COMP 112 - Lab 4

Due: 11:59pm, Friday December 5

Handin: place your `lab4.py` file in the folder `~/comp112/handin/lab4/`. Remember to issue the permissions command after you hand in the file:

```
setfacl -R -m u:rthorndy:rwx comp112/handin
setfacl -R -m m:rwx comp112/handin
```

For this lab you must provide a single python program that can perform all the calculations specified below, similar to the structure used in Lab 3. Your program will take a single command-line argument that tells the program which question is to be calculated, and then uses functions to produce the output.

For example, to execute Question 2, you would type:

```
$python lab4.py q2
```

You can use I/O redirection as well to make it easier to create test inputs, as was done in the previous two labs. For example, for Question 1, you need to provide 2 integers as input. This is tedious when testing, so you can put the two integers on separate lines in a file called "`q1input1.txt`", and send it to your program as input:

```
$python lab4.py q1 < q1input1.txt
```

Question 1 (execution code "q1")

Write a function that computes the time difference in hours and minutes between two user-supplied times.

The time of day will be entered as a 3- or 4-digit integer, in the form HHMM or HMM, with the hour specified in 24-hour format. Here are some examples:

10:15am -> `1015`
10:15pm -> `2215`
7:22am -> `722`
7:22pm -> `1922`
2:07pm -> `1407`
12:00am (midnight) -> `000`
12:15am -> `015`

You may assume that the first input is always before the second input, and that the inputs are less than 24 hours apart from each other. You may *not* assume that the first input is a smaller integer than the second input, however!! In other words, the time span may cross from one day to the next, past midnight.

The function should return a string (and the program should print this string by itself on a line) that states the integer number of hours elapsed, followed by a colon, followed by the integer number of minutes elapsed. Of course, the number of minutes should only be between 0 and 59, inclusive.

The hours should not be zero-padded (except for the actual zero value, which is just the single digit '0'); minutes *should* be zero-padded, so it will always be exactly 2 digits.

Here are some sample outputs:

```
1:15
10:04
0:15
19:00
```

Question 2 (execution code "q2")

You are to write a function that returns the boolean `True` or `False` to state whether a positive integer is a prime number of not.

Remember that a prime number is a positive integer that has no factors other than 1 and itself. The integer 2 is considered to be the smallest prime number.

The input is just a single positive integer, and the output is just the word True or False.

Question 3 (execution code "q3")

You are to write a function that returns one of three strings:

- "abundant" - if the input positive integer is abundant;
- "perfect" - if the input positive integer is perfect;
- "deficient" - if the input positive integer is deficient.

These three classifications relate to the sum of the number's proper divisors (i.e. not including itself): "abundant" means the sum is greater than the number itself; "perfect" means the sum is equal to the number itself; and "deficient" means the sum is less than the number itself.

For example:

- 12 is “abundant”, because the proper divisors of 12 are 1, 2, 3, 4 and 6, which add to 16, and 16 > 12;
- 6 is “perfect”, because the proper divisors of 6 are 1, 2 and 3, which add up to 6; and
- 9 is “deficient”, because the proper divisors of 9 are 1 and 3, which add up to 4, and 4 < 9.

The function takes a single, positive integer as input, and outputs one of the three strings as described above.

HINTS

- For Questions 2 & 3, there could be some common sub-task. Could you write a function that could be used in *both* questions, rather than duplicate the work separately for each?

- For Question 1, when entering integers, many shells and interpreters consider literal integers that start with ‘0’ (zero) to be in base-8 format (octal). This is true for Python, so if you try to use `x=input()`, and the user types in `015` (i.e. quarter past midnight), the interpreter will actually read this as the base-10 integer `13` (`1*8^1 + 5*8^0`).

  To get around this, you have to read the input using `raw_input()`. This will make Python first read it as text characters, so it won’t convert from octal … it will literally be the characters zero, one and five. You then convert ***the text string*** to an `int`, using `int()`. This bypasses the interpreter and just does a base-10 conversion:

  ```
  x = int(raw_input())
  ```

- For Question 1, one of the tricker steps involves converting that input integer into separate hours and minutes. There are different techniques, including:

  - string slices - keep the input as a raw text string, and chop it up using slices. Be careful with the different possible lengths of the input values.

  - integer division - you can use ‘/’ and ‘%’ to pull out the hour and minute values quite easily, and would be good practice in using these important operations!