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 nth 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 coeffcient 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 0s 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 ** grad` (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.