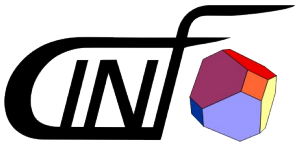
Center for Individual Nanoparticle Functionality

- The scientific Python ecosystem
- numpy array
- Indexing
- ufuncs (elementwise functions)
- Aggregates
- Broadcasting
- Masks and fancy indexing

Python part 2 - numpy

The program

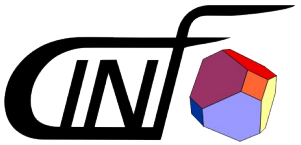


- Move through each of the topics in the outline
- For each topic there will be:
  - A presentation by me
  - A “type along” session, where we type the new things together
  - A few exerciser (the exercises are meant mainly for typing repetition, so most are simple)

- The scientific Python ecosystem
- numpy array
- Indexing
- ufuncs (elementwise functions)
- Aggregates
- Broadcasting
- Masks and fancy indexing

Python part 2 - numpy

The ecosystem for scientific computing in Python



- **numpy** – arrays and math functions
  - polynomials, sin, exp, etc.
- **scipy** – algorithms for arrays
  - fitting, smoothing, searching, FFT, etc..
- **matplotlib** – plotting
- **scitkits** – SciPy toolkits
  - scikit-learn: Machine learning
  - scikit-parse: Sparse matrices

The core

Installing extra packages

- (Possibly your operating system package systems, Linux, Mac)
- Use the tool of your Python package distribution
  - **Anaconda**: conda install pymysql
- For anything not in there, use pip:
  - pip install scikits.sparse

An ecosystem of sevaral sources

- Upside: There is a lot of stuff out there **A LOT**
- Downside: Need to learn a little something about installing extra packages
  - However, most likely all of what you will need is in Anaconda (numpy, scipy and matplotlib)

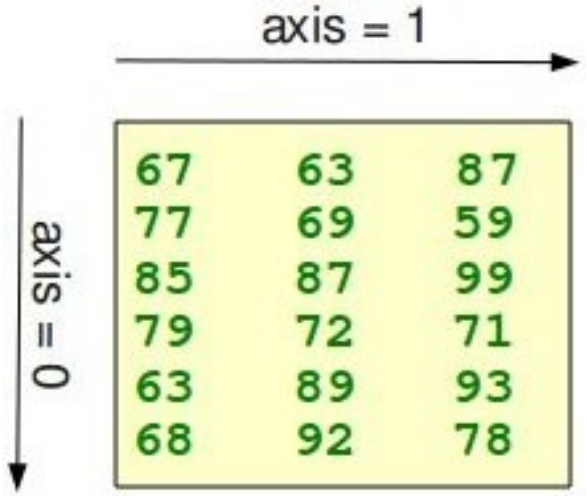
Center for Individual Nanoparticle Functionality

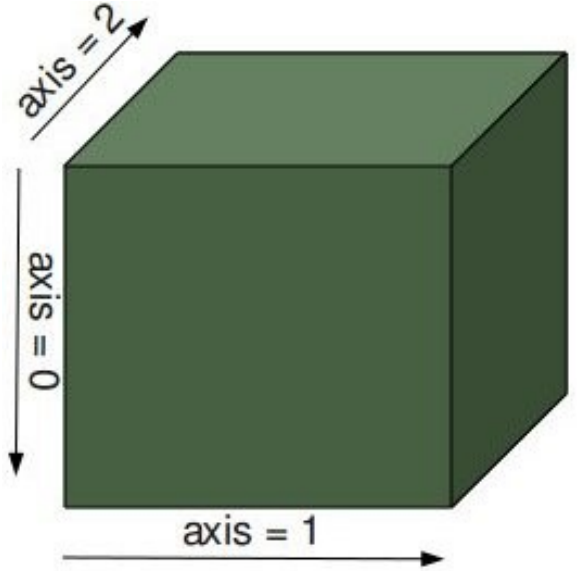
- The scientific Python ecosystem
- **numpy array**
- Indexing
- ufuncs (elementwise functions)
- Aggregates
- Broadcasting
- Masks and fancy indexing

