DA2004/DA2005: Labs

Lars Arvestad, Evan Cavallo, Christian Helanow, Anders Mörtberg, Kristoffer Sahlin

Contents

Lab rules

- All deadlines are **strict**. If you miss a deadline, you must redo the lab or project in the next interation of the course. Contact the course leader if this happens.
- If you know that you will not be able to finish a lab or project (for a legitimate reason such as sickness), inform the course leader **before** the deadline. If you only contact the course leader after the deadline, your lab will be counted as missed and you will have to hand it in during the next iteration of the course.
- You must work individually on the labs and the project. This means that you should write your own code and find your own solutions. You should not give solutions to each other or copy code from the Internet. All submissions are compared automatically and suspected cheating will be reported to the university's disiciplinary board, which can lead to suspension.
- You can search the Internet for Python commands, syntax, error messages, etc, but not for ready-made solutions. Search in English to get the most answers. You can find answers to many programming questions on the website https://stackoverflow.com/.
- Do not use functions from any library in a lab (that is, do not use `import` anywhere) unless you are given explicit permission.

- If you have a problem with anything other than programming itself, for example if you have a problem using the terminal, conflicts between libraries, etc, contact one of the teachers for help.
- Yor solutions should have a sensible structure. For example, you should not submit extremely long programs with the same copied code over and over instead of a loop. Unnecessarily complicated code can lead to loss of points.

Points

- Every lab is worth 10 points.
- You need to earn **at least 5 points per lab and a total of at least 25 points** to receive credit for the lab course.
- If you earn at least 35 points you will receive 1 bonus point on the project, and if you earn at least 45 points you will receive 2 bonus points.

Submission

- Submissions must be written in Python, not any other programming language.
- Solutions should be submitted in the form of `.py`-files. No file format other than `.py` will be accepted unless specified explicitly in the instructions.
- Solutions must be written in Python 3. It is not permitted to submit Python 2 code.
- Your solution should give the correct output for a given input.

Peer grading on PeerGrade

- Peer grading is a mandatory part of each lab. If you do not peer grade, you will lose 2 points on the lab.
- Be polite and give constructive feedback on others' code when you peer-grade. If you have a problem, contact the course leader.

Oral presentation

- In most cases no presentation of your submission is necessary, but if something is unclear we can request an oral presentation.
- In a oral presentation, you should be able to answer questions about your solution.

Lab 1

Temperature conversion

The main idea for this lab to get you familiar with the development environment either in the classrooms or on your own computer.

To receive credit for the lab, you must earn at least 5 of 10 points.

1.1 Learning goals

You should be able to write, run, and make changes in a small Python program.

1.2 Task

The program `convert.py` in the folder "Code for labs" on the course site is designed to ask for a temperature in Fahrenheit, read it from the user, convert it to the Celsius scale, and then print out the result. Here are your tasks:

1. Download, test-run, and study the `convert.py` program from the course site (see the file "Code for labs"). Does it work as intended?
2. Rewrite the function `fahrenheit_to_celsius` so that it calculates correctly. (2 points)
3. Extend the program with a function that converts from Celsius to Fahrenheit. (2 points)
4. Extend the program so that it asks which conversion you want to do. (2 points)
5. Extend the program so that it continues to ask for conversions until the user exits by entering `q` (short for `quit`). (2 points)
6. Extend the program so that it can also convert to and from Kelvin, both from and to Celsius and Fahrenheit. For full credit, the program should continue asking which conversion to do until the user inputs `q`. (2 points)
7. Submit your solution on PeerGrade.io. If you have not registered yet, do so using the signup code found on the course site. **Important:** use the *same name* that you are registered for the course with when you register on PeerGrade!
8. Peer review others' solutions on PeerGrade.io! This will become possible after the submission deadline.

Remember: it is always good to comment your code where necessary (to clarify the purpose of a line of code or a code block) and to document functions. You should also test all the functions you write carefully so that you know they run as you expect!

