RUTGERS

Python: the programing language

Sinisa Coh

Presentation for Physics 681 September 30th 2010

1. What is python

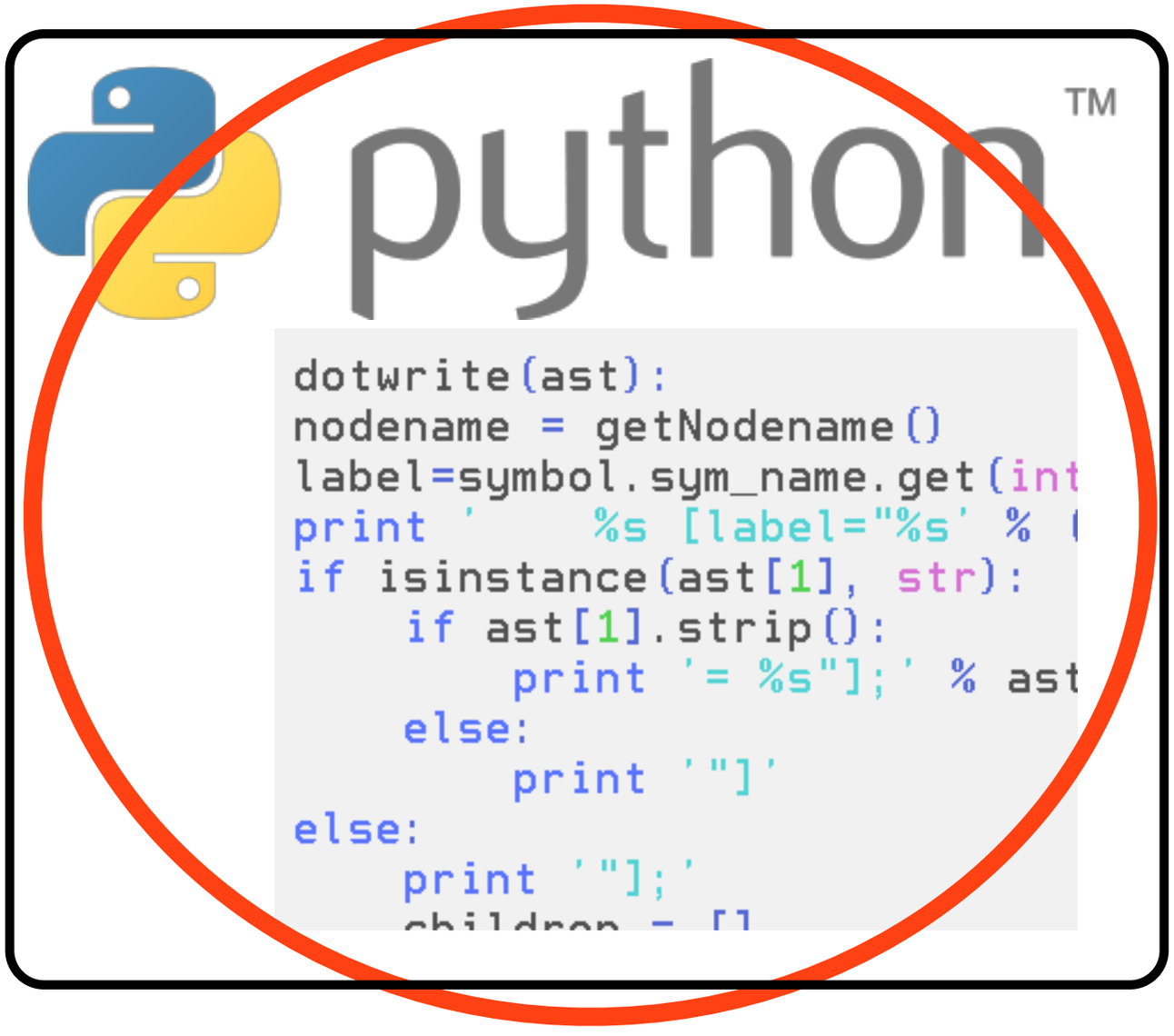
2. Basics of syntax

3. PythTB tight binding package

4. Examples

What is python?