Python part 2 - numpy

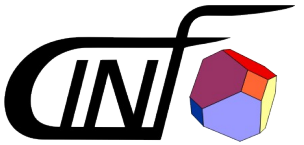
The numpy array

- Multidemonsional array
- Single typed
- Indexing and slicing
- Fancy indexing
- Attached aggregate methods:
  - sum, mean, min, max, etc





Create a **numpy** array



- Call **array()** on a sequence (of sequences)
- Use special functions for initializing arrays with default values:
- **arange(a, b, stepsize):** From a to b in steps of
- **linspace(a, b, n):** From a to b with n elements
- **ones(shape)** and **zeroes(shape):** Array of ones or zeroes with shape
- **empty(shape)**: Like above, but un-initialized
- **random.random(shape):** Random numbers

Creating a numpy array

```
In [1]: import numpy as np

In [2]: myarray = np.array([1, 2, 3])

In [3]: myarray
Out[3]: array([1, 2, 3])

In [4]: myarray = np.array([[1, 2], [3, 4]])

In [5]: myarray
Out[5]:
array([[1, 2],
       [3, 4]])
```

Creating a numpy array

```
In [2]: mylinspace = np.linspace(0.1, 0.7, 3)

In [3]: mylinspace
Out[3]: array([ 0.1,  0.4,  0.7])

In [4]: myones = np.ones((2, 3))

In [5]: myones
Out[5]:
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
```

Information about arrays

- **array.ndim:** Number of dimensions: 2
- **array.shape:** The shape: (2, 3)
- **array.size:** The total number of elements: 6
- **array.dtype:** The data type: float64

Information about an array

```
In [3]: myones.ndim
Out[3]: 2

In [4]: myones.shape
Out[4]: (2, 3)

In [5]: myones.size
Out[5]: 6
```

Reshape array

- **array.reshape(shape):** Reshapes an array to a new size
  - Fast (without copy if possible)
  - New view with different strides
- One dimensions size can be inferred, by settings it to -1

Reshaping

```
In [11]: myrandom = np.random.random(6)

In [12]: myrandom
Out[12]:
array([ 0.62830269,  0.90818518,  0.82203743,  0.56732933,
 0.26770021, 0.48429263])

In [13]: myrandomarray = myrandom.reshape((2, 3))

In [14]: myrandomarray
Out[14]:
array([[ 0.62830269,  0.90818518,  0.82203743],
       [ 0.56732933,  0.26770021,  0.48429263]])
```

Math with arrays

- All the normal math operators works elementwise
  - **+ - * ** / (careful)**
- Can also use scalars and arrays (elementwise)
  - array([1.0, 2.0]) * 2

Math with arrays

```
In [46]: a = np.arange(1, 2, 0.2)

In [47]: b = np.arange(2, 3, 0.2)

In [48]: a
Out[48]: array([ 1. ,  1.2,  1.4,  1.6,  1.8])

In [49]: b
Out[49]: array([ 2. ,  2.2,  2.4,  2.6,  2.8])

In [50]: a + b
Out[50]: array([ 3. ,  3.4,  3.8,  4.2,  4.6])

In [51]: a * b
Out[51]: array([ 2.  ,  2.64,  3.36,  4.16,  5.04])
```

array – Type along

Type along session in the terminal:
array

Types - Exercises

- Create with arange, linspace and random.random
- Reshape an array created with random.random
- Get information (ndim, shape, size, dtype) of that array
- Try simpel math on the arrays

Center for Individual Nanoparticle Functionality

- The scientific Python ecosystem
- numpy array
- **Indexing**
- ufuncs (elementwise functions)
- Aggregates
- Broadcasting
- Masks and fancy indexing

