Homework 1 Solutions **hw01.zip (hw01.zip)**

Solution Files

You can find the solutions in hw01.py (hw01.py).

Homework Questions

Q0: Welcome Survey

Please complete this welcome survey (https://goo.gl/forms/feBnXxIESOZi3QIk1) before you submit your homework. Your responses will not be visible to anyone outside the course.

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.

```
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
    """
    if b < 0:
        f = sub
    else:
        f = add
    return f(a, b)
```

Use Ok to test your code:

```
python3 ok -q a_plus_abs_b
```

If `b` is positive, we add the numbers together. If `b` is negative, we subtract the numbers. Therefore, we choose the operator `add` or `sub` based on the sign of `b`.

Video walkthrough: https://youtu.be/o9eUNrWTr3I (https://youtu.be/o9eUNrWTr3I)

Q2: Two of Three

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

    >>> two_of_three(1, 2, 3)
    13
    >>> two_of_three(5, 3, 1)
    34
    >>> two_of_three(10, 2, 8)
    164
    >>> two_of_three(5, 5, 5)
    50
    """
    return max(a*a+b*b, a*a+c*c, b*b+c*c)
    # Alternate solution
    return a**2 + b**2 + c**2 - min(a, b, c)**2
```

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
```

We use the fact that if `a>b` and `b>0`, then `square(a)>square(b)`. So, we can take the `max` of the sum of squares of all pairs. The `max` function can take an arbitrary number of arguments.

Alternatively, we can do the sum of squares of all the numbers. Then we pick the smallest value, and subtract the square of that.

Video walkthrough: https://youtu.be/oPN3OCGGb4M (https://youtu.be/oPN3OCGGb4M)

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
    """
    factor = n - 1
    while factor > 0:
        if n % factor == 0:
            return factor
        factor -= 1
```

> **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
```

Iterating from `n-1` to 1, we return the first integer that evenly divides `n`. This is guaranteed to be the largest factor of `n`.

Video walkthrough: https://youtu.be/pVgxbeL4DHQ (https://youtu.be/pVgxbeL4DHQ)

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 `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():
    return False

def t():
    print(1)

def f():
    print(2)
```

> **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
python3 ok -q with_if_function
```

The function `with_if_function` uses a call expression, which guarantees that all of its operand subexpressions will be evaluated before `if_function` is applied to the resulting arguments.

Therefore, even if `c` returns `False`, the function `t` will be called. When we call `t`, we print out `1`. Then, when we call `f`, we will also print `2`.

By contrast, `with_if_statement` will never call `t` if `c` returns `False`. Thus, we will only call `f`, printing `2`.

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
    """
    length = 1
    while n != 1:
        print(n)
        if n % 2 == 0:
            n = n // 2      # Integer division prevents "1.0" output
        else:
            n = 3 * n + 1
        length = length + 1
    print(n)                # n is now 1
    return length
```

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
```

We keep track of the current length of the hailstone sequence and the current value of the hailstone sequence. From there, we loop until we hit the end of the sequence, updating the length in each step.

> Note: we need to do floor division `//` to remove decimals.

Video walkthrough: https://youtu.be/lZZQ0BpsXIc (https://youtu.be/lZZQ0BpsXIc)

CS 61A (/)

Weekly Schedule (/weekly.html)

Office Hours (/office-hours.html)

Staff (/staff.html)

Resources (/resources.html)

Studying Guide (/articles/studying.html)

Debugging Guide (/articles/debugging.html)

Composition Guide (/articles/composition.html)

Policies (/articles/about.html)

Assignments (/articles/about.html#assignments)

Exams (/articles/about.html#exams)

Grading (/articles/about.html#grading)