Assignment Guidelines:

- This assignment covers material in Module 1.
- Submission details:
  - Solutions to these questions must be placed in files `a01q1.py`, `a01q2.py`, `a01q3.py`, and `a01q4.py`, respectively, and must be completed using Python 3.
  - Download the interface file from the course website 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 CS 116 Style Guide.
  - Be sure to review the Academic Integrity policy on the Assignments page.
  - Helper functions need design recipe elements but not examples and tests.
- Download the testing module from the course website. 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. For any questions with floating point values, you needn't get the exact answer that we do but you must be within this tolerance.
  - 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`.
  - You are always allowed to define your own helper functions, as long as they meet the assignment restrictions. Do **not** use Python constructs from later modules (e.g. `if` statements, loops, slicing, string methods and/or lists). Use only the functions and methods as follows:
    - `abs`
    - `len`
    - `max` and `min`
    - Any method or constant in the `math` module
    - Any basic arithmetic operation (including `+, -, *, /, //, %, **`)
    - Typecasting including `int()`, `str()`, `float()`
  - 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 Pickled Eggs

Pickled eggs are one of the staples in many bars in North America. Eggs are placed in a jar and then the jar is completely filled with brine. How much brine is needed to completely fill the jar after the eggs have been filled in? Write a function

`brine(num_eggs, egg_radius, jar_radius, jar_height)`

that consumes the number of identical spherical eggs, the radius of the eggs and the dimensions of the cylindrical jar in their respective parameters above and returns the volume of brine in the jar. You may assume that the jar can fit the number of eggs given.

Sample:

```
brine(20, 2.0, 5.25, 100.0) => 7988.808318691044
```

As always, unless otherwise stated, use a tolerance of 0.00001.

2 Rule of 72

In finance, a common question to ask is "How many years will it take to double an investment?" Recall that given an annual interest rate i and an initial amount P (for principal), the amount of money A after n years is given by

$$A = P(1+i)^n$$

This formula, while exact, can be hard to compute a value for n without a calculator. Instead, people use the "Rule of 72". To estimate how long it will take to a principal P to double, one can compute $72/(100 \cdot i)$. So for example, if your annual interest rate was 8%, we would compute using the rule of 72 that it would take $72/(100 \cdot 0.08) = 9$ years for your investment to double. Using the formula $A = P(1+i)^n$, we can that the exact value is $n = 9.006468342...$, remarkably close to the estimate. Your job is to compute the absolute error between these two values (that is, compute the absolute value of the difference between these two computations). Write the function

`doubling_time(i)`

that consumes a positive floating point interest rate and returns this error.

Sample:

```
doubling_time(0.08) => 0.006468342000587768
```

As always, unless otherwise stated, use a tolerance of 0.00001.

3 Reverse

Given a positive four-digit number `n`, write a function

`reverse(n)`

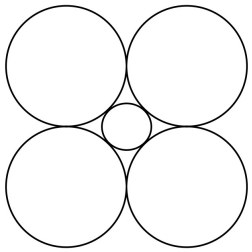
that returns the reverse of the number, that is, the number given by the digits of `n` in reverse order.

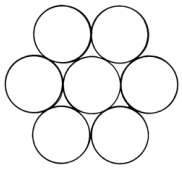
Sample:

```
reverse(1234) => 4321
reverse(1200) => 21
```

4 Tangent Circles

Consider the following two arrangements of circles:





Notice that the central circle is surrounded by 4 other circles of equal radii in the diagram on the left and the central circle on the right is surrounded by 6 circles of equal radii. A question one might ask is given a circle of any radius r, can one surround this circle with n other circles of equal radius s (not necessarily equal to r) such that consecutive circles are tangent and each circle is tangent to the central one? (Recall that two circles are tangent to each other if and only if they touch at a single point.) It turns out that this is always possible if $n \geq 3$. Your goal is given an inner radius r and a number of circles n, determine the size of the radius s of the outer circles. Write the function

`outer_radius(r, n)`

that returns the value of this radius. A reminder to use `check_within` with a tolerance of `0.00001`.

Sample:

```
outer_radius(1.0, 6) => 0.9999999999999999
outer_radius(2.0, 4) => 4.828427124746189
```