Introductory Linux Course

Python II

Nina Fischer
Dept. for Cell and Molecular Biology, Uppsala University
nina.fischer@icm.uu.se

August 26, 2016
Outline

- Short recap
- Functions
- Similarity of sequences
- Modules
- Command line arguments
- Read and write files
- Biopython
Short Recap – Creating a Program

- An arbitrary editor (nano, gedit, emacs, vi, ...) can be used to create the program code and save it as a file.
Short Recap – X server

- Login:
  
  $ ssh -X <username>@tintin.uppmax.uu.se

- Open editor, e.g., gedit:
  
  $ gedit sequence.py &

- In the terminal a new line, which starts with $ should appear. If not type <Ctrl C>

- Load module:
  
  $ module load python/3.5.0

- Then you can run your program in the terminal window:
  
  $ python3 sequence.py

**Advantage:** You can edit your program and switch easily between the terminal and editor.
Short Recap

We count from 0!

```
A = "ACGTCGA"
print(len(A))
for i in range(len(A)):
    print(i, A[i])
```

```python
7
for i in range [0,7[ 0 A
```
Short Recap

We count from 0!

```
A = "ACGTCGA"
print(len(A))
for i in range(len(A)):
    print(i, A[i])
```

```
7
for i in range [0,7[
0 A
1 C
```
Short Recap

We count from 0!

A = "ACGTCGA"
0 1 2
print(len(A))
for i in range(len(A)):
    print(i, A[i])

7
for i in range [0,7[
0 A
1 C
2 G
Short Recap

We count from 0!

A = "ACGTCGA"

\[
\begin{array}{c|c}
0 & A \\
1 & C \\
2 & G \\
3 & T \\
\end{array}
\]

```
print(len(A))
for i in range(len(A)):
    print(i, A[i])
```
A = "ACGTCGA"
    01234
print(len(A))
for i in range(len(A)):
    print(i, A[i])

for i in range [0,7[
0 A
1 C
2 G
3 T
4 C
Short Recap

We count from 0!

```python
A = "ACGTCGA"
0 1 2 3 4 5
print(len(A))
for i in range(len(A)):
    print(i, A[i])
7
for i in range [0,7[
0 A
1 C
2 G
3 T
4 C
5 G
```
Short Recap

We count from 0!

```python
A = "ACGTCGA"
print(len(A))
for i in range(len(A)):
    print(i, A[i])
```

```
0 A
1 C
2 G
3 T
4 C
5 G
6 A
```
Short Recap

We count from 0!

```python
A = "ACGTCGA"
print(len(A))
for i in range(len(A)):
    print(i, A[i])
```

```
7
for i in [0,7[  
0 A  
1 C  
2 G  
3 T  
4 C  
5 G  
6 A  
7
```
Functions

- Functions in programming languages are similar to mathematical functions
- In principal, functions are used to avoid repetitions of the same code
- Functions are code fragments, which can be given one or more arguments and can have a return value
- They have a name, a list of arguments and a code block, which does the actual work and are defined by `def`

```python
def product(x, y):
    return x * y

print(product(15, 5))
```
Functions

- **Fibonacci** numbers: infinite sequence, describes the growth of an idealized rabbit population (Leonardo Fibonacci, 1202)

- **Definition**: $f_0 = 0$
  $f_1 = 1$
  $f_n = f_{n-1} + f_{n-2} \ \forall n > 2$

- This can be written as recursive function (a function which calls itself again)

```python
def fib(x):
    if (x == 0):
        return 0
    elif (x == 1):
        return 1
    else:
        return fib(x-1) + fib(x-2)

for i in range(10):
    print(fib(i))
```

Nina Fischer

Python II
Similarity of Sequences

Biological intuition:

Sequence = string of characters

Informal:

Arrange sequences in such a way that similarity becomes visible

Example: Similarity of two DNA sequences

GATCGTT CG
| | | | |
CATGGTT GA
Similarity of Sequences

Distance function:

\[ d(a, b) = \begin{cases} 
0, & \text{if } a = b \\
1, & \text{otherwise} 
\end{cases} \]

