Homework 1: Variables & Functions, Control

作业链接：Homework 1: Variables & Functions, Control

Q1: Syllabus Quiz

教学大纲和课程政策，详见Syllabus Quiz和Syllabus & Course Policies。

Q2: 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.

题意就是实现如下函数 `f`：

$$f(a,b) = a + |b|$$

实现代码如下：

```
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 = lambda x, y : x - y
    else:
        f = lambda x, y : x + y
    return f(a, b)
```

另一种实现（使用 `lambda` 表达式）：

```
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 = sub
    else:
        f = add
    return f(a, b)
```

Q3: 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.**
>
> **Hint:** Consider using the `max` or `min` function:
>
> ```
> >>> max(1, 2, 3)
> 3
> >>> min(-1, -2, -3)
> -3
> ```

思路：最小的数可以用 `min()` 函数得到，次小的数可以用 `x` 、 `y` 、 `z` 三个数的和减掉最小值和最大值（ `max()` ）。
实现代码如下：

```
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 min(x, y, z) ** 2 + (x + y + z - min(x, y, z) - max(x, y, z)) ** 2
```

Q4: 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` .
> **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."

思路：找一个数 `n` 的最大因子，用循环把 `1~n-1` 都检查一遍就行了。
实现代码：

```
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 ***"
    ans = 1
    for i in range(1, n):
        if n % i == 0:
            ans = i

    return ans
```

Q5: 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, 'equal', 'not equal')
    'not equal'
    >>> if_function(3>2, 'bigger', 'smaller')
    'bigger'
    """
    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 `61A`, but `with_if_function` prints both `Welcome to` and `61A` on separate lines.

```
def with_if_statement():
    """
    >>> result = with_if_statement()
    61A
    >>> print(result)
    None
    """
    if cond():
        return true_func()
    else:
        return false_func()

def with_if_function():
    """
    >>> result = with_if_function()
    Welcome to
    61A
    >>> 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 ***"
```

思路：这个题主要是考查对 `if` 语句的理解和对函数传递参数的理解，观察 `with_if_statement()` 和 `with_if_function()` 两个函数，不难发现，要实现的三个函数有以下特点：

- cond()：返回 `True` 或者 `False` 。
- `true_func()` 和 `false_func()` ：调用输出函数，没有返回值，一个输出 `"Welcome to"` ，一个输出 `"61A"` 。

综合下来，实现代码如下：

```
def cond():
    "*** YOUR CODE HERE ***"
    return False


def true_func():
    "*** YOUR CODE HERE ***"
    print("Welcome to")


def false_func():
    "*** YOUR CODE HERE ***"
    print("61A")
```

Q6: 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:

思路：经典的 `3*n+1` 问题，注意最开始的 `n` 就算作一步。
实现代码如下：

```
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 ***"
    step = 1
    print(n)
    while True:
        if n % 2 == 0:
            n //= 2
        else:
            n = n * 3 + 1
        print(n)
        step += 1
        if n == 1:
            break

    return step
```

测评结果

运行测评代码：

```
python ok --local
```

测评结果如下：

```
=====================================================================
Assignment: Homework 1
OK, version v1.18.1
=====================================================================

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Running tests

---------------------------------------------------------------------
Test summary
    6 test cases passed! No cases failed.

Cannot backup when running ok with --local.
```