My favorite pythons



What is python?

General purpose high-level programming language

Interpreted, not compiled

Emphasizes code readability

Syntax and semantics are minimalistic

Comprehensive standard library

Why would you need python?

1. To make your homework for 681 class

2. Great tool for post-processing data from your favorite DFT code, high-energy simulator, biophysics code...

3. Easy access to many packages (numerical, data processing, plotting)

4. Complex data visualization

How to get python?

1. **Python** itself already installed on most UNIX systems

2. Afterwards need to install **numpy** package

3. After that install **matplotlib** package

4. Useful to have **ipython** installed as well

5. Useful to get an **editor** that supports python syntax

See also:
http://physics.rutgers.edu/grad/681/python
http://wiki.python.org/moin/PythonEditors

Basics of python + numpy + matplotlib

How to run python scripts? (1/3)

Python script files usually end with ".py"

example.py

If python is in your path you can then then execute it as

python example.py

There is no need for compiling!

How to run python scripts? (2/3)

If you make file executable with this command

chmod u+x example.py

and if example.py starts with

#! /usr/bin/env python

or

#! /usr/bin/python

you can run script simply as

./example.py

How to run python scripts? (3/3)

You can run python directly

python

or

ipython

and then start writing in your commands without having a separate file as example.py

How to get help?

Easiest if you have ipython installed

```
ipython
```

Then execute for example

```
?abs
```

and you will get help on function “abs”.

Or if you do

```
a [hit tab]
```

you will get all functions that start on “a”

Data types

Integer variable is initialized if there is no “.”

```
x=1
```

otherwise it is a float

```
y=3.1416
```

you can also make complex types

```
z=3.4 + 4.5j
```

Anything in quotation marks is a string

```
s=”Test”
```

Python lists

Python has quite flexible support for lists

```
a=[3,8,4,6,9]
```

And then you can access each element as

```
a[2]
```

Or you can access them backwards

```
a[-1]
```

Or you can access more elements at once

```
a[1:3]
```

Python lists

You can make multidimensional lists

```
a=[[8,5,3],[1,2,4],[2,3,6]]
```

They can contain elements of various types

```
b=[3.14,1,"Test",3+4j]
```

These lists are not convenient for numerical manipulations

```
a=[1,2]+[3,4]
```

will result in

```
a  -->  [1,2,3,4]
```

Numpy lists

For numerical manipulations, need to use numpy lists

First need to import numpy at the top

```
import numpy as nu
```

Then you can create numpy lists as

```
a=nu.array([[3,4],[2,5]])
```

Now you can do many linear algebra routines

Numpy lists

If you have these objects for example

```
a=nu.array([3,2,6])

b=nu.array([4,9,1])

mat=nu.array([[3,2,7],[4,1,0],[9,3,2]])
```

Then you can do

```
a+b

nu.dot(mat,a)

nu.linalg.det(mat)

nu.linalg.eig(mat)
```

For loops

For loops in python always are “looping” over python lists

```
lst=[3,54,61,1,7]
for i in lst:
    print i
```

Often you want to loop over integers from 0 to n-1

```
for i in range(5):
    print i
```

All blocks in python have to be indented

```
for i in range(5):
    print i
    for j in range(10):
        print j
        print i+j
```

If - else statement

Likewise if-else statement is also indented

```
if x > 0:
    print “X is positive”
    y=y+x
else:
print “X is negative”
y=y-x
if x < -5:
    print “do something”
    y=y+x*5
elif x > -2:
    ...
else:
    ...
```

Functions

Functions are defined as follows

```
def func(x,y,z):
    r=x*y+z
    return r

func(2,1,4)
```

You can also make them more fancy

```
def func(x,y,z=3,*args,**kwargs):
    ...

func(3,2,1,4,5,more="Test")
```

Importing packages

Python package can be imported in three ways

```
import numpy
```

```
import numpy as nu
```

```
from numpy import *
```

Usage of functions from numpy is in each case then

```
numpy.array([1,2,3])
```

```
nu.array([1,2,3])
```

```
array([1,2,3])
```

