CS 61A: Homework 4

Due by 11:59pm on Monday, 2/23

Instructions

Download hw04.zip. Inside the archive, you will find a file called hw04.py, along with a copy of the OK autograder.

Submission: When you are done, submit with `python3 ok --submit`. You may submit more than once before the deadline; only the final submission will be scored.

Using OK

The `ok` program helps you test your code and track your progress. The first time you run the autograder, you will be asked to log in with your @berkeley.edu account using your web browser. Please do so. Each time you run ok, it will back up your work and progress on our servers. You can run all the doctests with the following command:

```
python3 ok
```

To test a specific question, use the `-q` option with the name of the function:

```
python3 ok -q <function>
```

By default, only tests that fail will appear. If you want to see how you did on all tests, you can use the `-v` option:

```
python3 ok -v
```

If you do not want to send your progress to our server or you have any problems logging in, add the `--local` flag to block all communication:

```
python3 ok --local
```

When you are ready to submit, run `ok` with the `--submit` option:

```
python3 ok --submit
```

Readings: You might find the following references useful:

- Section 2.2

Table of Contents

Acknowledgements. This interval arithmetic example is based on Structure and Interpretation of Computer Programs,Section 2.1.4.

Introduction. Alyssa P. Hacker is designing a system to help people solve engineering problems. One feature she wants to provide in her system is the ability to manipulate inexact quantities (such as measured parameters of physical devices) with known precision, so that when computations are done with such approximate quantities the results will be numbers of known precision.

Alyssa's idea is to implement interval arithmetic as a set of arithmetic operations for combining "intervals" (objects that represent the range of possible values of an inexact quantity). The result of adding, subracting, multiplying, or dividing two intervals is itself an interval, representing the range of the result.

Alyssa postulates the existence of an abstract object called an "interval" that has two endpoints: a lower bound and an upper bound. She also presumes that, given the endpoints of an interval, she can construct the interval using the data constructor interval. Using the constructor and selectors, she defines the following operations:

```
def str_interval(x):
  """Return a string representation of interval x.

  >>> str_interval(interval(-1, 2))
  '-1 to 2'
  """
  return '{0} to {1}'.format(lower_bound(x), upper_bound(x))

def add_interval(x, y):
  """Return an interval that contains the sum of any value in interval x and
  any value in interval y.

  >>> str_interval(add_interval(interval(-1, 2), interval(4, 8)))
  '3 to 10'
  """
  lower = lower_bound(x) + lower_bound(y)
```

```
    upper = upper_bound(x) + upper_bound(y)
    return interval(lower, upper)

def mul_interval(x, y):
    """Return the interval that contains the product of any value in x and any
    value in y.

    >>> str_interval(mul_interval(interval(-1, 2), interval(4, 8)))
    '-8 to 16'
    """
    p1 = lower_bound(x) * lower_bound(y)
    p2 = lower_bound(x) * upper_bound(y)
    p3 = upper_bound(x) * lower_bound(y)
    p4 = upper_bound(x) * upper_bound(y)
    return interval(min(p1, p2, p3, p4), max(p1, p2, p3, p4))
```

Question 1

Alyssa's program is incomplete because she has not specified the implementation of the interval abstraction. Define the constructor and selectors in terms of two-element lists:

```
def interval(a, b):
    """Construct an interval from a to b."""
    "*** YOUR CODE HERE ***"

def lower_bound(x):
    """Return the lower bound of interval x."""
    "*** YOUR CODE HERE ***"

def upper_bound(x):
    """Return the upper bound of interval x."""
    "*** YOUR CODE HERE ***"
```

Question 2

Alyssa implements division below, by multiplying by the reciprocal of `y`. Ben Bitdiddle, an expert systems programmer, looks over Alyssa's shoulder and comments that it is not clear what it means to divide by an interval that spans zero. Add an assert statement to Alyssa's code to ensure that no such interval is used as a divisor:

```
def div_interval(x, y):
  """Return the interval that contains the quotient of any value in x divided
  by any value in y.

  Division is implemented as the multiplication of x by the reciprocal of y.

  >>> str_interval(div_interval(interval(-1, 2), interval(4, 8)))
  '-0.25 to 0.5'
  """
  "*** YOUR CODE HERE ***"
  reciprocal_y = interval(1/upper_bound(y), 1/lower_bound(y))
  return mul_interval(x, reciprocal_y)
```

