Intro to Python programming

Andreas

Today

- command-line tutorial
- basic Python

Tomorrow

- advanced Python
- real-world examples

Command-line

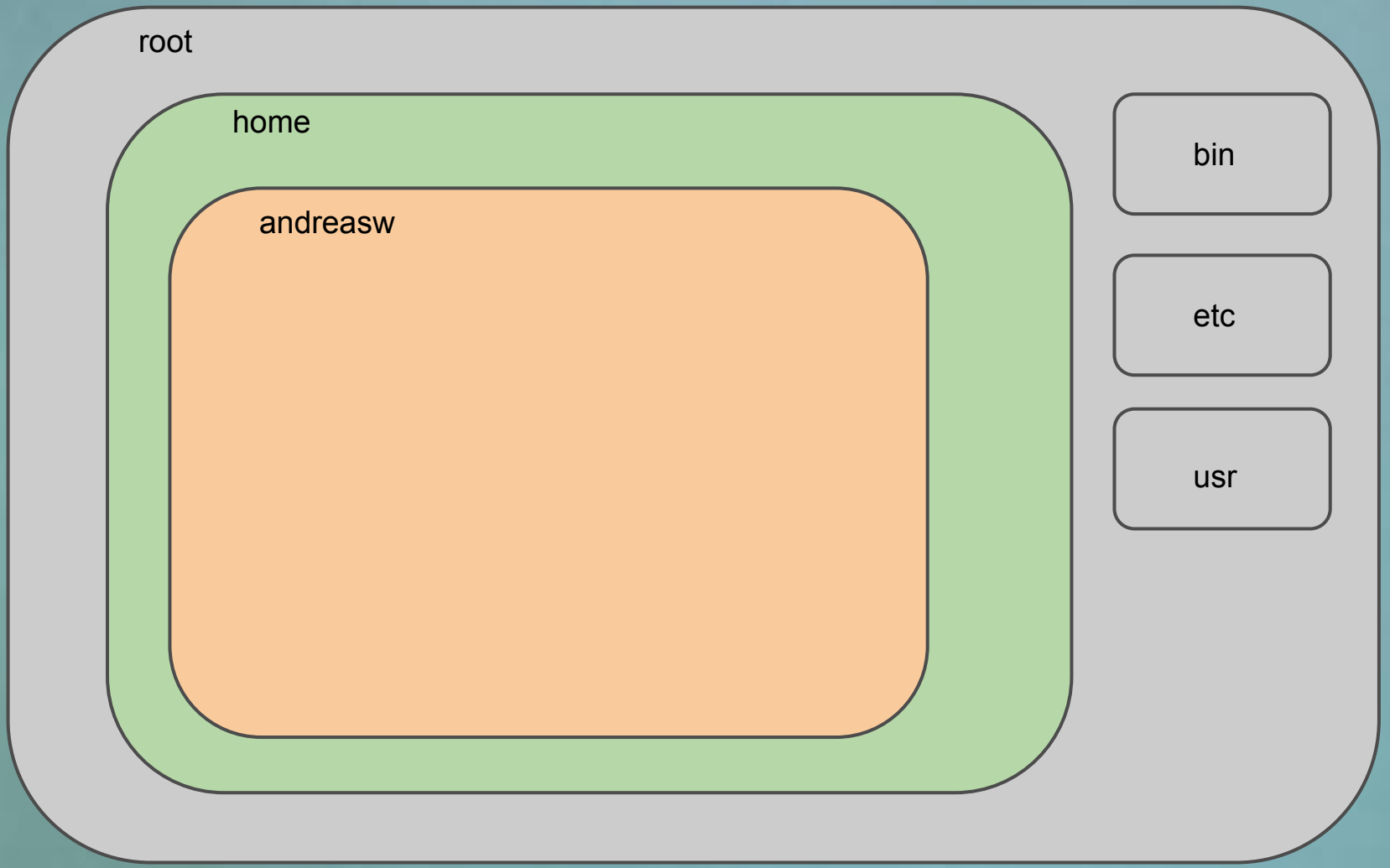
- text - based computer interface
- built-in to Linux and Mac OSX

