Mid-term exam

COMP9021, Session 1, April 2016

1 Longest sequence of deficient numbers

A number is said to be *deficient* if the sum of its proper divisors is strictly smaller than itself. For instance, the proper divisors of 10 are 1, 2 and 5, which add up to 8, which is strictly smaller than 10, hence 10 is deficient, whereas the proper divisors of 12 are 1, 2, 3, 4 and 6, which add up to 16, which is greater than or equal to 12, hence 12 is not deficient. Here is the list of all deficient numbers up to 50:

$$1, 2, 3, 4, 5, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17, 19, 21, 22, 23, 25, 26, 27, 29, 31, 32, 33, 34, 35, 37, 38, 39, 41, 43, 44, 45, 46, 47, 49, 50$$

Write a program that prompts the user to input two strictly positive integers a and b and finds out the range of the longest sequence of deficient numbers between a and b. When that sequence is not unique, the range should be that of the sequence closest to a. If a is smaller than b then that sequence is increasing, whereas if a is greater than b then that sequence is decreasing.

Here is a typical interaction:

```
$ python3 deficient.py
Enter two strictly positive numbers: 10 twenty
Incorrect input, giving up.
$ python3 deficient.py
Enter two strictly positive numbers: 10 20 30
Incorrect input, giving up.
$ python3 deficient.py
Enter two strictly positive numbers: 0 10
Incorrect input, giving up.
$ python3 deficient.py
Enter two strictly positive numbers: 10 10
The longest sequence of deficient numbers between 10 and 10 ranges between 10 and 10.
$ python3 deficient.py
Enter two strictly positive numbers: 12 12
There is no deficient number between 12 and 12.
$ python3 deficient.py
Enter two strictly positive numbers: 10 20
The longest sequence of deficient numbers between 10 and 20 ranges between 13 and 17.
$ python3 deficient.py
Enter two strictly positive numbers: 20 10
The longest sequence of deficient numbers between 20 and 10 ranges between 17 and 13.
$ python3 deficient.py
Enter two strictly positive numbers: 3 20
The longest sequence of deficient numbers between 3 and 20 ranges between 7 and 11.
$ python3 deficient.py
Enter two strictly positive numbers: 20 3
The longest sequence of deficient numbers between 20 and 3 ranges between 17 and 13.
```

If you do not have enough time, or if you find the whole question to be too difficult, simplify it in one or two of the following ways:

- only deal with the case where a is smaller than or equal to b;
- do not validate the input, assume that the user inputs two strictly positive integers.

Recall that if you want to exit a script, you execute the `exit()` function from the `sys` module. Also recall that `int()` is called to convert a string into an integer, and if the conversion fails then `int()` raises a `ValueError` exception.

The name of your program should be `deficient.py`. Just edit a file with that name in the working directory. It will be collected there and does not have to be "submitted."

2 Solving a multiplication

Write a program that solves a multiplication

```
  O E E
x   E E
  -----
E O E E
E O E
-------
O O E E
```

where `E` stands for an even digit and `O` stands for an odd digit. Of course the leading digit of any of the 5 numbers cannot be 0. Two occurrences of `E` can refer to the same even digit or to two distinct even digits, and two occurrences of `O` can refer to the same odd digit or to two distinct odd digits.

The output of your program should be of the form above, with of course the `E`s and the `O`s being replaced by appropriate digits. Still spaces are irrelevant; in particular, it is irrelevant whether digits are separated by spaces, whether `x` (the multiplication sign) is followed by spaces, whether there are leading spaces at the beginning of the last line. The number of `-` on the separating lines is also irrelevant.

Your program should not make any assumption on the actual solution (which is unique), except obvious ones on the ranges of some values.

The name of your program should be `multiplication.py`. Just edit a file with that name in your working directory. It will be collected there and does not have to be "submitted."

