Google Sheets Formulas

Capitalize the first letter of each word in a given string 1
Factorial of a given number 3
Extract the domain name from a given URL 5
String to title case 7
Calculate the sum of the even numbers in a range 10
Count the number of occurrences of a given substring in a string 12
Convert a number to its Roman numeral equivalent 15
Return the last number of characters of a string 18
Compound Interest 20

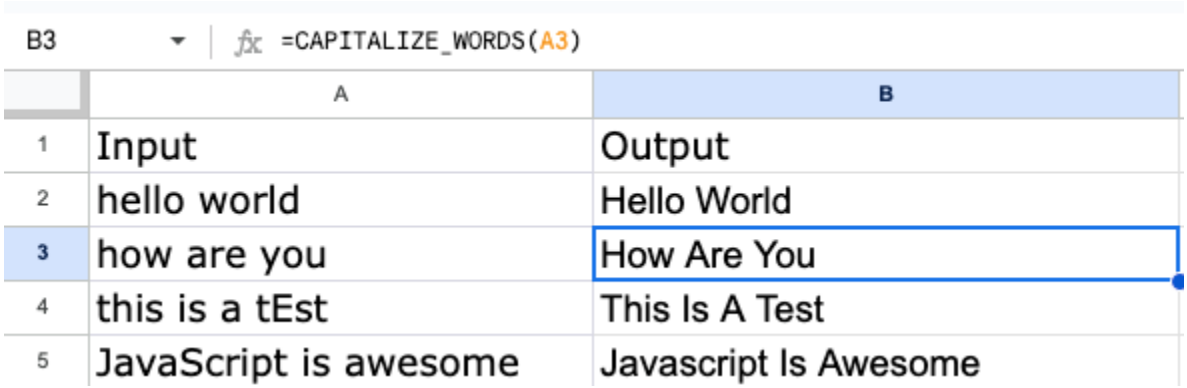
Capitalize the first letter of each word in a given string

```
function CAPITALIZE_WORDS(str) {
 const words = str.split(' ');
 for(let i=0;i<words.length;i++){
   words[i] = words[i].charAt(0).toUpperCase() +
words[i].slice(1).toLowerCase();
 }
 return words.join(' ');
}
```

The given code defines a function called CAPITALIZE_WORDS that takes a string str as an argument. The purpose of the function is to capitalize the first letter of each word in the given string and convert the rest of the letters to lowercase. The function follows these steps:

1. It starts by splitting the input string str into an array of words using the space (' ') as the delimiter. The resulting array is stored in the variable words.
2. It then iterates over each word in the words array using a for loop.
3. Inside the loop, it capitalizes the first letter of each word by accessing the first character using charAt(0), converting it to uppercase using the toUpperCase() method, and then concatenating it with the rest of the word. The rest of the word is obtained by slicing the word from the second character onwards and converting it to lowercase using the toLowerCase() method. The modified word is then assigned back to the words array at the same index.
4. Once all the words have been processed, the function joins the modified words back into a single string using the space (' ') as the delimiter, and returns the resulting capitalized string.

In summary, the CAPITALIZE_WORDS function takes a string, capitalizes the first letter of each word, converts the remaining letters to lowercase, and returns the modified string.



Input	Output
hello world	Hello World
how are you	How Are You
this is a tEst	This Is A Test
JavaScript is awesome	Javascript Is Awesome

Factorial of a given number

```
function FACTORIAL(n){
 if(n===0 || n===1){
   return 1;
 }else{
   return n * FACTORIAL(n-1);
 }
}
```

The given code defines a recursive function called FACTORIAL that calculates the factorial of a given number n. The factorial of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n. The function follows these steps:

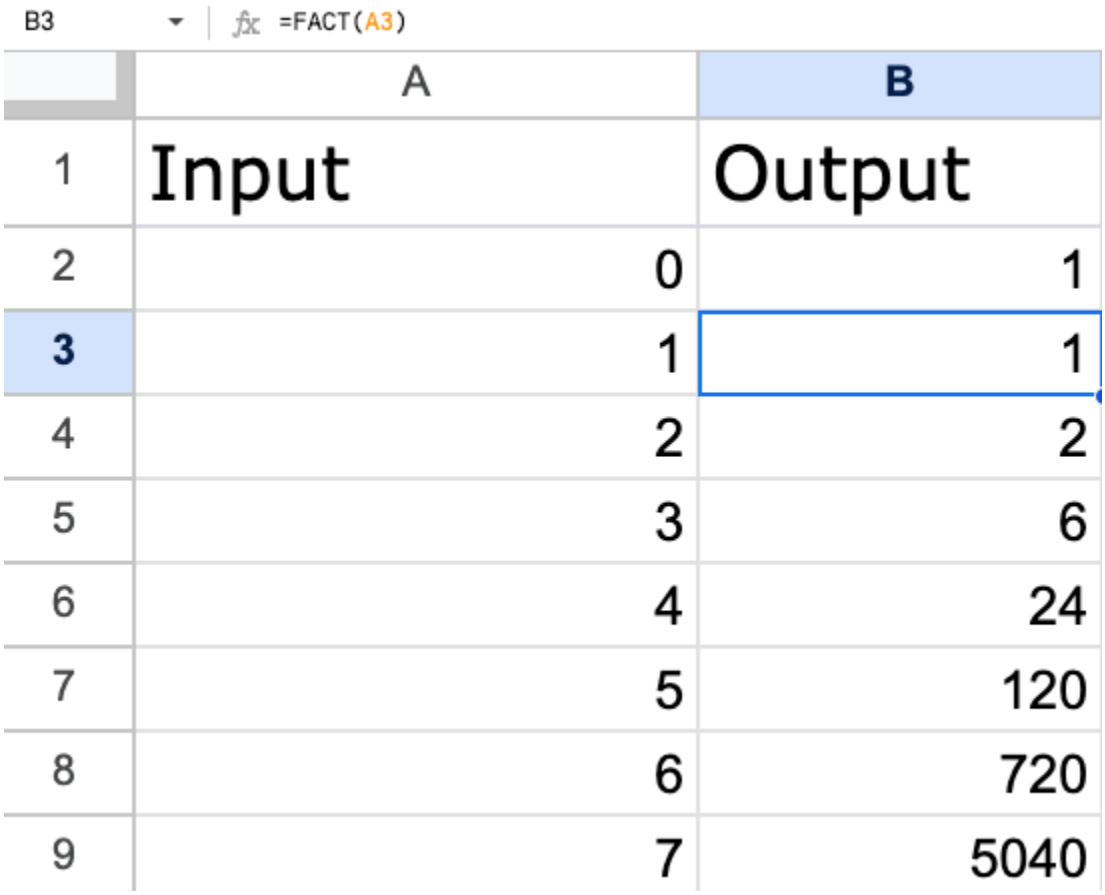
1. The function starts with a base case. It checks if the input n is equal to 0 or 1 using the strict equality operator ===. If n is either 0 or 1, it means we have reached the smallest possible factorial value, which is 1. In this case, the function returns 1.
2. If the input n is not 0 or 1, it means we need to calculate the factorial for a larger number. In this case, the function enters the else block.
3. Inside the else block, the function recursively calls itself with the argument n-1. This is done to calculate the factorial of the number n-1. The result of the recursive call is multiplied by n, which gives the factorial of n. The multiplication is performed using the * operator.
4. The recursive call continues until the base case is reached, at which point the function starts returning the factorial values upwards. Each recursive call multiplies the current number n with the factorial of n-1, until the original n is reached.
5. Finally, the calculated factorial value is returned as the result of the function.

To understand how this function works, let's consider an example:
scss
Copy code
FACTORIAL(5)
Since 5 is not equal to 0 or 1, the function enters the else block and makes a recursive call to FACTORIAL(4).

Similarly, FACTORIAL(4) makes a recursive call to FACTORIAL(3), and so on, until FACTORIAL(1) is reached.
At FACTORIAL(1), the base case is triggered, and the function returns 1.
Now, the function starts returning the factorial values upwards:

- FACTORIAL(1) returns 1 to FACTORIAL(2), which multiplies it by 2 and returns 2.
- FACTORIAL(2) returns 2 to FACTORIAL(3), which multiplies it by 3 and returns 6.
- FACTORIAL(3) returns 6 to FACTORIAL(4), which multiplies it by 4 and returns 24.
- FACTORIAL(4) returns 24 to FACTORIAL(5), which multiplies it by 5 and returns the final result 120.