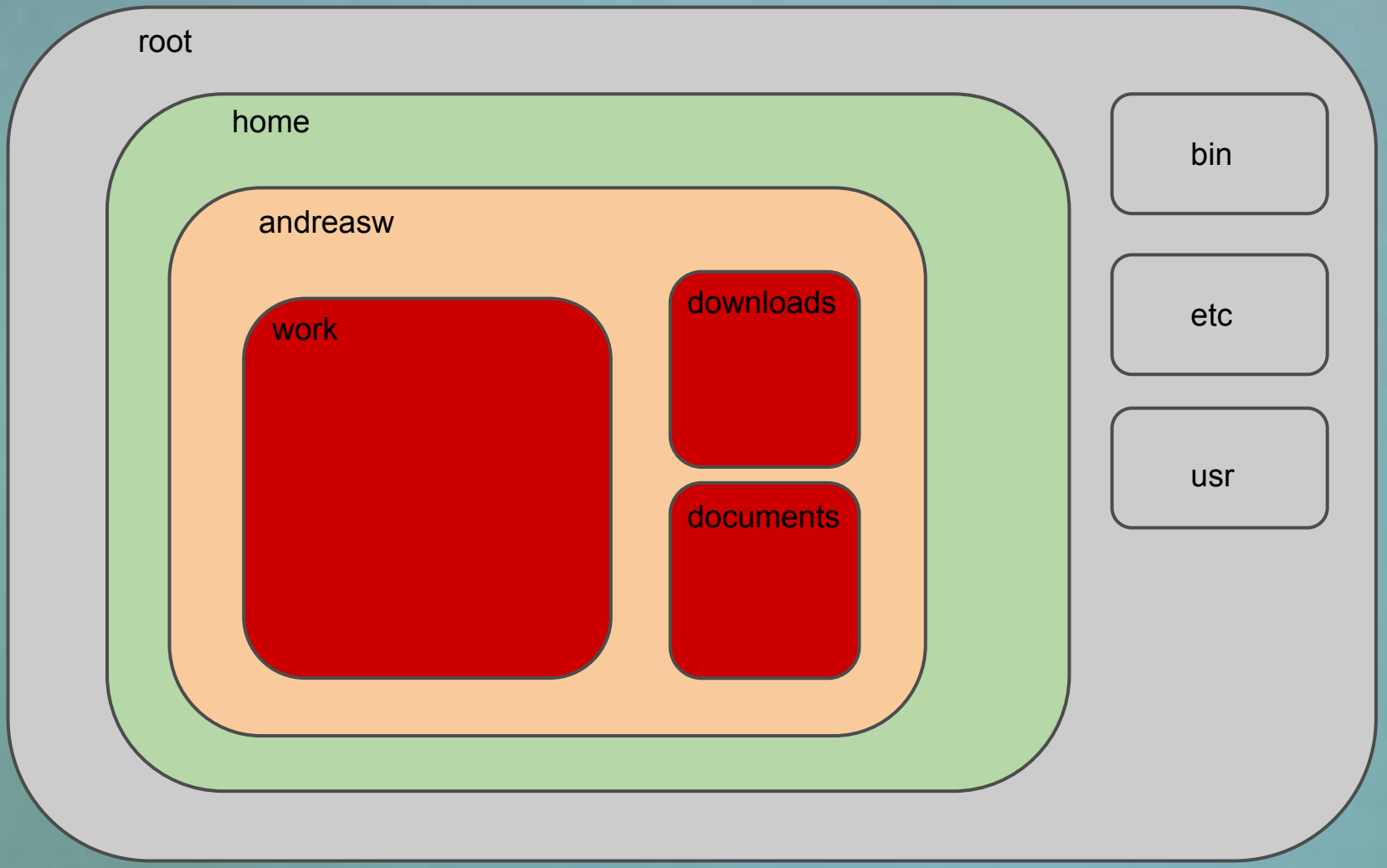
Why?

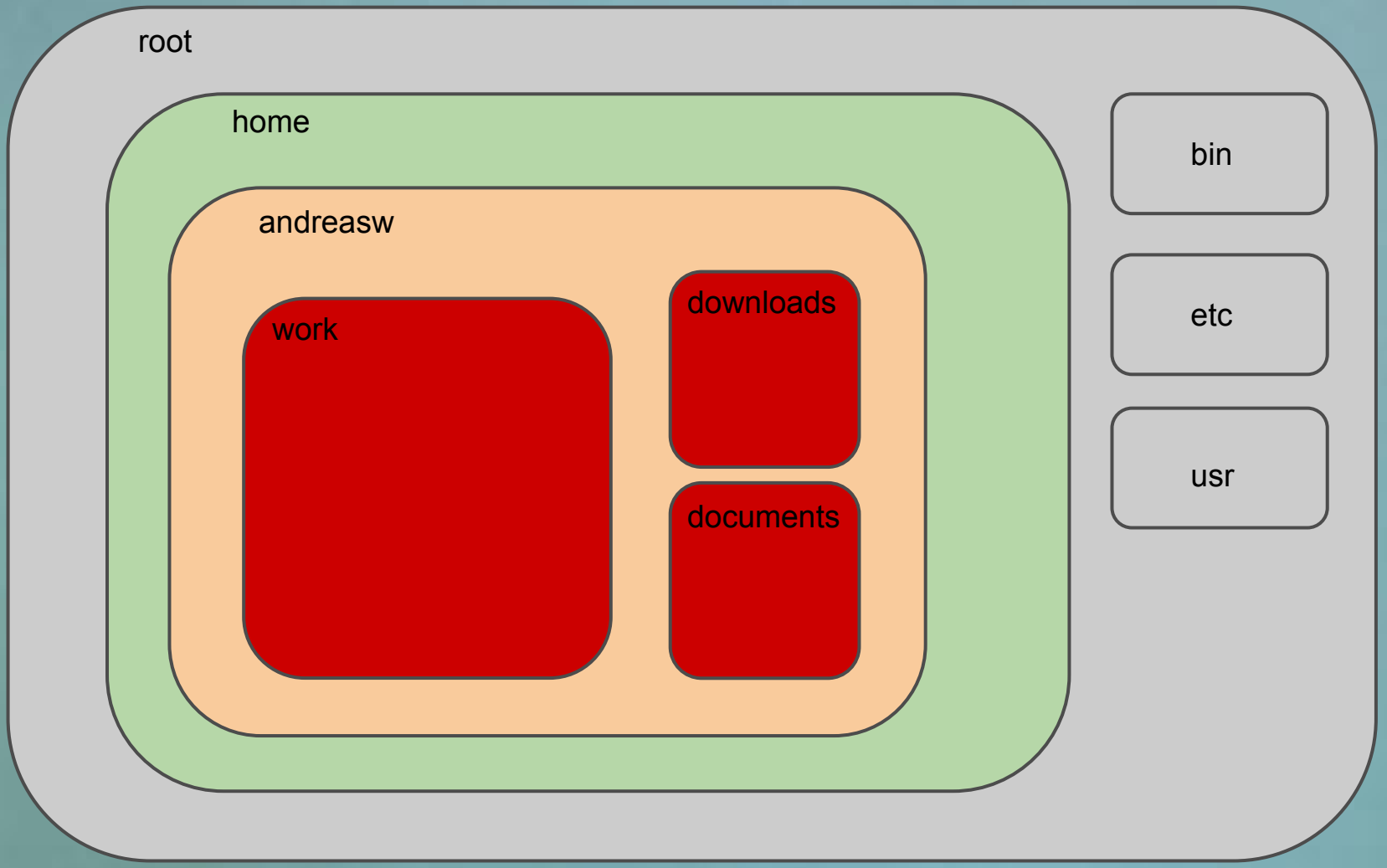
- more powerful commands
- batch commands
- handle huge files as stream
- automatable through e.g. Python
- many useful little programs already included