Python part 2 - numpy

Indexing arrays

- Work as with lists, but with super powers
- Single integer index [1]
- Slicing [2: 4]
- Slicing without numbers to get everythin [:4]
- Negative indexing [-1]
- But now for each dimension
- The indeces for each dimension is separated by comma: [3, 4] and [:, 0]

Indexing arrays

```
In [2]: a = np.arange(6).reshape((2, 3))

In [3]: a
Out[3]:
array([[0, 1, 2],
       [3, 4, 5]])

In [4]: a[1, 1]
Out[4]: 4

In [5]: a[1, :]
Out[5]: array([3, 4, 5])

In [6]: a[:, 1]
Out[6]: array([1, 4])

In [7]: a[:, -1]
Out[7]: array([2, 5])
```

indexing and slicing – Type along

Type along session in the terminal:
indexing and slicing

Types - Exercises

- Create a 2d array and on only one dimension:
  - index
  - slice with 2, 1 or 0 arguments
  - Use negative indeces
- The try on both dimension the same and notice that you can combine any way you want

Center for Individual Nanoparticle Functionality

- The scientific Python ecosystem
- numpy array
- Indexing
- **ufuncs (elementwise functions)**
- Aggregates
- Broadcasting
- Masks and fancy indexing

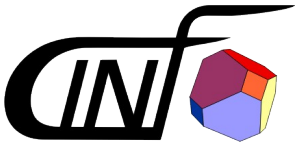
Python part 2 - numpy

ufuncs

- Short for **universal function**
- Can work on multidimensional arrays in an element-by-element fashion
- *Are vectorized forms* of functions that can take and return scalar input and output
- All the usual suspects:
  - exp, sqrt, reciprocal, sin, cos, tan, arcsin .., sinh ..,
  - and many many more:

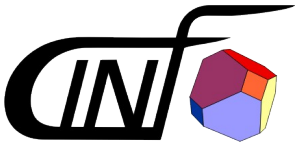
http://docs.scipy.org/doc/numpy/reference/ufuncs.html#available-ufuncs

ufuncs



- All the ordinary math operations ...
- … and all the ordinary comparison operators ...
- **… are ufuncs too**
- (I just didn't tell you before)

Using ufuncs



```
In [1]: import numpy as np

In [2]: import matplotlib.pyplot as plt

In [3]: x = np.linspace(0, np.pi*2, 1000)

In [4]: y = np.exp(np.sin(x) + 1)

In [5]: plt.plot(x, y)
Out[5]: [<matplotlib.lines.Line2D at 0x7fcb7a5bfd10>]

In [6]: plt.show()
```

ufuncs – Type along

Type along session in the terminal:
ufuncs

ufuncs - Exercises

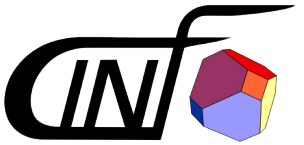
- Create x-values with arange
- Create y-values from different (more complicated) function expressions using a combination of ufuncs and ordinary math operators
- Plot as you experiment

Center for Individual Nanoparticle Functionality

- The scientific Python ecosystem
- numpy array
- Indexing
- ufuncs (elementwise functions)
- **Aggregates**
- Broadcasting
- Masks and fancy indexing