Plotting in matplotlib

Matplotlib is a package that can do very complex kind of plotting. Here is the simplest example

```
import pylab as pl

pl.plot([1,2,3],[1,4,9])

pl.title("Title")

pl.xlabel("x")

pl.ylabel("$x^2$")

pl.savefig("plot.pdf")
```

And much much more:

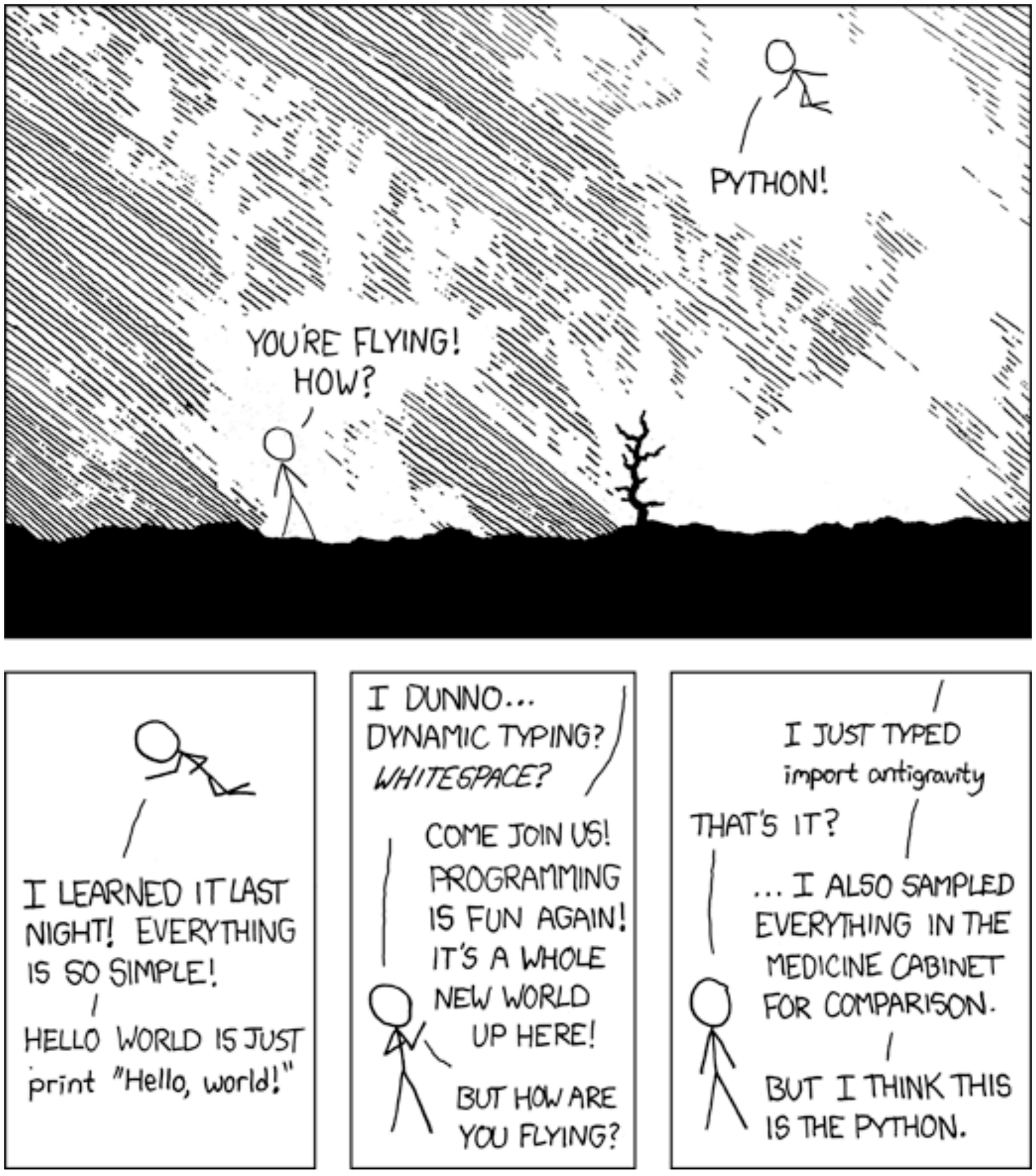
object-oriented programing

functional programming

...

