Lab 02 – Learning Python

Group: Estella Kinzel, Jonas Trenkler

Repository: https://github.com/JonasTrenkler/info3-lab02

Part 1 - Learning Python with Unit Tests

Topics and Issues

Instance Creation, Estella: https://github.com/htw-imi-info3/python-learning/issues/10

Dictionaries, Jonas T.: https://github.com/htw-imi-info3/python-learning/issues/7

Part 2 - Small Python Exercises

1. List Comprehensions were new to us, so we started reading the Python Tutorial on it, linked in the assignment. We started with a list that we filled using a for loop and worked through the examples using lambda syntax and eventually the List Complesension as short form. We played around these in the iPython REPL and came up with the following lines [28, 32]:

```
In [1]: list1 = []

In [2]: for i in range(1, 21):
   ...:     list1.append(i)
   ...:

In [3]: list1
Out[3]: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
     19, 20]

In [28]: list1 = [x**2 for x in range(1, 21)]

In [29]: list1
Out[29]:
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256,
    289, 324, 361, 400]

In [32]: list2 = [x for x in list1 if x%2 == 0]

In [33]: list2
Out[33]: [4, 16, 36, 64, 100, 144, 196, 256, 324, 400]
```

It took a while to figure out the lambda syntax, but the resulting list comprehensions were easier to read. Another thing to keep in mind is that the upper bound of range is exclusive, the lower one inclusive.

The file `list_comprehensions.py` summarizes all the tested code and the comments document it.

```python
""" This file summarizes the list Comprehension part of lab2, part 2.
The syntax is taken from the tutorial: https://docs.python.org/3/tutorial/datastructures.html#list-comprehensions """

list1 = []

for i in range(1, 21):
    list1.append(i)

print("Careful with the `for i in range` syntax in python, \
this has a side effect and creates a persisting variable, i:", i)
print("list1:", list1)

# approach using a function to square
def square(x):
    return x**2
# the map function returns a 'map object' that can be cast to a list
list_squares = list(map(square, range(1, 21)))
print("list_squares:", list_squares)

# it is possible to write this without defining the function square,
# we can write this with an anonymous fuction, a lambda

list_squares = list(map(lambda x: x**2, range(1, 21)))
print("list_squares:", list_squares)

# This is the basis for a List Comprehension which is basically just
# a short form of the above, we leave out the lambda and map function.
# It also uses square brackets, as when defining a list, instead of a cast
# The keyword `in` is used to point to the mapping iterable

list_squares = [x**2 for x in range(1, 21)]
print("list_squares:", list_squares)

# We can also filter the values with the keyword if, or iterate over more
# than one variable. This can be thought of as nested loops and if statements

list_even_squares = [x**2 for x in range(1, 21) if x**2 % 2 == 0]
print("list_even_squares:", list_even_squares)

# or using the list we created before as iterable, which seems more readable
list_squares = [x**2 for x in range(1, 21)]
```

```python
list_even_squares = [x for x in list_squares if x % 2 == 0]
print("list_even_squares:", list_even_squares)

# x should not be defined at this point,
# the above statements are all free of side effects
try:
    print(x)
except NameError:
    print("x is not defined here, List Comprehensions are free of side effects")

# This is great for simpler lists, but might become less readable
# if complex conditions are used, concatenated with `and`, `or` ...
# In this case it might be useful to move the condition checks into a function.

def complex_condition(x):
    # do some precalculations here ... or use many different conditions or cases
    # then apply complex condition, False is implicit when None is returned
    if x <= 200 and x**2 % 2 == 0:
        return True
    elif x == 400:
        return True

list_squares = [x**2 for x in range(1, 21)]
list_complex_filter = [x for x in list_squares if complex_condition(x)]
print("list_complex_filter:", list_complex_filter)
```

Running the file results in the following output, proving everything works as expected:

```
Careful with the `for i in range` syntax in python, this has a side effect and creates a persisting variable, i: 20
list1: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
list_squares: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400]
list_squares: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400]
list_squares: [1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400]
list_even_squares: [4, 16, 36, 64, 100, 144, 196, 256, 324, 400]
list_even_squares: [4, 16, 36, 64, 100, 144, 196, 256, 324, 400]
x is not defined here, List Comprehensions are free of side effects
list_complex_filter: [4, 16, 36, 64, 100, 144, 196, 400]
```

2. This part is still a WIP. The class stores information about a file, the name, the path, the filesize. I could be extended to include the time the file was last changed or a hash of the file. The module `os` is used to retrieve the file information, based on the path.

```python
import os

""" Class representing a File """
class myFile:
    def __init__(self, path):
        self.path = path
        # self.name = name
        # self.size = size

    def getSize(self):
        size = os.stat(self.name)
        print(size.st_size)

    def getName(self):
        name = os.getcwd()
        print(name)
```

Part 3: Finding large files

Our program is still a WIP. When finished it should prompt for a path and retrieve files that exceed a certain file size. Together with other file stats, this is useful, for example to filter an image folder.

```python
import os
import glob
from myFile import myFile

largeFiles = []
sortedFilesList = []
# path = 'C:/Users/estel/OneDrive/Bilder/Screenshots**/*.png' #path
    example

def getFilesList(path):
    return glob.glob(path, recursive=True)
    # files = glob.glob(path, recursive=True)
    # for file in files:
    #     # files.append(file)
    #     filesList.append(file)
    #     # print(file)

# def sortFilesList():
#     for file in found_file_paths:
#         if file.size >= 10.0:
#             largeFiles.append(file)
#             print(largeFiles)
#         else:
#             sortedFilesList.append(file)
#             print(sortedFilesList)

```

```python
if __name__ == "__main__":
    path = input("Enter a folder path: ")

    # overwrite default path for testing purposes
    path = 'C:/Users/Jonas/Pictures/Wallpaper/**/*.png'
    found_file_paths = getFilesList(path)
    print(found_file_paths)

    filesList = []
    for path in found_file_paths:
        # print(type(path))
        f = myFile(path)
        filesList.append(f)

    print(filesList[:3])
```