Lecture 7 – extra

**Why are we learning to write "oneliners"?! Isn't that nasty?!**

This topic will leave the - wrong! - impression that we are learning to write functions in one line. But does this exhibitionism really belong in Programming 1?! Isn't it actually nasty?

It doesn't. And it can be nasty.

Some students are really keen on this sport, the challenge. "But is it possible to make an oneliner for this too?" is quite a common question. Yes. Everything is possible. But it is usually without connection. Then, once you know how, it's often not even a challenge anymore. It's like asking yourself if it's possible to walk to Paris. Yeah, I just don't know why I would do it.

Why then are we learning to "write oneliners"?

We don't. We learn to write generators, iterators, derived lists, sets and dictionaries. Why? It's short and concise, perfectly understandable and faster - if you use common sense. So definitely something useful to know. But if we force it when we don't need to, we will of course write code that is longer, incomprehensible and slow.

So what we are going to do today is another tool that is available to us, and it must be used, like all tools, at the right time.

The second purpose of this is to get used to a different way of thinking about progrmail. So far, we have looked at programs - especially the code that compiles a list, a dictionary, a set - *in a procedural way*. We've described *the steps of* building a list, a dictionary, a set -- each element is computed separately and *added* with append, addor whatever. Today's programs will be *declarative*: instead of telling you how to build a list, we'll just tell you what it should contain. The code won't even be that different, it will just be reversed a bit. What will have to be completely different is the way we think when we write and read.

Apart from these two reasons, writing functions in one line is of course also a fun sport - even when the result is not suitable for production code. :)

**"Declaratory record"**

You already know what we are going to do. In mathematics, we sometimes describe a set by listing its elements.



Sometimes we just describe their properties (with some care [not to get Russell on us](https://en.wikipedia.org/wiki/Barber_paradox)). We would describe the set of multiples of all natural numbers whose square is less than 120 as follows:



**Derived lists**

Let's start with what we have known for the longest: lists. So far, we have written them by listing their elements in square brackets,

Ein Bild, das Schrift, Text, Handschrift, Grafiken enthält.

Automatisch generierte Beschreibung

We will continue to use parentheses, but instead of listing the elements inside them, we will write the expression that makes them up. Suppose I wanted to list the roots of the numbers in the list above (by some strange coincidence, the list above contains exactly the squares). I would do this:

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte Beschreibung

So: I wrote square brackets, as I always do when defining a list, but instead of listing the elements, I said what they should be. I said that the elements of my new list should be the *square root of x*, where xis the elements of the list k. What if I wanted to have squares of numbers instead? Same schmorn:

Ein Bild, das Schrift, Grafiken, weiß, Text enthält.

Automatisch generierte Beschreibung

The form of what we write is always [a for expression variable and something], where *the variable* is the name of some variable (e.g. x, the *expression* is some expression that (usually, but not necessarily) contains that variable (e.g. sqrt(x)or x \*\* 2), and the *something* is something over which a for loop can be passed, i.e. a list, a dictionary, a set, or something we don't know yet but will learn about today.

So when we construct such a *derived list*, we have to think:

* what do we want in this list? What term do I use to describe (calculate) an element?
* what (which list, range-a...) do I have to go through to get these elements.

Let's look at some more examples.

I have a list of names, say,

Ein Bild, das Text, Schrift, Reihe, Screenshot enthält.

Automatisch generierte Beschreibung

How would you calculate the average length of a name? I have a sum function that can add numbers in a list. So I just need a list of lengths (instead of a list of names). Let's look at the two "points" above. What do I want in the list? The lengths of the names. How do I get them? Obviously by some len(name), where a nameis just any name in the name list. In this sentence we have answered both questions -- what am I going to compute (len(name)) and what am I going to loop over (over names). So we know: we need

Ein Bild, das Schrift, Text, weiß, Grafiken enthält.

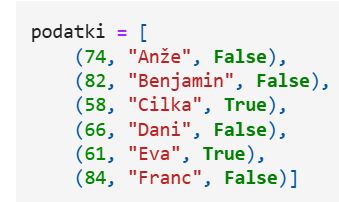
Automatisch generierte Beschreibung

Now I just add this up and divide by the length of the list.

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

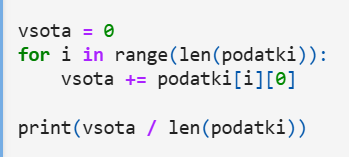
Automatisch generierte Beschreibung

The average length of the name is not particularly interesting. Let's calculate the average weight instead and, to avoid boredom, we will calculate it from these triples (weight, name, is\_female):



Here's how to do it: loop over the data, calculate the sum of the first elements and divide by the length of the list.