http://docs.python.org/tutorial

http://physics.rutgers.edu/grad/681/python


PYTHON!
YOU'RE FLYING!
HOW?
I LEARNED IT LAST NIGHT! EVERYTHING IS SO SIMPLE!
HELLO WORLD IS JUST print "Hello, world!"
I DUNNO... DYNAMIC TYPING? WHITESPACE?
COME JOIN US! PROGRAMMING IS FUN AGAIN! IT'S A WHOLE NEW WORLD UP HERE!
BUT HOW ARE YOU FLYING?
I JUST TYPED import antigravity
THAT'S IT?
... I ALSO SAMPLED EVERYTHING IN THE MEDICINE CABINET FOR COMPARISON.
BUT I THINK THIS IS THE PYTHON.

xkcd.org

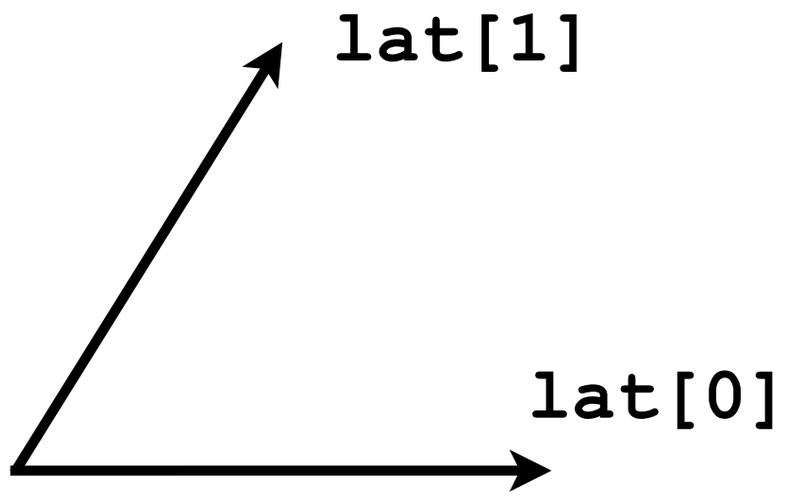
PythTB tight binding package

Specifying model

To specify (orthogonal) tight binding model you need

1. Unit cell vectors
2. Positions of tight binding orbitals
3. On-site energies
4. Hopping parameters

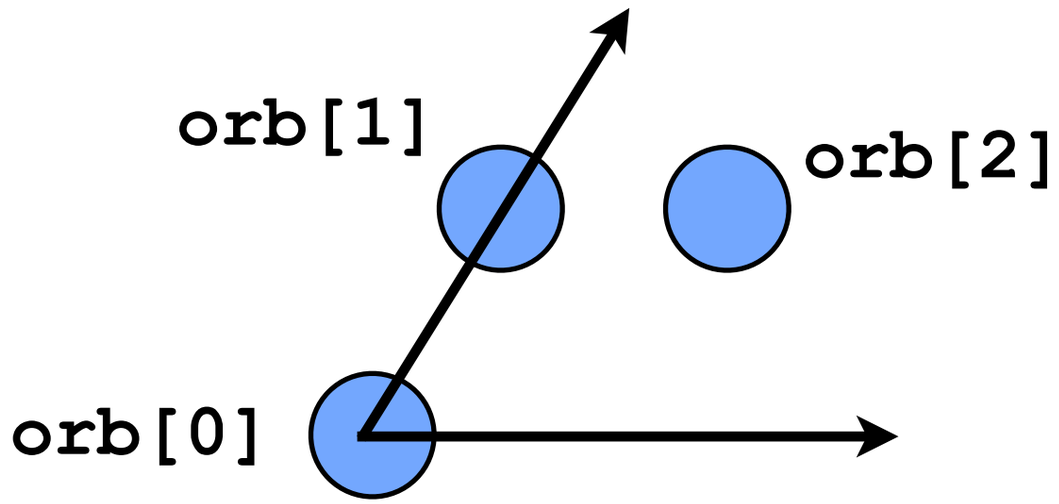
Unit cell vectors



Need to define a matrix whose rows are coordinates of unit cell vectors

```
lat=[[1.0,0.0],[0.2,0.9]]
```

