DA2004/DA2005: Labs

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

Content

Lab rules

- All deadlines are **strict.** If the deadline is missed, the lab or project must be redone at the next one course opportunity. Contact the course leader if this happens.
- If you know that you will not be able to finish a lab or the project (due to a valid reason, e.g. illness), inform the course leader **before** the deadline. If you contact the course leader only after the deadline , it will be counted as missed and you will have to do it again at the next course opportunity.
- You have to work individually with the labs and the project. This means that you must write your own code and find your own solutions. You must not give solutions to each other or copy code from the Internet. All submissions are automatically compared and suspected cheating is reported to the university's disciplinary committee, which may lead to suspension.
- You can Google for Python commands, syntax, error messages, etc, but not for ready-made solutions. Google in English to get est answer. On the site https://stackoverow.com/ nns probably the answer to many of your questions.

- Do not use functions from any library in the labs unless it is explicitly stated that you will get them (ie, do not use “import” anywhere).
- Do you have problems with things that have nothing to do with the programming itself, e.g. if you have problems using the terminal, libraries that are in conic, etc., contact one of the teachers for help.
- The solution must have a reasonable structure, ie it is not ok to have extremely long programs with the same copied code over and over again instead of e.g. a loop. Unnecessarily complicated code may lead to point deductions.

Scoring

- Each laboratory is worth 10 points.
- You must get **at least 2 points per laboratory and a total of at least 25 points** to be approved for the laboratory course.
- If you reach 35 points, you get 1 bonus point for the project, and if you reach 45 points, you get man 2 bonus points.

Submission

- Submissions may not be made in any language other than Python. • Solutions

must be submitted in the form of .py files. No lformat other than .py is accepted otherwise specified explicitly in the instructions.

- Solutions must be written in Python 3. Submissions in Python 2 are therefore not permitted.
- Your solution must provide the correct output for a given input.

Peer correction on PeerGrade

- Peer correction is a mandatory part of every lab. If you don't do it, the lab is counted as missed and you have to redo the lab at the next course session.
- Keep a good tone and give constructive feedback on other people's code when you peer-correct. In case of problems contact the course leader.

Accounting

- In normal cases, no report is required, but in the event of ambiguities, we may require an oral report. • During reporting,

you must be able to answer questions about the solution orally.

Lab 1

Temperature conversion

The main purpose of this laboratory is for you to familiarize yourself with the development environment either in the halls or on your own computer.

To pass the lab you must have at least 2 out of 10 points, but remember that to pass the lab part at the end of the course you must have at least 25 of the maximum 50 points in the labs.

1.1 Learning objectives

You should be able to write, run, and modify a small Python program.

1.2 Assignment

The program converta.py in the "Code for labs" folder on the course homepage is supposed to ask for a temperature in Fahrenheit, read in from the user, convert to the Celsius scale and then print the result.

Here are your details:

1. Download, test run and study the konvertera.py program from the course homepage (see the "Code for labs" folder). Does it work as it should?
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. Expand the program so that it asks for the conversion you want to do. (2 points)
5. Extend the program so that it continues to ask for conversions, until exiting through to type q (short for “quit”). (2 points)
6. Extend the program so that it can also convert to and from Kelvin from both Celsius and Fahrenheit. For full points, the program must still ask what conversion to do right up until the user types q. (2 points)

7. Submit your solution to PeerGrade.io. If you haven't registered yet, do so using the registration code found on the course website. **Important:** use your real name/the same name as you are registered in the course with when you register on Peergrade!

8. Peer-review other solutions on PeerGrade.io! It will be possible after the deadline.

Remember: it's always good to comment the code where necessary (to clarify the purpose of e.g. a line of code or a block of code) and to document functions. All functions you write should also be thoroughly tested so that you know they work as intended!

Lab 2

Polynomial

In this task we will represent polynomials using lists of coefficients. The word "representation" here roughly means "storage method in computer", and it means concretely that a polynomial like 1+3x+7xˆ2 is stored in Python as the list [1,3,7]. In general, the coefficient of the term of degree n is therefore stored at position n of the list .

2.1 Learning objectives

- You will see how to give an abstract concept (polynomial) a concrete representation in the computer and how to make calculations on it. • You must be able to write small simple functions • You must be able to work with the data structure list.

2.2 Submission

Submission of the laboratory must be done as usual on PeerGrade. Don't forget to peer correct!

Remember: it's always a good idea to comment the code where necessary (to clarify the purpose of e.g. a line of code or a block of code) and to document functions.

To pass the lab you must have at least 2 out of 10 points, but remember that to pass the lab part at the end of the course you must have at least 25 of the maximum 50 points in the labs.