Python part 2 - numpy

Aggregates



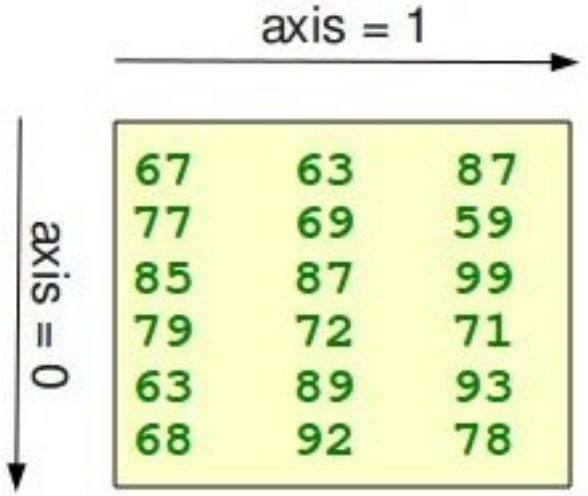
- **Aggregates** are functions on arrays that return smaller arrays (reduces them)
- There are many aggregate functions:
  - min, max, sum, prod, mean, std, var, any, all, median, percentile
  - argmin, argmax (Returns indeces instead of values)
- Variants that ignore NaN
  - nanmin, nanmax etc.
- Some are available on the arrays, some in numpy
  - np.percentile

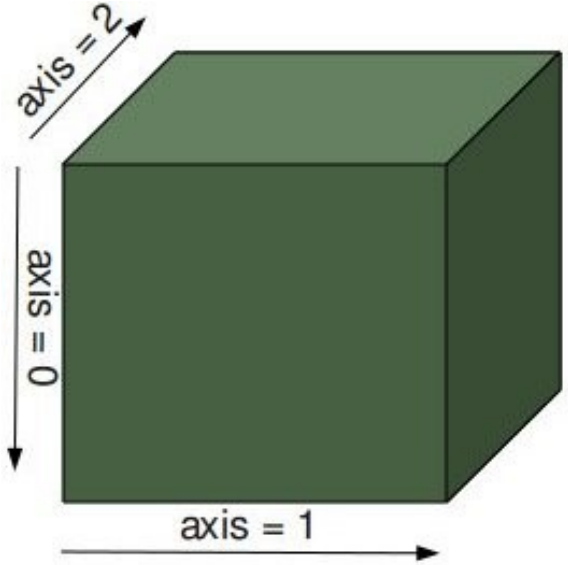
Aggregates

```
In [25]: a = np.arange(6).reshape(2, 3)

In [26]: a
Out[26]:
array([[0, 1, 2],
       [3, 4, 5]])

In [27]: a.sum()
Out[27]: 15

In [28]: a.min()
Out[28]: 0
```

Aggregates axis argument

- Remember the array axis
- Aggregate functions **all** takes an optional *axis* argument
  - array.sum(axis=0)
- Will do the operation **along that axis**





Aggregates with axis

```
In [34]: a
Out[34]:
array([[0, 1, 2],
       [3, 4, 5]])

In [35]: a.sum(axis=0)
Out[35]: array([3, 5, 7])

In [36]: a.sum(axis=1)
Out[36]: array([ 3, 12])

In [37]: a.mean(axis=0)
Out[37]: array([ 1.5,  2.5,  3.5])
```

aggregates – Type along

Type along session in the terminal:
aggregates

aggregates - Exercises

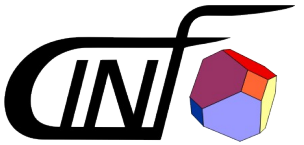
- Make a 2d (or 3d) array
- Test several of the aggegates like, min, sum, prod etc. on this array
- Experiment with the axis argument

Center for Individual Nanoparticle Functionality

- The scientific Python ecosystem
- numpy array
- Indexing
- ufuncs (elementwise functions)
- Aggregates
- **Broadcasting**
- Masks and fancy indexing

