Introduction to Python and ASE

Ask Hjorth Larsen
asklarsen@gmail.com

Nano-bio Spectroscopy Group
and ETSF Scientific Development Centre
Universidad del País Vasco UPV/EHU

May 29, 2017

This minicourse

Python

- Learn basic scripting
- Intro to scientific libraries: NumPy, SciPy, Matplotlib, ...

ASE — the Atomic Simulation Environment

- Fully scriptable tools and workflows for atomistic simulations
- Written in Python

GPAW

- DFT code written in Python and C using ASE
- Calculations are Python scripts; no “input files”

Python

"The only way to learn a new programming language is by writing programs in it. The first program to write is the same for all languages" — Kernighan & Ritchie, "Programming in C"

Write your first Python program

- Open a file, say, `hello.py`, in your favourite editor
- Type: `print('hello, world!')`
- Save the file.
- Run: `python3 hello.py`

(In Python2: `print 'hello, world!'`,
but as of Python3, `print` is a function!)

The interactive interpreter

```
askhl@jormungandr:~$ python
Python 2.7.9 (default, Jun 29 2016, 13:08:31)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more in
>>> print('hello, world!')
hello, world!
>>> 2 + 2
4
>>> [x**2 for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
>>>
```

- Use the interactive interpreter to play around and try stuff.
- Interactive interpreter is the best "pocket calculator".

Basics of Python

Language features

- General-purpose language suitable for scripting and rapid application development
- No type declarations — dynamic typing
- Memory management, bounds checks, ...

"Boundary conditions"

- Standard implementation of Python is CPython, written in C
- Python programs are run by the Python interpreter
- Other Pythons: Jython, IronPython, PyPy
- First version from 1991. Ongoing transition from Python2 to Python3

Python basics

```
>>> a = 3
>>> a * 7
21
>>> x = [1, 2, a, 'hello']
>>> x.append(7)
>>> x
[1, 2, 3, 'hello', 7]
>>> x[2]  # Access an element
3
>>> x[1] = 17  # Set an element
>>> x
[1, 17, 3, 'hello', 7]
>>> x[-1]  # Negative indices count from the end
7
```

Common built-in types

- Numeric types: `5`, `5.0`, `1j`, `True`, `False`
- `str`: `'hello, world!'`
- `list`: `[x, y, z]`
- `tuple`: `(x, y, z)` (like list, but cannot be modified)
- `dict`: maps objects to objects;
  `{'banana': 'yellow', 'apple': 'red', 7: [17, 42]}`
- `None` — value which represents lack of a value

Control structures

```
# Implementation of Danish drinking game
for i in range(1, 101):
    txt = str(i)
    if i == 5 + 7:
        print('fum bum sum')
    elif i % (5 * 7) == 0:
        print('fum bum multiplum')
    elif '5' in txt and '7' in txt:
        print('fum bum')
    elif '5' in txt:
        print('fum')
    elif '7' in txt:
        print('bum')
    else:
        print(txt)
```

Output:

```
1
2
3
4
fum
6
bum
8
9
10
11
fum bum sum
13
14
fum
```

Indentation controls scope! Indent using four spaces.

"Pythonic" or not

```
symbols = ['H', 'He', 'Li' , 'Be', 'B', 'C']
names = ['hydrogen', 'helium', 'lithium',
         'beryllium', 'boron', 'carbon']

for i in range(len(symbols)):  # C/Fortran-style
    print(symbols[i])

for sym in symbols:  # Python-style
    print(sym)

for i, sym in enumerate(symbols):
    print('Element {} is {}'.format(i + 1, sym))

for sym, name in zip(symbols, names):
    print('{} is {}'.format(sym, name))
```

- C: Loop over a number and use the number to index the list.
- Python: Loop over the list.

File I/O

```
# Print to a file:
fd = open('goethe.txt', 'w')
fd.write('Wer reitet so spät\n'
         ' durch Nacht und Wind?\n')
fd.write('Es ist der Vater\n'
         ' mit seinem Kind.\n')

# Read lines from a file:
fd = open('goethe.txt')
for line in fd:
    print(line, end='')

# Or read everything at once:
text = open('goethe.txt').read()
```

"Batteries included"

The Python standard library

- `math` mathematical functions, `math.sin(2.0 * math.pi)`
- `os`, `sys` interact with OS or system; `os.system('ls -l')`
- `subprocess` run and talk to subprocesses
- `shutil` work with files (copy, etc.)
- `pickle` serialization — read and write arbitrary objects
- `re` regular expressions
- `glob`, `fnmatch` expand filenames; `glob('data/*.txt')`
- `argparse` parse command-line arguments
- `urllib` open web pages

and much more.