2.3 Data

In the lab, polynomials must therefore be represented as lists, below are some of your examples of how polynomials can be implemented as lists:

Polynomial	Python representation
ÿÿ 4	[0, 0, 0, 0, 1]
2 4ÿÿ 3 + 5ÿÿ	[0, 0, 4, 5]
2 5 + 4ÿÿ + 3ÿÿ 2ÿÿ + ÿÿ 3 4 +	[5,4,3,2,1]

Note: you can assume that the lists you are working with only contain numbers.

The function in Python-len polynom.py (nns on the course homepage in the "Code for labs" folder) contains a function, poly_to_string, which converts polynomials represented as lists to strings.

Start by creating a l labb2.py, to get started with the lab, and copy over the function poly_to_string there. You must now solve all the tasks below by adding the necessary code to solve problems specified in the tasks. As you will see, you also have to (later) modify the function poly_to_string (see task 2).

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

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

Assume that the polynomials p and q are dened as below.

```
p := 2 + x^2 q :=
-2 + x + x^4
```

Write code to store the list representation of these two polynomials in the variables p and q in Python. That is to say,

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

where the contents of the lists must be filled in. Test that you have written correctly in the following way:

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

Here >>> is the “prompt” in the Python interpreter, i.e. poly_to_string(p) is a command to be executed by Python and on the line after comes the result. This may look different on your computer and be achieved in different ways, e.g. in Spyder you can instead write:

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

and then observe the result in the console on the right after running the program. In that case you will not see '2 + 0x + 1xˆ2' but only 2 + 0x + 1xˆ2, i.e. ' will not be printed. These types of tests are very helpful when developing the code, but should not be included in the final version that you submit. However, it can be good to include the tests in the form of commenters to make it easier for those who have to read through and test the code.

2.3.2 Task 2 (3 points)

Change in poly_to_string so that:

- The empty list is converted to 0. •
Terms with coecient 1 are written without coecient. That is, 1xˆ2 should instead be written as xˆ2. • Terms with coefficient -1 put a minus in front of the term, but the one is not printed. For example. 2 + -1xˆ2 is instead written as 2 + -xˆ2.
- Terms with coecient 0 are not printed. That is, 0 + 0x + 2xˆ2 must be simplified to 2xˆ2. • A list containing only 0 as elements, e.g. [0, 0, 0] is written as 0.

Test the function! The output should look like this:

```
>>> 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 zeroes at the end of a polynomial and **returns the** result. Tip: use a while loop and the pop() function.

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

Dene any polynomials with zeros at the end

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

and test the function:

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

>>> drop_zeroes(q0) []
```

b) Write a function that tests when two polynomials are equal by ignoring all trailing zeros and then tests for equality:

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

Test that the code works:

```
>>> eq_poly(p,p0)
True
```

```
>>> eq_poly(q,p0)
False

>>> eq_poly(q0,[])
True
```

Note: the drop_zeroes and eq_poly functions should **return** their results and not just print it out.
The difference can be hard to grasp at first as the result looks similar when you run the code, but there is a very big difference between a function that returns something and one that just prints something. See the end of 2.5.1 in the compendium for more information on this.

Also note that the code ni ck for poly_to_string returned the result string. Does your solution to problem 2 work the same way? If not, go back and repeat.

2.3.4 Task 4 (2 points)

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

Suggested algorithm:1

- Iterate over the terms of the polynomial by iterating over the coefficients.
- Keep track of the degree of the current term and the sum of the terms you have summed so far. In each iteration, calculate the value of the term as coeff * x ** degree (remember raised is **). Then add the value of the term to the sum. •
When you have finished iterating, you return the sum.

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 Exercise 5 (3 points) a) Dene

negation of polynomials (ie change the sign of all coefficients and **return the** result).

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

1The proposed algorithm is not the most efficient, if you want to optimize, you can instead implement Horner's algorithm: https://sv.wikipedia.org/wiki/Horners_algorithm

b) Deny addition of polynomials (ie add the coefficients and **return the** result).

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

c) Dene subtraction of polynomials.

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

Tip: keep in mind that p - q can be dened as p + (- q), i.e. to subtract the polynomial q from p , you can first take the negation of q and then add with p.

Test that the functions work:

```
# 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
```

Note: the comments are only there to explain what the tests are testing. Can you think of good tests to find possible bugs in the code?

2.3.6 Task 6 (0 points)

Read through, rewrite and document your code. Since the correction must be objective, you must not leave your name in the file you submit.

Tip: read through “Rules of Thumb for Programming” under Resources on the course homepage for recommendations on how to write good code.

Lab 3

Iteration, l handling, error handling and lookup tables

This lab contains a number of independent tasks involving basic algorithms, lookup tables, lhandling, and error handling.

To pass the lab you must have at least 2 out of 10 points, but remember that to pass the lab part at the end of the course you must have at least 25 of the maximum 50 points in the labs.

