Homework 2

● Graded

Student
Sangwon Ji

Total Points
2 / 2 pts

Autograder Score
2.0 / 2.0

Autograder Results

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

False
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Scoring tests

---------------------------------------------------------------------
Doctests for product

>>> from hw02 import *
>>> 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
Score: 1.0/1

---------------------------------------------------------------------
Doctests for accumulate

>>> from hw02 import *
>>> 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
>>> # 2 + (1^2 + 1) + (2^2 + 1) + (3^2 + 1)
>>> accumulate(lambda x, y: x + y + 1, 2, 3, square)
19
>>> # ((2 * 1^2 * 2) * 2^2 * 2) * 3^2 * 2
>>> accumulate(lambda x, y: 2 * x * y, 2, 3, square)
576
>>> accumulate(lambda x, y: (x + y) % 17, 19, 20, square)
16
Score: 1.0/1

---------------------------------------------------------------------
Doctests for funception

>>> from hw02 import *
>>> def func1(num):
... return num + 1
>>> func2_1 = funception(func1, 0)
>>> func2_1(3) # func1(0) * func1(1) * func1(2) = 1 * 2 * 3 = 6
6
>>> func2_2 = funception(func1, 1)
>>> func2_2(4) # func1(1) * func1(2) * func1(3) = 2 * 3 * 4 = 24
24
>>> func2_3 = funception(func1, 3)
>>> func2_3(2) # Returns func1(3) since start >= stop
4
>>> func2_4 = funception(func1, 3)
>>> func2_4(3) # Returns func1(3) since start >= stop
4
>>> func2_5 = funception(func1, -2)
>>> func2_5(-3) # Returns None since start < 0
>>> func2_6 = funception(func1, -1)
>>> func2_6(4) # Returns None since start < 0
Score: 1.0/1

---------------------------------------------------------------------
Doctests for num_eights

>>> from hw02 import *
>>> num_eights(3)
0
>>> num_eights(8)
1
>>> num_eights(88888888)
8
>>> num_eights(2638)
1
>>> num_eights(86380)
2
>>> num_eights(12345)
0
```

```
>>> num_eights(8782089)
3
>>> from construct_check import check
>>> # ban all assignment statements
>>> check(HW_SOURCE_FILE, 'num_eights',
... ['Assign', 'AnnAssign', 'AugAssign', 'NamedExpr', 'For', 'While'])
True
Score: 1.0/1

---------------------------------------------------------------------
Doctests for waves

>>> from hw02 import *
>>> waves(1)
True
>>> waves(10001)
False
>>> waves(12233121)
False
>>> waves(1313)
True
>>> waves(12332023213)
True
>>> from construct_check import check
>>> # ban all loops
>>> check(HW_SOURCE_FILE, 'waves',
... ['For', 'While'])
True
Score: 1.0/1

---------------------------------------------------------------------
Doctests for count_coins

>>> from hw02 import *
>>> count_coins(15)
6
>>> count_coins(10)
4
>>> count_coins(20)
9
>>> count_coins(100) # How many ways to make change for a dollar?
242
>>> count_coins(200)
1463
>>> from construct_check import check
>>> # ban iteration
>>> check(HW_SOURCE_FILE, 'count_coins', ['While', 'For'])
True
Score: 1.0/1

---------------------------------------------------------------------
Point breakdown
product: 1.0/1
```

accumulate: 1.0/1
funception: 1.0/1
num_eights: 1.0/1
waves: 1.0/1
count_coins: 1.0/1

Score:
Total: 6.0

Cannot backup when running ok with --local.

Final Score:2.0

Submitted Files

hw02.py Download

```
from operator import add, mul

square = lambda x: x * x

identity = lambda x: x

triple = lambda x: 3 * x

increment = lambda x: x + 1


HW_SOURCE_FILE = __file__


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
    """
    x, total = 1, 1
    while x<= n: # Until x is bigger than n
        x, total = x + 1, term(x) * total
    return total



def accumulate(merger, start, n, term):
    """Return the result of merging the first n terms in a sequence and start.
    The terms to be merged are term(1), term(2), ..., term(n). merger 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
    >>> # 2 + (1^2 + 1) + (2^2 + 1) + (3^2 + 1)
    >>> accumulate(lambda x, y: x + y + 1, 2, 3, square)
    19
    >>> # ((2 * 1^2 * 2) * 2^2 * 2) * 3^2 * 2
    >>> accumulate(lambda x, y: 2 * x * y, 2, 3, square)
    576
    >>> accumulate(lambda x, y: (x + y) % 17, 19, 20, square)
    16
    """
    x, total = 1, start
    while x<=n :
        x, total = x+1, merger(term(x), total)
    return total


def summation_using_accumulate(n, term):
    """Returns the sum: term(1) + ... + term(n), using accumulate.

    >>> summation_using_accumulate(5, square)
    55
    >>> summation_using_accumulate(5, triple)
    45
    >>> # You aren't expected to understand the code of this test.
    >>> # Check that the bodies of the functions are just return statements.
    >>> # If this errors, make sure you have removed the "***YOUR CODE HERE***".
    >>> import inspect, ast
    >>> [type(x).__name__ for x in
ast.parse(inspect.getsource(summation_using_accumulate)).body[0].body]
    ['Expr', 'Return']
    """
    return accumulate(add, 0, n, term)


def product_using_accumulate(n, term):
    """Returns the product: term(1) * ... * term(n), using accumulate.