Lab 2

Polynomials

In this lab, we will represent polynomials using lists of coefficients. The word “representation” here means approximately “way of storing in a computer”. Concretely, we will represent a polynomial like `1+3x+7x^2` in Python as the list `[1,3,7]`. In general, we will store the coefficient of degree `n` in the list’s `n`th entry.

2.1 Learning goals

- You will see how to give an abstract concept (like a polynomial) a concrete representation in a computer and how to do calculations with those representations.
- You should be able to write small, simple functions.
- You should be able to work with the data structure `list`.

2.2 Submission

The lab should be submitted as usual via PeerGrade. Don’t forget to peer-grade after the submission deadline!

Remember: it’s always a good idea to add a comment where it would help explain the purpose of a line or code block to someone reading your code. Likewise, document your functions!

To receive credit for the lab, you must earn at least 5 of 10 points.

2.3 Tasks

In this lab, we will represent polynomials as lists. Below are some more examples:

Polynomial	Python representation
x^4	`[0, 0, 0, 0, 1]`
$4x^2 + 5x^3$	`[0, 0, 4, 5]`
$5 + 4x + 3x^2 + 2x^3 + x^4$	`[5,4,3,2,1]`

Obs: you can assume that the lists you work with only contain numbers.

The Python file `polynom.py` (see "Code for labs" on the course site) contains a function, `poly_to_string`, which converts a polynomial represented by a list to a string.

To get started with the lab, create a file `labb2.py` and copy over the function `poly_to_string`. Complete the tasks below by adding the necessary code to solve the specified problems. As you will see, you will also need to modify the function `poly_to_string` (see task 2).

Obs: you may not use functions from any library in the lab, i.e. you should not use the "import" keyword anywhere in your solution.

2.3.1 Task 1 (0 points, but necessary for tests later in lab)

Suppose that the polynomials `p` and `q` are defined as below.

$$\begin{aligned} p &:= 2 + x^2 \\ q &:= -2 + x + x^4 \end{aligned}$$

Write Python code to store list representations for these two polynomials in the variables `p` and `q`. That is, write

```
p = [...]
q = [...]
```

with the contents of the lists filled in. Check that you encoded the polynomials correctly using `poly_to_string`. You should get the following:

```
>>> poly_to_string(p)
'2 + 0x + 1x^2'

>>> poly_to_string(q)
'-2 + 1x + 0x^2 + 0x^3 + 1x^4'
```

Here, we write >>> for the "prompt" of the Python interpreter: `poly_to_string(p)` is a command that should be run by Python, and the lines after are the results. The prompt may look different depending on your computer and interpreter. You can also check the results by adding

```
print(poly_to_string(p))
```

to your file and running it, then observing what is printed in the console. In this case you should not see `'2 + 0x + 1x^2'` but rather `2 + 0x + 1x^2` (without surrounding quote marks). This type of test is very useful when developing code, but you should make sure to remove any `print` calls used for testing in the final code you hand in. It can however be good to add comments with tests and expected results to make it easy for someone reading your code to go through and test it.

2.3.2 Task 2 (3 points)

Edit the `poly_to_string` function so that:

- The empty list is converted to `0`.
- Terms with coefficent 1 are written without a coefficient. For example, `1x^2` should instead be written `x^2`.

- Terms with coefficient `-1` add a minus before the term, but the `1` is not written. For example `2 + -1x^2` should instead be `2 + -x^2`.
- Terms with coefficient `0` are not written. For example, `0 + 0x + 2x^2` should be simplified to `2x^2`.
- A list that contains only `0`s as elements, for example `[0, 0, 0]`, should be written as `0`.

