

Introduction to Programming Homework 8

Due Friday Nov 18

You will turn in your homework **via GitHub!** Please use this link to start your repository :

<https://classroom.github.com/assignment-invitations/1052153ba07d06b632692d444e737228>

Exercise 1 (Moving files around)

Write a module called `file_sorter.py`

- **a.** Generate a file with at least 50 lines of random text using your `generate_random_text` from the previous homework. Write a function called `make_files(path_to_random_text)` which
 - creates a directory called `sorting_dir`, if it doesn't already exist
 - takes the path to your random text file and creates a blank file in `sorting_dir` for **every word in the file of random text**
 - the names of the blank files should be `<word>.ext`, for example if your random text file contains a line

```
M TiRmMn weyfZVKT WNftXrUrjuLmECV
```

you should generate files with names `M.ext`, `TiRmMn.ext`, `weyfZVKT.ext`, and `WNftXrUrjuLmECV.ext` for that line. Make sure to generate files for **all** lines in your random text file.

- you can either just open and close a file to make a blank file or you can use the function

```
import os
def touch(file_name):
    with open(file_name, 'a') as fp:
        os.utime(fp)
```

- **b.** Write a function called `first_letter_sort(sorting_dir)`, which takes a directory and **moves/sorts all the files** in `sorting_dir` into subdirectories by the first letter of the file name. The subdirectories should just be named by the **uppercase** first letter they represent (and they must be created if they don't already exist).
 - For example, the files `M.ext`, `TiRmMn.ext`, `weyfZVKT.ext`, and `WNftXrUrjuLmECV.ext` would be sorted into directories `M/`, `T/` and `w/` based on the first letter. Note that `weyfZVKT.ext` and `WNftXrUrjuLmECV` would both go into `w/`.

Exercise 2 (Ring of Dual Numbers)

Create a module called `dual_numbers.py`.

- **a.** Create a class called `DualNumber` which should represent an element of the ring $\mathbb{R}[\epsilon]/(\epsilon^2)$. Here, \mathbb{R} will just be floats.
 - your `__init__` method should build a dual number $z = a + b\epsilon$
 - define **readonly** property attributes `.real` and `.dual` that return a and b , respectively, for `DualNumber(a,b)`.
 - this means you **don't** need to write setters.
 - define `==` for `DualNumber`.
 - for example, `DualNumber(1,2) == DualNumber(1,2)` should be `True`.
 - make it so that `str(DualNumber(1.67,1.4)) == '1.67 + 1.4 eps'` and `repr(DualNumber(1.7,1)) == 'DualNumber(1.7,1)'`
 - your `DualNumber` should support all ring operations, i.e. `+`, `-`, `*`.
 - for example, `DualNumber(1,2) + DualNumber(2,1) == DualNumber(3,3)` should be `True`.
 - define `**` such that `z**x` works for an `int x >= 0` and a `DualNumber z`.
 - define `/` for `DualNumber` but return `NotImplemented` when you cannot perform the division.
 - all of these methods should return **new** `DualNumber` instances.

Note : the idea here is that, once created, a `DualNumber` is immutable.

- **b.** If we want to do operations like `5 + DualNumber(2,1)`, we need to define some extra methods. Since `__add__` is always called on the **left** operand, `5 + DualNumber(2,1)` will fail as `int` doesn't know how to add a `DualNumber`. In the case of failure, python will try to call the `__radd__` method on the **right** operand. Your tasks are :
 - allow for adding `int` and `float` to a `DualNumber`
 - make sure that `__add__(self, other)` works when `other` is a `DualNumber`, `float`, or `int` and define a method `__radd__(self, other)` to just return `self + other`.
 - allow for subtracting a `DualNumber` from `int` and `float` (you'll need to define `__rsub__`)
 - allow for multiplying a `DualNumber` by `int` and `float` (you'll need to define `__rmul__`)
 - allow for `/` with `int` and `float` (you'll need to define `__rtruediv__`)

At the end, calls like `5 + DualNumber(2,1)`, `5.4*DualNumber(2,1)`, and `1.2/DualNumber(2.7,1)` should all work.

- **c.** define a global function `derivative(f, a)`, which takes a **rational function** $f(x)$ defined with only the `+`, `-`, `*`, `/`, and `**` operations performed on `x`, and returns $f'(a)$.
 - for example, let

```
def f(x) :
    return x**2 + 2
```

then `derivative(f, 1.5)` should be `3`. (or some very close float).

- hint : use dual numbers!