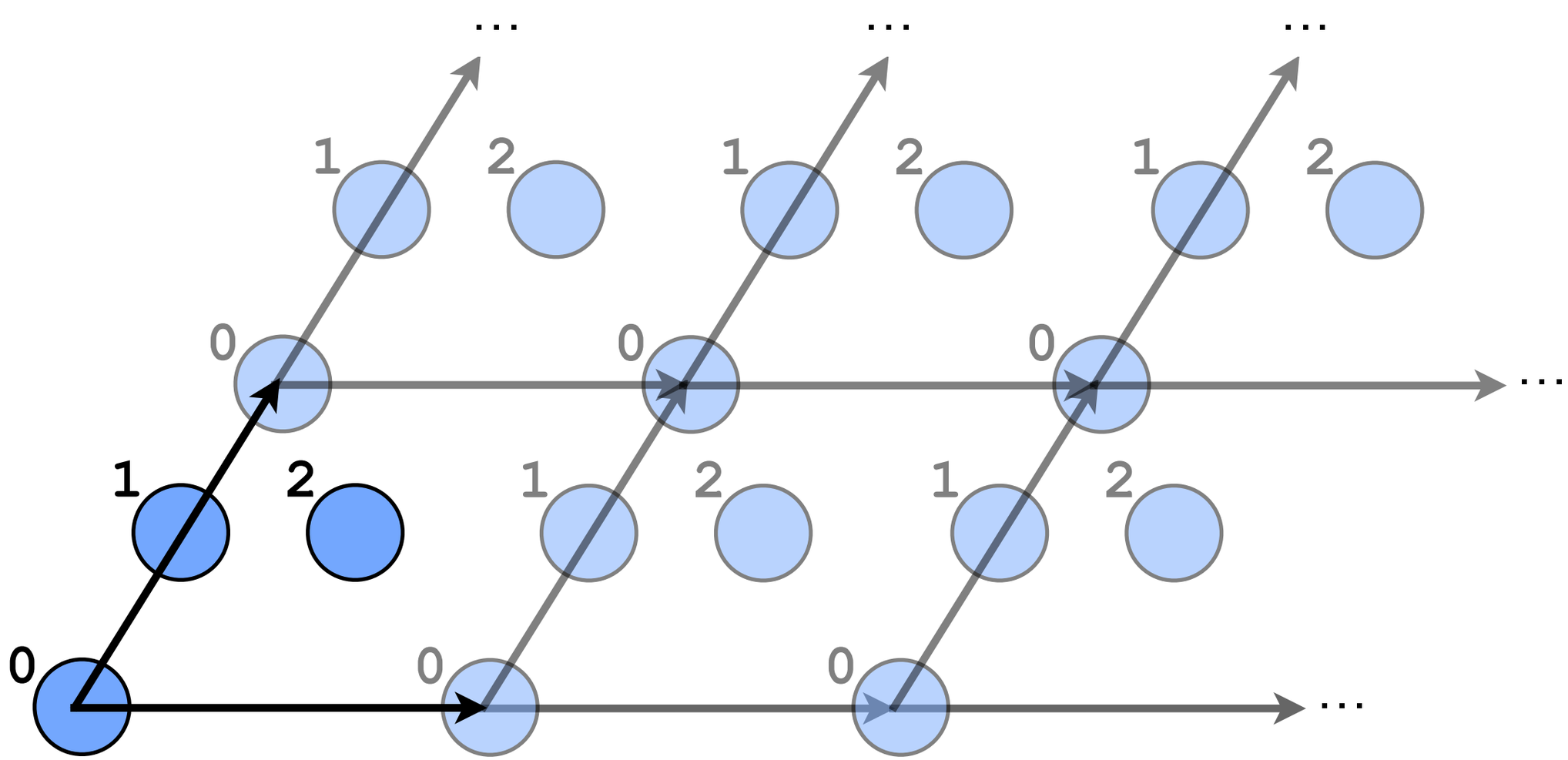
Orbital positions



In terms of **reduced** coordinates of unit cell vectors need to specify coordinates of orbitals

```
orb=[[0.0,0.0],[0.0,0.5],[0.5,0.5]]
```