Navigating the file system

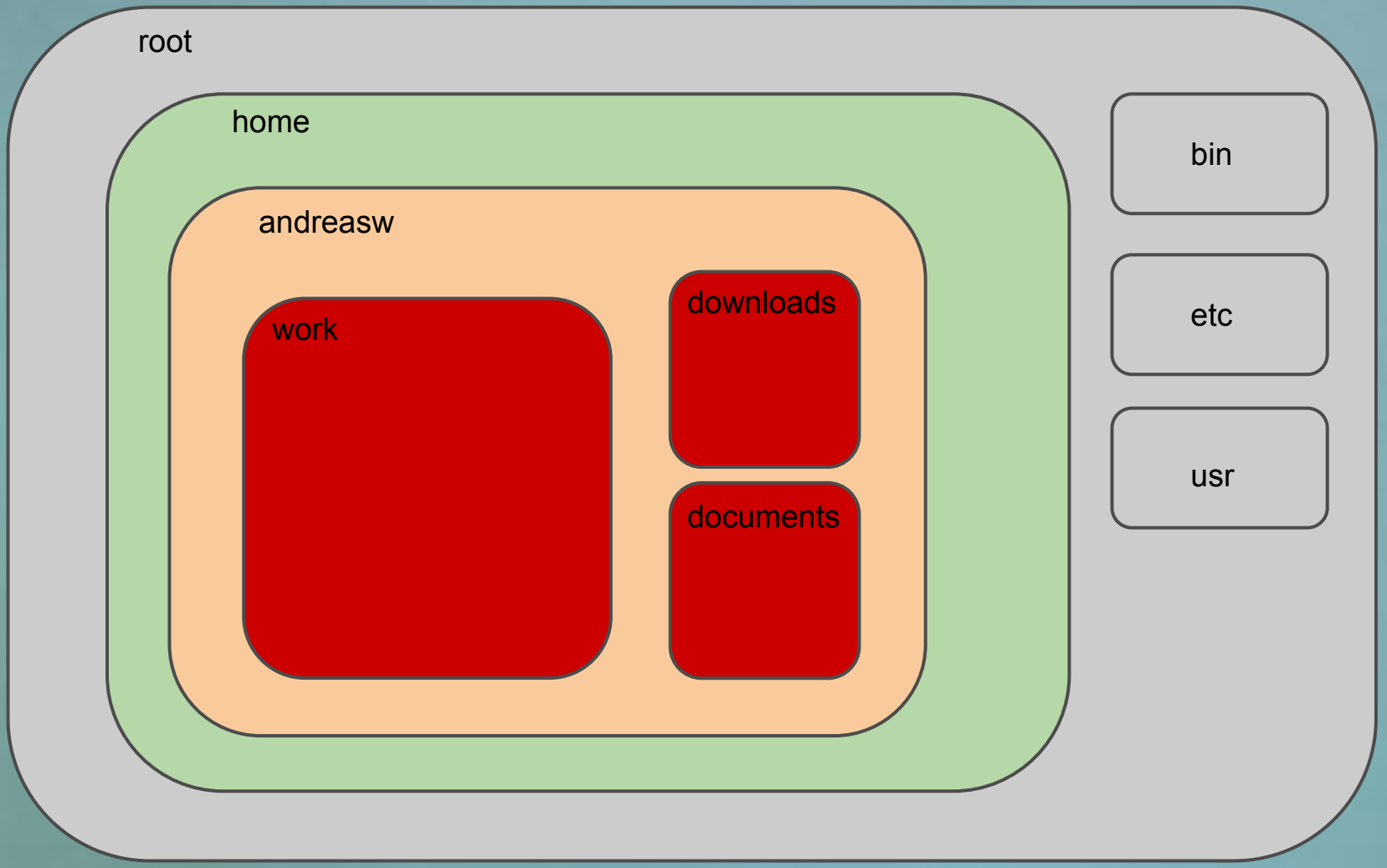

root
home
bin
etc
usr


root
home
andreasw
bin
etc
usr

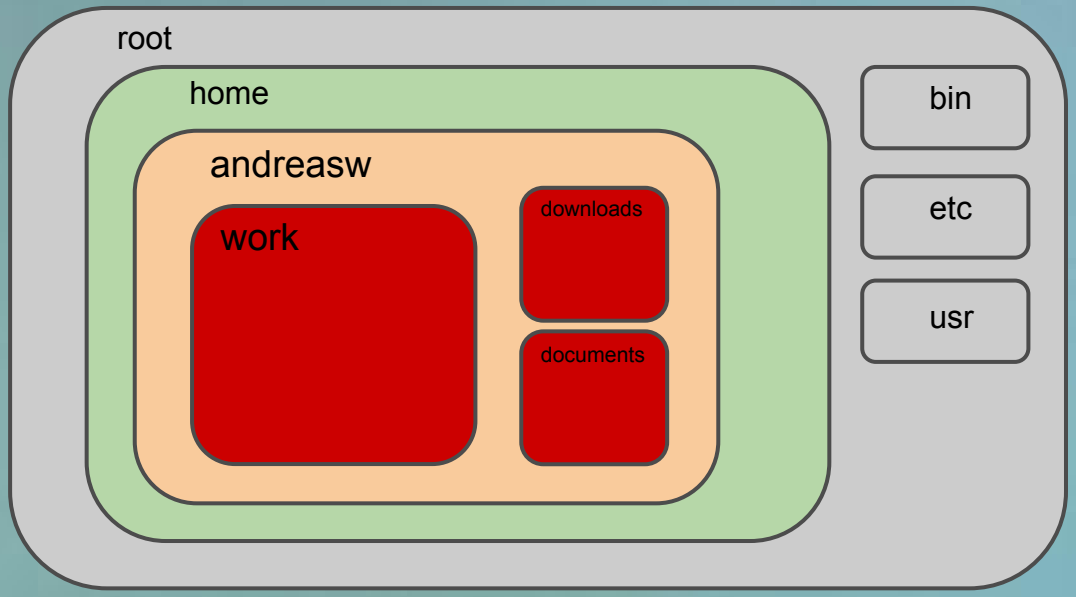

root
home
andreasw
work
downloads
documents
bin
etc
usr



- abbreviations:
  root: /
  andreasw: ~/
  current dir: ./
- absolute path: /home/andreasw/work/
- relative path: ~/work/
- autocomplete: [TAB]