First, let's see how we would have done it the old-fashioned way. Please, not like this:



This is better:

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte Beschreibung

If you want your professor to have a lasting good memory of you, this is what you will do:

Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

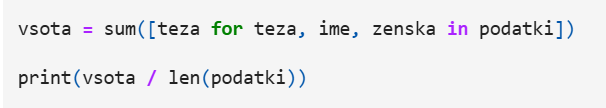
Automatisch generierte Beschreibung

Now let's do as we have learned today. We can do

Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

or, better, the latest version



**Derived lists and conditions**

When deriving lists, you can add a condition to select the elements to be added.

How would you make a list of squares of all odd numbers up to 100? A list of squares of all numbers up to 100 is trivial, all we need to do is write:

Ein Bild, das Text, Schrift, weiß, Grafiken enthält.

Automatisch generierte Beschreibung

If you want to pick only odd numbers, you can add a condition to the list derivation.



This is not really necessary, we could simply say

Ein Bild, das Schrift, Text, Grafiken, weiß enthält.

Automatisch generierte Beschreibung

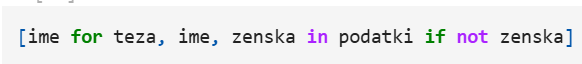
What if you want a list of all numbers up to 30 that are not divisible by 7 and do not contain the digit 7?

Ein Bild, das Text, Screenshot enthält.

Automatisch generierte Beschreibung

The first part of the condition ensures that the number is not divisible by 7 (the remainder after division by 7 must be different from 0). The second part ensures that the number does not contain 7: simply convert the number to a string and check that it does not contain 7.

How would you calculate the sum of the weights of the men in the above list of data? We add a condition to the loop. To start with, let's just make a list of the names of all the men.



Ein Bild, das Text, Schrift, weiß, Kalligrafie enthält.

Automatisch generierte Beschreibung

In the definition of our list, we add an ifat the end, after the for, and a condition that must be satisfied by the element that we are interested in.

We also pick up the weights in the same way.

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung

Add the elements of this list with sum, and we have.

Ein Bild, das Text, Schrift, Reihe, Screenshot enthält.

Automatisch generierte Beschreibung

Now you know why I'm so tired of not using for i in range(len(s)): because with it, instead of a nice, short, transparent [thesis for thesis, name, woman and data if not woman], we get [data[i][0] for i in range(len(data)) if not data[i][2]]. Of course, you can also work the other way if you like. We live in freedom; no law forbids masochism.

Let us make a list of all the factors of a given number n. Take, say, n = 60. The list of divisors is then the list of all x, where xcomes from the interval 1 ≤ x ≤ n, for which the remainder after dividing nby xis 0 (i.e. xdivides n).

Ein Bild, das Text, Schrift, weiß, Grafiken enthält.

Automatisch generierte Beschreibung

Now we can quickly see if a given number is perfect: perfect numbers are (remember?) numbers that are equal to the sum of their divisors (excluding themselves).

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

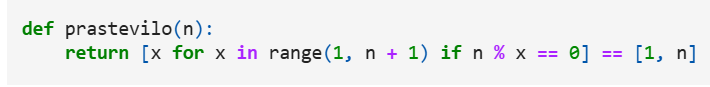
Automatisch generierte Beschreibung

Even more remarkable are prime numbers, numbers without divisors. That is, those that are divisible only by 1 and by themselves.

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte Beschreibung

If the sharersdidn't have the function yet - no big deal. You would still find out whether a number is a prime number in a single line. Moreover, we could only go from 2 to n (without n); so we would leave out 1 and n and declare a number prime if it is divisible only by 1 and by itself.



Of course, it is simpler to change the bounds in the rangeso that they do not include 1and n.



Better still, we should remember that empty lists are untrue.

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte Beschreibung

For an encore, let's add something we won't explain.

Ein Bild, das Schrift, Text, Grafiken, weiß enthält.

Automatisch generierte Beschreibung

This function - in any form - can be used to quickly list all primes between 2 and 100:

Ein Bild, das Text, Screenshot, Software enthält.

Automatisch generierte Beschreibung



Of course, we can do without a function; we just write what the function does into the condition.

Ein Bild, das Text, Screenshot enthält.

Automatisch generierte Beschreibung

We prefer not to write such things, as they quickly become obscure.

**General pattern**

In general: any piece of code that looks like this

Ein Bild, das Text, Screenshot, Schrift, Reihe enthält.

Automatisch generierte Beschreibung

**Multiplication**

Sets are constructed exactly like lists, except that instead of square brackets, we use curly brackets. This gives all divisors 60

Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

**Dictionaries**

Same thing. If we want to make a dictionary that contains numbers up to 10 as keys and their squares as values, we write

Ein Bild, das Text, Screenshot, Schrift, weiß enthält.

Automatisch generierte Beschreibung

**Some useful features**

These things really come to life when used in combination with a few features. Some of them we already know, some of them will only make sense today.

**enumerate**

So far, I've been annoyed that we use it when we need both the index of an element and its value. Now it will be even more useful, as it will significantly improve the clarity of derived lists, sets and dictionaries.

Now that we are just as clever, we also know how to write enumerates if they did not already exist.

