Introductory Linux Course

Python II

Martin Dahlö – UPPMAX

Author: Nina Fischer

Dept. for Cell and Molecular Biology, Uppsala University

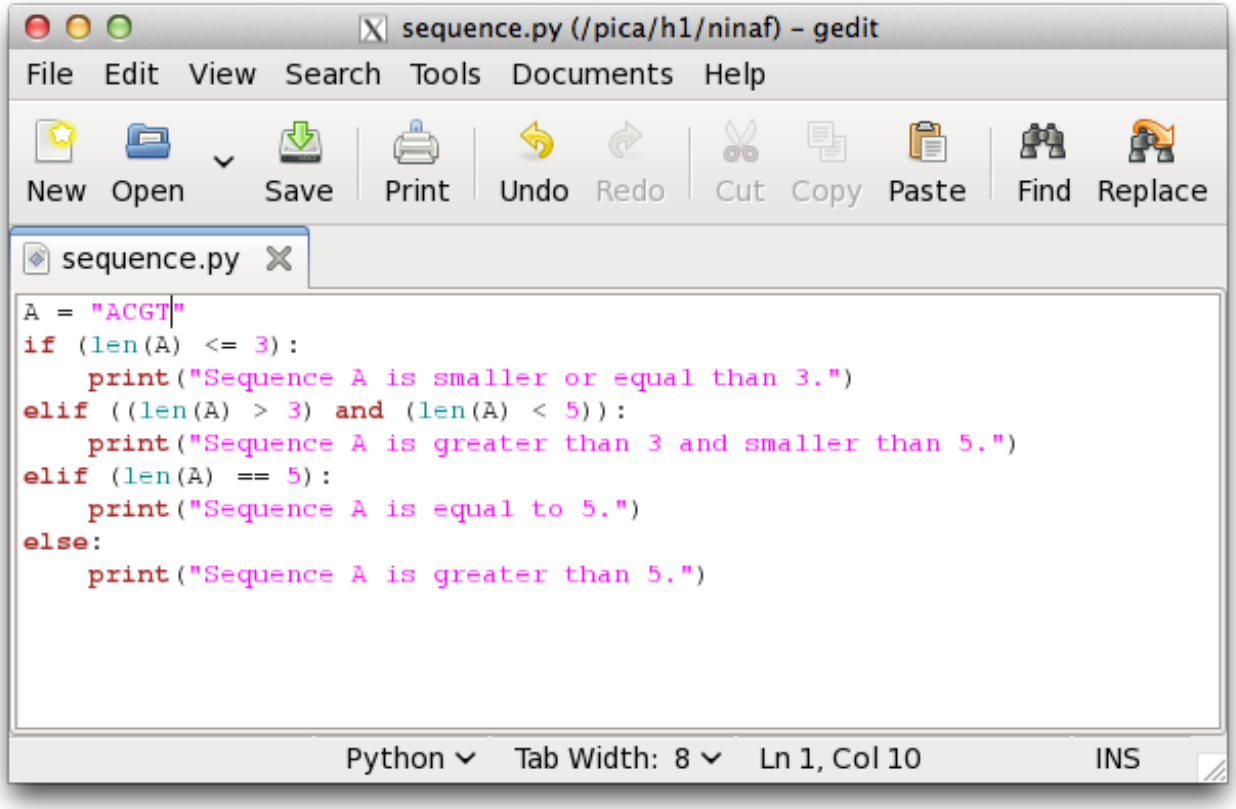
August, 2018

Outline

- Short recap
- Functions
- Similarity of sequences
- Modules
- Command line arguments
- Read and write files

Short Recap – Creating a Program

- An arbitrary editor (nano, gedit, emacs, vim, …) can be used to create the program code and save it as a file



Short Recap – X server

- Login:
  `$ ssh -X <username>@rackham.uppmax.uu.se`
- Open editor, e.g., gedit:
  `$ gedit sequence.py &`
- In the terminal a new line, which starts with **`$`** should appear. If not press **`<Ctrl C>`**
- Load module:
  `$ module load python3/3.6.0`
- 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"
      0123456

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
6 A
```

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**

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

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

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

```
GATCGTTCG
  ||  |||
CATGGTTGA
```

Similarity of Sequences

Distance function:

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

Example:

A = GATCGTTCG, B = CATGGTTGA

```
GATCGTTCG
  ||  |||
CATGGTTGA

100100011
```

$distance = 4$

Similarity of Sequences

```
# Simple calculation of distances
A = "GATCGTTCG"
B = "CATGGTTGA"
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)
```

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

Similarity of Sequences with Function