- abbreviations:
  root: /
  andreasw: ~/
  current dir: ./
- absolute path: /home/andreasw/work/
- relative path: ~/work/
- autocomplete: [TAB]

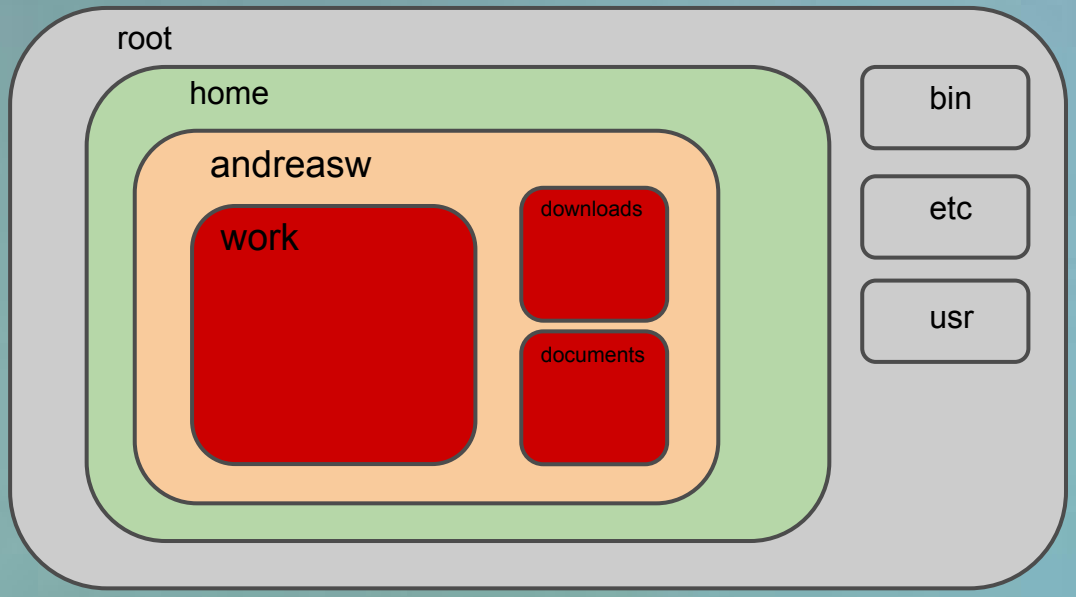


Explore your file system!
Whats in your home folder?
How many users are there on your system?
Try an absolute path to get to home!

- absolute path: /home/andreasw/work/
- relative path: ~/work/
- autocomplete: [TAB]

- move into "work": cd work
- move out of current folder: cd ..
- move back to home: cd
- get folder contents: ls

- abbreviations:
  root: /
  andreasw: ~/
  current dir: ./
  parent dir: ../



- absolute path: /home/andreasw/work/
- relative path: ~/work/
- autocomplete: [TAB]

- move into "work": cd work
- move out of current folder: cd ..
- move back to home: cd
- get folder contents: ls

- abbreviations:
  root: /
  andreasw: ~/
  current dir: ./
  parent dir: ../

Manipulating the file system

Moving

- cd (change directory)
- ls (list directory)
- pwc (print working directory)

Manipulating

- mkdir [dir] (make directory)
- rm -r [dir] (remove directory)
- touch [file] (create empty file)
- rm [file] (remove file)

Beware: there is no trash!

- cp [oldpath] [newpath] (copy file)
- mv [oldpath] [newpath] (move file)

Moving

- cd (change directory)
- ls (list directory)
- pwc (print working directory)

Create a folder at home
Create a file in it
Are they there?
Move the file one level up
Remove both

Go to root
Create a directory in root

Manipulating

- mkdir [dir] (make directory)
- rm -r [dir] (remove directory)
- touch [file] (create empty file)
- rm [file] (remove file)

Beware: there is no trash!

- cp [oldpath] [newpath] (copy file)
- mv [oldpath] [newpath] (move file)

Moving

- cd (change directory)
- ls (list directory)
- pwc (print working directory)

Create a "python_course" directory
Move the course files there
Unpack them with

tar xzvf [files]

Manipulating

- mkdir [dir] (make directory)
- rm -r [dir] (remove directory)
- touch [file] (create empty file)
- rm [file] (remove file)

Beware: there is no trash!

- cp [oldpath] [newpath] (copy file)
- mv [oldpath] [newpath] (move file)

Getting information about files

ls (list files)

- ls -l (long output)
- ls -h (human readable filesizes)

wc (wordcount)

- wc -l [file] (count lines in file)

man (manual)

- man ls (read manual on the 'ls' command)

ls (list files)

- ls -l (long output)
- ls -h (human readable filesizes)

wc (wordcount)

- wc -l [file] (count lines in file)

How big are your course files?
And how long?