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

**zip**

We already know about zip: we give it some lists and it returns a new list - a list of targets with the elements of the lists we sent as arguments.

Well, as of today we know: it actually returns a pair generator, not a list of pairs.

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte Beschreibung

Ignore those 0x10193c308(or whatever is displayed) - the printout said we made a zipand not a list. Of course we can "generate" a list from it.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Let's write a function that tells how many letters match the given two words. For example, MELODY and EMPLOYER match in six letters.

MELODY

EMPLOYER

EMPLOYEE

To make our work easier, let us first remember that Truebehaves as if it were 1and False as if it were 0.

Ein Bild, das Text, Schrift, Design, Screenshot enthält.

Automatisch generierte Beschreibung

Take, for example, the words MELODY and EMPLOYER.

Ein Bild, das Text, Schrift, weiß, Design enthält.

Automatisch generierte Beschreibung

Combine them into a list of letter pairs (but only add theleaf in the printout to see what we've actually sown).

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

Automatisch generierte Beschreibung

Let's loop over the list of letter pairs and create a new list containing True if the letters are the same and False if they are different.

Ein Bild, das Text, Schrift, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

Now let's count how many True-'s there are. How? If Falseis the same as 0 and Trueis the same as 1, then we simply calculate the sum of the elements of the list.

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte Beschreibung

So if you wanted to write a function that tells you how many letters the two words match, you would say

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte Beschreibung

Let's write a function that calculates the Euclidean distance between two points whose coordinates are represented by a list. Let's have

Ein Bild, das Schrift, Zahl, Diagramm, Design enthält.

Automatisch generierte Beschreibung

Sometimes we would solve the problem like this:

Ein Bild, das Text, Schrift, Handschrift, weiß enthält.

Automatisch generierte Beschreibung

Now we know how to do it much, much more simply. Can we make a list of pairs? We can.

Ein Bild, das Text, Schrift, Screenshot, Handschrift enthält.

Automatisch generierte Beschreibung

Can we make a list of pair differences? We can.

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte Beschreibung

Actually, we need a list of squares of the differences.

[(x - y) \*\* 2 for x, y and zip(a, b)]

And now we just need to add it all up.

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte Beschreibung

The function that calculates the Euclidean distance is tore



Once you get used to it, these things are eminently readable. The program is an almost verbatim natural language description: the Euclidean distance is the square root of the sum of the squares of the differences of the elements taken "in parallel" from the two lists.

**All and any**

The functions alland anylook trivial, but in fact they are very useful. The first, all, receives a generator (or, of course, a list, a set, a string, a dictionary, etc.) and returns True if it generates true things (say, True itself).

A number nis prime if all remainders after division by i(for each ifrom 2to n) are different from 0. With all, this translates almost literally into Python.

Ein Bild, das Text, Schrift, weiß, Screenshot enthält.

Automatisch generierte Beschreibung

The anyfunction returns True if what is passed as an argument contains at least some true value. Let's show how it works with inverse prime numbers: a number is composite if the remainder after division by some other number is 0.

Ein Bild, das Text, Schrift, weiß, Screenshot enthält.

Automatisch generierte Beschreibung

Incidentally, we notice that alland anyare linked by de Morgan's rule, which we should remember from mathematics. A number is prime if it is not composite,

Ein Bild, das Text, Schrift, weiß, Grafiken enthält.

Automatisch generierte Beschreibung

If, instead of calling compound, we insert the function code compound, we get

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte Beschreibung

If we just reverse the condition, we see that

**not** any(n **%** i **==** 0 **for** i **in** range(2, n))

same as

all(**not** n **%** i **==** 0 **for** i **in** range(2, n))

**itertools.count**

There is even more sugar in the itertools module.

The countfunction of itertoolsis like a range, which can be given at most a lower bound or not yet a lower bound. count()generates all numbers from 0 to infinity, and count(n)generates all numbers from nonwards.

I use it most often with zip. zip(count(), s)is the same as enumerate(s). That's not very useful - obviously enumerate(s)is simpler and clearer than zip(count(), s). What happens is that you zipover three lists at once, and you need indexes. This can be written as

**for** i, (x, y, z) **and** enumerate(zip(s, t, u)):

or we can write

**for** i, x, y, z **and** zip(count(), s, t, u)

And here's where it will come in handy, if you remember it

**itertools.chain**

If we have several generators (or lists, dictionaries, etc.) and we want to loop through them all, one by one, we build them with chain.

Ein Bild, das Text, Screenshot, Schrift enthält.

Automatisch generierte Beschreibung

Of course, this is the same as for c and "Anna" + "Berta" + "Cilka":, but the beauty is that we haven't added the strings, it's a chain. This is nice especially when we're not chaining strings, but something that can't be added.

**Itertools**

Anyone who liked this will also enjoy the other [goodies from the itertools module](https://docs.python.org/3/library/itertools.html#itertools.chain).