Orbital positions



This is just a basis for a crystal, which is periodic

```
orb=[[0.0,0.0],[0.0,0.5],[0.5,0.5]]
```

Object file representing model

```
lat=[[1.0,0.0],[0.2,0.9]]
orb=[[0.0,0.0],[0.0,0.5],[0.5,0.5]]
```

Using these parameters we can now generate the object which will represent our tight-binding model

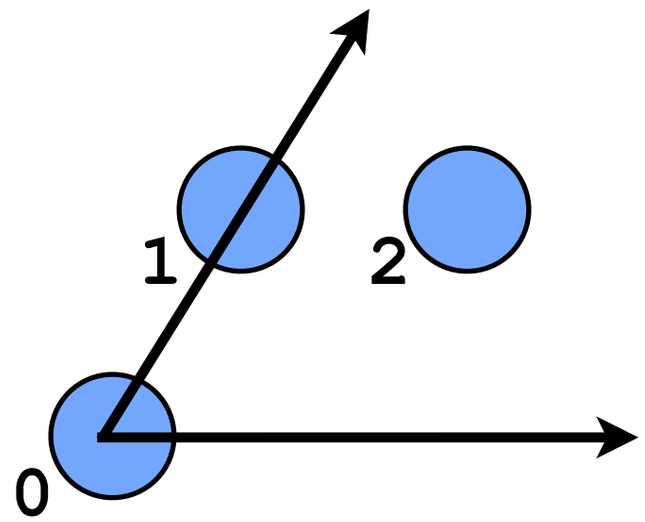
```
my_model=tb_model(2, 2, lat, orb)
```

You can think of `my_model` as variable which contains all information about the model.

First two parameters are dimensionality of **k** and **r** space

For now, our model has all hamiltonian matrix elements set to zero

Specify onsite energies



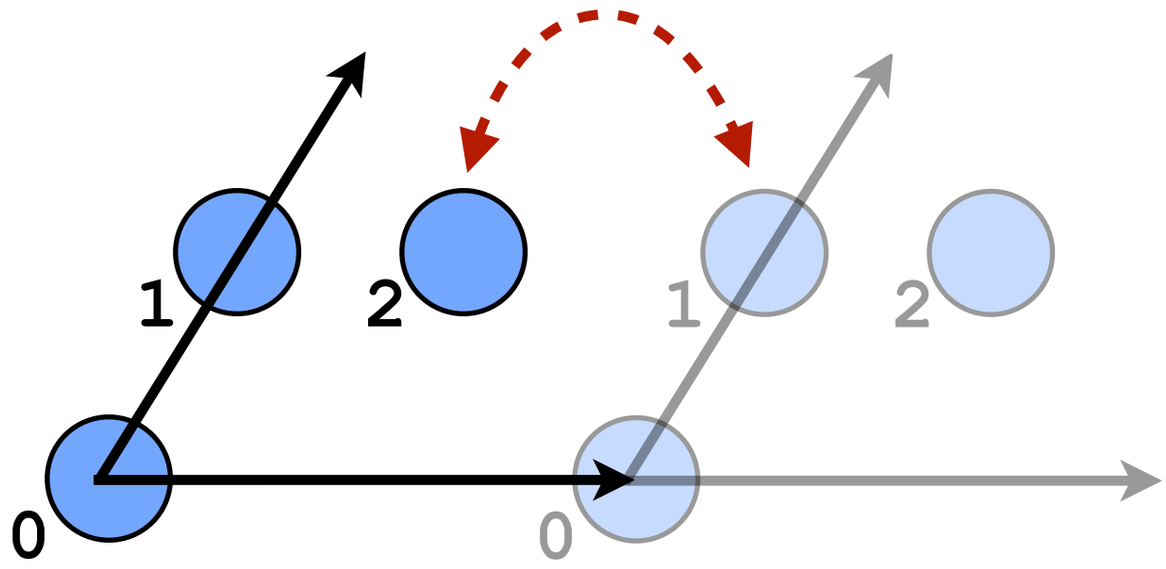
For each orbital need to specify onsite energy

```
my_model.set_onsite([1.0, -1.0, 0.0])
```