Create a new file
How big and long is it?

man (manual)

- man ls (read manual on the 'ls' command)

cat (concatenate)

- cat [file] (cat file contents)
- cat [file1] [file2] (cat both files)
- cat [file1] [file2] > [file3] (cat both files into file3)

nano (word processor)

- nano [file] (open file to process)
- nano [filename] (creates new file)
- within nano: Ctrl + X to exit

less (simple file viewer)

- less [file] (open file to view)
- within less: Ctrl+C to exit

cat (concatenate)

- cat [file] (cat file contents)
- cat [file1] [file2] (cat both files)
- cat [file1] [file2] > [file3] (cat both files into file3)

nano (word processor)

- nano [file] (open file to process)
- nano [filename] (creates new file)
- within nano: Ctrl + X to exit

Open a new file in nano
Write something
Save it

Do it again for a 2nd file

Cat both into a new file

Open it with less

less (simple file viewer)

- less [file] (open file to view)
- within less: Ctrl+C to exit

The cool stuff:
batch manipulating files

head and tail (slice file)

- head [file] (print first 20 lines of file)
- head -n 5 [file] (print first 5 lines of file)
- same for tail

cut (extract columns)

- cut -f 2 [file] (print second column)
- cut -f 1,3- [file] (print 1st and all columns from 3rd)

head and tail (slice file)

- head [file] (print first 20 lines of file)
- head -n 5 [file] (print first 5 lines of file)
- same for tail

cut (extract columns)

- cut -f 2 [file] (print second column)
- cut -f 1,3- [file] (print 1st and all columns from 3rd)

Extract from "denovo_snps.vcf"
1 SNP = 1 line

Understand the contents
Try head and tail
Make a new file containing
only the Contigs

grep (search)

- grep [word] [file] (print lines having the word)
- grep -v [word] [file] (print lines not having the word)
- grep -P $"term" [file] (print lines with exactly the term)

| (pipe commands)

- send output of command1 as input to command2
- grep [term] [file] | wc -l (count lines having the term)

grep (search)

- grep [word] [file] (print lines having the word)
- grep -v [word] [file] (print lines not having the word)
- grep -P $"term" [file] (print lines with exactly the term)

| (pipe commands)

- send output of command1 as input to command2
- grep [term] [file] | wc -l (count lines having the term)

Analyze "denovo_snps.vcf"
1 SNP = 1 line

Any SNPs in an *unc*?
How many SNPs total ?
How many in exons?
How many in exons on Contig0?

awk (advanced pattern matching)

- awk '$2 > 10' (print lines where 2nd column > 10)
- awk '$2 == "T" ' (print lines where 2nd column is "T")

sort and uniq

- sort [file] (sort file according to 1st column)
- sort -k2n [file] (numerical sort file according to 2nd column)
- uniq (only print directly repeated lines)
- uniq -c (print counts of directly repeated lines)

awk (advanced pattern matching)

- awk '$2 > 10' (print lines where 2nd column > 10)
- awk '$2 == "T" ' (print lines where 2nd column is "T")

sort and uniq

- sort [file] (sort file according to 1st column)
- sort -k2n [file] (numerical sort file according to 2nd column)
- uniq (only print directly repeated lines)
- uniq -c (print counts of directly repeated lines)

Analyze "denovo_snps.vcf"

How many SNPs in "C"?
How many from "A" to "G"?
How many in MA_46?
Which MA has the most SNPs?

Basic Python

Outline

- quick intro

- types of objects
- comparisons
- assign names to variables
- manipulate strings

- container types: lists and dictionaries
- shorten code by using loops

- import foreign modules
- use advanced object methods
- read+ write files

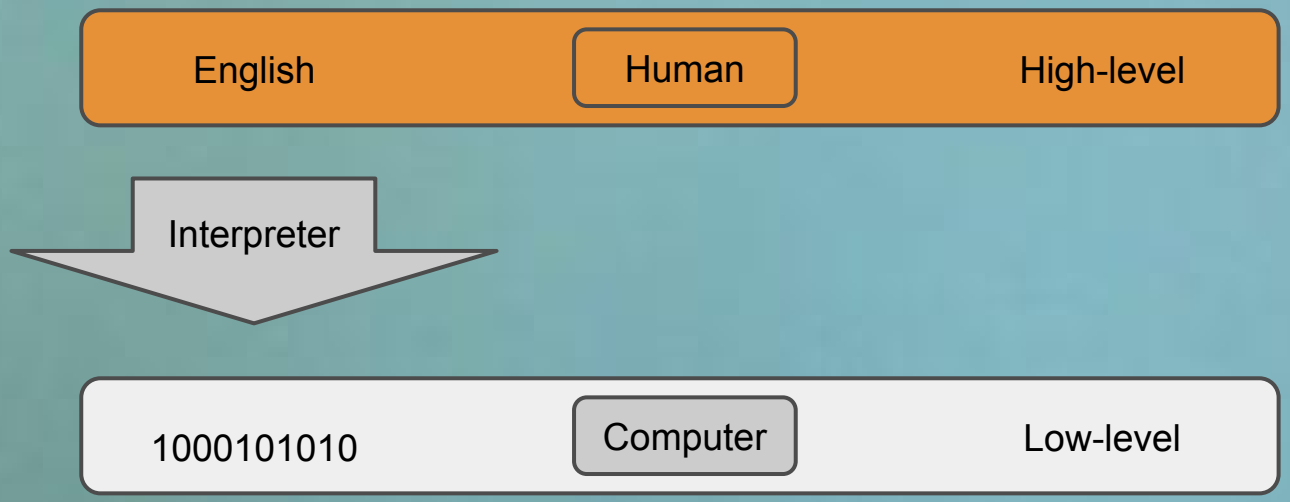
- real world examples 1 and 2

Python

- named after Monty Python
- code is in simple textfiles, compiling during execution
- high-level scripting language