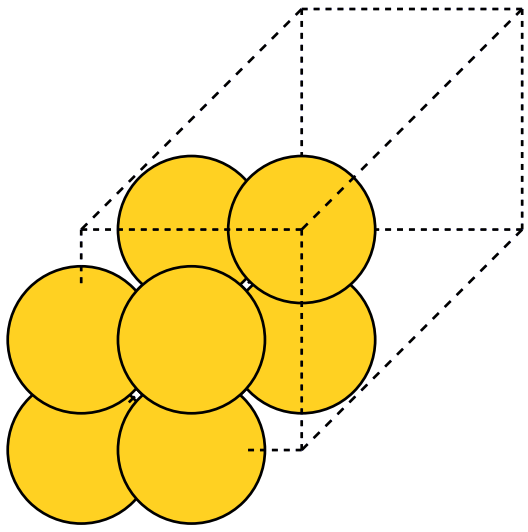
```
import math

def gauss(a, x):
    return math.exp(-0.5 * (x / a)**2)

class Gaussian:
    def __init__(self, a):
        self.a = a

    def calculate(self, x):
        return math.exp(-0.5 * (x / self.a)**2)

def main():
    print(gauss(5.0, 0.3))
    g = Gaussian(5.0)
    print(g.calculate(0.3))
    calc = g.calculate
    print(calc(0.3))

main()
```

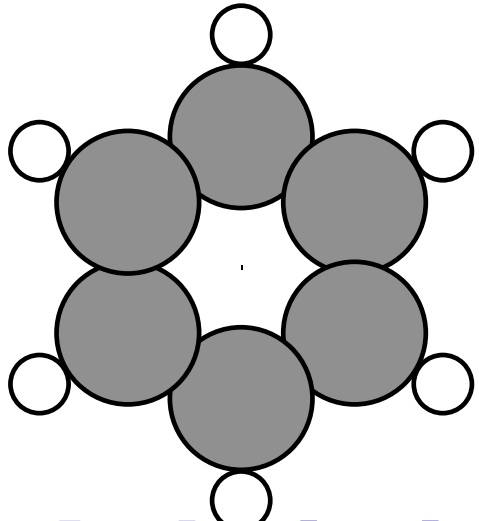
ASE

- Started as an object-oriented Python interface to the old ultrasoft pseudopotential planewave code Dacapo
- S.R. Bahn, K.W. Jacobsen, "An object-oriented scripting interface to a legacy electronic structure code". Computing in Science & Engineering, 4(3):56–66, 2002.
- BDFL: Jens Jørgen Mortensen, DTU Physics
- Very large number of contributors
- Now has interfaces to many codes, and many tools.
- New reference paper: A.H. Larsen, J.J. Mortensen *et al.*, "The Atomic Simulation Environment – A Python library for working with atoms": J. Phys. Cond. Matt. (Available as Psi-k Highlight of the Month, January 2017)

Build and view structures

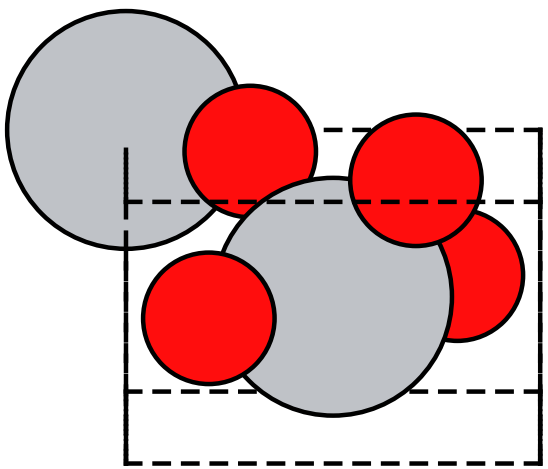
```
from ase import Atoms
from ase.visualize import view

a = 2.04
gold = Atoms('Au', pbc=True,
             cell=[[0, a, a],
                   [a, 0, a],
                   [a, a, 0]])
print(gold)
view(gold.repeat((2, 2, 2)))

from ase.build import molecule
view(molecule('C6H6'))
```





