CS 61A: Solutions for Homework 1

Solutions: You can find the file with solutions for all questions here.

Table of Contents

Question 1

We've seen that we can give new names to existing functions. Fill in the blanks in the following function definition for adding `a` to the absolute value of `b`, without calling `abs`.

```
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)
```

We choose the operator `add` or `sub` based on the sign of `b`.

Question 2

Write a function that takes three *positive* numbers and returns the sum of the squares of the two largest numbers. Use only a single expression 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
    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)
```

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.

Question 3

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` returns the number `1`, but `with_if_function` does not (it can do *anything* else):

```
def with_if_statement():
    """
    >>> with_if_statement()
    1
    """
    if c():
        return t()
    else:
        return f()

def with_if_function():
    return if_function(c(), t(), f())

def c():
    return False

def t():
    1/0

def f():
    return 1
```

Note: No tests will be run on your solution to this problem.

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.

By contrast, `with_if_statement` will never call `t` if `c` returns `False`.

Question 4

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.

The sequence of values of `n` is often called a Hailstone sequence, because hailstones also travel up and down in the atmosphere before falling to earth. 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" outpu
        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?

Question 5: Challenge Problem (optional)

Write a one-line program that prints itself, using only the following features of the Python language:

- Number literals
- Assignment statements
- String literals that can be expressed using single or double quotes
- The arithmetic operators `+`, `-`, `*`, and `/`
- The built-in `print` function
- The built-in `eval` function, which evaluates a string as a Python expression
- The built-in `repr` function, which returns an expression that evaluates to its argument

You can concatenate two strings by adding them together with `+` and repeat a string by multipying it by an integer. Semicolons can be used to separate multiple statements on the same line. E.g.,

```
>>> c='c';print('a');print('b' + c * 2)
a
bcc
```

Hint: Explore the relationship between single quotes, double quotes, and the `repr` function applied to strings.

Place your solution in the multi-line string named `challenge_question_program` in `hw01.py`.

Note: No tests will be run on your solution to this problem.