Test your function! Here are some examples with the expected output:

```
>>> poly_to_string(p)
'2 + x^2'

>>> poly_to_string(q)
'-2 + x + x^4'

>>> poly_to_string([])
'0'

>>> poly_to_string([0,0,0])
'0'

>>> poly_to_string([1,2,3])
'1 + 2x + 3x^2'

>>> poly_to_string([-1, 2, -3])
'-1 + 2x + -3x^2'

>>> poly_to_string([1,1,-1])
'1 + x + -x^2'
```

2.3.3 Task 3 (2 points)

a) Write a function `drop_zeroes` that removes all zeros at the end of a polynomial and **returns** the result. *Tips*: use a `while`-loop and the `pop()` function.

```
def drop_zeroes(p_list):
    # here be code
```

Define some polynomials with zeroes at the end, for example:

```
p0 = [2,0,1,0]       # 2 + x^2 + 0x^3
q0 = [0,0,0]         # 0 + 0x + 0x2
```

and use these to test your function:

```
>>> drop_zeroes(p0)
[2, 0, 1]
>>> drop_zeroes(q0)
[]
>>> drop_zeroes([])
[]
```

b) Write a function that tests when two polynomials are equal by ignoring all zeroes at the end and then comparing the rest for equality:

```
def eq_poly(p_list,q_list):
   # here be code
```

Example tests:

```
>>> eq_poly(p,p0)
True
>>> eq_poly(q,p0)
False
>>> eq_poly(q0,[])
True
```

Obs: The functions `drop_zeroes` and `eq_poly` should **return** their results, not just print them out. The difference can be difficult to understand for a beginner, since the result may look the same when you run the code, but there is a big difference between a function that returns something and one that just prints something. See the end of section 2.5.1 in the compendium for more on this.

Note that the code you were given for `poly_to_string` returned the result as a string. Does your solution to Task 2 look similar? If not, go back and make sure you are returning.

2.3.4 Task 4 (2 points)

Write a function named `eval_poly` that takes a polynomial and a value in a variable `x` and **returns** the polynomials value at the point `x`.

Suggestions for your algorithm:[1]

- Iterate over the polynomial's terms by iterating over the coefficients.
- Keep track of the degree of the current term and the sum of the values of the terms you have seen so far. In each step of the iteration, calculate the value of the term as `coeff * x ** degree` (recall that `**` is exponentiation). Then add the term's value to the sum.
- When you have finished interating, return the final sum.

Sample tests:

```
>>> eval_poly(p,0)
2
>>> eval_poly(p,1)
3
>>> eval_poly(p,2)
6
>>> eval_poly(q,2)
16
>>> eval_poly(q,-2)
12
```

2.3.5 Task 5 (3 points)

a) Define negation of polynomials (that is, flip the sign of all coefficients and **return** the result).

[1]The following algorithm is not the most efficient one. If you want to optimize, you can instead implement Horner's method: https://en.wikipedia.org/wiki/Horner%27s_method

```
def neg_poly(p_list):
    # here be code
```

b) Define addition of polynomials (that is, add the coefficients and **return** the result).

```
def add_poly(p_list,q_list):
    # here be code
```

c) Define subtraction of polynomials.

```
def sub_poly(p_list,q_list):
    # here be code
```

Tips: notice that `p - q` can be defined as `p + (- q)`. That is, to subtract the polynomial `q` from `p` you can first take the negation of `q` and then add it to `p`.

Sample tests:

```
# p + q = q + p
>>> eq_poly(add_poly(p,q),add_poly(q,p))
True

# p - p = 0
>>> eq_poly(sub_poly(p,p),[])
True

# p - (- q) = p + q
>>> eq_poly(sub_poly(p,neg_poly(q)),add_poly(p,q))
True

# p + p != 0
>>> eq_poly(add_poly(p,p),[])
False

# p - q = 4-x+x^2-x^4
>>> eq_poly(sub_poly(p,q),[4, -1, 1, 0, -1])
True

# (p + q)(12) = p(12) + q(12)
>>> eval_poly(add_poly(p,q),12) == eval_poly(p,12) + eval_poly(q,12)
True
```

Obs: The comments are only there to explain what these tests are testing. Can you think of more good tests that might turn up in your code?

2.3.6 Task 6 (0 points)

Read through, clean up, and document your code. In order that grading be objective, you should not include your name in the file you hand in.

Tips: read through "Basic principles of programming" under Resources on the course site for recommendations on how to write good code.

Lab 3

Dictionaries, file handling, and error handling

3.1 Learning goals

- You should be able to work with dictionaries.
- You should be able to read and write data from files.
- You should be able to raise and handle errors.

