Python: Advanced stuff - Lab

Table of contents

- 1. Get started
  - 1.1. Assignment
  - 1.2. Hints
    - 1.2.1. Run a Python program
    - 1.2.2. Enable interactive mode
- 2. Duplicate finder
  - 2.1. Problem
    - 2.1.1. Assignment 1
    - 2.1.2. Assignment 2
    - 2.1.3. Assignment 3
  - 2.2. Hints
    - 2.2.1. Checking for equivalent files
- 3. Date object
  - 3.1. Problem
    - 3.1.1. Assignment 1
    - 3.1.2. Assignment 2
    - 3.1.3. Assignment 3
- 4. Solutions
- Bibliography
- Licenses

1. Get started

1.1. Assignment

1. Set up the development environment
2. Try out the examples provided in the previous lectures
   1. Python: Basic stuff - Pt. 1
   2. Python: Basic stuff - Pt. 2
   3. Python: Advanced stuff
3. Complete the previous assignments
   1. Python: Basic stuff - Lab

1.2. Hints

1.2.1. Run a Python program

```
$ python <your_program>.py
```

Depending on your system, you may have to use `python` or `python3`

1.2.2. Enable the interactive mode

```
$ python
```

Depending on your system, you may have to use `python` or `python3`

2. Duplicate finder

2.1. Problem

In a large collection of files, there may be more than one copy of the same file, potentially stored in different directories or with different file names. The goal is to search for duplicates. Use this directory as an example.

2.1.1. Assignment 1

Write a function called `is_image` that takes a path and a list of file extensions, and returns `True` if the path ends with one of the extensions in the list (check out the `str` method `endswith`). Here's an outline of the function that includes doctests. Fill in the function and then check that all tests pass.

```
def is_image(path, extensions):
    """
    Check whether the path ends with one of the extensions

    path: string file path
    extensions: list of extensions

    >>> is_image('photo.jpg', ['jpg', 'jpeg'])
    True
    >>> is_image('PHOTO.JPG', ['jpg', 'jpeg'])
    True
    >>> is_image('notes.txt', ['jpg', 'jpeg'])
    False
    """
    return False
```

2.1.2. Assignment 2

Write a function called `add_path` that takes as arguments a path and a `defaultdict` (see here) and return the updated `defaultdict`. `add_path` add the digest of the path as a key and append the path to a list as a value.

Note that if two files contain the same data, they will have the same digest. If two files differ, they will almost always have different digests. A digest is the output of an hash function. Use `md5` for this purpose.

Here's an outline of the function that includes doctests. Fill in the function and then check that all tests pass.

```
def add_path(path, d):
    """
    Compute the digest of path, add the digest to d as key and append
path to a list as value

    path : path to a file
    d : defaultdict of lists

    >>> add_path('data/photos/feb-2023/photo1.jpg', defaultdict(list))
    defaultdict(<class 'list'>, {'dace5bcdd614b5a23e465b1edc406bc3':
['data/photos/feb-2023/photo1.jpg']})
    """
    return None
```

2.1.3. Assignment 3

Write a version of `walk` (see here) called `walk_images` that takes a directory and walks through the files in this directory and its subdirectories. For each file, it should use `is_image` to check whether it's an image file and `add_path` to add it to the `defaultdict`. Here's an outline of the function.

```
def walk_images(dirname, d):
    """
    Walk the directory tree and return a defaultdict of lists where
    - the key is the digest of the image
    - the value is a list of paths to the images with the same digest

    dirname : path to a directory
    d : defaultdict of lists
    """
```

Add the following `main` function and make it sure it gets called when `duplicate_finder` is executed as a script.

```
def main():
    d = defaultdict(list)
    walk_images(".", d)
    for digest, paths in d.items():
        if len(paths) > 1:
```

```
            for path in paths:
                print(path)
```

Expected output:

```
$ python duplicate_finder.py
./data/photos/mar-2023/photo2.jpg
./data/photos/jan-2023/photo1.jpg
```

2.2. Hints

2.2.1. Checking for equivalent files

The `hashlib` module provides several hash functions, such as `md5`. The following `md5_digest` function takes a filename as a parameter and returns its digest.

```
import hashlib

def md5_digest(filename):
    data = open(filename, 'rb').read()
    md5_hash = hashlib.md5()
    md5_hash.update(data)
    digest = md5_hash.hexdigest()
    return digest
```

`open(filename, 'rb')` opens `filename` for reading in binary mode. In binary mode, the contents are not interpreted as text but treated as a sequence of bytes.

3. Date object

3.1. Problem

Implement a `Date` class for dates in the format `yyyy-mm-dd`.

3.1.1. Assignment 1

Write an `__init__` method (see special methods) that takes `year`, `month`, and `day` as parameters and assigns the parameters to attributes. Use the interactive mode to create an object that represents June 22, 1933.

3.1.2. Assignment 2

Write `__str__` method (see special methods) that uses an f-string to format the attributes and returns the result. If you test it with the `Date` you created in §3.1.1, the result should be `1933-06-22`.

3.1.3. Assignment 3

Make `Date` objects totally ordered. Check that

1. `1933-06-22 > 1900-01-01`
2. `1933-06-22 <= 2000-01-01`
3. `1933-06-22 == 1933-06-22`
4. The method `sort` works correctly on a list of `Date` objects

4. Solutions

Problem	Program	Input
§2	duplicate_finder.py	photos
§3	date.py	n/a

Bibliography

Author	Title	Year
Downey, A.	Think Python	2024

Licenses