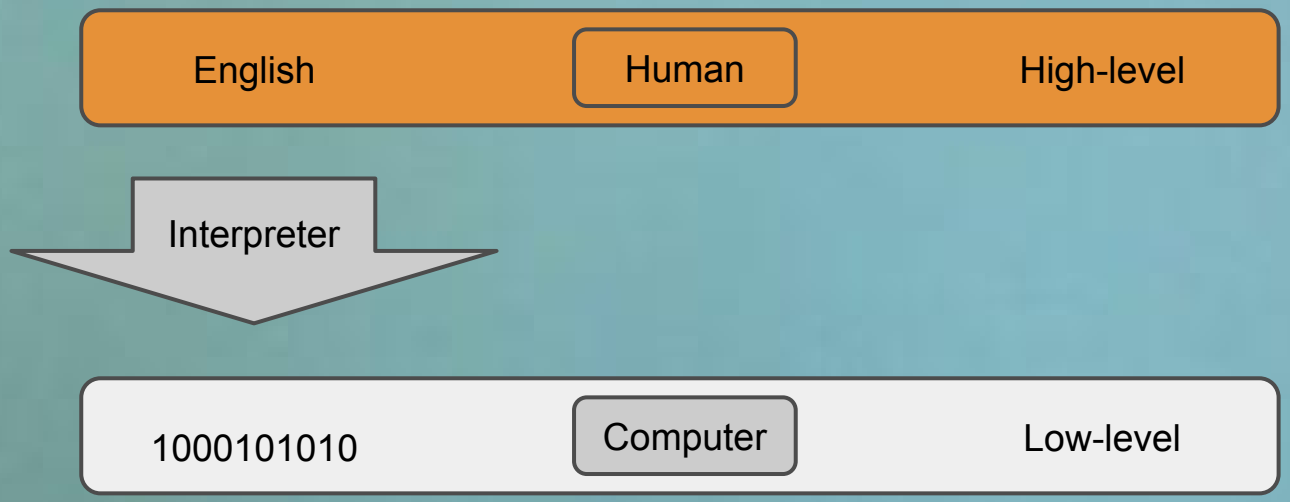
Python

- named after Monty Python
- code is in simple textfiles, compiling during execution
- high-level scripting language



Python

- named after Monty Python
- code is in simple textfiles, compiling during execution
- high-level scripting language



Pros: easy to learn, very readable code

Con: potentially slower than compiled languages

Types of objects

- each object in the program has a type

- a full number is of type int (integer)
- a decimal number is of type float

- a sequence of letters is of type string
- strings are always in "string"

Types of objects

- each object in the program has a type

- a full number is of type int (integer)
- a decimal number is of type float

- a sequence of letters is of type string
- strings are always in "string"

Type 'python' in bash

1+1
2*10

type(10)
type(10.0)
type("10")

10/3
10.0/3
10/0
"10" + "10"

Types of objects

- each object in the program has a type

- a full number is of type int (integer)
- a decimal number is of type float

- a sequence of letters is of type string
- strings are always in "string"
- strings are immutable

- a True/False switch is of the type bool (Boolean)
- 'if clauses' execute code dependent on a comparison

- Comparisons:

==	equal
!=	not equal
>	bigger
>=	bigger or equal
A in B	A contained in B

Types of objects

```
type("True")
type(True)

1 == 1
1 == "1"
1 != "1"

if 5 > 2: print "Its true!"
```

- each object in the program has a type

- a full number is of type int (integer)
- a decimal number is of type float

- a sequence of letters is of type string
- strings are always in "string"
- strings are immutable

- a True/False switch is of the type bool (Boolean)
- 'if clauses' execute code dependent on a comparison

- Comparisons:

==	equal
!=	not equal
>	bigger
>=	bigger or equal
A in B	A contained in B

Variables

- variables are names for objects

- a = 1
- myname = "Andreas"

- a = 1
- b = a

- myname = "Andreas"
- myname = myname + "Weller"
- myname += "Weller"

Variables

- variables are names for objects

- a = 1
- myname = "Andreas"

- a = 1
- b = a

- myname = "Andreas"
- myname = myname + "Weller"
- myname += "Weller"

a = 1
b = a
a = 2

What is b now?
What is the type of b?

Working with strings

- strings cannot be changed
- we can only create a new string from the old string

- all objects consisting of parts are iterable
- strings are iterable

A n d r e a s
0 1 2 3 4 5 6

Working with strings

- strings cannot be changed
- we can only create a new string from the old string

- all objects consisting of parts are iterable
- strings are iterable

A n d r e a s
0 1 2 3 4 5 6

- string indexing:

string[n]	retrieve the n-th element of the string
string[0:5]	from 0 to before 5
string[1:-1]	from 1 to before the last
string[::2]	from 0 to end, using every 2nd item

Working with strings

- strings cannot be changed
- we can only create a new string from the old string

- all objects consisting of parts are iterable
- strings are iterable

A n d r e a s
0 1 2 3 4 5 6

- string indexing:

string[n]	retrieve the n-th element of the string
string[0:5]	from 0 to before 5
string[1:-1]	from 1 to before the last
string[::2]	from 0 to end, using every 2nd item

Try it!

What is:
string[:-2]
string[::-1]
len(string)

string[2] = "A"
Is 1234 iterable?
Is "1234" iterable?

Hello world!

- python scripts are text files ending with .py
- lines beginning with # are ignored as comments
- print [object] (prints object to the commandline)

Open Komodo Edit
Open new file

write:
 # my very first script!
print "Hello World!"

Save it as hello_world.1.py

Execute as
python hello_world.1.py

Try also
python hello_world.1.py > output.txt

List

- variables often need to be grouped

- simplest container: list
- a list is an ordered, mutable collection of objects (of any type)
- lists have square brackets

List

- variables often need to be grouped

- simplest container: list
- a list is an ordered, mutable collection of objects (of any type)
- lists have square brackets

Open shell

letters = ["A", "B", "C"]

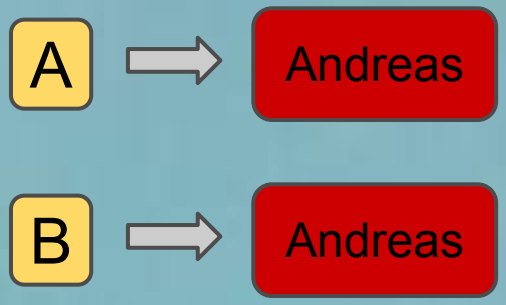
letters[0]
letters[2] = "D"

numbers = letters
numbers[0] = 1
What happened to letters?