3.1 Learning objectives

• You should be able to translate an algorithm into code. • You must be able to work with lookup tables. • You must be able to read and write data from l. • You must be able to use error handling.

3.2 Submission

Submission of the laboratory must be done as usual on PeerGrade. Don't forget to peer correct!

Remember: it's always good to remember to comment the code where necessary (to clarify the purpose of eg a line of code or a block of code) and read through your code before submitting!

Tip: use documentation strings in all functions you've written so you can easily find out what the input is and what the function does.

No modules ("libraries") may be used, i.e. no imports. Put all the code in an l like in the last lab. Functions first, then the main program that calls the functions. On task 1, you may not use built-in functions for sorting, such as sort or sorted.

Remember to remove your name from the code, in case e.g. Spyder put it in the len.

3.3 Data

3.3.1 Task 1: deposit sorting (2 points)

Insertion sort (eng.: insertion sort) is a common sorting algorithm, i.e. a method for sorting a list of elements. The idea behind this algorithm is similar to the way you might sort a deck of cards: for each card in the deck, insert it into the correct slot in a pile of sorted cards.

We can divide this into two sub-problems:

a) Write a function that inserts an element xi into an already sorted list sorted_list:

```
def insert_in_sorted(x,sorted_list): # here be code
```

Algorithm idea:

1. Assume sorted_list is sorted.
2. Iterate over all indexes in < len(sorted_list) until you find some element sorted_list[i] that satisfies sorted_list[i] > x and then insert x.
3. If there is no sorted_list[i] greater than x: insert xi at the end.

Tests:

```
>>> insert_in_sorted(2,[]) [2] >>>
 insert_in_sorted(5,[0,1,3,4]) [0, 1, 3,
4, 5] >>> insert_in_sorted(2,[0 ,1,2,3,4]) [0, 1, 2,
 2, 3, 4] >>> insert_in_sorted(2,[2,2]) [2, 2, 2]
```

b) Write insertion sort using insert_in_sorted:

```
def insertion_sort(my_list): # here be
         code
```

Algorithm idea:

1. Initialize a variable out with the empty list.
2. For each element xi my_list insert it into out using your function insert_in_sorted.
3. Return out.

Tests:

```
>>> insertion_sort([12,4,3,-1]) [-1, 3, 4, 12]
 >>> insertion_sort([]) []
```

Note: for scoring, insertion_sort must use insert_in_sorted.

3.3.2 Problem 2: sparse matrices (1 point)

A matrix can be represented in Python as a list containing equally long lists of numbers.

For example, the matrix:

ÿÿ 1 0 0 2 ÿÿÿ
ÿÿ 0 8 0 0 ÿÿÿ
ÿ 0 0 0 5 ÿÿ

can be represented as [[1, 0, 0, 2], [0, 8, 0, 0], [0, 0, 0, 5]].

A matrix is sparse if it contains mostly zeros. If you represent such a matrix as a list of lists, you need to use up quite a lot of computer memory, especially if the matrix is very large. Consider a matrix with many millions of rows and columns containing only a handful of non-zero elements. A better way to represent this type of matrix is as a lookup table from coordinates to nonzero elements.

If the coordinates are of the form (row, column) and we start counting from zero, then the matrix above can be written in the following way as a lookup table:

{(0, 0): 1, (0, 3): 2, (1, 1): 8, (2, 3): 5}

Write a function matrix_to_sparse that takes in a matrix represented as a list of lists and produces a lookup table like above. You can assume that the matrix has the correct form (ie that all lists are the same length).

Tests:

```
>>> matrix_to_sparse([[1,0,0,2],[0,8,0,0],[0,0,0,5]]) {(0, 0): 1, (0, 3 ): 2,
(1, 1): 8, (2, 3): 5} >>> matrix_to_sparse([[0,0,0,0],[0,0,0,0],[0,0 ,0,0],
[0,0,0,0]]) {} >>> matrix_to_sparse([[0,0],[0,0],[0,0],[0,10]]) {(3, 1): 10}
```

3.3.3 Task 3: management (1 point)

Write a function annotate(f) that takes an lname f as parameter and prints to a new l annotated_f with original text, line number (counted from 0), total number of words up to and including that line.

Example: if len infile.txt contains:

```
A Dead Statesman

I could not you; I dared not rob: Therefore
I lied to please the mob.
Now all my lies are proven untrue And I
must face the men I slew.
What tale shall serve me here among Mine
angry and defrauded youth?
```

So running annotate('infile.txt') should produce an l annotated_infile.txt containing :

```
A Dead Statesman 0 3
 1 3
```

```
I could not you; I dared not rob: 2 11 Therefore I
lied to please the mob. 3 18 Now all my lies are proven
untrue 4 25 And I must face the men I slew. 5 33 What
tale shall serve me here among 6 40 Mine angry and
defrauded youth? 7 45
```

3.3.4 Task 4: string search in ler (2 points) a) Write a function

find_matching_lines(h,s) that takes a **lhandle** (eng: handle) h and a string s. The function must return both line numbers (counted from 0) and content for the rows containing the string in the form of a list of tuples.

