http://www-inst.eecs.berkeley.edu/~cs61a/sp15/hw/hw02/

CS 61A: Homework 2

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

Instructions

Download hw02.zip. Inside the archive, you will find a file called hw02.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 1.6

Table of Contents

Several doctests refer to these one-argument functions:

```
def square(x):
  return x * x

def triple(x):
  return 3 * x

def identity(x):
  return x

def increment(x):
  return x + 1
```

Question 1

Implement piecewise, which takes two one-argument functions, f and g, along with a number b. It returns a new function that takes a number x and returns either f(x) if x is less than b, or g(x) if x is greater than or equal to b.

```
def piecewise(f, g, b):
  """Returns the piecewise function h where:

  h(x) = f(x) if x < b,
      g(x) otherwise

  >>> def negate(x):
  ...     return -x
  >>> abs_value = piecewise(negate, identity, 0)
  >>> abs_value(6)
  6
  >>> abs_value(-1)
  1
  """
  "*** YOUR CODE HERE ***"
```

Question 2

If f is a numerical function and n is a positive integer, then we can form the nth repeated application of f, which is defined to be the function whose value at x is f(f(...(f(x))...)). For example, if f adds 1 to its argument, then the nth repeated application of f adds n. Write a function that takes as inputs a function f and a positive integer n and returns the function that computes the nth repeated application of f:

```
def repeated(f, n):
  """Return the function that computes the nth application of f.

  >>> add_three = repeated(increment, 3)
  >>> add_three(5)
  8
  >>> repeated(triple, 5)(1) # 3 * 3 * 3 * 3 * 3 * 1
  243
  >>> repeated(square, 2)(5) # square(square(5))
  625
  >>> repeated(square, 4)(5) # square(square(square(square(5))))
  152587890625
  """
  "*** YOUR CODE HERE ***"
```

Hint: You may find it convenient to use compose1 from the textbook:

```
def compose1(f, g):
  """Return a function h, such that h(x) = f(g(x))."""
  def h(x):
    return f(g(x))
  return h
```

Question 3: Challenge Problem (optional)

The logician Alonzo Church invented a system of representing non-negative integers entirely using functions. The purpose was to show that functions are sufficient to describe all of number theory: if we have functions, we do not need to assume that numbers exist, but instead we can invent them.

Your goal in this problem is to rediscover this representation known as Church numerals. Here are the definitions of `zero`, as well as a function that returns one more than its argument:

```
def zero(f):
  return lambda x: x

def successor(n):
  return lambda f: lambda x: f(n(f)(x))
```

First, define functions `one` and `two` such that they have the same behavior as `successor(zero)` and `successsor(successor(zero))` respectively, but do not call `successor` in your implementation.

Next, implement a function `church_to_int` that converts a church numeral argument to a regular Python integer.

Finally, implement functions `add_church`, `mul_church`, and `pow_church` that perform addition, multiplication, and exponentiation on church numerals.

```
def one(f):
  """Church numeral 1: same as successor(zero)"""
  "*** YOUR CODE HERE ***"

def two(f):
  """Church numeral 2: same as successor(successor(zero))"""
  "*** YOUR CODE HERE ***"

three = successor(two)

def church_to_int(n):
  """Convert the Church numeral n to a Python integer.

  >>> church_to_int(zero)
  0
  >>> church_to_int(one)
  1
  >>> church_to_int(two)
  2
```

```
    >>> church_to_int(three)
    3
    """
    "*** YOUR CODE HERE ***"

def add_church(m, n):
    """Return the Church numeral for m + n, for Church numerals m and n.

    >>> church_to_int(add_church(two, three))
    5
    """
    "*** YOUR CODE HERE ***"

def mul_church(m, n):
    """Return the Church numeral for m * n, for Church numerals m and n.

    >>> four = successor(three)
    >>> church_to_int(mul_church(two, three))
    6
    >>> church_to_int(mul_church(three, four))
    12
    """
    "*** YOUR CODE HERE ***"

def pow_church(m, n):
    """Return the Church numeral m ** n, for Church numerals m and n.

    >>> church_to_int(pow_church(two, three))
    8
    >>> church_to_int(pow_church(three, two))
    9
    """
    "*** YOUR CODE HERE ***"
```