Copy vs reference

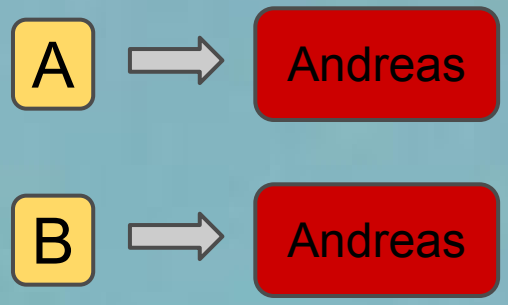
A = "Andreas"

B = A

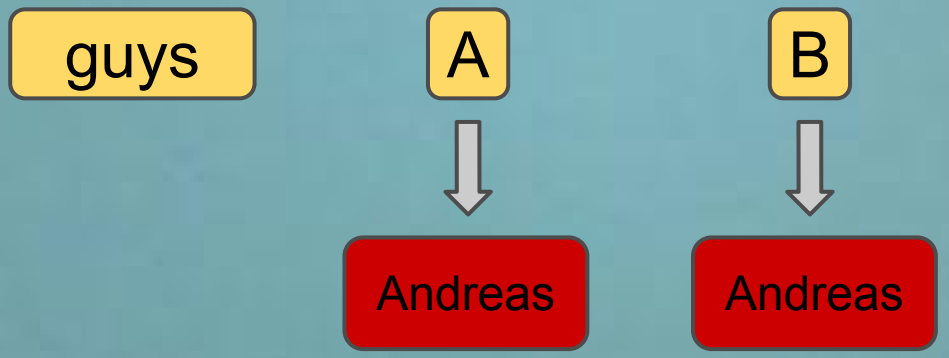


Copy vs reference

```
A = "Andreas"
B = A
```

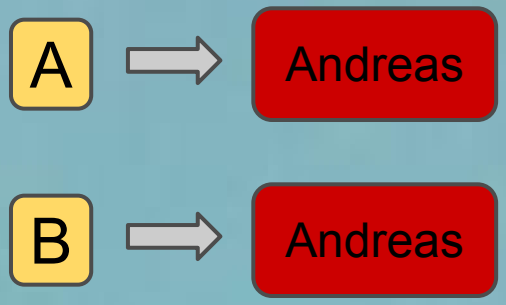


```
guys = [A, B]
students = guys
```

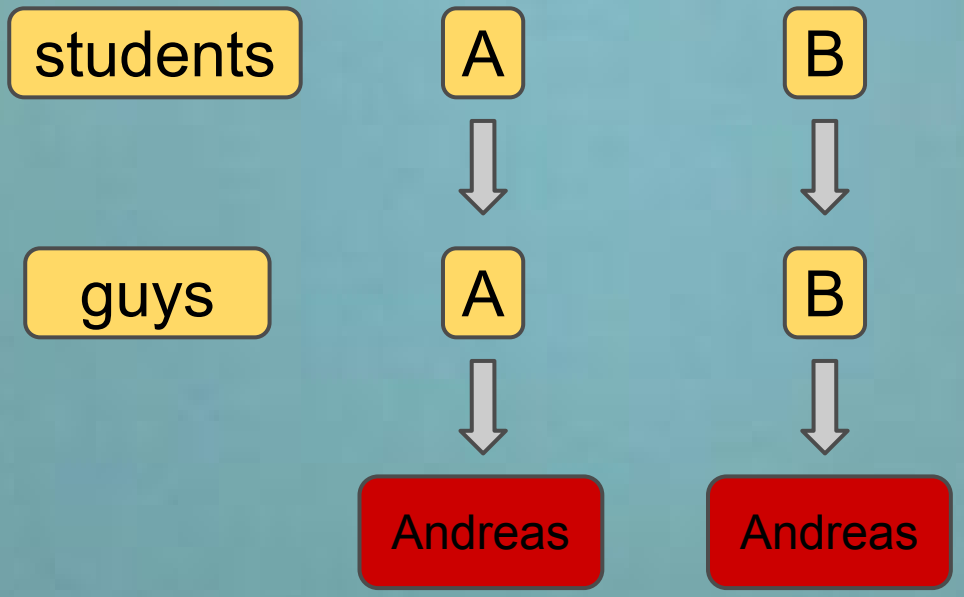


Copy vs reference

```
A = "Andreas"
B = A
```



```
guys = [A, B]
students = guys
```



loops

- loops carry out some code repeatedly, either
  - once for each element of an iterable (for-loop)
  - as long as a conditions is met (if-loop)

- a name for each element can be chosen freely
- loops start with a ':'
- the code to be looped is indented

loops

- loops carry out some code repeatedly, either
  - once for each element of an iterable (for-loop)
  - as long as a conditions is met (if-loop)

- a name for each element can be chosen freely
- loops start with a ':'
- the code to be looped is indented

```
people = ["Andreas", "Vlad", "Robbie"]
for name in people:
        print "Hello " + name
```

Save it as hello_world.2.py
Execute

Dictionary

- a dictionary is an unordered, mutable collection of key: value pairs

- keys must be unique strings or numbers
- values can be any object
- dictionaries have curled brackets

Dictionary

- a dictionary is an unordered, mutable collection of key: value pairs

- keys must be unique strings or numbers
- values can be any object
- dictionaries have curled brackets

Open shell

weight = {"A":70, "J":80}

weight["A"]
weight["V"] = 60

What happens if we do *weight["A"] = 100?*
What happens if we do *weight["R"] = 60?*

Summary

- the Python shell is nice to quickly test code
- scripts are text files ending in .py

- objects have a type, e.g. string, int, bool
- strings are immutable, but can be indexed with string[x:y]
- if statements are used to execute code dependent on a condition

- lists are ordered collections of any objects
- dictionaries are unordered pairs of:
  - unique keys
  - any objects as value

- if/while loops are used to compress repetitive code

Writing Python scripts

Objects and Methods

English:

- Object Verb (Subject)
  - Dog runs
  - Dog eats cat

Python:

- Object.Method(Argument)
  - Dog.runs()
  - Dog.eats(cat)

Every object has a unique set of methods.

