Lab Report

Nathalie Huppert (nathalieclaire), Tobias Bayer (abductedRhino)
Assignment: bkleinen.github.io/classes/ss2023/info3/labs/lab-02-python
Repository: github.com/nathalieclaire/lab_02_small_python_exercises

1 Python Learning with Unit Tests

First, we create a new issue with our test classes "test_addition" and "test_class-definition". We work on our tests and close the issue with our last commit. Next, we create a new issue to ask for review. Then we review the test classes of the others and made small changes. See this and that for Nathalie's changes and this and that for Tobias' changes.

2 Small Python Exercises

2.1 Lists

Create a list of numbers between 1 and 20

To create a list of numbers between 1 and 20, we call the `range` constructor with the arguments `start=1`,`stop=21`. We use the returned `range` Object as a parameter in the `list` constructor.

```
list1 = list(range(1, 21))
```

Create a list containing the squares of list1, using List Comprehension

The power operator in python is `**`. To calculate the squares for each item in our `list`, we use a List Comprehension with `x ** 2` as an expression. Next and last in line is a `for` clause. It is part of every List Comprehension, and we use it to iterate all `list1` items. We assign the new `list` Object to `list2`.

```
list2 = [x ** 2 for x in list1]
```

Create a list with only the even values in list1, using List Comprehension

In this List Comprehension, we use `y` as statement, as there is no need to change the `list1` items. We use the same `for` clause as before and follow up with an `if-statement`. It uses the modulo operator to determine if `y` is evenly divisible by two. If this returns `True`, `list[y]` is included in the returned `list`.

```
list3 = [y for y in list1 if y % 2 == 0]
```

2.2 Sorting

Create your own datatype

Our datatype is called `Person` and it's constructor takes parameters `Name`,`Age`. To define our own `class`, we simply declare `class Person`.

```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
```

We define `__init__` to allow the construction of `Person` Objects with our own parameters. Every new instance of `Person` that is created will call it during construction.

Make it sortable

We want to sort Persons by age, low to high. To make a datatype sortable with the `<` operator, it needs to implement `__lt__`. `Fred < Frederike` calls `Fred.__lt__(Frederike)`. If Fred is younger than Frederike, this should return `True`. To archive this, `__lt__` returns `Fred.age < Frederike.age`.

```
class Person:
    # details omitted
    def __lt__(self, other):  # we sorted our persons depending on the age
the person has
        return self.age < other.age
```

All we have to do to sort a `list` of `Person` Objects is to put them in a `list` and call the `sort()`method.

```
a = Person('Anton', 18)
b = Person('Betina', 20)
c = Person('Chad', 30)
d = Person('Dolly', 21)
people = [a, b, c, d]
people.sort()
```

3 Lambdas and List Comprehensions Applied

Use glob to generate the list of files in a directory.

To use `glob`, we need to `import glob` first. The other thing we need is a directory with some test files. We want to simulate the detection of duplicates in the "Downloads" directory. We create the aforementioned directory in our project root. Next, we create some `.txt` files. Some are empty, some contain the same content and others have similar names but differing contents. We create a list of `globs`. All files in "Downloads" that end with `.txt` are included in the list. We can test this by printing the `list` on the console. We use iglob because it can handle large directories better. So if we were to work with a larger directory, `iglob` has the benefit that it has a better runtime compared to `glob.glob`.

```
globs = glob.iglob("./Downloads/*.txt",
                   recursive=True)


#  prints all the file names
def print_globs():
    for globi in globs:
        print(globi)
```

Define Duplicate Files - how to you check if two files contain the same text?

Under normal circumstances, files have the same content if they produce the same md5 hash. Although unlikely in our case, collisions can occur. This means that a file outputs the same hash but the content is not the same. We `import hashlib` to gain access to to `hashlib.md5()`. This allows us to define a method `get_md5()` that returns the `hexdigest` of a file.

```
def get_md5(path):
    hasher = hashlib.md5()
    with open(path, 'rb') as file:
        data = file.read()
        hasher.update(data)
    return hasher.hexdigest()
```

Sketch the algorithm to detect and document duplicate files in pseudo code

Create a data structure that supports detection and documentation of the duplicate files.

We went over this step by accident because we implemented the code instantly. However, what the `get_md5` method does is, it gets the md5 hash code of every file. In the `find_doublettes` method all new hashes are appended to a `list`. Before appending a new hash to a list, the method checks if it already exists in the list and if yes, it prints it out.

```
def find_doublettes(file_paths):
    hashes = []
    for globi in file_paths:
        md = get_md5(globi)
        if hashes.__contains__(md):
            print(globi + ', ' + md)
        else:
            hashes.append(md)
```

How do you test your script?

We test our script manually, meaning that we evaluate the printed output of `find_doublettes`, because a proper test `setUp()` and `tearDown()` would have meant for one of us to allow writing and, in case of `tearDown()`, removing files on our system. Not to say that this is something otherworldly - but we do make mistakes and were afraid that we could accidentally delete files on our computer.

We test the `get_md5` method anyway with a test-hash we took from the "test.txt" file, but we had problems with testing the other method as it just printed something on the console and doesn't actually return a value.

```
class TestPart3(unittest.TestCase):
    def testGet_md5(self):
        with open('test.txt', 'w') as file:
            file.write('')

        test_hash = 'd41d8cd98f00b204e9800998ecf8427e'
        actual_hash = part3.get_md5('test.txt')
        self.assertEqual(test_hash, actual_hash)
        os.remove('test.txt')
```

not bored this time.

4 Sources

Ranges
List Comprehensions
Object.It
Class Objects
iglob advantages hashlib