```
# Simple calculation of distances
A = "GATCGTTCG"
B = "CATGGTTGA"
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)
```

```
def d(a, b):
    distance = 0
    for i in range(len(a)):
        if (a[i] != b[i]):
            distance += 1
    return distance
```

Similarity of Sequences with Function

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

# Function d(a, b) for calculating the distance
def d(a, b):
    distance = 0
    for i in range(len(a)):
        if (a[i] != b[i]):
            distance += 1
    return distance

# Print the result
print("Distance of sequences A and B: ", d(A,B))
```

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

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

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

```
from math import pi
r = 5.6
print("Circumference: ", 2 * pi * r)
```

Modules

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

```
from math import pi
r = 5.6
print("Circumreference: ", 2 * pi * r)
```

```
import math
r = 5.6
print("Circumreference: ", 2 * math.pi * r)
```

- In module **`math`** π is already (with full accuracy) defined
- **`import`** loads its functionality from module **`math`**

Modules

- There exist many useful additional modules which can be imported with:

```
import <module_name>
```

- Examples:
  - `import math / from math import pi`
  - `import numpy`
  - `import Bio / from Bio import SeqIO`
  - …
- These modules often 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 (**`from x import y`**/ **`from Dist import d`**) or the entire module (**`import Dist`**)

Dist.py

```
# Module implemetation
def d(a, b):
    distance = 0
    for i in range(len(a)):
        if (a[i] != b[i]):
            distance += 1
    return distance
```

```
# Calculates distance
import Dist # loads module
A = "GATCGTTCG"
B = "CATGGTTGA"
dist = Dist.d(A,B)
```

Command Line Arguments

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

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

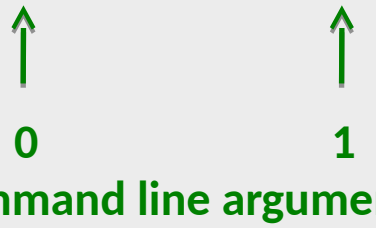
trivial2.py

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

sequence1.txt

```
GATCGTTCG
```

sequence2.txt

```
CATGGTTGA
```

Read and Write Files

- Program, which calculates distance between two sequences

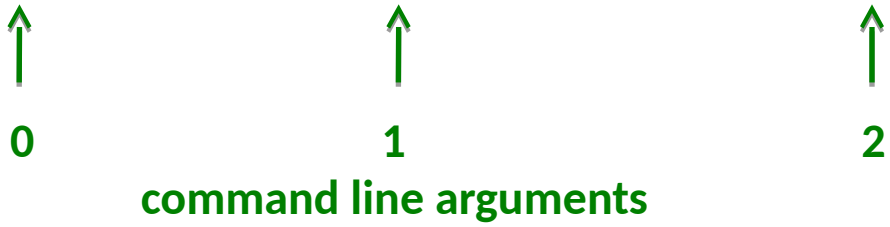
readSeq.py

```
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()

print("Distance between A and B is " + str(Dist.d(A,B)))
```

Execution of the Program

- In order to execute the program we call the interpreter from the command line using the python filename as first argument, and the sequence files as 2nd and 3rd arguments

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



Read and Write Files

- 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()

outputfile = open(sys.argv[3], 'w')
outputfile.write("Distance between A and B is " + str(Dist.d(A,B)))
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
```

Read and Write Files

```
import sys
import Dist

inputfile1 = open(sys.argv[1], 'r')
inputfile2 = open(sys.argv[2], 'r')
Asequences = inputfile1.readlines()
B = inputfile2.readline()
inputfile1.close()
inputfile2.close()

outputfile = open(sys.argv[3], 'w')
for A in Asequences:
    outputfile.write("Distance between “, A, ” and “, B, “ is " +
      str(Dist.d(A,B)) + "\n")

outputfile.close()
```

Read and Write Files

```
import sys
import Dist

inputfile1 = open(sys.argv[1], 'r')
inputfile2 = open(sys.argv[2], 'r')
Asequences = inputfile1.readlines()
Bsequences = inputfile2.readlines()
inputfile1.close()
inputfile2.close()

outputfile = open(sys.argv[3], 'w')
for A in Asequences:
    for B in Bsequences:
        outputfile.write("Distance between “, A, ” and “, B, “ is " +
          str(Dist.d(A,B)) + "\n")

outputfile.close()
```

Many other Functionalities

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

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

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

```
import glob
print(glob.glob("*"))
```

Many other Functionalities

- You can run other programs from within python

```
import os
os.system(<bash commands>)
```

- 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