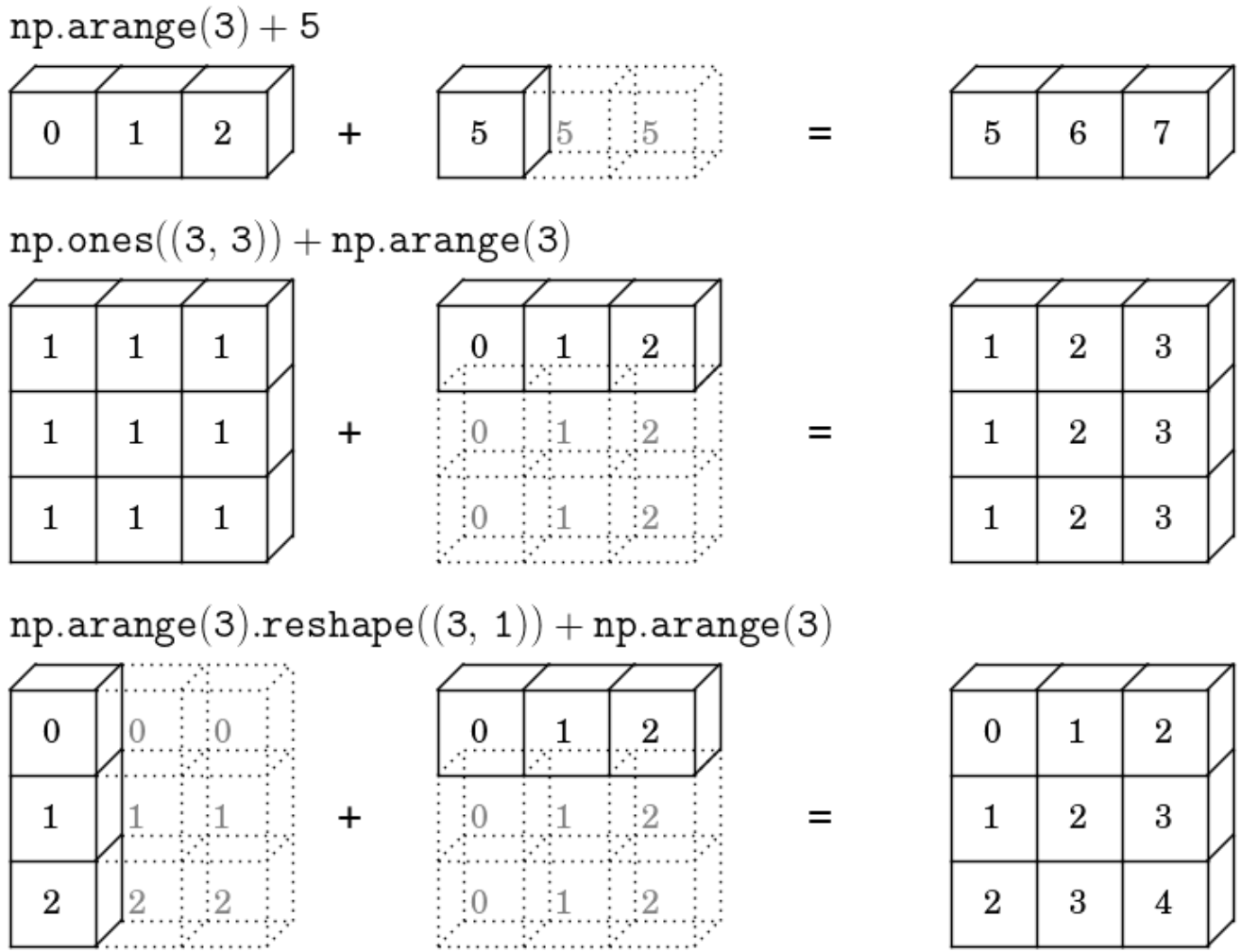
Python part 2 - numpy

Broadcasting



- A set of rules that makes ufuncs operate on arrays of different sizes and/or dimensions
- One of the more advanced features
- Can be used to make to
  - Repeat a smaller array in calculations with larger
  - To make certain **outer** operations on arrays
- Makes it possible to repeat parts of an array without copying

Broadcasting



http://studiosmile.pl/master-invalid-floating-point-operation.htm
https://speakerdeck.com/jakevdp/losing-your-loops-fast-numerical-computing-with-numpy-pycon-2015