Example:

A = GATCGTTTCG, B = CATGGTTTGA

GATCGTTTCG
|| || |||
CATGGTTTGA
Similarity of Sequences

Distance function:

\[ d(a, b) = \begin{cases} 
0, & \text{if } a = b \\
1, & \text{otherwise} 
\end{cases} \]

Example:

A = GATCGTTTCG, B = CATGGTTGA

\[
\begin{align*}
\text{GAT} & \quad \text{CG} \\
\text{GTT} & \quad \text{TTCG} \\
\text{CAT} & \quad \text{GGTT} \\
\text{GA} & \quad \text{TGA} \\
\end{align*}
\]

100100011

\[ distance = 4 \]
Similarity of Sequences

A = "GATCGTTTCG"
B = "CATGGTTGA"

d(a, b) = \begin{cases} 
0, & \text{if } a = b \\
1, & \text{otherwise}
\end{cases}
# Simple calculation of distances

A = "GATCGTTTCG"

B = "CATGGTTGA"

# Calculate the distance for each position and sum it up

for i in range(len(A)):
    if (A[i] == B[i]):
        d = 0
    else:
        d = 1

\[
d(a, b) = \begin{cases} 
0, & \text{if } a = b \\
1, & \text{otherwise}
\end{cases}
\]
# Simple calculation of distances

A = "GATCGTTTCG"
B = "CATGGTTTGA"
distance = 0

# Calculate the distance for each position and sum it up

for i in range(len(A)):
    if (A[i] == B[i]):
        d = 0
    else:
        d = 1
    distance = distance + d

print("Distance of sequences A and B: ", distance)
# Simple calculation of distances

A = "GATCGTTCG"
B = "CATGTTGGA"

distance = 0

# Calculate the distance for each position and sum it up

for i in range(len(A)):
    if (A[i] == B[i]):
        d = 0
    else:
        d = 1
    distance = distance + d

print("Distance of sequences A and B: ", distance)
Similarity of Sequences with Function

```python
# Simple calculation of distances
A = "GATCGTTCG"
B = "CATGGTTGA"
distance = 0

# Function d(a, b) for calculating the distance
def d(a, b):
    if (a == b):
        return 0
    else:
        return 1

# Calculate the distance for each position
for i in range(len(A)):
    distance = distance + d(A[i], B[i])

print("Distance of sequences A and B: ", distance)
```

\[
d(a, b) = \begin{cases} 
0, & \text{if } a = b \\
1, & \text{otherwise}
\end{cases}
\]
Modules

- A program can be divided into smaller, better manageable units, so called modules
- Improves reusability of code: when someone wrote something useful, someone else can use the same code in his program
- Example: Circumference

```python
r = 5.6
pi = 3.1415
print("Circumference: ", 2 * pi * r)
```

```python
from math import pi
r = 5.6
print("Circumference: ", 2 * pi * r)
```
In module `math` \( \pi \) is already (with full accuracy) defined

`import` loads its functionality from module `math`
Modules

- Functionality of modules are not implemented inside the language Python
- There exist many useful additional modules which can be imported with:

  ```python
  import <module_name>
  ```

- Examples:
  - `import math / from math import pi`
  - `import numpy`
  - `import Bio / from Bio import SeqIO`
  - `...`

- These modules need to be installed separately
Modules

- Own modules can be written
- For this a Python file with the name of the module is written, which can be used by other programs
- `import` can load parts of the module (`import x from y`/`import d from Dist`) or the entire module (`import Dist`)

Dist.py

```python
# Module implements # distance function
def d(a, b):
    if (a == b):
        return 0
    else:
        return 1
```

```python
# Calculates distance
import Dist # loads module
A = "GATCGTTCG"
B = "CATGGTTGA"
dist = 0
for i in range(len(A)):
    dist = dist + Dist.d(A[i], B[i])
```
Command Line Arguments

