Lab 02: Functions and Control

AUTHOR
Christian Osendorfer

Implement in Python

Implement the following functions which are specified by the description in the respective comments. Test using the available (if any) **doctests** and come up with more doctest if possible. Don't `import` any other module (you are allowed to `import operator` or `from operator import *`. And you should find out what `import` does.)

```
        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
            """
            # YOUR CODE HERE
```

```
        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
            """
            # YOUR CODE HERE
```

```
        def largest_factor(n):
            """Return the largest factor of n (n > 1 !) that is smaller than n.
            A factor divides a number evenly.

            >>> 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
```

```
def double_eights(n):
    """Return true if n has two eights in a row.
    >>> double_eights(8)
    False
    >>> double_eights(88)
    True
    >>> double_eights(2882)
    True
    >>> double_eights(880088)
    True
    >>> double_eights(12345)
    False
    >>> double_eights(80808080)
    False
    """
    # YOUR CODE HERE
```

```
def getKthDigit(n, k):
    """
    Return the kth digit of n (an integer), starting from 0,
    counting from the right.

    >>> getKthDigit(789, 0)
    9
    >>> getKthDigit(789, 1)
    8
    >>> getKthDigit(789, 2)
    7
    >>> getKthDigit(789, 3)
    0
    >>> getKthDigit(-789, 0)
    9
    """
    # YOUR CODE HERE
```

```
def setKthDigit(n, k, d):
    """
    n is an integer, k is a non-negative integer and d is non-negative
    single digit (0 ⩽ d ⩽ 9). Return the number n with the kth digit
    replaced with d.

    >>> setKthDigit(468, 0, 1)
    461
    >>> setKthDigit(468, 1, 1)
    418
    >>> setKthDigit(468, 2, 1)
    168
    >>> setKthDigit(468, 3, 1)
    1468
    """
    # YOUR CODE HERE
```

```
def sum_digits(y):
    """Sum all the digits of y. y is always nonnegative. Floor division
    and modulo might be helpful.

    >>> sum_digits(10) # 1 + 0 = 1
    1
    >>> sum_digits(4224) # 4 + 2 + 2 + 4 = 12
    12
    >>> sum_digits(1234567890)
    45
    >>> a = sum_digits(123) # use return rather than print in your code!
    >>> a
    6
    """
    # YOUR CODE HERE
```

Compute without a computer

Determine the results of the provided python code snippets by evaluating them without using a computer. If `print` is used, also provide the strings (order matters!) that are printed on the console.

```
def xk(c, d):
    if c == 4:
        return 6
    elif d >= 4:
        return 6 + 7 + c
    else:
        return 25
xk(10, 10)
xk(10, 6)
xk(4, 6)
xk(0, 0)
```

```
def how_big(x):
    if x > 10:
        print('huge')
    elif x > 5:
        return 'big'
    elif x > 0:
        print('small')
    else:
        print("nothing")
how_big(7)
how_big(12)
how_big(1)
how_big(-1)
```

```
def bake(cake, make):
    if cake == 0:
        cake = cake + 1
        print(cake)
    if cake == 1:
        print(make)
    else:
        return cake
    return make
bake(0, 29)
bake(1, "mashed potatoes")
```

```
def ab(c, d):
    if c > 5:
        print(c)
    elif c > 7:
        print(d)
    print('foo')
ab(10, 20)
```

```
def f(x):
    print(x)
    x = 2 * x
    return x + 1

def g(x):
    print(x)
    return f(x + 4)

def ct2(x):
    x = x // 2
    y = g(x)
    print(y)
    return y % (x + 3)

print(ct2(5) + 1)
```

For Fun: Simple Yahtzee

Write a simplified form of the dice game Yahtzee. In this version, the goal is to get 3 matching dice, and if you can't do that, then you try to get at least 2 matching dice. The game is played like so:

1. Roll 3 dice.
2. If you do not have 3 matching dice:
    i. If you have 2 matching dice (a pair), keep the pair and roll one die to replace the third die.
    ii. Otherwise, if you have no matching dice, keep the highest die and roll two dice to replace the other two dice.

