1 Goal

In the last class you have been taught about variables. The variables that we used were of types int, float, string and bool. In addition we introduced the conditional statement if. Today, we will look at a new type: list. Manipulating lists of variable sizes requires repetition, which is also part of today’s topic.

2 Practical

2.1 Lists

Start up the Python shell by typing python in your terminal. Type the following

```python
>>> a = [2, 3, 5, 7, 11]
```

The variable a is of type list, you can confirm this by typing

```python
>>> type(a)
```

Q1. Try evaluating type("hello"), type(3), type(3.14159), type(False). What happens when you force a type using type conversions, e.g. type(bool("False"))?

Just like the other types, variables of type list can also be printed. Let’s try it:

```python
>>> print a
```

To access for instance the first element in the list a, you can do the following

```python
>>> a[0]
```

The length of a can be obtained using

```python
>>> len(a)
```
Q2. What happens when you evaluate \(a[5]\) and why is that?

Note that more complex expressions are allowed as indices, e.g. \(a[1+6/3]\) equals \(a[3]\).

Q3. Which element is returned when evaluating \(a[a[1]]\)? Describe in your own words what the expression exactly means.

It is possible to obtain a slice of a list in the following way

```python
>>> a[1:3]
```

Using the ‘+’ operator lists can be concatenated:

```python
>>> b = [13, 17, 19]
>>> a = a + b
```

Q4. What happens when you do \(a = a + []\)? Try evaluating \(len([])\). Do you know what \([]\) represents?

The function \texttt{range} generates a list of integers according to the specified pattern. Try the following:

```python
>>> range(13)
>>> range(0, 13)
>>> range(5, 13)
>>> range(0, 13, 1)
>>> range(0, 13, 2)
>>> range(0, 13, 3)
```

So \texttt{range} takes at most three integers arguments. In case only one argument \(n\) is specified, a list of integers starting from 0 up to (and excluding) \(n\) is generated. It is possible to specify the starting value by calling \texttt{range} with two or three arguments. The third argument corresponds to the step size.

Q5. Why does \texttt{range(-1)} evaluate to \([]\)?

Q6. What is the difference between \texttt{range(0, 13, 1)}, \texttt{range(0, 13, 2)} and \texttt{range(0, 13, 3)}?

It is important to realize that Python allows for lists whose elements are not of the same type:

```python
>>> a[0]="apples"
>>> type(a[0])
>>> type(a[1])
```

In the previous class you learned to access command-line arguments using \texttt{sys.argv}. Do the following in the Python shell:

```python
>>> import sys
>>> type(sys.argv)
```

So \texttt{sys.argv} is just a list and therefore all the list operations that you have just learned apply to it as well!
2.2 Repetition

Suppose that you are given a list of (positive) numbers whose size you do not know beforehand, and that you want to determine the biggest number in the given list. Since you have no idea about the size of the list, the indexing techniques that you just learned do not suffice. Instead, you need to be able to iterate over/loop through all elements in the given list so that you can evaluate them one-by-one.

The `for` statement allows you to loop through all elements in a list. Do the following—don’t forget to indent the second line using either a tab or four spaces

```python
>>> for number in range(10):
...     print number*10
...
```

Q7. Describe what just happened.

Let’s dissect the previous `for` statement. The generic syntax is: `for <VARIABLE> in <LIST>:`. The indented statements that follow the colon are called the body of the for-loop. The body is executed as many times as the size of `<LIST>`. Every subsequent iteration `<VARIABLE>` will hold the subsequent value in the list.

Now let’s try to find the maximum number in the following list:

```python
>>> X = [30, 3, 5, 100, 23, 189, 12, 32, 24, 85]
```

We start by defining a variable `max` which will hold the maximum number found thus far. Since we know that the list only contains positive numbers, we can initialize the variable as follows:

```python
>>> max = 0
```

We iterate over the elements in `X` as follows:

```python
>>> for x in X:
...
```

For every element `x` we need to check whether it’s bigger than `max`. If so, we set `max` to `x`.

```python
...     if (x > max):
...         max = x
...
```

That’s it! Type `max` to confirm that the maximum has indeed been found.

Q8. Why is the statement `max = x` indented twice?
3 Exercises

1. Implement a script called `sum.py` which takes—via the command line—two integer arguments \(k, n\) such that \(k \leq n\) as input and prints the result of:

\[
\sum_{i=k}^{n} i = k + (k + 1) + \ldots + n.
\]

In your script first check whether \(k \leq n\), if that’s not the case print an error message instead.

For example \(\sum_{i=2}^{5} i = 2 + 3 + 4 + 5 = 14\):

```
% ./sum.py 2 5
14

% ./sum.py 5 2
Error: 5 > 2
```

2. Optional: Implement a script called `factorial.py` which takes—again, via the command line—an integer \(n \geq 1\) as input and computes and prints the result of:

\[
\prod_{i=1}^{n} i = 1 \times \ldots \times (n-1) \times n.
\]

In case \(n < 1\), produce an error message instead. Example:

```
% ./factorial.py 10
3628800

% ./factorial.py -10
Error: -10 < 1
```

Hint: don’t forget that command-line arguments are of type `string` and therefore you need to use an appropriate type conversion.

The scripts are due tomorrow! Please hand them in on paper.