# Lecture 1: Introduction to Python

Hendrik Weimer

Institute for Theoretical Physics, Leibniz University Hannover

Quantum Physics with Python, 04 April 2016

# Goals of this course

- Learn to use the Python language for physics problems
- Master the Quantum Toolbox in Python (QuTiP) library
- Get ready to write code for research projects
- Have fun!

# What this course is not about

- Algorithms
- Learning to write code
- Environmental aspects (installing packages, using clusters)

# Some useful references

- The Python Tutorial: `https://docs.python.org/3/tutorial/`
- PEP 0008 – Style Guide for Python Code: `https://www.python.org/dev/peps/pep-0008/`
- J. R. Johansson: *Lectures on scientific computing with Python* and *Lectures on QuTiP*, `https://jrjohansson.github.io/computing.html`

# Outline of the course

1. **Introduction to Python**
2. SciPy/NumPy packages
3. Plotting and fitting
4. QuTiP: states and operators
5. Ground state problems
6. Non-equilibrium dynamics: quantum quenches
7. Quantum master equations
8. Generation of squeezed states
9. Quantum computing
10. Grover's algorithm and quantum machine learning
11. Student presentations

# Why Python?

- Combines advantages of high-level and low-level languages
- Rich standard library (e.g., including a web server)
- $>$75,000 modules on PyPI (Python Package Index)
- 5th most popular language according to the TIOBE index

# Python 2 and 3

- Python 3.0 was released in 2008
- Many GNU/Linux distributions still use 2.7 as the default (will change in 2016/2017)
- Almost all important packages are ready for Python 3 (see: `http://py3readiness.org/`)
- `E_kin = 1/2*m*v**2`

$\Rightarrow$ We will use Python 3

# Python specifics: Whitespaces

- Control flow via indentation
- No braces

```
if a is True:
    b = c
else:
    b = d
```

- Use a good editor to assist you!

# Dynamical typing

```
a = 1
print(type(a))
a = 3.14
print(type(a))
a = 'foo'
print(type(a))
```

Output:

```
<class 'int'>
<class 'float'>
<class 'str'>
```

# String functions

```
str = 'foomatic'
print(str.find('bar'))
print(str.find('foo'))
print('foobarbaz'.strip('baz'))
```

Output:

```
-1
0
foobar
```

# Lists

```
a = [1, 1, 2, 3, 5, 8]
print(a[2])
print(a[1:4])
print()

a = [0]*4
print(a)
a.append(1)
print(a)
```

Output:

```
2
[1, 2, 3]

[0, 0, 0, 0]
[0, 0, 0, 0, 1]
```

# Dictionaries

```
b = {'name': 'Alice', degree: 'B.Sc.'}
print(b['name'])
print(b.keys())
```

Output:

```
Alice
dict_keys(['degree', 'name'])
```

# Simple for loops

```
a = [1, 3, 5, 7]
for i in a:
    print i
print()

for i in range(4):
    print i
```

Output:

```
1
3
5
7

0
1
2
3
```

# Defining functions

```
def foo():
    print("I am foo!")

def bar(baz):
    print("And I am bar, called with", baz, "as an argument!")

foo()
bar("qux")
```

Output:

```
I am foo!
And I am bar, called with qux as an argument!
```

# Using a “main” function

```
def main():
    ...

if __name__ == "__main__":
    main()
```

# Comments and docstrings

```
# This is a comment

def important_function(foo, bar=0):
    """This is a docstring describing the use of important_function.

    Keyword arguments:

    foo -- description of the first function argument
    bar -- description of the first function argument (default 0)
    """

    # More comments describing what is going on in the code
```

See also: PEP 0257: Docstring Conventions,
https://www.python.org/dev/peps/pep-0257/

# Importing modules

Good:

```
import math
from math import sqrt
```

Bad:

```
from math import *
```

# Let's get started!

Let's write our first Python program!