    >>> product_using_accumulate(4, square)
    576
    >>> product_using_accumulate(6, triple)
    524880
    >>> # You aren't expected to understand the code of this test.
    >>> # Check that the bodies of the functions are just return statements.
    >>> # If this errors, make sure you have removed the "***YOUR CODE HERE***".
    >>> import inspect, ast
    >>> [type(x).__name__ for x in
ast.parse(inspect.getsource(product_using_accumulate)).body[0].body]
```

```
    ['Expr', 'Return']
    """
    return accumulate(mul, 1, n, term)

def funception(func1, start):
    """ Takes in a function (function 1) and a start value.
    Returns a function (function 2) that will find the product of
    function 1 applied to the range of numbers from
    start (inclusive) to stop (exclusive)

    >>> def func1(num):
    ...     return num + 1
    >>> func2_1 = funception(func1, 0)
    >>> func2_1(3)    # func1(0) * func1(1) * func1(2) = 1 * 2 * 3 = 6
    6
    >>> func2_2 = funception(func1, 1)
    >>> func2_2(4)    # func1(1) * func1(2) * func1(3) = 2 * 3 * 4 = 24
    24
    >>> func2_3 = funception(func1, 3)
    >>> func2_3(2)    # Returns func1(3) since start >= stop
    4
    >>> func2_4 = funception(func1, 3)
    >>> func2_4(3)    # Returns func1(3) since start >= stop
    4
    >>> func2_5 = funception(func1, -2)
    >>> func2_5(-3)    # Returns None since start < 0
    >>> func2_6 = funception(func1, -1)
    >>> func2_6(4)    # Returns None since start < 0
    """
    def func2(stop):
        if start <0:
            return None
        elif start >= stop:
            return func1(start)
        else:
            total = 1
            for i in range(start, stop):
                total = total * func1(i)
            return total
    return func2



def num_eights(n):
    """Returns the number of times 8 appears as a digit of n.

    >>> num_eights(3)
    0
    >>> num_eights(8)
    1
    >>> num_eights(88888888)
    8
```

```
    >>> num_eights(2638)
    1
    >>> num_eights(86380)
    2
    >>> num_eights(12345)
    0
    >>> num_eights(8782089)
    3
    >>> from construct_check import check
    >>> # ban all assignment statements
    >>> check(HW_SOURCE_FILE, 'num_eights',
    ...       ['Assign', 'AnnAssign', 'AugAssign', 'NamedExpr', 'For', 'While'])
    True
    """
    if n < 8:
        return 0
    else:
        if n% 10 == 8:
            return 1 + num_eights(n//10)
        else:
            return num_eights(n//10)


def waves(n):
    """Return whether n is balanced.

    >>> waves(1)
    True
    >>> waves(10001)
    False
    >>> waves(12233121)
    False
    >>> waves(1313)
    True
    >>> waves(12332023213)
    True
    >>> from construct_check import check
    >>> # ban all loops
    >>> check(HW_SOURCE_FILE, 'waves',
    ...       ['For', 'While'])
    True
    """
    #def crest(prev, curr, next):
        #return prev < curr and next < curr
   # def trough (prev, curr, next):
        #return curr < prev and curr < next
    #def plateau(prev, curr, next):
        #return prev == curr == next
    def helper(n, count, prev):
        curr, next, rest = n % 10, (n // 10) % 10, n // 10
        if n == 0:
                return True
```

```
        if prev > curr:
            if curr == next:
                # decrease stay 2
                return False
            else:
                # Decrease stay Increase or Decrease Decrease Decrease
                return True and helper(n//10 ,count +1 , n%10)
        elif prev <curr:
            # Increase stay stay or Increase Increase Increase, or Increease stay decrease
                return True and helper(n//10, count +1, n%10)
        else:
            if curr < next:
                # stay 2 increase
                return False
            else:
                # stay stay decrease or stay x3
                return True and helper(n // 10, count, n % 10)
    return helper(n // 10, 0, n % 10)


def next_larger_coin(coin):
    """Returns the next larger coin in order.
    >>> next_larger_coin(1)
    5
    >>> next_larger_coin(5)
    10
    >>> next_larger_coin(10)
    25
    >>> next_larger_coin(2) # Other values return None
    """
    if coin == 1:
        return 5
    elif coin == 5:
        return 10
    elif coin == 10:
        return 25


def next_smaller_coin(coin):
    """Returns the next smaller coin in order.
    >>> next_smaller_coin(25)
    10
    >>> next_smaller_coin(10)
    5
    >>> next_smaller_coin(5)
    1
    >>> next_smaller_coin(2) # Other values return None
    """
    if coin == 25:
        return 10
    elif coin == 10:
        return 5
```

```
    elif coin == 5:
        return 1


def count_coins(total):
    """Return the number of ways to make change using coins of value of 1, 5, 10, 25.
    >>> count_coins(15)
    6
    >>> count_coins(10)
    4
    >>> count_coins(20)
    9
    >>> count_coins(100) # How many ways to make change for a dollar?
    242
    >>> count_coins(200)
    1463
    >>> from construct_check import check
    >>> # ban iteration
    >>> check(HW_SOURCE_FILE, 'count_coins', ['While', 'For'])
    True
    """
    def combinations(amount, coin):
        if amount == 0:
            return 1
        elif coin == None:
            return 0
        elif amount <0:
            return 0
        else:
            current = combinations( amount - coin, coin)
            smaller = combinations (amount, next_smaller_coin(coin))
            return current + smaller
    return combinations(total, 25)

print(waves(1223121))
```