3. Repeat step 2 one more time.
4. Finally, compute your score like so:
    i. If you have 3 matching dice, your score is 20 + the sum of the matching dice
        - So if you have 4-4-4, your score is 20+4+4+4, or 32.
    ii. If you only have 2 matching dice, your score is 10 + the sum of the matching dice.
        - So if you have 4-4-3, your score is 10+4+4, or 18.
    iii. If you have no matching dice, your score is the highest die. * So if you have 4-3-2, your score is just 4.

Come up with some Python code that plays this game. We will use *top-down design* and break it up into smaller, more manageable pieces. That is, first write some helper functions that do part of the work. Then those helper functions will make your final function much easier to write. Be sure to write your functions in the same order as given here, since later functions will make use of earlier ones!

Note

Represent a hand of 3 dice as a single 3-digit integer. So the hand 4-3-2 will be represented by the integer 432.

`handToDice(hand)`

The function `handToDice(hand)` takes a hand, which is a 3-digit integer, and returns 3 values, each of the 3 dice in the hand. For example:

```
>>> handToDice(123)
(1,2,3)
>>> handToDice(214)
(2,1,4)
```

`diceToOrderedHand(a, b, c)`

The function `diceToOrderedHand(a, b, c)` takes 3 dice and returns them in a hand, which is a 3-digit integer. However, even if the dice are unordered, the resulting hand must be ordered so that the largest die is on the left and smallest die is on the right. For example:

```
>>> diceToOrderedHand(1,2,3)
321
>>> diceToOrderedHand(6,5,4)
654
>>> diceToOrderedHand(1,4,2)
421
```

Tip

You can use `max(a,b,c)` to find the largest of 3 values, and `min(a,b,c)` to find the smallest.

`playStep2(hand, dice)`

The function `playStep2(hand, dice)` plays step 2 as explained above. This function takes a hand, which is a 3-digit integer, and it also takes dice, which is an integer containing all the future rolls of the dice. For example, if dice is 5341, then the next roll will be a 1, then the roll after that will be a 4, then a 3, and finally a 5. Note that in a more realistic version of this game, instead of hard-coding the dice in this way, we'd probably use a random-number generator.

With that, the function plays step 2 of the given hand, using the given dice to get the next rolls as needed. At the end, the function returns the new hand, but it has to be ordered, and the function also returns the resulting dice (which no longer contain the rolls that were just used). For example:

```
>>> playStep2(413, 2312)
(421, 23)
```

Here, the hand is 413, and the future dice rolls are 2312. What happens? Well, there are no matching dice in 413, so we keep the highest die, which is a 4, and we replace the 1 and the 3 with new rolls. Since new rolls come from the right (the one's digit), those are 2 and 1. So the new hand is 421. It has to be sorted, but it already is. Finally, the dice was 2312, but we used 2 digits, so now it's just 23. We return the hand and the dice, so we return (421, 23).

Here are some more examples. Be sure you carefully understand them:

```
>>> playStep2(413, 2345)
(544, 23)
>>> playStep2(544, 23)
(443, 2)
>>> playStep2(544, 456)
(644, 45)
```

Note

- You may wish to use `handToDice(hand)` at the start to convert the hand into the 3 individual dice.
- Then, you may wish to use `diceToOrderedHand(a, b, c)` at the end to convert the 3 dice back into a sorted hand.

`score(hand)`

The function `score(hand)` takes a 3-digit integer hand, and returns the score for that hand as explained in step 4 above. For example:

```
>>> score(432)
4
>>> score(532)
5
>>> score(443)
10+4+4
>>> score(633)
10+3+3
```

Note

The structure of this function is actually quite similar to the previous function.

`PlayThreeDiceYahtzee(dice)`

The last function is `PlayThreeDiceYahtzee(dice)`. This function takes one value, the dice with all the rolls for a game of 3-Dice Yahtzee. The function plays the game – it does step 1 and gets the first 3 dice (from the right), then it does step 2 twice (by calling `playStep2`, which you already wrote), and then it computes the score (by calling `score`, which you already wrote). The function should return two values – the resulting hand and the score for that hand. For example:

```
        >>> PlayThreeDiceYahtzee(2312413)
        (432, 4)
        >>> PlayThreeDiceYahtzee(2315413)
        (532, 5)
        >>> PlayThreeDiceYahtzee(2345413)
        (443, 18)
```