Instructions

In this homework, you are required to complete the problems described in section 2. The starter code for these problems is provided in hw01.py, which is distributed as part of the homework materials in the code directory.

Readings: You might find the following references to the textbook useful:

- Section 1.2
- Section 1.3
- Section 1.4
- Section 1.5

Required Problems

In this section, you are required to complete the problems below and submit your code with Ok to get your answer scored.

Remember, you can use `ok` to test your code:

```
$ python3 ok
```

Problem 1: A Sub Abs B (100pts)

Fill in the blanks in the following function for subtracting a with the absolute value of b, without calling `abs`. You may **not** modify any of the provided code other than the two blanks.

```
from operator import add, sub, mul, neg

def a_sub_abs_b(a, b):
    """Return a-abs(b), but do not call abs.

    >>> a_sub_abs_b(2, 3)
    -1
    >>> a_sub_abs_b(2, -3)
    -1
    >>> # a check that you didn't change the return statement!
    >>> import inspect, re
    >>> re.findall(r'^\s*(return .*)',
inspect.getsource(a_sub_abs_b), re.M)
    ['return h(a, b)']
    """
    if b >= 0:
        h = _____
    else:
        h = _____
    return h(a, b)
```

Hint: You may want to use `add`, `sub`, `mul` and/or `neg` that imported from `opeartor`.

Here is the documentation of these operators.

Test your implementation with `python3 ok -q a_sub_abs_b`.

在大多数作业中，大家可以用`python3 ok -q 函数名`的方式来测试对应的函数，后面就不再赘述了。

Problem 2: Two of Three (100pts)

Write a function that takes three positive numbers and returns the sum of the squares of the two largest numbers. **Use only a single line for the body of the function.**

```
def two_of_three(x, y, z):
    """Return a*a + b*b, where a and b are the two largest
members of the
    positive numbers x, y, and z.

    >>> two_of_three(1, 2, 3)
    13
    >>> two_of_three(5, 3, 1)
    24
    >>> two_of_three(10, 2, 8)
    164
    >>> two_of_three(5, 5, 5)
    50
    >>> # check that your code consists of nothing but an
expression (this docstring)
    >>> # and a return statement
    >>> import inspect, ast
    >>> [type(x).__name__ for x in
ast.parse(inspect.getsource(two_of_three)).body[0].body]
    ['Expr', 'Return']
    """
    return _____
```

Hint: Consider using the max or min function:

```
>>> max(1, 2, 3)
3
>>> min(-1, -2, -3)
-3
```

Problem 3: Largest Factor (100pts)

Write a function that takes an integer x that is **greater than** 1 and returns the largest integer that is smaller than x and evenly divides x.

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

    >>> 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
    """
    "*** YOUR CODE HERE ***"
```

Hint1: To check if b evenly divides a, you can use the expression a % b == 0, which can be read as, "the remainder of dividing a by b is 0."

Hint2: You may want to review what is iteration?

Problem 4: If Function vs Statement (100pts)

```
def if_function(condition, true_result, false_result):
    """Return true_result if condition is a true value, and
    false_result otherwise.

    >>> if_function(True, 2, 3)
    2
    >>> if_function(False, 2, 3)
    3
    >>> if_function(3==2, 3+2, 3-2)
    1
    >>> if_function(3>2, 3+2, 3-2)
    5
    """
    if condition:
        return true_result
    else:
        return false_result
```

Despite the doctests above, this function actually does not do the same thing as an `if` statement in all cases.

To prove this fact, write functions `c`, `t`, and `f` such that `with_if_statement` prints the number 2, but `with_if_function` prints both 1 and 2.

```
def with_if_statement():
    """
    >>> result = with_if_statement()
    2
    >>> print(result)
    None
    """
    if c():
        return t()
    else:
        return f()

def with_if_function():
    """
    >>> result = with_if_function()
    1
    2
    >>> print(result)
    None
    """
    return if_function(c(), t(), f())

def c():
    "*** YOUR CODE HERE ***"

def t():
    "*** YOUR CODE HERE ***"

def f():
    "*** YOUR CODE HERE ***"
```

Hint: If you are having a hard time identifying how an if statement and if_function differ, consider the rules of evaluation for if statements and call expressions.

For this problem, you can test your implementation with:

- `python3 ok -q with_if_statement`
- `python3 ok -q with_if_function`

Problem 5: Hailstone (100pts)

Douglas Hofstadter's Pulitzer-prize-winning book, *Gödel, Escher, Bach*, poses the following mathematical puzzle.

1. Pick a positive integer x as the start.
2. If x is even, divide it by 2.
3. If x is odd, multiply it by 3 and add 1.
4. Continue this process until x is 1.

The number x will travel up and down but eventually end at 1 (at least for all numbers that have ever been tried -- nobody has ever proved that the sequence will terminate). Analogously, a hailstone travels up and down in the atmosphere before eventually landing on earth.

Breaking News (or at least the closest thing to that in math). There has been a recent development in the hailstone conjecture that shows that almost all numbers will eventually get to 1 if you repeat this process. This isn't a complete proof but a major breakthrough.

This sequence of values of x is often called a Hailstone sequence. Write a function that takes a single argument with formal parameter name x, prints out the hailstone sequence starting at x, and returns the number of steps in the sequence:

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

    >>> a = hailstone(10)
    10
    5
    16
    8
    4
    2
    1
    >>> a
    7
    """
    "*** YOUR CODE HERE ***"
```

Hailstone sequences can get quite long! Try 27. What's the longest you can find?

Problem 6: Falling Factorial (100pts)

Let's write a function `falling`, which is a "falling" factorial that takes two arguments, n and k, and returns the product of k consecutive numbers, starting from n and working downwards. (This problem is an advanced version of Problem 2, Lab 01.)

```
def falling(n, k):
    """Compute the falling factorial of n to depth k.

    >>> falling(6, 3)  # 6 * 5 * 4
    120
    >>> falling(4, 3)  # 4 * 3 * 2
    24
    >>> falling(4, 1)  # 4
    4
    >>> falling(4, 0)
    1
    """
    "*** YOUR CODE HERE ***"
```

Problem 7: Double Ones (100pts)

Write a function that takes in a number and determines if the digits contain two adjacent 1s. (Reviewing Problem 4 and 5 in Lab 01 might be helpful here!)

```
def double_ones(n):
    """Return true if n has two ones in a row.

    >>> double_ones(1)
    False
    >>> double_ones(11)
    True
    >>> double_ones(2112)
    True
    >>> double_ones(110011)
    True
    >>> double_ones(12345)
    False
    >>> double_ones(10101010)
    False
    """
    "*** YOUR CODE HERE ***"
```

Submit

Submission: When you are done, submit your code as instructed in lab00. You may submit more than once before the deadline; the highest score will be recorded.

To submit your work when you are done:

```
$ python3 ok --submit
```