Question 3

Using reasoning analogous to Alyssa's, define a subtraction function for intervals:

```
def sub_interval(x, y):
  """Return the interval that contains the difference between any value in x
  and any value in y.

  >>> str_interval(sub_interval(interval(-1, 2), interval(4, 8)))
  '-9 to -2'
  """
  "*** YOUR CODE HERE ***"
```

Question 4

After considerable work, Alyssa P. Hacker delivers her finished system. Several years later, after she has forgotten all about it, she gets a frenzied call from an irate user, Lem E. Tweakit. It seems that Lem has noticed that the formula for parallel resistors can be written in two algebraically equivalent ways:

```
par1(r1, r2) = (r1 * r2) / (r1 + r2)
```

or

```
par2(r1, r2) = 1 / (1/r1 + 1/r2)
```

He has written the following two programs, each of which computes
the parallel_resistors formula differently::

```
def par1(r1, r2):
  return div_interval(mul_interval(r1, r2), add_interval(r1, r2))

def par2(r1, r2):
  one = interval(1, 1)
  rep_r1 = div_interval(one, r1)
  rep_r2 = div_interval(one, r2)
  return div_interval(one, add_interval(rep_r1, rep_r2))
```

Lem complains that Alyssa's program gives different answers for the two ways of computing. This is a serious complaint.

Demonstrate that Lem is right. Investigate the behavior of the system on a variety of arithmetic expressions. Make some intervals a and b, and show that par1 and par2 can give different results. You will get the most insight by using intervals whose width is a small percentage of the center value:

```
# These two intervals give different results for parallel resistors:
"*** YOUR CODE HERE ***"
```

Note: No tests will be run on your solution to this problem. Any answer will be accepted, but please attempt to answer the question correctly.

Question 5

Eva Lu Ator, another user, has also noticed the different intervals computed by different but algebraically equivalent expressions. She says that the problem is multiple references to the same interval.

The Multiple References Problem: a formula to compute with intervals using Alyssa's system will produce tighter error bounds if it can be written in such a form that no variable that represents an uncertain number is repeated.

Thus, she says, par2 is a better program for parallel resistances than par1. Is she right? Why? Write a function that returns a string containing a written explanation of your answer:

```
def multiple_references_explanation():
  return """The mulitple reference problem..."""
```

Note: No tests will be run on your solution to this problem. Any answer will be accepted, but please attempt to answer the question correctly.

Question 6

Write a function quadratic that returns the interval of all values f(t) such that t is in the argument interval x and f(t) is aquadratic function:

```
f(t) = a*t*t + b*t + c
```

Make sure that your implementation returns the smallest such interval, one that does not suffer from the multiple references problem.

Hint: the derivative f'(t) = 2*a*t + b, and so the extreme point of the quadratic is -b/(2*a):

```
def quadratic(x, a, b, c):
  """Return the interval that is the range of the quadratic defined by
  coefficients a, b, and c, for domain interval x.

  >>> str_interval(quadratic(interval(0, 2), -2, 3, -1))
  '-3 to 0.125'
  >>> str_interval(quadratic(interval(1, 3), 2, -3, 1))
  '0 to 10'
  """
  "*** YOUR CODE HERE ***"
```

Question 7: Challenge Problem (optional)

Write a function polynomial that takes an interval x and a list of coefficients c, and returns the interval containing all values off(t) for t in interval x, where:

```
f(t) = c[k-1] * pow(t, k-1) + c[k-2] * pow(t, k-2) + ... + c[0] * 1
```

Like quadratic, your polynomial function should return the smallest such interval, one that does not suffer from the multiple references problem.

Hint: You can approximate this result. Try using Newton's method.

```
def polynomial(x, c):
  """Return the interval that is the range of the polynomial defined by
  coefficients c, for domain interval x.

  >>> str_interval(polynomial(interval(0, 2), [-1, 3, -2]))
  '-3 to 0.125'
  >>> str_interval(polynomial(interval(1, 3), [1, -3, 2]))
  '0 to 10'
```

```
>>> str_interval(polynomial(interval(0.5, 2.25), [10, 24, -6, -8, 3]))
'18.0 to 23.0'
"""
"*** YOUR CODE HERE ***"
```