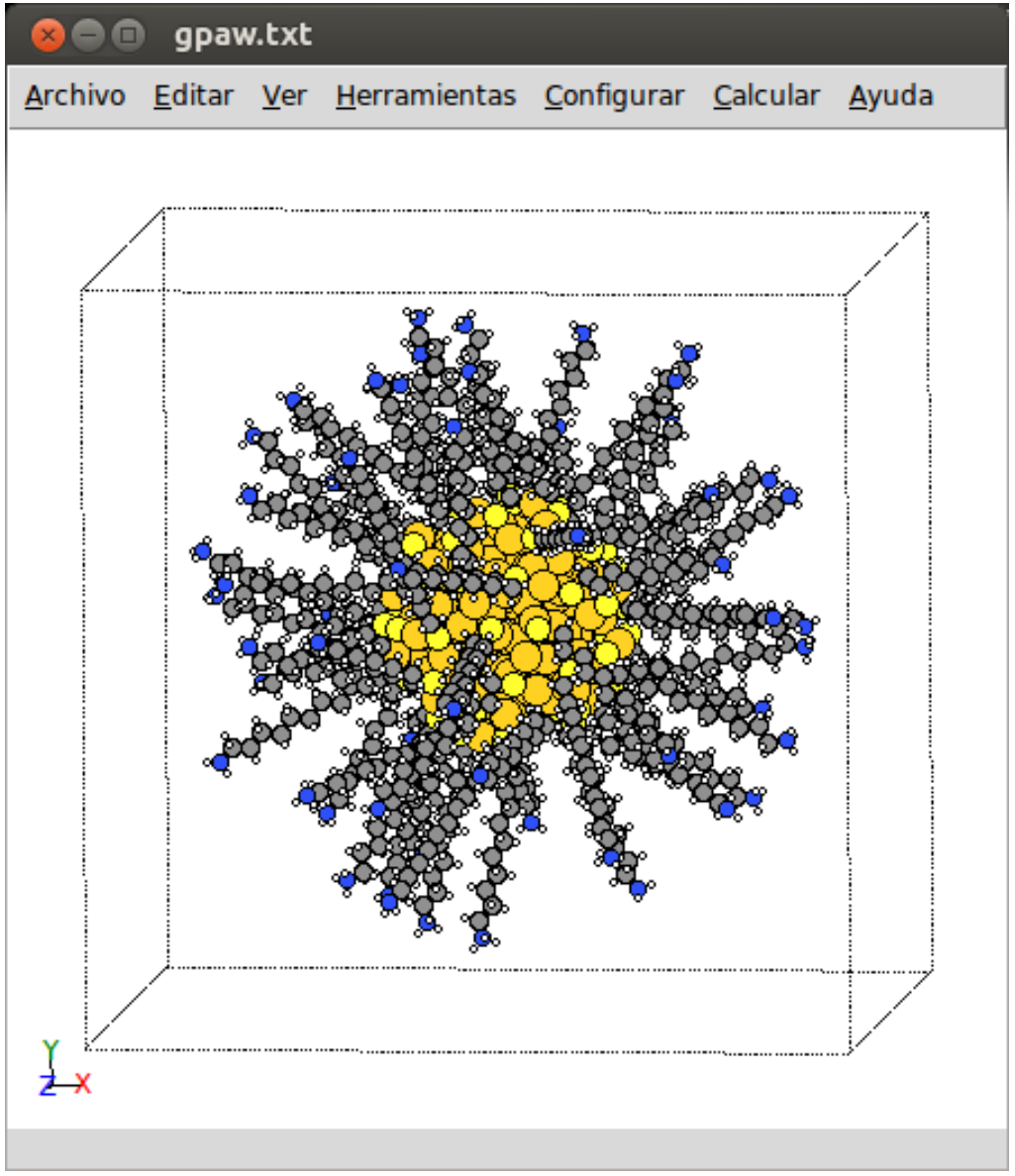
Example: Bulk rutile

```
from ase.lattice.spacegroup import crystal
a = 4.6
c = 2.95
rutile = crystal(['Ti', 'O'],
                 basis=[(0, 0, 0),
                        (0.3, 0.3, 0.0)],
                 spacegroup=136,
                 cellpar=[a, a, c, 90, 90, 90])
```



Try the ASE GUI

- Run `ase gui` (previously: `ase-gui`)
- Build nanoparticle or something else
- Select, move atoms (Ctrl+M)
- Save to your favourite format



