Homework 1

● Graded

Student

Sangwon Ji

Total Points

2 / 2 pts

Autograder Score

2.0 / 2.0

Autograder Results

```
=====================================================================
Assignment: Homework 1
OK, version v1.18.1
=====================================================================

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Scoring tests

---------------------------------------------------------------------
Doctests for a_plus_abs_b

>>> from hw01 import *
>>> a_plus_abs_b(2, 3)
5
>>> a_plus_abs_b(2, -3)
5
>>> a_plus_abs_b(-1, 4)
3
>>> a_plus_abs_b(-1, -4)
3
Score: 1.0/1

---------------------------------------------------------------------
Doctests for a_plus_abs_b_syntax_check

>>> from hw01 import *
>>> # You aren't expected to understand the code of this test.
>>> import inspect, re
>>> re.findall(r'^\s*(return .*)', inspect.getsource(a_plus_abs_b), re.M)
['return f(a, b)']
Score: 1.0/1

---------------------------------------------------------------------
Doctests for two_of_three

>>> from hw01 import *
>>> two_of_three(1, 2, 3)
5
>>> two_of_three(5, 3, 1)
```

```
10
>>> two_of_three(10, 2, 8)
68
>>> two_of_three(5, 5, 5)
50
Score: 1.0/1

---------------------------------------------------------------------
Doctests for two_of_three_syntax_check

>>> from hw01 import *
>>> # You aren't expected to understand the code of this test.
>>> import inspect, ast
>>> [type(x).__name__ for x in ast.parse(inspect.getsource(two_of_three)).body[0].body]
['Expr', 'Return']
Score: 1.0/1

---------------------------------------------------------------------
Doctests for largest_factor

>>> from hw01 import *
>>> largest_factor(15) # factors are 1, 3, 5
5
>>> largest_factor(80) # factors are 1, 2, 4, 5, 8, 10, 16, 20, 40
40
>>> largest_factor(13) # factor is 1 since 13 is prime
1
Score: 1.0/1

---------------------------------------------------------------------
Doctests for hailstone

>>> from hw01 import *
>>> a = hailstone(10)
10
5
16
8
4
2
1
>>> a
7
>>> b = hailstone(1)
1
>>> b
1
Score: 1.0/1

---------------------------------------------------------------------
Point breakdown
a_plus_abs_b: 1.0/1
a_plus_abs_b_syntax_check: 1.0/1
two_of_three: 1.0/1
```

```
two_of_three_syntax_check: 1.0/1
largest_factor: 1.0/1
hailstone: 1.0/1

Score:
Total: 6.0

Cannot backup when running ok with --local.
--------------------------------------------------------------------
Final Score:2.0
```

Submitted Files

hw01.py

Download

```python
from operator import add, sub


def a_plus_abs_b(a, b):
    """Return a+abs(b), but without calling abs.

    >>> a_plus_abs_b(2, 3)
    5
    >>> a_plus_abs_b(2, -3)
    5
    >>> a_plus_abs_b(-1, 4)
    3
    >>> a_plus_abs_b(-1, -4)
    3
    """
    if b < 0:
        f = sub
    else:
        f = add
    return f(a, b)


def a_plus_abs_b_syntax_check():
    """Check that you didn't change the return statement of a_plus_abs_b.

    >>> # You aren't expected to understand the code of this test.
    >>> import inspect, re
    >>> re.findall(r'^\s*(return .*)', inspect.getsource(a_plus_abs_b), re.M)
    ['return f(a, b)']
    """
    # You don't need to edit this function. It's just here to check your work.


def two_of_three(i, j, k):
    """Return m*m + n*n, where m and n are the two smallest members of the
    positive numbers i, j, and k.

    >>> two_of_three(1, 2, 3)
    5
    >>> two_of_three(5, 3, 1)
    10
    >>> two_of_three(10, 2, 8)
    68
    >>> two_of_three(5, 5, 5)
    50
    """
    return i*i + j*j + k*k - max(i,j,k) * max(i,j,k)


```

```python
def two_of_three_syntax_check():
    """Check that your two_of_three code consists of nothing but a return statement.

    >>> # You aren't expected to understand the code of this test.
    >>> import inspect, ast
    >>> [type(x).__name__ for x in ast.parse(inspect.getsource(two_of_three)).body[0].body]
    ['Expr', 'Return']
    """
    # You don't need to edit this function. It's just here to check your work.


def largest_factor(n):
    """Return the largest factor of n that is smaller than n.

    >>> largest_factor(15) # factors are 1, 3, 5
    5
    >>> largest_factor(80) # factors are 1, 2, 4, 5, 8, 10, 16, 20, 40
    40
    >>> largest_factor(13) # factor is 1 since 13 is prime
    1
    """
    i = n
    while n > 1:
        i = i -1
        if n % i == 0:
            return i


def hailstone(n):
    """Print the hailstone sequence starting at n and return its
    length.

    >>> a = hailstone(10)
    10
    5
    16
    8
    4
    2
    1
    >>> a
    7
    >>> b = hailstone(1)
    1
    >>> b
    1
    """
    length1 = 0

    while n !=1:
        print(n)
        if n % 2 ==0: # If n is even
```

```python
            n = n//2
        else: # If n is odd
            n = 3 * n + 1
        length1 += 1

    print(n)
    length1 += 1

    return(length1)

```