Exercises

A number of exercises and their solutions can be found here. There are separate folders with exercises for:

- basic python
- numpy
- pandas

These folders also contain suggested solutions, either with the exercise itself or in separate files. The remainder of this note introduces some of the python exercises.

Modules and packages

The slides can be found here, look here for some simple demos. Some of the traps you may walk into using imports and setting up packages can be found here:

http://python-notes.curiousefficiency.org/en/latest/python_concepts/import_traps.html

If working with virtual environments and/or creating distributions has no secrets for you, you may want to take a look at these.

Suggested exercises:

1. Create a virtual environment, activate and install some package using pip.

2. Create trivial script using package.

3. Create requirements listing.

4. Probably for a later time: make your own distribution, put it on the test PyPI server, install it and remove it again.

See https://hynek.me/articles/sharing-your-labor-of-love-pypi-quick-and-dirty/ for general instructions. See https://packaging.python.org/guides/using-testpypi/ for instructions on how to use the test server.

Calculator

Build a simple calculator. See calculator.py for some suggestions.

Google exercises

The Google exercises are mainly about string manipulations / searches. They involve reading and writing files and passing arguments to your scripts. See https://developers.google.com/edu/python/ for background and instructions.

Mutable data

The code below is an attempt at a function that adds a value to a list of values stored under the *k* key in *table*. When the key is new, the function should give the caller the option to specify an initial list of values to use. There are several things that go wrong here. Why? How would you resolve this?

```
table = {}
last_added = ()
def add(k, v, start=[]):
     last_added = (k,v)
     print("Adding: ", last_added)
     if k in table:
         table[k].append(v)
     else:
         table[k] = start.append(v)
add("x", 2)
add("y", 5)
add("x", 5)
add("z",3,[2])
add("z",4)
print("Last added to table {}: {}", table, last_added)
```

DIY Python

One can learn a lot from simply trying to implement some of the built-in functions, such as map, filter and reduce.

With the knowledge on iterators in mind, you could also try to write your own for statement (as a function), using the exec function that allows you to execute any string as a Python block. Just to understand how for works, you should refuse or be extreme cautious to include exec in production code!

Comprehension

1. List Comprehension

a = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

Write a list comprehension that takes this list and makes a new list with only the even elements in this list.

2. Another List Comprehension

Find all x where x is a natural number less than or equal to 100 and x is a perfect square. This can be solved using a for-loop as:

```
Res = []
for i in range(1,101):           #the iterator
    if int(i**0.5)==i**0.5:      #conditional filtering
        res.append(i)            #output-expression
```

Create a list comprehension doing the same

3. Matrix Flattening

Take a 2-D matrix as input and return a list with each row placed on after the other.

The Python code with a FOR-loop could look as follows:

```
def flatten(matrix):
    flat = []
    for row in matrix:
        for x in row:
            flat.append(x)
    return flat
```

Create a matrix (using list comprehension!) and test flatten. Create a list comprehension version that achieves the same result.

Other options to pursue:

- extend to n-dimensional matrices
- or to trees

4. Dictionary Comprehension

Take two lists of the same length as input and return a dictionary with the first list as keys and the second as values.

The Python code with a FOR-loop could look like this:

```
def makedict(keys, values):
    dic = {}
    for i in range(len(keys)):
         dic[keys[i]] = values[i]
    return dic
```

Create a solution using dictionary comprehension, using e.g.: country =

['Germany', 'France', 'Belgium', 'England', 'Spain', 'Italy'] capital = ['Berlin',

'Paris', 'Brussels', 'London', 'Madrid', 'Rome']

Hint: look at the zip generator.

Dictionary comprehension is just one way to achieve this. Look up the dict type and see what other (built-in) options there are.

Generator

1. infinite number generator
   Write function that generates infinite number of integers.

2. firstn
   Write a generator that generates first n items of an iterator.

3. Fibonacci
   Write Fibonacci function as generator