This line specifies this matrix element

<i|H|i>

Specify hopping matrix elements



Also need to specify all hopping terms you want to have

```
my_model.set_hop(thop, 2, 1, [1, 0])
```

This line will specifies both

<i|H|j+R> and **<j+R|H|i>**

Solving model

Energy at a given k-point can now easily be calculated

```
my_model.solve_one([0.3, 0.4])
```

k-vector is specified in reduced coordinates in reciprocal space

You can also solve for many k-points at once

```
my_model.solve_all([[0.3, 0.4],[0.1,0.2]])
```

You can also create arbitrary path in k-space

```
p=[[0.0, 0.0],[0.0, 0.5],[0.5, 0.5]]
kpts=k_path(p, 100)
my_model.solve_all(kpts)
```

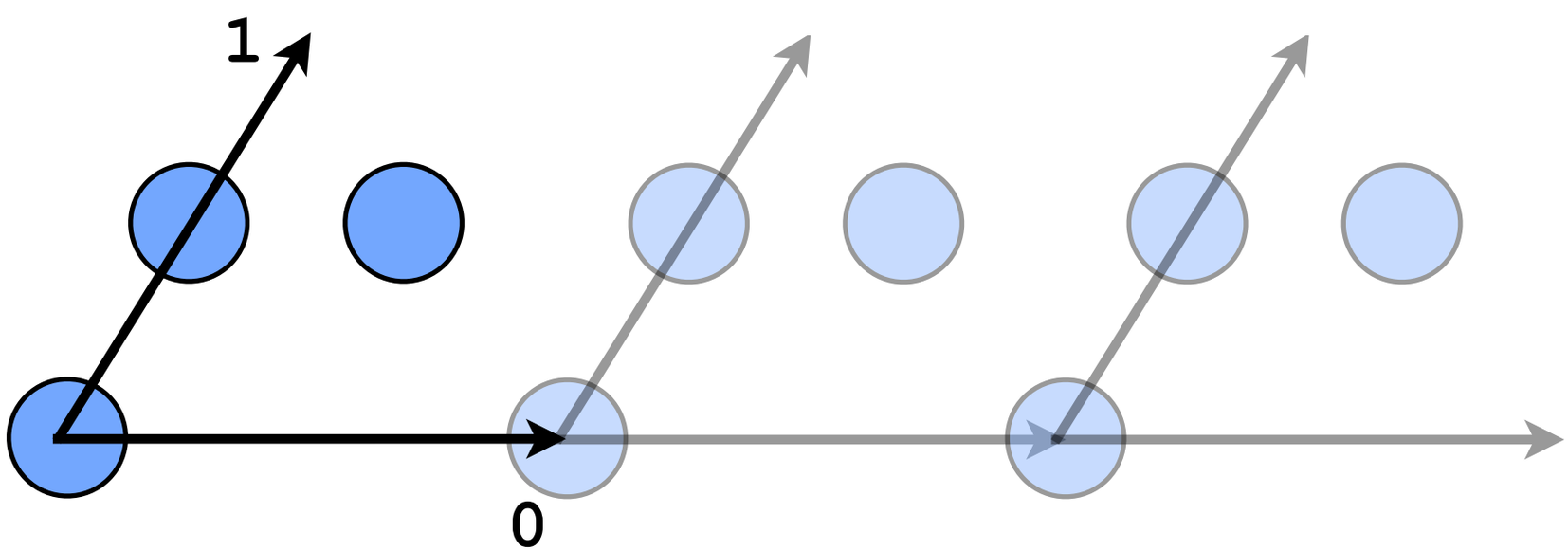
Special cases

Can specify dimensionality of k-space to be smaller than dimensionality of real space

```
my_model=tb_model(1, 2, lat, orb)
```

