CONTENTS

1 Python crash course ........................................ 3
   1.1 Python command line .................................. 3
   1.2 Basic data types ..................................... 3
   1.3 Operators ............................................. 4
   1.4 Variables ............................................. 4
   1.5 More data types ...................................... 5
   1.6 Control flow ......................................... 6
   1.7 Defining functions .................................. 8
   1.8 Python program ....................................... 8
   1.9 Task: Reading a coordinate file ..................... 9
Files for this practical can be found at http://becksteinlab.physics.asu.edu/pages/courses/2013/SimBioNano/03/
Contents:
Resources:
- http://www.swaroopch.com/notes/Python_en:Table_of_Contents
- http://www.python.org/doc/
- http://docs.python.org/tutorial/

1.1 Python command line

Run python: opens a python commandline, the Python interpreter:

```
$ python
Type a command:
```
```
>>> print("Hello World!")
Hello World!
```
Exit with CTRL+d or `exit()`

1.2 Basic data types

- numbers: integers, floating point numbers, complex numbers:
  
  ```
  42
  3.14152
  3.2 - 0.3j
  ```

- strings (use single or double quotes)

  ```
  "Hello World!"
  ‘Hello World’
  ```

  triple quotes: span multiple lines - can contain single quotes

- escaping: backslash:

  ```
  ‘What’s your name?’
  "What’s your name?"
  ```
print(\n) outputs
- ‘n’ : newline
- ‘t’ : tab

• conversion:
  
  float("3")
  str(3)
  int(3.3)

### 1.3 Operators

Python as a calculator:

```python
g>>> 3 + 10
>>> 3 - 10
>>> 3 * 101
>>> 3/10  # !
>>> 3./10  # !
```

• arithmetic: + - * /

• power: **

• modulo: %

• integer division: // (note that Python 2.x does integer division _by default_ if all numbers are integers, i.e. 2/3 == 0 so use 2./3.)

• comparison: < > == >= != ==

• boolean: and, or, not

### Strings:

• + concatenates (as does writing strings adjacent to each other "a" + "b" == "ab")

• single and double quotes are equivalent. Triple (single or double) quotes can span multiple lines.

### 1.4 Variables

Assign values to names:

```python
answer = 42
x = 0.1234
y = 2
z = -2.5 + 0.2j
pi = 3.14152
hero = "Batman"
sidekick = "Robin"

a = b = c = 0
q = (x > 0)
print q
```
and do something with it
x * y
z + x

team = hero + " and " + sidekick

i = 0
i = i + 1
i += 1

Operator precedence:
- see table http://www.swaroopch.com/notes/Python_en:Operators_and_Expressions
- can be changed with parentheses

1.5 More data types

1.5.1 Lists

With brackets:

```
bag = [1, 3, "cat", 5, "dog"]
empty = []
```

Indexed (starting at 0):

```
bag[0]
bag[1]
bag[-1]
```

```
>>> bag[10]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

Length: len function:

```
len(bag)
```

Slicing:

```
bag[0:2]
bag[:2]
bag[2:]
bag[1:3]
bag[-2:]
```

```
bag[:] # returns new list (makes a copy)
```

Also works on strings:

```
ga = "Four score and seven years ago"
len(ga)
 ga[:4]
ga[15:20]
ga[15:15+len("seven")]
```
Iterating:

```python
for thing in bag:
    print("I have a %s" % (thing,))
```

### 1.5.2 Tuples

A tuple is a sequence that can be indexed like a list but cannot be changed:

```python
point = (11.2, -3.4)
x = point[0]
y = point[1]
# error
point[0] = 20
```

### 1.5.3 Dictionaries

Dictionaries are containers that can be indexed with arbitrary keys:

```python
ages = {'Einstein': 42, 'Dirac': 31, 'Feynman': 47}
ages['Dirac']
ageis['Heisenberg'] = 1932 - 1901
print(ages)
```

Another way to create a dictionary:

```python
ages = dict(Einstein=42, Dirac=31, Feynman=47)
```

Note that the order of elements in a dictionary is undefined.

Iterating:

```python
for key in ages:
    print("%s got the Nobel prize at age %d" % (key, ages[key]))
```

or over pairs of (key, value):