Main features

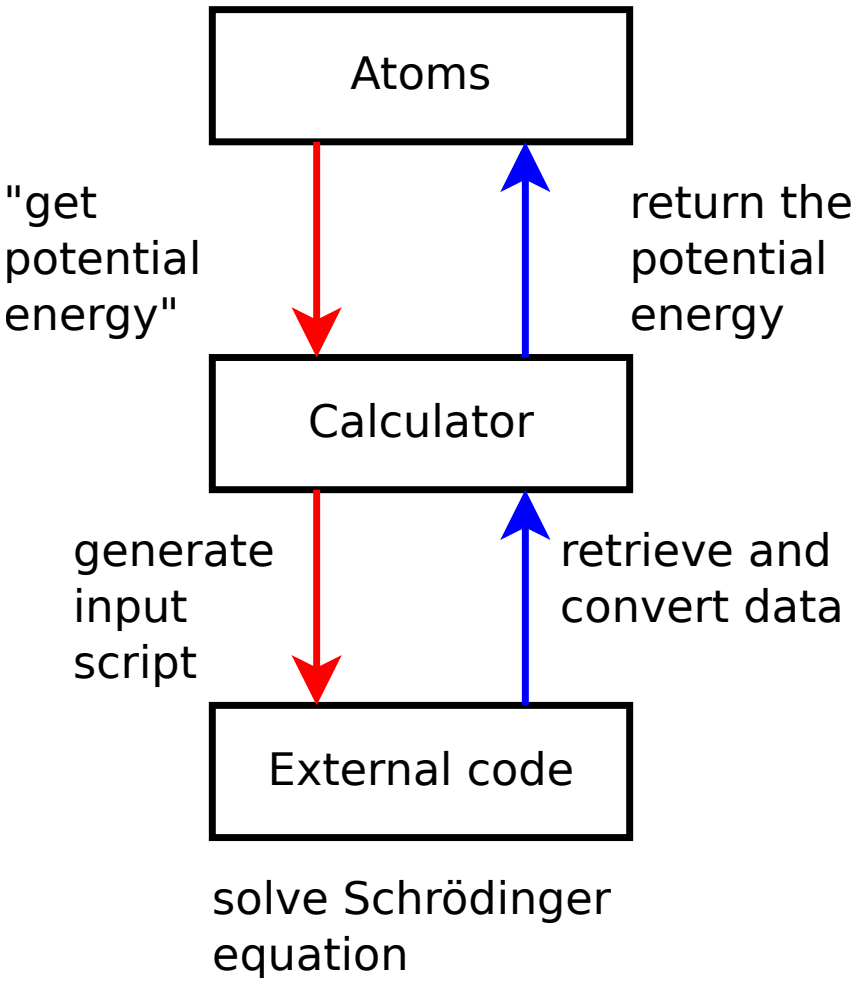
- The `Atoms` object
- Set up molecules, crystals, surfaces and more using provided modules augmented by scripting
- Use GUI to visualize structures
- Read and write many file formats (xyz, cube, xsf, cif, pdb, ...)
- Call external codes from Python using the ASE Calculator interface

Calculator

- A calculator can take `Atoms` as input and produce energies and forces as output
- Most calculators call an external DFT code
- Some calculators: EMT, GPAW, NWChem, Abinit, VASP
- There are 30+ calculators.

Structure optimization

```
from ase import Atoms
from ase.optimize import BFGS
from gpaw import GPAW

system = Atoms('H2O', positions=[[-1, 0, 0],
                                 [1, 0, 0],
                                 [0, 0, 1]])
system.center(vacuum=3.0)
system.calc = GPAW(mode='lcao', basis='dzp')

opt = BFGS(system,
           trajectory='opt.traj',
           logfile='opt.log')
opt.run(fmax=0.05)
```



Interface through file I/O

- ASE creates inputfile, runs programme (see figure)

Calculator daemon

- Calculator runs in background
- Read/write using sockets

Direct linking

- Everything within one process $\rightarrow$ efficient *and* nice
- Also rather complicated