Here lattice vectors and orbital positions are still two dimensional but k-vector is one dimensional

By convention, first vector is the one that is periodic

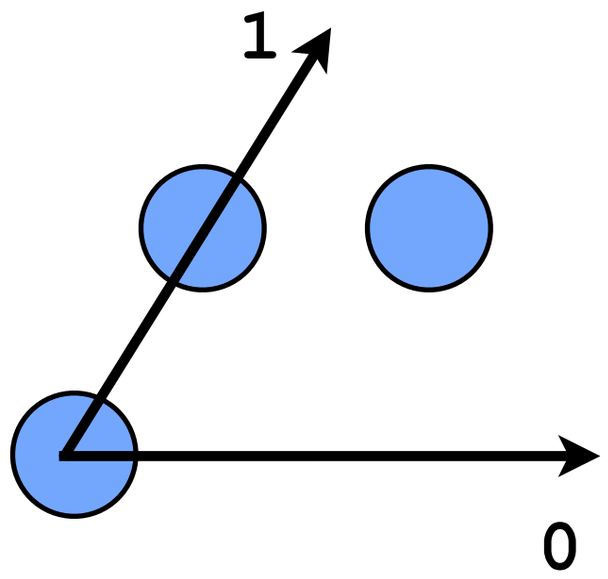


Special cases

Can also make model zero dimensional

```
my_model=tb_model(0, 2, lat, orb)
```

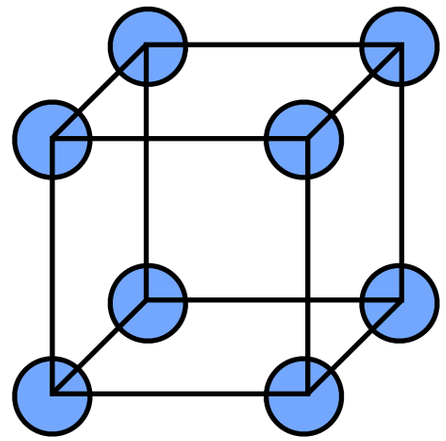
Now there is no k-vector



PythTB examples

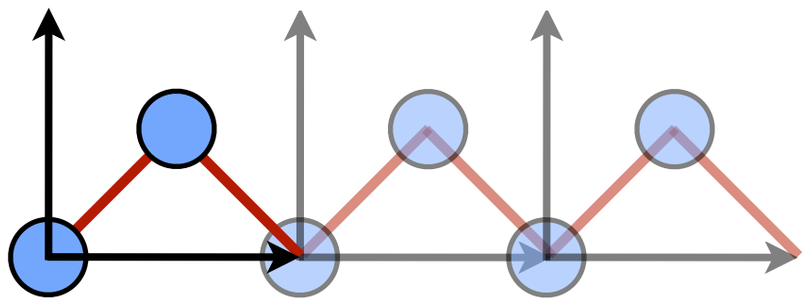
Simple cubic model

```
from pythtb import *
lat=[[1.0,0.0,0.0],
     [0.0,1.0,0.0],
     [0.0,0.0,1.0]]
orb=[[0.0,0.0,0.0]]
my_model=tb_model(3,3,lat,orb)
my_model.set_onsite([3.0])
my_model.set_hop(-1.0,0,0,[1,0,0])
my_model.set_hop(-1.0,0,0,[0,1,0])
my_model.set_hop(-1.0,0,0,[0,0,1])
```



Chain model with p orbitals

```
from pythtb import *
lat=[[1.0,0.0],
     [0.0,1.0]]
orb=[[0.0,0.0],
     [0.0,0.0],
     [0.0,0.0],
     [0.5,0.5],
     [0.5,0.5],
     [0.5,0.5]]
my_model=tb_model(1,2,lat,orb)
my_model.set_hop(-1.0,0,1,[0,0])
...
```



Remaining examples on the website:

www.physics.rutgers.edu/pythtb