Example (with infile.txt as above):

```
>>> hinfile = open('infile.txt') >>>
find_matching_lines(hinfile, 'the mob')
[(3, 'Therefore I lied to please the mob.\n')] >>> hinfile.close()
>>> hinfile = open('infile.txt') >>> find_matching_lines(hinfile,
'the')

[(3, 'Therefore I lied to please the mob.\n'), (5, 'And I must face the men I slew.\n')] >>> hinfile.close() >>> with
open( 'infile.txt') as h: find_matching_lines(h, 'summer')

...
[]
```

Note: The search must be case sensitive, so "the" is not the same as "The". **Tip:** How does in work for strings?

Note: find_matching_lines must take an lhandle, so it must not contain any call to open but it is assumed that open is run before the function is called as shown in the tests above.

b) Write a find_lines() function that prompts the user for an l and a string, and uses the find_matching_lines function to print the lines where the string was found.

Example: a run of find_lines(), with infile.txt as above, might look like this:

```
Hello, which file do you want to search in? infile.txt Ok, searching in
"infile.txt".

What do you want to search for? the The result
after searching for "the" is:

Line 3: Therefore I lied to please the mob.
Line 5: And I must face the men I slew.
```

It is up to you whether the program should continue to ask and whether you should be able to change l to search in, etc. As long as the user can choose l and string and that the code uses find_matching_lines , you get points.

3.3.5 Task 5: position search in clay (4 points)

a) Write a function save_rows(h) that takes a lhandle (eng: handle) h and saves row numbers as keys and rows as values in a lookup table. The function should then return the lookup table.

Example: if the input number infile2.txt contains:

```
Hey you
the moon revolves around the earth two
chairs and the table
```

then the following should happen during a run:

```
>>> with open('infile2.txt') as hinfile2:
...             save_rows(hinfile2)
{0: 'Hey you', 1: 'the moon revolves around earth', 2: 'two chairs and the table'}
```

Note: note that there are no \ni at the end of the strings. **b)**

Write a function lookup that takes in a lookup table d as above and two coordinates r and c which correspond to row and column id and returns the character of the position that the coordinates correspond to.

The idea is that lookup should be used together with save_rows (see subtask **c))** and each character in len can be said to lie on a row and in a column. For example, the word “Hey” in infile2.txt occupies the coordinates (0,0), (0,1), (0,2). In the same way, the word "chairs" occupies the coordinates (2,4), (2,5), ...,(2,9).

We assume a 0-indexed coordinate system (as used in programming).

These cases must be handled:

- If the row and/or column is not nns id, the program should throw a LookupError. [1]
- If the position is a space, "Space" must be returned.

Tests:

```
>>> with open('infile2.txt') as hinfile2: d =
...             save_rows(hinfile2) print(lookup(d,0,0))
...             print(lookup(d,2,9)) print(lookup(d,2 ,10))
...
...
hrs
pp
Space
```

But if you run, for example, lookup(d,3,0) or lookup(d,0,7), a LookupError must be raised. **c)** Use

save_rows and lookup to write a code snippet (a “program”) that:

1. Asks the user for an l, reads the len into the lookup table (using save_rows)
2. Asks the user to provide coordinates for row number and column number.
3. Uses lookup to retrieve and then print the character in len at the location of the coordinate.

The user must be able to give your coordinates until he chooses to write exit , whereupon the program ends. You choose the structure of this code snippet yourself, e.g. can you dene a function main() that takes no parameters but contains the code containing 1-3 above:

1For documentation on which special cases are nndenied in Python see: https://docs.python.org/3/library/exceptions.html

```
def save_rows(...):
      ...

def lookup(...):
      ...

def main():
      infile2 = input('Enter a file: ') indexed_file
      = save_rows(...)
      # More code here for steps 2 and 3
```

If lookup throws an exception because the coordinates are outside the text, it should be caught with try-except. Then the message Warning: Out of bounds, try again! is printed and the program continues.

Example: if you run the program with len infile2.txt as above, running the program should look like this:

```
At any point type "exit" to quit.

Provide row: 0
Provide column: 3

Space

Provide row: 1
Provide column: 1

hrs

Provide row: 3
Provide column: 1

Warning: Out of bounds, try again!

Provide row: 2
Provide column: 9

pp

Provide row: exit
```

Lab 4

Coming later

Lab 5

Coming later