3.2 Submission

The lab should be submitted as usual via PeerGrade. Don't forget to peer-grade after the submission deadline!

Remember: it's always a good idea to add a comment where it would help explain the purpose of a line or code block to someone reading your code. Likewise, document your functions so that a reader can easily understand what inputs they take and what the function does.

You may not use functions from any library in the lab, i.e. you should not use the "import" keyword anywhere in your solution. Submit all your code in a single file as in previous labs: functions first, then the main program that calls the functions.

Remember to remove your name from your code if for example Spyder adds it to your file.

3.3 Tasks

The goal of this lab is to create a program that loads information about a music library from a file and helps the user create playlists from their library.

Our music library will be text files in a certain format. A typical music library might look like this (`80s_library.txt` in the "Code for labs" file on the course site):

```
Kate Bush,Get Out of My House,5:51
Kate Bush,Rocket's Tail,4:07
```

```
Talking Heads,Burning Down The House,4:01
Talking Heads,Houses in Motion,4:33
Talking Heads,Moon Rocks,5:45
```

A library consists of several lines of text, each of which has an artist name, song name, and song length separated by commas (`,`). The length of a song is stored as a number of minutes followed by a number of seconds, separated by a colon (`:`).

Note: This is not a perfect format for storing a music library—how would we store a song with a comma in its name? Most "comma-separated value" data formats have ways to get around this issue (see Wikipedia or the Python `csv` library), but we will keep things simple for this lab.

3.3.1 Task 1: Parsing and printing song lengths (1 point)

Our data format stores song lengths in a human-readable string format, but if we want to calculate with song lengths, it will be more useful to store a length as a single integer, the total number of seconds. Here is a function that converts an integer to a string using Python's *f-strings*:

```python
def seconds_to_string(seconds):
    return f"{seconds // 60}:{seconds % 60:02d}"
```

For example, `seconds_to_string(101)` returns the string `'1:41'`. Write a function `string_to_seconds(s)` that does the opposite conversion: given a string in the minute-second format, it should return the total number of seconds as an integer. If given a string *not* in this format, it should instead raise a `ValueError`. A minute-second string is correctly formatted when it consists of a non-negative integer followed by a colon (`:`) followed by two digits that make a non-negative integer less than `60`.

Sample tests:

```
>>> string_to_seconds('1:41')
101
>>> string_to_seconds('207:53')
12473
>>> string_to_seconds('0:00')
0
>>> string_to_seconds('4:5')
Traceback (most recent call last):
  ...
ValueError
>>> string_to_seconds('2:70')
Traceback (most recent call last):
  ...
ValueError
>>> string_to_seconds('hej')
Traceback (most recent call last):
  ...
ValueError
```

3.3.2 Task 2: Loading a library from a file (4 points)

When we load a library into Python, we will store it in hierarchical, dictionary-based format. A library will be represented as a dictionary with artist names as keys. The value of the dictionary at an artist will be *another* dictionary with that artist's songs as keys and their lengths (as integers) as values.

a) Write a function `read_library` that takes a filename string as an argument, reads the library stored in that file, and converts it to a dictionary of the form described above. (2 points)

Your function should have the following behavior in exceptional cases:

- If the file does not exist, the function should raise a `FileNotFoundError`.
- If the input file is malformed (if some line does not have 3 comma-separated entries or the time is not in minutes-second format), then the function should raise a `ValueError`.

You can assume that the same song does not occur twice in the library file.

Sample test:

```
>>> read_library('80s_library.txt')
{'Talking Heads': {
  'Burning Down The House': 241,
  'Houses in Motion': 273,
  'Moon Rocks': 345
  },
 'Kate Bush': {
  'Get Out of My House': 351,
  "Rocket's Tail": 247}
}
```

Note: you can assume that the input file ends in a newline (`'\n'`) character.

b) Write a function `print_library(lib)` that takes a library in dictionary format and *prints* its contents organized by artist. In addition to individual song information, your function should print out the total number of songs and total length for each artist and for the entire library, as shown in the example below. (2 points)

Sample execution:

```
>>> lib = read_library('80s_library.txt')
>>> print_library(lib)
Talking Heads (3 songs, 14:19)
- Burning Down The House (4:01)
- Houses in Motion (4:33)
- Moon Rocks (5:45)
Kate Bush (2 songs, 9:58)
- Get Out of My House (5:51)
- Rocket's Tail (4:07)
Total: 5 songs, 24:17
```

3.3.3 Task 3: Creating playlists (3 points)

Next, we'll write a function to create a playlist from a music library based on a theme (such as `House` or `Rock` music).