```python
for (name, age) in ages.items():
    print("%s got the Nobel prize at age %d" % (name, age))
```

### 1.6 Control flow

A small number of statements allow you to make decisions and implement loops

See Also:

- http://www.swaroopch.com/notes/Python_en:Control_Flow

```python
• if
    age = 21
    if age >= 21:
        print "Ok, you can have a drink."
    elif age >= 18:
```

---

Chapter 1. Python crash course
You may vote.

elif age >= 16:
    print "Drive a car!"
else:
    print "You’re too young to do anything reckless."

while

Sample code:

# Fibonacci series:
# the sum of two elements defines the next
a, b = 0, 1
while b < 10:
    print b
    a, b = b, a+b

  – multiple assignments
  – while loop: conditions
  – white space: block (body of the loop)

See Also:

http://docs.python.org/tutorial/introduction.html#first-steps-towards-programming

• for loops over a list (or something that behaves like a list):

    for a in range(10):
        print a, a**2, 1./a

    bag = ["pen", "laptop", "shades", "phone", "coins"]
    for thing in bag:
        print "I have a "+ thing + " in my bag"

range() function:

print range(-2,2)
print range(-2,2,0.1)

Note that in Python 2.x you should use the xrange() function in loops as it has much better performance than range(). It does not matter in Python 3.x.

excercise:

    for thing in bag:
        if thing[-1] == "s":
            print "I have "+ thing + " in my bag"
        else:
            print "I have a "+ thing + " in my bag"

• break: terminate a loop prematurely
• continue: immediately proceed with the next iteration of a loop

Putting things together

nmax = 150
for n in xrange(2, nmax+1):
    d = 2
    while d*d < nmax:
        if n % d == 0:

1.6. Control flow
Practical 03: Introduction to Python Documentation, Release 1.0

```
break
d += 1
else:
    print "Prime number: ", n
```

Note: `xrange()` is faster than `range()` in 99% of cases.

### 1.7 Defining functions

builtin functions like `len()`

Reusable code with arguments:

```python
def funcname(arg1, arg2, opt1=val1, ...):
    COMMANDS
    return VAL
```

The positional arguments have to be provided. Optional arguments have default values.

The return value can be _any_ python data type, i.e. you can return tuples, dicts, ... any object or collection of objects.

Examples:

```python
def greeting(name):
    print "Hello ", name

def u_harm(x,x0,k):
    energy = 0.5*k*(x-x0)**2
    return energy

def fib(n):
    """Return last two Fibonacci numbers less than n."""
    a, b = 0, 1
    while a < n:
        print a,
        last_a, last_b = a, b  # save
        a, b = b, a+b
    return last_a, last_b  # can return multiple values!
```

(more later...)

### 1.8 Python program

Write a python program:

```
$ vi helloworld.py

#!/usr/bin/env python
# author: I
# program: helloworld

print("Hello World!")
```

- shabang magic
• # (“octothorpe”, “hash”, “pound”): comments
• print function

Run the program:

$ python helloworld.py
Hello World!

or:

$ chmod a+x helloworld.py
$ ./helloworld.py
Hello World!

Now make an intentional mistake:

print("Hello World!\n")

Gives :: File “helloworld.py”, line 4

^ SyntaxError: invalid syntax

White space at beginning of line is important:

#!/usr/bin/env python
print("Hello World!\n")
print("Goodbye")

yields an error::

loki:03 oliver$ ./helloworld.py

File "./helloworld.py", line 5 print("Goodbye") ^

IndentationError: unexpected indent

• leading whitespace is crucial

• be consistent: either 1 TAB or 4 spaces (spaces are recommended)

set up vi appropriately:

" Python: see http://wiki.python.org/moin/Vim
autocmd BufRead,BufNewFile *.py syntax on
autocmd BufRead,BufNewFile *.py set ai
autocmd BufRead *.py set smartindent cinwords=if,elif,else,for,while,with,try,except,finally,def,class

" indentation
" add to sourcefiles:
" # vim: tabstop=4 expandtab shiftwidth=4 softtabstop=4
set modeline
au FileType python setl autoindent tabstop=4 expandtab shiftwidth=4 softtabstop=4

1.9 Task: Reading a coordinate file

Get the data file http://becksteinlab.physics.asu.edu/pages/courses/2013/SimBioNano/03/Ar_L16_N64.xyz Using your downloader script cd1:
Footnote: Look at the file in VMD

1. Open /Applications/VMD
2. File -> New Molecule: browse to Ar_L16_N64.xyz and load
4. Look at scene with mouse - click + moving: rotates - scroll wheel: zoom - switch between [r]otation and [t]ranslation by pressing 'r' or 't' (or use the menu Mouse)
5. File -> Quit

1.9.1 File structure: XYZ format

Look at the file as text:

```
less Ar_L16_N64.xyz
```

The XYZ file format is a very simple format to store positions of particles. It is described in VMD’s XYZ Plugin. Basically, a XYZ file looks like this:

```
N
title text
atom1 x y z
atom2 x y z
...
atomN x y z
```

The first line is the number of atoms. The second a string. From the third line onwards, each line contains a symbol for the particle (“atomX”) and the cartesian coordinates. All entries are white-space separated.

1.9.2 Data structures

- atoms: list `[‘Ar’, ‘Ar’, …]`
- coordinates: list `coord = [[x,y,z], [x,y,z], …]` so that we can access

```
coord[0]
coord[3][2]  # <-- z of atom 3
```

note: atom numbering starts with 0 (Python!)

Reading the file interactively in the Python interpreter:

```python
filename = "Ar_L16_N64.xyz"
xyz = open(filename, "r")
xyz.readline()
xyz.readline()
line = xyz.readline()
print line
line.split()
atom, x, y, z = line.split()
xyz.close()
```
Practical 03: Introduction to Python Documentation, Release 1.0

Looping through a file, line by line:

```python
xyz = open(filename, "r")
for line in xyz:
    print '>>> ', line
xyz.close()
```

(Note: when typing interactively, finish loop with empty line)

**Note:** The opened file is a “object” (what we named `xyz` in the example): Objects are “thingies” that have methods (=functions) and attributes (=variables). For right now, remember the above code as the way to deal with files.

Building lists:

```python
coord = []
coord.append([1,2,3])
coord.append([0,2.2,5])
len(coord)
print coord
```

Now put it all together: We write a small script `reader.py` that

- stores the atoms in a list `atoms`
- stores coordinates in a list `coordinates`
- number of atoms in variable `n_atoms`
- title in variable `title`
- and prints number of atoms and title

**Script reader.py:**

```python
#!/usr/bin/env python
# read xyz coordinate file

filename = "Ar_L16_N64.xyz"
atoms = []
coordinates = []
xyz = open(filename)
n_atoms = int(xyz.readline())
title = xyz.readline()
for line in xyz:
    atom,x,y,z = line.split()
    atoms.append(atom)
    coordinates.append([float(x), float(y), float(z)])
xyz.close()

print("filename: %s" % filename)
print("title: %s" % title)
print("number of atoms: %d" % n_atoms)
```

`title` comes with newline:

```python
title = title.strip()
```

or:

```python
title = xyz.readline().strip()
```

1.9. Task: Reading a coordinate file
Your task: add a check that the number of atoms $n_{\text{atoms}}$ is really the same as the number of atoms read. Print an error message if the numbers are not equal.

```python
if len(atoms) != n_atoms:
    print("ERROR: file contains \$d\ atoms instead of the stated number \$d\ \% (n_atoms, len(atoms))")
    print("number of atoms in file: \$d\ \% len(atoms))")
    print("number of coordinates: \$d\ \% len(coordinates))")
```

Next (or if you’re quick: do it as a bonus challenge):

1. package the above code as a function:

```python
atoms, coordinates = read_xyz(filename)

def read_xyz(filename):
    """Read filename in XYZ format and return lists of atoms and coordinates.
    If number of coordinates do not agree with the statd number in
    the file it will raise a ValueError.
    """
    atoms = []
    coordinates = []

    xyz = open(filename)
    n_atoms = int(xyz.readline())
    title = xyz.readline()
    for line in xyz:
        atom,x,y,z = line.split()
        atoms.append(atom)
        coordinates.append([float(x), float(y), float(z)])
    xyz.close()

    if n_atoms != len(coordinates):
        raise ValueError("File says \$d\ atoms but read \$d\ points." \% (n_atoms, len(coordinates))

    return atoms, coordinates
```

2. write a xyz writer:

```python
write_xyz(filename, atoms, coordinates)

• open a file for writing: xyz = open(fn, "w")
• write a line: xyz.write("...
\n")
```