HOMEWORK 3

Due Thursday, Feb 2, at 11pm

Please enter your answers into a Jupyter notebook and submit by the deadline via canvas.

Problem 1. Sieve of Erasthotenes.

Using the `isprime(n)` function from before, write a function `primes(m)` that returns a list of prime numbers between 1 and m. Approximately how many operations does it take to make this list?

The sieve of erasthotenes is a method for producing these primes more efficiently. The idea is to start with a list of numbers,

$$2, 3, 4, 5, 6, 7, 8, 9, 10, \ldots, m$$

strike out all the multiples of 2 (starting from 4),

$$2, 3, \cancel{4}, 5, \cancel{6}, 7, \cancel{8}, 9, \cancel{10}, \ldots, m$$

then strike out all the multiples of the next unmarked number, which is 3,

$$2, 3, \cancel{4}, 5, \cancel{6}, 7, \cancel{8}, \cancel{9}, \cancel{10}, \ldots, m$$

the the next..., which is 5...

The remaning numbers are the prime numbers.

Implement the sieve of Erasthotenes using Python lists and make a function `primes(m)` that returns a list of prime numbers between 1 and m.

One thing you may (or may not) find useful is to create a list where every element is the same. `xs = [0 for i in range(10)]` will have value `[0,0,0,0,0,0,0,0,0,0]`.

Problem 2. Quick problems about lists. Use list comprehension (i.e. the `[blah for i in some_list]`) syntax, and list comprehension with `if`, i.e. (i.e. the `[blah for i in some_list if some_condition]`), list concatenation, number*list etc. (try not to use for loops).

Produce the following lists in Python.

(1) `[1,3,5,7,..,51]`
(2) `[21,23,25,27,..,51]`
(3) `[1,2,4,8,16,.., 1048576]`

(4) `[1,-1,1,-1,..,1,-1]` (length 20)

(5) `[0,1,1,1,..,1]` (length 20)

(6) `[1,0,1,0,0,1,0,0,0,1,0,0,0,0,1,0,0,0,0,0,1,..]` (length 100; looping might be easier for this one, but can you do it with only one loop?)

Problem 3. For each of the following, write the described function. Please pay attention to the description of the function, some of them say to return a new list, which means that your function shouldn't modify `xs`, but should make a fresh list and return that; others (e.g. `reverse`) ask you to modify xs (these should return `xs` anyway). Check each function on at least two examples:

(1) `f(xs,k)`, returns the kth from last element of `xs`.
example: `f([1,2,3,4], 1)` should return `4`.
example: `f([1,2,3,4], 3)` should return `2`.

(2) `damean(xs)`, returns the mean (average) of the elements of xs.
example: `damean([1,2,3,4])` should return `2.5`.

(3) `appearso(xs,ys)` returns the number of times `xs` appears in `ys` as a subsequence.
example: `appearso([1,2],[1,2,3,4,1,1,2,1])` should return `2`.

(4) `reverse(xs)`, reverses the order of the lements of `xs` and returns `xs`.
example: if `xs = [1,2,3,4]`, then `reverse(xs)` should return `[4,3,2,1]`, and afterwards if we print `xs`, we should see `[4,3,2,1]`.

(5) `shift(cs)`, shifts the elements to the left by one, i.e. `shift([1,2,3,4])` is `[2,3,4,1]`, and returns `xs`.

(6) `ispalin(xs)` returns True if xs is a palindrome (a palindrom is a sequnce that is the same in reverse, example: "amanaplanacanalpanama").

(7) `no_duplicate(xs)`, returns a new list with the same elements as x but without any duplicate elements.
example: `no_duplicate([1,2,3,8,1,4,4,5,1])` returns `[1,2,3,8,4,5]`.

(8) `dupli(xs,k)`, returns a new list with the same elements as x but each one is duplicated k times. e.g. `dupli([1,2,3,2,4], 3)` is `[1,1,1,2,2,2,3,3,3,2,2,2,4,4,4]`.

Problem 4. Prime factorize like a boss. This is a hard problem. You may need to work on it for a long time, start from small numbers and simple examples and don't get discouraged!

The aim is to print out all the prime factors of numbers 1,...,m. The first idea for the algorithm would be the following:

```
for i in 1,...,m
    print: factors of i are:
    for j in 1,...,i
        if j is prime and divides i, print j (as many times as it divides i)
```

But this would be very inefficient because if we are trying to factorize, say 54, and we know 2 is a factor, then finding the remaining factors is the same as finding the factors of 27, which we may have already factorized. The idea for this problem is to use lists to store values that we compute so that we don't do redundant computations.

The underlying mathematical point is that if p is a prime that divides n, then the prime factors of n, are p and the prime factors of n/p.

We will use a lists of lists to keep the factors of the smaller numbers:

```
m = 20
pfactors = [[] for i in range(m)]

for i in range(2,m):
  ... find the first prime factor j of i,
  ... add the previously computed factors of j to the factors of i (factors[i].
      extend(?))
  ... add j to the factors of i, by using factors[i].append(?)
```

Print out all the factors up to $m = 200$.