Objects and Methods

English:

- Object Verb (Subject)
  - Dog runs
  - Dog eats cat

Python:

- Object.Method(Argument)
  - Dog.runs()
  - Dog.eats(cat)

Every object has a unique set of methods.

Open Komodo

name = "aaaaaa"
type name. and wait
what methods are there for strings?

list1 = ["aaa", "bbb"]
type list1. and wait
what methods are there for lists?

Important methods

String:

string.split("\t")	splits the string into a list (using tab as the delimiter)
string.strip()	removes whitespaces from start/end of the string
string.replace("old", "new")	replaces old with new in the string

Important methods

String:

string.split("\t")	splits the string into a list (using tab as the delimiter)
string.strip()	removes whitespaces from start/end of the string
string.replace("old", "new")	replaces old with new in the string

List:

list.sort()	sort the list
list.min()	return smallest item
list.sum()	return sum of items
"\t".join(list)	returns a string of list items with tabs between them

Important methods

String:

string.split("\t")	splits the string into a list (using tab as the delimiter)
string.strip()	removes whitespaces from start/end of the string
string.replace("old", "new")	replaces old with new in the string

List:

list.sort()	sort the list
list.min()	return smallest item
list.sum()	return sum of items
"\t".join(list)	returns a string of list items with tabs between them

Dict:

dict.keys()	returns a list of the keys
dict.values()	returns a list of the values

Modules

- modules provide extra functions
- modules are imported by "import module"

module sys (system)

- reads arguments after scriptname
- saves them as the list sys.argv

Modules

- modules provide extra functions
- modules are imported by "import module"

module sys (system)

- reads arguments after scriptname
- saves them as the list sys.argv

```
import sys
name = sys.argv[1]
print "hello " + name
```

save as hello_world.3.py
python hello_world.3.py yourname

Reading and writing files

- files are accessed as a file object in read mode
- files are then read line by line
- files can be written into a file object in write mode

```
import sys

input_name = sys.argv[1]
output_name = sys.argv[2]

output = open(output_name, "w")

with open(input_name, "r") as input:
        for row in input:
                print row,
                output.write(row)
```

Most common beginner mistakes

1. code first, think later
2. write too much code before testing it
3. don't cross-check results
4. mix up types of objects
5. use the wrong type of brackets: (), [], {}
6. forget the ':' when starting a loop
7. forget the brackets after an empty argument
8. get confused by nested loops
9. try to concatenate string and int in the output
10. unintentionally overwrite keys in dictionary

Real life example 1: reduce sequence length

- nexus files are important for phylogenetics
- the main part consists of rows like:
  - RS2333 AGTTCGATGCTGTGATTGTAG

- goal: reduce the sequence lengths to a target number

Real life example 1: reduce sequence length

- nexus files are important for phylogenetics
- the main part consists of rows like:
  - RS2333 AGTTCGATGCTGTGATTGTAG

- goal: reduce the sequence lengths to a target number

parse a number as a commandline argument

read the contents of a nexus file
split each row into contents

if there are 2 parts (= a row with sequence info):
    reduce the second part (the sequence) to the target size
    combine it to a string again
    print the string

if there are more parts:
    print the row

Real life example 1: reduce sequence length

- nexus files are important for phylogenetics
- the main part consists of rows like:
  - RS2333 AGTTCGATGCTGTGATTGTAG

- goal: reduce the sequence lengths to a target number

```
import sys

size = int(sys.argv[1])
filename = sys.argv[2]

with open(filename, "r") as nexus:
    for row in nexus:
        row = row.strip("\n")
        fields = row.split()

        if len(fields) == 2:
            name = fields[0]
            seq = fields[1]
            seq = seq[:size]
            result = name + "\t" + seq
            print result

        else:
            print row,
```

Real life example 2: find upregulated genes

- gene_upregulation.txt consists of gene ID and p-value
- gene_names.txt consists of ID and homolog

- Goal: get all homologs for genes with $p < 0.05$

Real life example 2: find upregulated genes

- gene_upregulation.txt consists if gene ID and pvalue
- gene_names.txt consists of ID and homolog

- Goal: get all homologs for genes with $p < 0.05$

read gene_names.txt
split each row
save ID: homolog pairs into a dictionary

read gene_upregulation.txt
split each row
test if p is below 0.05

if yes: get homolog for that ID from the dictionary

Real life example 2: find upregulated genes

- gene_upregulation.txt consists if gene ID and pvalue
- gene_names.txt consists of ID and homolog

- Goal: get all homologs for genes with p < 0.05

```
import sys
homolog_dict = {}

with open("gene_names.txt", "r") as names:
    for row in names:

        row = row.strip("\n")
        fields = row.split()
        if len(fields) > 3:

            gene_id = fields[3]
            gene_homolog = fields[1]
            homolog_dict[gene_id] = gene_homolog

#########################################################################

with open("gene_upregulation.txt", "r") as upregulation:
    for row in upregulation:

        row = row.strip("\n")
        fields = row.split()
        gene_id = fields[0]

        if gene_id in homolog_dict:

            p_value = float(fields[2])
            if p_value < 0.05:
                homolog = homolog_dict[gene_id]
                print homolog +"\t"+ str(p_value)
```