String to Integer

Problem Description: Write a recursive function to convert a given string into the number it represents. That is input will be a numeric string that contains only numbers, you need to convert the string into corresponding integer and return the answer.

Sample Input:

"2029"

Sample Output:

2029

How to approach?

<u>Iterative approach</u>

We have to implement a function that converts a string to an integer. Before we look at this problem, let's discuss the approach for a much simpler problem. What we had to convert a character ('4') to an integer (4)? And before we even discuss that, do you know what happens when you store a character in an integer? For instance what will be the value of i after the following piece of code is executed:

```
char ch = 'a';
int i = ch;
```

'i' will have the value 97. Do you know why? The reason is that every key on your keyboard is mapped to a numerical value. This numerical value is called the ASCII code for that key. ASCII stands for American Standard Code for Information Interchange. You can visit http://www.asciitable.com/ to find out the ASCII values for every key on your keyboard. For this question, we are interested in the ASCII values of '0' to '9' (which are unfortunately not 0 to 9 as you would expect). They are from 48 to 57. That means '0' maps to 48, '1' to 49 and so on till '9' which maps to 57.

Coming back to the question, how will we convert a character to an integer? Let's say we want to convert '4' to 4. Let's see how we'll do it in code.

```
char ch = '4';
int i = ch - '0'
```

You're probably wondering what exactly does the second line of the code do? When we perform arithmetic operations like '+' or '-' to characters and store them in an integer, then the operation is done on the ASCII value of those characters. In this case, '4' maps to 52, '0' maps to 48. Therefore, the variable `i` will store the value of 52 - 48, or 4.

Now, that we know how to convert a character to an integer, let's see how to convert a string to an integer. Note that a string is just a collection of characters, so we can use the same method that we discussed above and construct our integer.

We'll discuss the approach for this problem with a sample input. Let the input be str = "935". We will iterate through this string and store the integer in a variable called `num`. There will be three iterations in total because the string has three characters. Let `i` be the looping variable.

1. `i = 0 -> str[i] = '9'` We will just store (`'9'-'0'`) in `num` and move on. (num = 9)

2. `i = 1 -> str[i] = '3'` We will get the next digit by (`'3'-'0'`) and then 'append' it to `num`. The way we append a digit to an integer is to multiply the integer by 10 and then add the digit. So we will do something like this:

   ```
   int nextDigit = str[i] - '0'
   num = num*10 + nextDigit
   ```

   Note that this would work even for the first character if we initialise `num` with 0. (num = 93)

3. `i = 2 <- str[i] = '5'` Very similar to the previous step. We get the digit 5 and append it to `num`.
   (num = 935)

By the time we iterate through the loop, `num` would have the complete integer stored in it. Then we can print it, return it or use it further in our program.

Recursive approach

Now that we know the iterative approach, the recursive one is not very hard to grasp. We'll get a string (let's say "324"). The first thing we'll check is whether our string is empty. If it is, we can return 0. If not, then we do some work.

To think of this problem recursively, we'll split this string in two parts, one of which will be a single character and the other will be the rest of the string.

Now we have two ways to split this string. Taking the example string, either we can do "3" + "24" or we can do "32" + "4". i.e. the single character part of the string can either be the string's first character or its last character.

We will discuss the second way of splitting. In this way, we will recurse on our function passing the substring of the string that starts at the first character and goes till the second last character. And we will store the result of that substring. Taking the example string we will do something like this.

```
function solve(str): //str is "324"
  //Base case goes here
  len <- str.length()
  smallAnswer <- solve(str.sub(0, len - 1)); //str.sub is "32"
```

Now, smallAnswer will contain the integer value of the substring (in our case, 32). Now, we will get the last character of our original string, and convert that to an integer (we know how) and then 'append' it to smallAnswer. ('appending a digit to an integer is also discussed above'). Then we can finally return the smallAnswer that has the appended digit.

The pseudo-code for both approaches is given on the next page

Iterative Approach

```
function convertStringToInteger(str):

  num <- 0

  for char in str:

    nextDigit <- char - ‘0’
    num <- (num * 10) + nextDigit

  return num
```

Recursive Approach

```
function convertStringToInteger(str):

  len <- str.length()

  if(len = 0): //Base case
    return 0

  smallAnswer <- convertStringToInteger(str.substring(0, len - 1))

  nextDigit <- str[len - 1] - ‘0’

  finalAnswer <- (smallAnswer * 10) + nextDigit

  return finalAnswer
```

Further exercise: How would you change these codes to accommodate for negative values, i.e. convert “-487” to -487, etc.