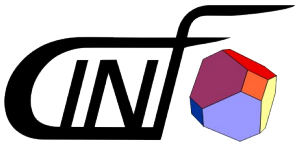
The rules of broadcasting

1) If number of dimension differ, left pad the smaller shape with 1s
2) If the dimensions do not match, then broadcast to the dimension with size=1
3) If neither of the non-matching dimentions are 1, then raise an error

broadcasting – Type along

Type along session in the terminal:
broadcasting

More on nearest neighboors



- For a more complete 3d example, see this presentation:
- https://speakerdeck.com/jakevdp/losing-your-loops-fast-numerical-computing-with-numpy-pycon-2015
- By Jake VanderPlas

broadcast - Exercises

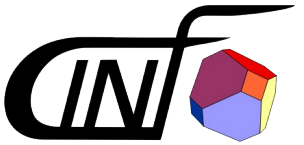
- While making sure that you understand the broadcasting rules thet governs each of these examples:
  - Experiment with doing math between scalars and 1s or 2d arrays
  - Experiment with math between 1d and 2d arrays
  - Experiment with an outer operation between 2 1d arrays

Center for Individual Nanoparticle Functionality

- The scientific Python ecosystem
- numpy array
- Indexing
- ufuncs (elementwise functions)
- Aggregates
- Broadcasting
- **Masks and fancy indexing**

Python part 2 - numpy

Masking and fancy indexing



- Arrays can be indexed by ints and slices
- An optional third int in a slice gives the step
  - myarray[::2] (every other)
  - myarray[::-1] (reverses)
- But they can also be indexed by sequences
  - myarray[[1, 3, 5]]
- And with a boolean arrays, to select only elements

fancy indexing – Type along

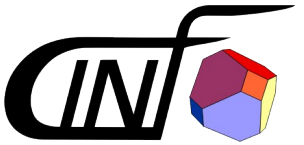
Type along session in the terminal:
fancy indexing

fancy indeces - Exercises

- Reverse the order of the rows in a 2d-array
- Set all values in a random array between 0.2 and 0.8 to infinity
- Select a list of rows from an array

On Python and for loops

For loops over arrays are slow



- Python is fast to develop in, because it is dynamic, dynamically typed and high level
- Python is slow for repeated simple tasks because it is dynamic, dynamically typed and high level
- With the tools we learned today, loops can be pushed into the compiled layer, which is very fast
- Numerical problems in Python must be vectorized!!!

- Numerical problems in Python must be vectorized!!!
- If for-loops over arrays cannot be avoided, certain techniques can be used to speed them up
- Out of scope for this session

Center for Individual Nanoparticle Functionality

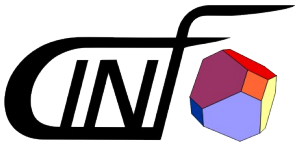
- **Summing up**

Python part 2 - numpy

Summary

- Express the complete vectorized problem with:
  - Indexing and slicing
  - ufuncs
  - Aggregates
  - Broadcasting
  - (Fancy indexing)
- Numpy has a wide range of general purpose ufuncs for many purposes (look for them)
- Have func with numpy

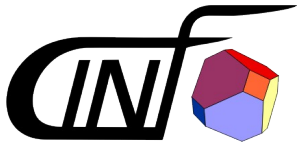
References and more material



- “Loosing your Loops: Fast Numerical Computing with Numpy” Jake VanderPlas,

  https://www.youtube.com/watch?v=EEUXKG97YRw
  https://speakerdeck.com/jakevdp/losing-your-loops-fast-numerical-computing-with-numpy-pycon-2015
- “100 numpy exercises”

  http://www.labri.fr/perso/nrougier/teaching/numpy.100/
- Documents on http://www.numpy.org/ including “Numpy for Matlab® Users”
- And much more, use google



Hope you had fun.
See you for the rest of part 2 tomorrow.

scipy and matplotlib

THE END