Homework 1: Functions, Control

hw01.zip (hw01.zip)

Due by 11:59pm on Monday, September 9

Instructions

Download hw01.zip (hw01.zip).

Submission: When you are done, submit the assignment by uploading all code files you've edited to Gradescope. You may submit more than once before the deadline; only the final submission will be scored. Check that you have successfully submitted your code on Gradescope. See Lab 0 (../../lab/lab00#task-c-submitting-the-assignment) for more instructions on submitting assignments.

Using Ok: If you have any questions about using Ok, please refer to this guide. (../../articles/using-ok)

Readings: You might find the following references useful:

- Section 1.1 (https://www.composingprograms.com/pages/11-getting-started.html)
- Section 1.2 (https://www.composingprograms.com/pages/12-elements-of-programming.html)
- Section 1.3 (https://www.composingprograms.com/pages/13-defining-new-functions.html)
- Section 1.4 (https://www.composingprograms.com/pages/14-designing-functions.html)
- Section 1.5 (https://www.composingprograms.com/pages/15-control.html)

Grading: Homework is graded based on correctness. Each incorrect problem will decrease the total score by one point. **This homework is out of 2 points.**

Getting Started Videos

Required Questions

Q1: A Plus Abs B

Python's `operator` module contains two-argument functions such as `add` and `sub` for Python's built-in arithmetic operators. For example, `add(2, 3)` evalutes to 5, just like the expression `2 + 3`.

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.

```
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 = _____
    else:
        f = _____
    return f(a, b)
```

Use Ok to test your code:

```
python3 ok -q a_plus_abs_b
```

Use Ok to run the local syntax checker (which checks that you didn't modify any of the provided code other than the two blanks):

```
python3 ok -q a_plus_abs_b_syntax_check
```

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.**

```python
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 _____
```

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
```

Use Ok to run the local syntax checker (which checks that you used only a single line for the body of the function):

```
python3 ok -q two_of_three_syntax_check
```

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`, use the expression `a % b == 0`, which can be read as, "the remainder when dividing `a` by `b` is 0."

Use Ok to test your code:

```
python3 ok -q largest_factor
```

Q4: 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
    >>> b = hailstone(1)
    1
    >>> b
    1
    """
    "*** YOUR CODE HERE ***"
```

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

> Note that if `n == 1` initially, then the sequence is one step long.
> **Hint:** If you see 4.0 but want just 4, try using floor division `//` instead of regular division `/`.

Use Ok to test your code:

```
python3 ok -q hailstone
```

Curious about hailstone sequences? Take a look at this article:

- In 2019, there was a major development (https://www.quantamagazine.org/mathematician-terence-tao-and-the-collatz-conjecture-20191211/) in understanding how the hailstone conjecture works for most numbers!

Check Your Score Locally

You can locally check your score on each question of this assignment by running

```
python3 ok --score
```

This does NOT submit the assignment! When you are satisfied with your score, submit the assignment to Gradescope to receive credit for it.

Submit Assignment

Submit this assignment by uploading any files you've edited **to the appropriate Gradescope assignment.** Lab 00 (../../lab/lab00/#submit-with-gradescope) has detailed instructions.