Calculators

Basic properties

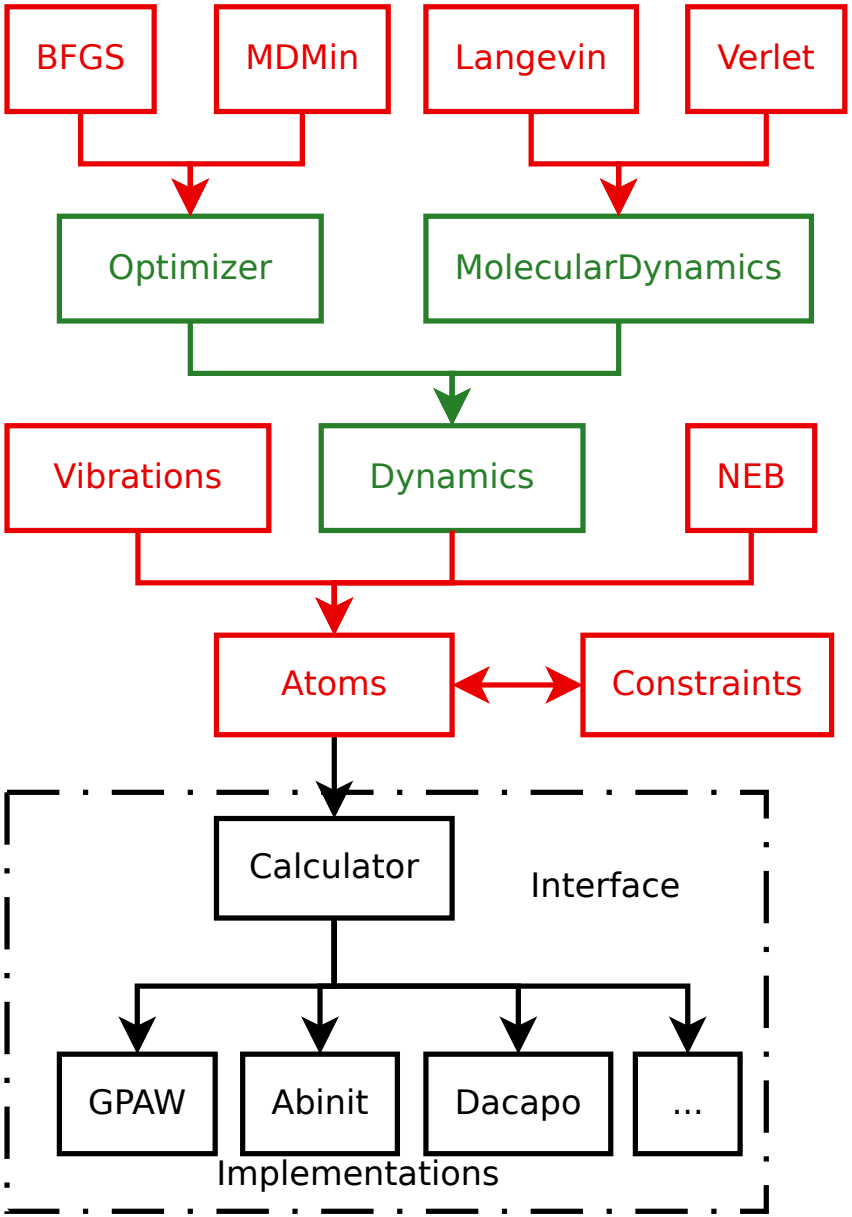
- `atoms.get_potential_energy()`
- `atoms.get_forces()`
- `atoms.get_stress()`
- `atoms.get_dipole_moment()`

Electronic structure calculators

- `calc.get_eigenvalues()`
- `calc.get_occupations()`
- `calc.get_pseudo_density()`
- `calc.get_ibz_k_points()`

Some algorithms using energies and forces

- Gradient-based structure optimizations with constraints
- Global optimizations: minima/basin hopping, genetic algorithm
- Molecular dynamics with different controls
- Saddle-point searches (for transition states)
- Vibrational modes (molecules and phonons)


BFGS
MDMin
Langevin
Verlet
Optimizer
MolecularDynamics
Vibrations
Dynamics
NEB
Atoms
Constraints
Calculator
Interface
GPAW
Abinit
Dacapo
...
Implementations

Set up structures

- `ase.build.molecule` G2 molecule test set
- `ase.build.bulk`
- `ase.spacegroup.crystal` From spacegroup
- `ase.lattice.cubic, tetragonal, ...`
- `ase.build.surface`

Skim features on web page!

Tutorials

- https://www.python.org/
- https://docs.python.org/3/tutorial/
- https://wiki.fysik.dtu.dk/ase/
- https://wiki.fysik.dtu.dk/ase/tutorials/tutorials.html