Assignment Guidelines:

- This assignment covers material in Module 2.
- Submission details:
  - Solutions to these questions must be placed in files a02q1.py, a02q2.py, a02q3.py, and a02q4.py, respectively, and must be completed using Python 3.
  - Download the interface file from the course Web page to ensure that all function names are spelled correctly and each function has the correct number and order of parameters.
  - All solutions must be submitted to MarkUs. No solutions will be accepted through email, even if you are having issues with MarkUs.
  - Verify using MarkUs and your basic test results that your files were properly submitted and are readable on MarkUs.
  - For full style marks, your program must follow the Python section of the CS116 Style Guide.
  - Helper functions need design recipe elements but not examples and tests.
- Download the testing module from the course web page. Include `import check` in each solution file.
  - When a function produces a floating point value, you must use `check.within` for your testing. Unless told otherwise, you may use a tolerance of 0.00001 in your tests.
  - Test data for all questions will always meet the stated assumptions for consumed values.
- Restrictions:
  - Do not import any modules other than `math` and `check`.
  - Do not use Python constructs from later modules (e.g. loops and lists). Do not use any other Python functions not discussed in class or explicitly allowed elsewhere. See the allowable functions post on Piazza. You are always allowed to define your own helper functions, as long as they meet the assignment restrictions.
  - While you may use global `constants` in your solutions, do not use global `variables` for anything other than testing.
  - Read each question carefully for additional restrictions.
  - The solutions you submit must be entirely your own work. Do not look up either full or partial solutions on the Internet or in printed sources.
1 Conditional Terms

In a simplified version of the rules for the standings for Computer Science (CS) students, a CS student can be in one of three different standings, namely "fine", "conditional" or "out". These are determined as follows:

- If your **new CS average** (that is, the average of all of your CS courses) exceeds or equals 60%, then you are not in conditional standing; everything is fine. We call this "fine".
- If your **CS average** is below 60%, you are given a conditional term. We will call this "conditional". In your conditional term:
  - If after the current term, your **new CS average** is greater than or equal to 60%, then you are not in conditional standing anymore (you are "fine").
  - If in this current term, your **CS term average** is 65% or better but your **new CS average** is still below 60%, you are put once again in a "conditional" term.
  - If you fail to achieve a **CS term average** of 65% or better and your **new CS average** is still below 60%, then you are out of CS and we say you are "out".

Your goal is to write a function

```
standing(cur_cs_avg, cur_num_cs_courses, c1, c2)
```

that consumes a student's current CS average in `cur_cs_avg` and the student's total number of courses to get this average in `cur_num_cs_courses` as well as the grades of two new CS courses that the student is taking this term in `c1` and `c2`. Your function should return one of "fine", "conditional" or "out" according to the above rules.

You may make the following assumptions but should document these appropriately:

- Each of `c1` and `c2` is a natural number between 0 and 100.
- The value of `cur_cs_avg` is a rational number between 0 and 100.
- The value of `cur_num_cs_courses` is at least 1.

Sample:

```
standing(59.0, 1, 40, 40) => "out"
standing(59.0, 1, 61, 60) => "fine"
standing(30.0, 1, 65, 65) => "conditional"
standing(60.0, 1, 40, 40) => "conditional"
standing(80.0, 2, 40, 40) => "fine"
```

2 Mortgages

To purchase a home, prospective buyers combine a down payment and a mortgage. A down payment is the initial money a person has to use towards the purchase of a house and a mortgage is a loan a person can get to borrow money to purchase a house (note it is possible to not need a mortgage).

Each payment period, part of your payment goes towards paying interest and the remainder (if any) of your payment goes towards your principal, that is, the amount that will reduce your mortgage. These mortgage loans are given a fixed interest rate which we call the annual rate and are required to be paid back in a fixed number of years, that is, banks will figure out how much you need to pay monthly in order for your loan to be completely repaid in a set amount of time. Your goal in this question is to compute this amount. To compute how much your monthly payment $P$ will be, a simple formula can be used:

\[
P = \frac{ar/12}{1 - (1 + r/12)^{-12n}}
\]

where $a$ is the initial amount you are borrowing, $r$ is the annual interest rate and $n$ is the number of years you will have the mortgage.
However, there is a catch. If your down payment is less than 20% of the cost of the house, the government of Canada forces you to purchase CMHC mortgage loan insurance based on the percentage of your down payment was made. This amount is added to your mortgage total. As of March 17th, 2017, this is computed as a percentage of the mortgage amount and is based on the percentage of the cost of the house that you paid up front in your down payment as is indicated by the table headings below:

<table>
<thead>
<tr>
<th>At least 5% and less than 10%</th>
<th>At least 10% and less than 15%</th>
<th>At least 15% and less than 20%</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.00%</td>
<td>3.10%</td>
<td>2.80%</td>
</tr>
</tbody>
</table>

For example, if a house cost $100 000 and you had $10 000 as a down payment (that is 10% of the total), you would need a $92 790 mortgage computed by:

$$(100000 - 10000) + (100000 - 10000) \cdot 0.031 = 92790$$

where the 0.031 is the 3.1% from the table above.

Write a function

```python
monthly_payment(cost_of_house, down_payment, annual_rate, years)
```

which computes the monthly payment of the mortgage needed. Note the following:

- `cost_of_house` is a positive natural number.
- `years` is between 1 and 30 inclusive.
- `down_payment` is a positive natural number that is at least 5% of the cost of the house and is no greater than the cost of the house.
- `annual_rate` is a value in the open interval (0, 1).

Sample:

```python
monthly_payment(100000, 10000, 0.05, 30) => 498.11678399296454
monthly_payment(100000, 10000, 0.05, 25) => 542.4410995152548
```

3 Perfection

Let \(\sigma(n)\) represent the sum of all positive proper divisors of an integer \(n\), that is, the sum of the positive divisors not equal to the number. For example, \(\sigma(16) = 15\) since the positive proper divisors of 16 are 1, 2, 4, 8 and their sum is 15. Then, \(n\) is...

- ... abundant if and only if \(\sigma(n) > n\).
- ... perfect if and only if \(\sigma(n) = n\).
- ... deficient if and only if \(\sigma(n) < n\).

Write a function `perfection(n)` that determines whether or not a number is abundant, deficient or perfect. Return exactly one of "abundant", "perfect" or "deficient" depending on the above scheme. You may assume that \(n\) is a natural number between 1 and 1000 inclusive.

Sample:

```python
perfection(6) => "perfect"
perfection(3) => "deficient"
perfection(24) => "abundant"
```
4 Lucky Sevens

Write a function `lucky_sevens(a, b)` which returns the number of times the digit 7 would appear if you wrote all the integers between the integers `a` and `b` inclusive. You may assume that `-10000 < a <= b < 10000` holds and that `|b - a| <= 1000`.

Sample:

`lucky_sevens(1, 10)` => 1
`lucky_sevens(5, 70)` => 8
`lucky_sevens(76, 78)` => 4