Iterating

Using generators/iterators in python

Enable native scrolling

Generators

```
1 def cubes(n):
2     for i in range(n):
3         yield(i*i*i)
4
5 for c in cubes(11):
6     print(c)
```

generators

- produce a stream of values
- do not produce complete result at once
- stop execution after producing a value

Generators

Extend the previous example with `print` statements before and after the `yield` statement to better understand the program flow

Fibonacci

```
1 a= 0
2 b= 1
3
4 print(a, b)
5
6 for i in range(10):
7     (a, b)= (b, a+b)
8
9     print(b)
```

The program above prints the first 12 fibonacci numbers

Write a generator function `def fibonacci(n)` that `yield`s the first `n` fibonacci numbers

List(iter)

In Python generators can be used in many places, not just `for` loops.

You can, for example, use `list(fibonacci(n))` to get a list of every value the generator yields

Calculate and print a list of the first 12 fibonacci numbers

There is also a `sum` function that calculates the sum of all the values a generator yields

Use `sum` to print the sum of the first 12 fibonacci numbers

a for a in b

```
1 list(a*a for a in range(10))
2
3 list(a for a in range(10) if a%2)
```

If a generator function can be expressed in a single line it can be written as a list comprehension

Execute the list comprehensions above and find out what they do

checksum

Uni Bremen matriculation numbers contain a checksum to detect typos

If you consider the digits

```
a|b|c|d|e|f|s
4|0|2|9|6|8|5
```

the checksum is calculated like this:

```
s= (1*a + 2*b + 3*c + 4*d + 5*e + 6*f)%11
```


checksum

```
1 num= 4019661
2 dgts= list((num//10**i)%10 for i in range(6, -1, -1))
```

Using you newly-earned knowledge of list comprehensions:

what is the purpose of the above two lines?

checksum

```
1 acc= 0
2
3 for (i, n) in enumerate(dgts[:6]):
4     acc+= (i+1)*n
```

Given the for-loop above, that calulates the checksum:

use `sum` to write a list comprehension that calculates the same result

checksum

```python
def checksum(mat):
    digits= list(
        (mat//10**i)%10 for i in range(6, -1, -1)
    )

    csum= sum(
        (i+1)*n
        for (i, n)
        in enumerate(digits[:6])
    )%11

    return(csum == digits[-1])
```

The function above takes a matriculation number, calculates the checksum and compares it to the one in the number

csv

```python
import csv

reader= csv.reader(
    open('25_student_list.csv', 'r'),
    delimiter= ';'
)

for entry in reader:
    print(entry)
```

Download the student list and run the above script in the same directory

Use `int` to convert the matriculation number field to an integer and only print those lines that contain a valid matriculation number

hexdump

```
Address   Hex                                             ASCII
00000000  0c 94 5f 00 0c 94 87 00  0c 94 87 00 0c 94 87 00  |.._.............|
00000010  0c 94 87 00 0c 94 87 00  0c 94 87 00 0c 94 87 00  |................|
00000020  0c 94 87 00 0c 94 87 00  0c 94 87 00 0c 94 87 00  |................|
00000030  0c 94 87 00 0c 94 87 00  0c 94 87 00 0c 94 87 00  |................|
00000040  0c 94 71 08 0c 94 87 00  0c 94 fa 02 0c 94 3d 03  |..q...........=.|
00000050  0c 94 87 00 0c 94 87 00  0c 94 87 00 0c 94 87 00  |................|
```

When analyzing binary files hexdumps are an useful tool

they show the hexadecimal representation of every byte in the file

HTML hexdump

```
for (index, chunk) in to_chunks(binary, 16):
    as_hex= ' '.join(
        '{:02x}'.format(byte) for byte in chunk
    )
```

...

Run the program above with this binary file and view the file it outputs

Look at the function `read_binary` and check the documentation of `open` to find out what the parameters to the `open` call mean and why they are used

Edit the program, so that 32 bytes are printed per line instead of 16

Edit the program, so that the decimal representation of each byte is printed instead of the hexadecimal representation