- Always try to separate data and program code
- Read data from command line (e.g., DNA sequence)

```python
s = "ABCD"
for i in range(len(s)):
    print(s[i])
```

```python
import sys
s = sys.argv[1]
print("s = ", s)
for i in range(len(s)):
    print(s[i])
```

trivial2.py
Execution of the Program

- In order to execute the program we call the interpreter from the command line using the filename as first argument and the sequence as second argument.

```
$ python3 trivial2.py AGCT
s = AGCT
A
G
C
T
```

```
trivial2.py
import sys
s = sys.argv[1]
print("s = ", s)
for i in range(len(s)):
    print(s[i])
```
Read and Write Files

- We have several sequences stored in files and want to do something with them

- **Example:** Calculate distance between sequences and write distances to file

```plaintext
sequence1.txt  GATCGTTTCG
sequence2.txt  CATGTTTTGA
```
Program, which calculates distance between two sequences and writes distance to a file

```
import sys
import Dist
inputfile1 = open(sys.argv[1], 'r')
inputfile2 = open(sys.argv[2], 'r')
A = inputfile1.readline()
B = inputfile2.readline()
inputfile1.close()
inputfile2.close()

d = 0
for i in range(len(A)):
    d = d + Dist.d(A[i], B[i])
```
Execution of the Program

- In order to execute the program we call the interpreter from the command line using the filename as first argument, the first file as second and the third file as third argument.

```
$ python3 readSeq.py sequence1.txt sequence2.txt
```

command line arguments

0 1 2
Program, which calculates distance between two sequences and writes distance to a file

```python
import sys
import Dist
inputfile1 = open(sys.argv[1], 'r')
inputfile2 = open(sys.argv[2], 'r')
A = inputfile1.readline()
B = inputfile2.readline()
inputfile1.close()
inputfile2.close()

outputfile = open(sys.argv[3], 'w')
d = 0
for i in range(len(A)):
    d = d + Dist.d(A[i], B[i])
outputfile.write("Distance between A and B is " + str(d))
outputfile.close()
```
Read and Write Files

- We have several sequences stored in files and want to do something with them
- **Example:** Calculate distance between sequences and write distances to file

```
sequence1b.txt
GATCGTTCG
TCGTT
ATCGTAA
GTGGTTGA
AGTCGT

sequence2.txt
CATGGTTGA
```

Nina Fischer
Python II
import sys
import Dist
inputfile1 = open(sys.argv[1], 'r')
inputfile2 = open(sys.argv[2], 'r')
sequences = inputfile1.readlines()
B = inputfile2.readline()
inputfile1.close()
inputfile2.close()
outputfile = open(sys.argv[3], 'w')
for A in sequences:
    d = 0
    for i in range(len(A)):
        d = d + Dist.d(A[i], B[i])
    outputfile.write("Distance between A and B is \n")
outputfile.close()
Read a fasta file `many_sequences.fasta` using the BioPython fasta-parser (files stored in `/proj/g201611/python/`)

```python
# import fasta-parser from BioPython
from Bio import SeqIO

# read all sequences stored in fasta file
fasta_file = open("many_sequences.fasta", 'r')
all = list(SeqIO.parse(fasta_file, "fasta"))

# print out all sequence ids and sequences
for record in all:
    print(record.id, str(record.seq))
```
Biopython

- On tintin (or milou) start the Biopython environment:
  
  ```
  module load bioinfo-tools biopython
  ```

- Make sure that you unload other Python modules

- Then start the Python-interpreter with:
  ```
  python3
  ```

- Or run your python program with:
  ```
  python3 <your_program>.py
  ```
Many other Functionalities

- There are many other ways how to read in files from the command line
  One of the fastest ways:

```python
with open("<your_file>") as f:
    for line in f:
        <do something with line>
```

- You can list all files from a directory, e.g.

```python
import glob
print(glob.glob("*"))
```
Many other Functionalities

- You can run other programs from within python
  ```python
  import os
  os.system(<run_program>)
  ```

- You can start python programs within bash scripts

- And many other things...
References

- http://www.diveintopython.net
  A full book about Python freely available for download
- http://openbookproject.net/thinkcs/python/english2e/
  „How to think like a computer scientist“
  With examples in Python!

More information:
- www.biopython.org
- www.stackoverflow.com