First, we need to decide on a format for playlists. Unlike a library, a playlist is meant to be played in order, so we'll represent it as a list of tuples. Each tuple will have three elements: one for the artist, one for the song, and one for the song's length (as an integer). Here's an example of a playlist:

```
[('Talking Heads', 'Houses in Motion', 273),
 ('Talking Heads', 'Moon Rocks', 345)]
```

a) Write a function `make_playlist(library, theme)` that takes a library (in dictionary form) and a "theme" string. Your function return a playlist consisting of all songs in the library whose names contain the theme as a substring. If there are *no* songs matching the theme, `make_playlist` should raise a `ValueError`. (2 points)

Sample execution:

```
>>> lib = read_library('80s_library.txt')
>>> make_playlist(lib, "House")
[('Talking Heads', 'Burning Down The House', 241),
 ('Talking Heads', 'Houses in Motion', 273),
 ('Kate Bush', 'Get Out of My House', 351)]
>>> make_playlist(lib, "Rock")
[('Talking Heads', 'Moon Rocks', 345),
 ('Kate Bush', "Rocket's Tail", 247)]
>>> make_playlist(lib, "Bluegrass")
Traceback (most recent call last):
  ...
ValueError
```

Note: The order of songs in your playlists does not need to match the examples above.

Tip: You can use the `in` operator to test whether one string is a substring of another. For example, `'el' in 'hello'` returns `True`, while and `'hej' in 'hello'` and `'hl' in 'hello` return `False`.

b) Write a function `write_playlist(playlist,filename)` that takes a playlist and a filename as arguments and saves the playlist in the given file, using the same comma-separated format as we use for libraries. (1 point)

For example, after running

```
>>> lib = read_library('80s_library.txt')
>>> house = make_playlist(lib, "House")
>>> write_playlist(house, "house_music.txt")
```

the file `house_music.txt` should contain the following text:

```
Talking Heads,Burning Down The House,4:01
Talking Heads,Houses in Motion,4:33
Kate Bush,Get Out of My House,5:51
```

3.3.4 Task 4: Putting it all together (2 points)

For your final task, we'll combine what we've done so far into a complete program. Write a function `main()` that

1. asks the user for a music library filename;
2. prints out the contents of the library;
3. asks the user for a playlist theme;
4. asks the user for a filename to save the playlist;
5. saves the playlist for the theme in that file.

Your function should handle the following exceptional situations:

- if the music library filename given by the user does not exist or the file is malformed, it should ask for a different filename.
- if there are no songs matching the theme, it should ask the user for a different theme.

Sample execution:

```
>>> main()
Which music library do you want to load? nonexistent.txt
That file does not exist.
Which music library do you want to load? 80s_library.txt
Talking Heads (3 songs, 14:19)
- Burning Down The House (4:01)
- Houses in Motion (4:33)
- Moon Rocks (5:45)
Kate Bush (2 songs, 9:58)
- Get Out of My House (5:51)
- Rocket's Tail (4:07)
Total: 5 songs, 24:17
Enter a playlist theme: Pythonwave
No songs match this theme.
Enter a playlist theme: Rock
Where do you want to save the playlist? rock_music.txt
Saved. Goodbye!
```

After this execution, the file `rock_music.txt` should contain the following text:

```
Talking Heads,Moon Rocks,5:45
Kate Bush,Rocket's Tail,4:07
```

3.3.5 Task 5 (0 points)

Read through, clean up, and document your code. In order that grading be objective, you should not include your name in the file you hand in.

Tips: read through "Basic principles of programming" under Resources on the course site for recommendations on how to write good code.