Lab 02: Practice with Functions

COSC 102 - Spring '24

Goal: In this lab you will further acclimate yourself to Java syntax by writing code in functions. Additionally, you will gain practice designing test caes.

1 Overview and Restrictions

For this assignment, you will be working with **two** `.java` files, described below:

- **`Lab02Code.java:`** you will write the write the functions described below in this file
- **`Lab02Client.java:`** this file will contain **only** the `main` method and will be used to call and test the functions in `Lab02Code`.

Additionally, as alluded to above, you are required to document the test cases you use in the `main` method of your `Lab02Client` code. **You will be graded** on the scope/quality/variety of these test cases – only using the test cases in this document **is not sufficient**.

Lastly, you are not allowed to use **`Strings`**, **`String` methods** or any other data structure we've yet to discuss in class (*ex:* fixed-length arrays, lists, etc)

2 Your Task

Complete the following functions in your **`Lab02Code.java`**. For each function **document a variety of test cases** in your **`Lab02Client.java`**.

2.1 Order Of The Number

Define a function called **`checkOrder`** which accepts as argument **one** `int` - **`'num'`**. This function returns an **`int`** indicating what is the order of the number provided as an argument. In other words, this function returns how many digits long the argument number is.

Lastly, when zero is provided as an argument to check order with, it should still return one as its result. Also for negative values treat them as though they were positive

Examples:

```
checkOrder(1234) -> 4
checkOrder(123) -> 3
checkOrder(12) -> 2
```

2.2 Raise To The Power

Define a function called **`getPower`** which accepts as argument two `int`s - **`'base'`** and **`'pow'`**. This function returns an **`int`** indicating what is the result of the `'base'` raised to the power of `'pow'`.
In other words, for the cube of number 2 (*i.e.* `2` to the power of `3`) -> $\mathbf{2}^{3} = \mathbf{8}$

Examples:

```
getPower(2, 2) -> 4
getPower(2, 3) -> 8
getPower(3, 3) -> 27
```

2.3 Is It Armstrong Number

In number theory, an Armstrong Number is a number that is the sum of its own digits each raised to the power of the number of digits. It is used in security systems. The number **153** is an armstrong number because $\mathbf{1}^3 + \mathbf{5}^3 + \mathbf{3}^3 = \mathbf{1} + \mathbf{125} + \mathbf{27} = \mathbf{153}$. 153 is a three digit number so when you take the sum of each digit raised to the power of 3 equals to the original number, then the number is deemed to be an armstrong number.

Define a function called **isArmstrongNumber** which accepts as argument two `int`s - **'num'** and **'order'**. This function returns a **boolean** indicating if the number **'num'** is indeed an armstrong number. You must use the functions you wrote in Tasks 2.1 and 2.2 in your solution.

Examples:

```
isArmstrongNumber(153, 3) -> true
isArmstrongNumber(132, 3) -> false
isArmstrongNumber(1634, 4) -> true
```

2.4 Num Reverse

Define a function named **numReverse** which accepts an argument `int` **num** and returns the *reverse* of `num` as an `int`. This means that, the leftmost digit of the returned value will be the rightmost digit of `num`; the second-leftmost digit of the returned value the second-rightmost digit of `num`, and so on.

Any trailing zeroes (*ex:* the two rightmost digits of **1400**) will be ignored in the returned value. If a non-positive argument is provided for `num`, this function instead returns **-1**.

As a reminder, you **may not** utilize another data type (*i.e.* `String` or array) or use any of respective functionalities of these types. Instead remember your modulus operator (*ex:* `x % y`) and the effects of integer division!

Examples:

```
numReverse(2734) -> 4372
numReverse(120) -> 21
```

2.5 Roll Chance

Define a function named **rollChance** which accepts three argument `int`s: **dieSides**, **sum**, and **rolls**. This function simulates taking a pair of dice with `dieSides` number of sides and rolling them `rolls` times. The function then returns the percentage of times the total of the rolled dice equaled `sum`.

The percentage will be returned as a **double** decimal value. For example, the chance of rolling a sum of `7` given two six-sided dice is roughly **1/6**, thus as `rolls` increases the returned value should become closer and closer to **0.1666**. If a non-positive value is provided for `dieSides`, `sum`, or `rolls`, this function returns **-1.0**.

Lastly, you must use the **Math.random()** function to handle your "dice rolling". This function returns a random `double` between **0** (inclusive) and **1** (exclusive). You **may not** use any other Java random functionality.

Remember, you can convert a `double` to an `int` using a type-cast, which will **truncate** any decimal component:

```
double x = 14.7;
int y = (int)x;
System.out.println(y); //prints 14
```

Examples:

Chance of rolling 12 w/ two six-sided dice is **1/36**, thus `rollChance(6, 12, rolls)` should near **0.0277**
Chance of rolling 17 w/ two twenty-sided dice is **1/25**, thus `rollChance(20, 4, rolls)` should near **0.04**

2.6 Most Divisible

Lastly, define a function named **`mostDivisible`** which accepts two `int` arguments **`bound1`** and **`bound2`**. This function returns the integer between `bound1` and `bound2` (inclusive) which is evenly divisible by the **most single digit positive integers**.

For example, given `bound1 = 22` and `bound2 = 28`, the most divisible number would be **`24`**. This is because `24` is divisible by **`6`** single digit integers: `1, 2, 3, 4, 6, and 8` – more than any other number between `22` and `28`.

You may assume `bound1` and `bound2` are both **positive** and **at least two digits** long, but they **may not** be in order (*i.e.* `bound2` could be <= `bound1`). If invalid value(s) are provided for the bounds, this function returns **`-1`**. Additionally, If multiple values in the range are tied for most divisible, any of them may be returned.

Example:

```
mostDivisible(22, 28) -> 24
```

3 Submission

Upload both your **`Lab02Code.java`** (containing your code) and **`Lab02Client.java`** (containing your test cases) to the submission link on your lab's Moodle page. This lab is due:

- **Tuesday, February 6th** by **11:59PM** for lab sections **A** , **B**, and **C** (which meet on Wednesday)
- **Wednesday, February 7th** by **11:59PM** for lab section **D** (which meets on Thursday)