3 Decoding a number

Write a program that implements a function named `decode`, called as `decode(base, number)` where `base` is any number b between 1 and 9, and `number` is any nonnegative integer n, seen as encoding a set S of nonnegative integers such that for all nonnegative integers i, i belongs to S iff the ith digit of n written in base b is not equal to 0 (counting the digits from 0 and starting from the right). For instance, in base 3, 586 reads as 210201 because

$$586 = 2 \times 3^5 + 1 \times 3^4 + 0 \times 3^3 + 2 \times 3^2 + 0 \times 3^1 + 1 \times 3^0$$

and so, with `base` equal to 3, 586 encodes $\{5, 4, 2, 0\}$.

Here is a typical interaction:

```
$ python3
...
>>> from decoding import *
>>> decode(2, 11)
{3, 1, 0}
>>> decode(3, 586)
{5, 4, 2, 0}
>>> decode(8, 4223296)
{7, 4, 2}
>>> decode(9, 0)
{}
```

For this question, the output has to be **exactly** as shown, there has to be a single space after every comma and only after every comma. Note that the digits are output from largest to smallest.

You do not have to validate input, you can assume that the arguments provided to the function `decode()` are what they should be (a number between 1 and 9 for the first argument, and an arbitrary nonnegative number for the second argument).

Recall that `dir(list)` will list all list methods (that might or might not be useful to you depending on your approach), and `help(print)` will provide you with the necessary information about the `print()` function.

The name of your program should be `decoding.py`. Just edit a file with that name in the working directory. It will be collected there and does not have to be "submitted."

4 Words built from characters

Write a program that opens a file named `text_file.txt`, stored in the working directory, prompts the user for characters all on one line, and outputs all words in the text built from those characters, classified in increasing length, and for a given length output lexicographically. We make the following assumptions:

- The text in `text_file.txt` consists of nothing but lowercase and uppercase letters, hyphens, commas, full stops, double quotation marks, spaces and new line characters.
- Words in the text are longest sequences of letters and hyphens, so:
  - except possibly for the first word, words are preceded by a space or a new line character or a double quotation mark;
  - every word is followed by either a space, a new line character, a comma, a full stop or a double quotation mark.
- Commas and full stops precede double quotation marks, not the other way around.

Moreover,

- Spaces in the sequence of characters input by the user are ignored.
- The case of letters is irrelevant, both for the letters in the sequence of characters input by the user and for the letters in the text file.
- Words are output all lowercase.

For instance, if the user inputs `AAABC n2-` and the text contains `Ann` and `ann`, then it would be found out that both `Ann` and `ann` are built from `AAABC n2-`, and only `ann` would be output.

You are provided with a sample file `text_file.txt`. Here is a possible interaction when this file is being used.

```
$ python3 words.py
Enter characters (spaces will be ignored): ADN   A D N   A   D   N
Words of length 1 built from these characters, in lexicographic order:
    a
Words of length 3 built from these characters, in lexicographic order:
    and
$ python3 words.py
Enter characters (spaces will be ignored): cluuud IN    DeD 23*
Words of length 2 built from these characters, in lexicographic order:
    in
Words of length 4 built from these characters, in lexicographic order:
    nice
Words of length 8 built from these characters, in lexicographic order:
    included
```

```
$ python3 words.py
Enter characters (spaces will be ignored): CLGS   AOEI
Words of length 1 built from these characters, in lexicographic order:
    a
Words of length 2 built from these characters, in lexicographic order:
    as
    is
    so
Words of length 4 built from these characters, in lexicographic order:
    also
Words of length 8 built from these characters, in lexicographic order:
    colleges
$ python3 words.py
Enter characters (spaces will be ignored): NSCRT - oooe+*
Words of length 2 built from these characters, in lexicographic order:
    so
    to
Words of length 3 built from these characters, in lexicographic order:
    one
    too
Words of length 4 built from these characters, in lexicographic order:
    once
Words of length 5 built from these characters, in lexicographic order:
    terse
Words of length 6 built from these characters, in lexicographic order:
    sooner
Words of length 13 built from these characters, in lexicographic order:
    not-so-secret
```

Recall that `dir(str)` will list all string methods, and help(str.method_name) will provide you with a description of `str.method_name()`. Recall that `sorted()` returns a sorted list of its input.

The name of your program should be `words.py`. Just edit a file with that name in the working directory. It will be collected there and does not have to be "submitted."