Homework 2: Higher Order Functions

作业链接：Homework 2: Higher Order Functions

Important Functions

一些文档测试可能基于以下函数：

```
from operator import add, mul, sub

square = lambda x: x * x

identity = lambda x: x

triple = lambda x: 3 * x

increment = lambda x: x + 1
```

Q1: Product

仿照能够计算`term(1) + ... + term(n)`的`summation(n, term)`函数，编写一个能够计算`term(1) * ... * term(n)`的`product(n, term)`函数。

思路：注意`range`函数的区间是左闭右开的。

实现代码如下：

```
def product(n, term):
    """Return the product of the first n terms in a sequence.
    n -- a positive integer
    term -- a function that takes one argument to produce the term

    >>> product(3, identity)  # 1 * 2 * 3
    6
    >>> product(5, identity)  # 1 * 2 * 3 * 4 * 5
    120
    >>> product(3, square)    # 1^2 * 2^2 * 3^2
    36
    >>> product(5, square)    # 1^2 * 2^2 * 3^2 * 4^2 * 5^2
    14400
    >>> product(3, increment) # (1+1) * (2+1) * (3+1)
    24
    >>> product(3, triple)    # 1*3 * 2*3 * 3*3
    162
    """
    "*** YOUR CODE HERE ***"
    pdt = 1
    for i in range(1, n + 1):
        pdt *= term(i)

    return pdt
```

Q2: Accumulate

在`product`和`summation`函数的基础上，进行泛化，`combiner`参数指定运算符，`base`参数指定初始值。

实现代码如下：

```
def accumulate(combiner, base, n, term):
    """Return the result of combining the first n terms in a sequence and base.
    The terms to be combined are term(1), term(2), ..., term(n).  combiner is a
    two-argument commutative function.

    >>> accumulate(add, 0, 5, identity)  # 0 + 1 + 2 + 3 + 4 + 5
    15
    >>> accumulate(add, 11, 5, identity) # 11 + 1 + 2 + 3 + 4 + 5
    26
    >>> accumulate(add, 11, 0, identity) # 11
    11
    >>> accumulate(add, 11, 3, square)   # 11 + 1^2 + 2^2 + 3^2
    25
    >>> accumulate(mul, 2, 3, square)    # 2 * 1^2 * 2^2 * 3^2
    72
    >>> accumulate(lambda x, y: x + y + 1, 2, 3, square)
    19
    >>> accumulate(lambda x, y: 2 * (x + y), 2, 3, square)
    58
    >>> accumulate(lambda x, y: (x + y) % 17, 19, 20, square)
    16
    """
    "*** YOUR CODE HERE ***"
    for i in range(1, n + 1):
        base = combiner(base, term(i))

    return base
```

实现`accumulate`函数之后，用该函数实现`product`和`summation`函数：

```
def summation_using_accumulate(n, term):
    """Returns the sum of term(1) + ... + term(n). The implementation
    uses accumulate.

    >>> summation_using_accumulate(5, square)
    55
    >>> summation_using_accumulate(5, triple)
    45
    >>> from construct_check import check
    >>> # ban iteration and recursion
    >>> check(HW_SOURCE_FILE, 'summation_using_accumulate',
    ...       ['Recursion', 'For', 'While'])
    True
    """
    "*** YOUR CODE HERE ***"
    return accumulate(add, 0, n, term)
```

```
def product_using_accumulate(n, term):
    """An implementation of product using accumulate.

    >>> product_using_accumulate(4, square)
    576
    >>> product_using_accumulate(6, triple)
    524880
    >>> from construct_check import check
    >>> # ban iteration and recursion
    >>> check(HW_SOURCE_FILE, 'product_using_accumulate',
    ...       ['Recursion', 'For', 'While'])
    True
    """
    "*** YOUR CODE HERE ***"
    return accumulate(mul, 1, n, term)
```

Q3: Make Repeater

实现`make_repeater(func, n)(x)`函数，返回`func(func(...func(x)...))`，其中有n个`func`.

思路：使用上述实现的`accumulate`和`compose1`函数可以很容易得到目标函数，也可以循环嵌套得到。

实现代码如下：

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

    >>> add_three = make_repeater(increment, 3)
    >>> add_three(5)
    8
    >>> make_repeater(triple, 5)(1) # 3 * 3 * 3 * 3 * 3 * 1
    243
    >>> make_repeater(square, 2)(5) # square(square(5))
    625
    >>> make_repeater(square, 4)(5) # square(square(square(square(5))))
    152587890625
    >>> make_repeater(square, 0)(5) # Yes, it makes sense to apply the function
zero times!
    5
    """
    "*** YOUR CODE HERE ***"

    # return accumulate(compose1, lambda x : x, n, lambda x : func)

    base = lambda x : x
    for i in range(1, n + 1):
        base = compose1(base, func)

    return base
```

使用如下命令进行测评：

```
python3 ok --local
```

结果如下：

```
=====================================================================
Assignment: Homework 2
OK, version v1.18.1
=====================================================================

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    5 test cases passed! No cases failed.

Cannot backup when running ok with --local.
```

Q4: Church numerals

使用高阶函数（`Higher Order Function`）构造一个简单的丘奇代数系统。

实现代码如下：

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


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


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

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

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 ***"
    f = lambda x: x + 1
    return n(f)(0)


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 func(f):
        return lambda x: m(f)(n(f)(x))
    return func


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 ***"
    base = zero
    for i in range(0, church_to_int(n)):
        base = add_church(base, m)
    return base


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 ***"
    base = one
    for i in range(0, church_to_int(n)):
        base = mul_church(base, m)
    return base
```

使用如下命令进行测评：

```
python3 ok -q church_to_int
python3 ok -q add_church
python3 ok -q mul_church
python3 ok -q pow_church
```

结果如下（4个结果都是一样的）：

```
rex@rex-virtual-machine:~/CS61A/HW/HW02/hw02$ python3 ok -q mul_church --local
=====================================================================
Assignment: Homework 2
OK, version v1.18.1
=====================================================================

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    1 test cases passed! No cases failed.

Cannot backup when running ok with --local.
```