Homework 01: Variables & Functions, Control

> Adapted from cs61a of UC Berkeley.

Instructions

Readings: You might find the following references useful:

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

Note: Some of these readings necessary for the homework questions **will not be covered until lecture3 on Control**.

Starter Files

Get your starter file by cloning the repository: https://github.com/JacyCui/sicp-hw01.git

```
git clone https://github.com/JacyCui/sicp-hw01.git
```

`hw01.zip` is the starter file you need, you might need to unzip the file to get the skeleton code.

```
unzip hw01.zip
```

`README.md` is the handout for this homework. `solution` is a probrab solution of the homework. However, I might not give my solution exactly when the homework is posted. You need to finish the task on your own first. If any problems occurs, please make use of the comment section.

Required Questions

Q1: A Plus Abs B

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

```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 check that you didn't change the return statement!
    >>> import inspect, re
    >>> re.findall(r'^\s*(return .*)', inspect.getsource(a_plus_abs_b), re.M)
    ['return f(a, b)']
    """
    if b < 0:
        f = _____
    else:
        f = _____
    return f(a, b)
```

Use Ok to test your code:

```
python3 ok -q a_plus_abs_b --local
```

Q2: Two of Three

Write a function that takes three *positive* numbers as arguments and returns the sum of the squares of the two smallest 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 smallest members of the
    positive numbers x, y, and z.

    >>> 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
    >>> # check that your code consists of nothing but an expression (this
docstring)
    >>> # 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
> ```

Use Ok to test your code:

```
python3 ok -q two_of_three --local
```

Q3: Largest Factor

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

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

> **Hint:** 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."

Use Ok to test your code:

```
python3 ok -q largest_factor --local
```

Q4: If Function vs Statement

Let's try to write a function that does the same thing as an `if` statement.

```
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 `cond`, `true_func`, and `false_func` such that `with_if_statement` prints the number `47`, but `with_if_function` prints both `42` and `47`.

```
def with_if_statement():
    """
    >>> result = with_if_statement()
    47
    >>> print(result)
    None
    """
    if cond():
        return true_func()
    else:
        return false_func()

def with_if_function():
    """
    >>> result = with_if_function()
    42
    47
    >>> print(result)
    None
    """
    return if_function(cond(), true_func(), false_func())

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

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

def false_func():
    "*** 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.

Use Ok to test your code:

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

Q5: Hailstone

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

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

The number `n` 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.

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

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

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

Use Ok to test your code:

```
python3 ok -q hailstone --local
```

Doctest

Run the doctest of `hw01.py` like:

```
python3 -m doctest hw01.py
```

If nothing happened, you should pass all the doctests.