → origin
- → extent
- → cmap

Add the colorbar at a side of image

Set the desired levels for contours

Plot the contours

Plot the arrows

Save the image onto a file, with desired quality

File: plot_wind_2d.py - contd.:

```
im = plt.imshow(WM, interpolation='bilinear', or
             extent=[0, maxI, 0, maxJ], cmap=cm

cbar = plt.colorbar(im, orientation='vertica
cbar.set_label("Wind speed (m/s)")

maxzs = np.max(ZS)
minzs = np.min(ZS)
levels = np.linspace(minzs, maxzs, 25) # Set

plt.contour(arrayI,arrayJ,ZS,levels,colors=                    s=0.5)

plt.plot([arrayI[SYMX]],[arrayJ[SYMY]],"o",color='black', ms=8)

plt.quiver(arrayI[::5],arrayJ[::5],WUT[::5,                 ows
         headwidth=6,headlength=6)

plt.title("Wind speed - hour 007 or hour 000 MST")
plt.xlabel('Km')
plt.ylabel('Km')
plt.tight_layout()
plt.show()
plotfile = 'wind_speed.png'
plt.savefig(plotfile, dpi=200)
print("Created file:", plotfile)
```

Show the image. Note:
→ interpolation
→ origin
→ extent
→ cmap

Add the colorbar at a side of image

Set the desired levels for contours

Plot the contours

Plot the arrows

Save the image onto a file, with desired quality