So, FACTORIAL(5) returns 120, which is the factorial of 5.
In summary, the FACTORIAL function uses recursion to calculate the factorial of a given number n by repeatedly multiplying the current number with the factorial of the preceding number until it reaches the base case.



Input	Output
0	1
1	1
2	2
3	6
4	24
5	120
6	720
7	5040

Extract the domain name from a given URL


```
function GET_DOMAIN(url){
 let domain = '';
 let matches =
url.match(/^https?\:\/\/([^\/?#]+)(?:[\/?#]|$)/i);
 if(matches && matches[1]){
   domain = matches[1];
 }
 return domain;
}
```

The given code defines a function called GET_DOMAIN that extracts and returns the domain name from a given URL. The domain name represents the network location of a website. The function follows these steps:

1. It starts by declaring and initializing two variables: domain and matches. The domain variable will store the extracted domain name, while the matches variable will be used to store the result of the regular expression match.
2. The regular expression pattern used in the match method is /^https?\:\/\/([^\/?#]+)(?:[\/?#]|$)/i. Let's break down this pattern:
    - ^ asserts the start of the string.
    - https? matches either "http" or "https".
    - \:\/\/ matches the literal characters "://".
    - ([^\/?#]+) captures one or more characters that are not "/", "?", or "#" as a group.
    - (?:[\/?#]|$) matches either "/", "?", or "#" non-capturing group or the end of the string.
    - /i specifies that the match should be case-insensitive.
3. This regular expression pattern is used to match and capture the domain name portion of the URL.
4. The match method is called on the url string, passing the regular expression pattern as an argument. The method attempts to find a match for the pattern in the url string.
5. The result of the match method is stored in the matches variable. If a match is found, the matches variable will be an array containing the entire matched string as the first element, followed by the captured groups.
6. The if statement checks if matches is truthy (not null or undefined) and if matches[1] exists. Since we are interested in the captured group at index 1 (the domain name), the condition checks if the element at index 1 of matches exists.
7. If the condition in the if statement is true, it means a match was found, and the domain name is extracted from matches[1]. The domain name is assigned to the domain variable.
8. Finally, the domain variable, which either contains the extracted domain name or an empty string, is returned as the result of the function.

To understand how this function works, let's consider an example:
rust

Copy code
GET_DOMAIN('https://www.example.com/path/to/page.html')
The function will extract the domain name from the given URL and return it.
In this case, the regular expression pattern matches the "https://" portion of the URL and captures "www.example.com" as the domain name.
The extracted domain name is then assigned to the domain variable, and the function returns "www.example.com" as the result.
In summary, the GET_DOMAIN function extracts and returns the domain name from a given URL by using a regular expression match to capture the relevant portion of the URL.

Input (url)	Output
https://www.example.com	www.example.com
http://example.com/index.html	example.com
https://subdomain.example.org/path?query=string	subdomain.example.org
http://localhost:3000/	localhost:3000

String to title case

```
function TITLE_CASE(str){
 const words = str.toLowerCase().split(' ');
 words.forEach((ele,ind) =>{
   words[ind] = ele.charAt(0).toUpperCase()+ele.slice(1);
 })
 return words.join(' ');
}
```

B3 =TITLE_CASE(A3)

	A	B
1	Input	Output
2	hello world	Hello World
3	THE QUICK BROWN FOX	The Quick Brown Fox
4	cAn YoU cOnVeRt ThIs?	Can You Convert This?
5	tEStIng tiTle caSE	Testing Title Case
6	ALL CAPS	All Caps

The given code defines a function called TITLE_CASE that converts a given string str into title case. Title case is a writing style where the first letter of each word is capitalized, and the rest of the letters are in lowercase. The function follows these steps:

1. It starts by declaring a constant variable words and assigns it the result of manipulating the input string str. The string is first converted to lowercase using the toLowerCase() method to ensure consistency. Then, the split() method is used to split the string into an array of words using the space (' ') as the delimiter. The resulting array of words is stored in the words variable.
2. The forEach() method is called on the words array, which iterates over each element (word) of the array. For each word, the function executes the provided callback function.
3. Inside the callback function, the current word is accessed using the ele parameter. The index of the current word in the array is accessed using the ind parameter.
4. The callback function capitalizes the first letter of each word by using the charAt(0) method to access the first character of the word, converting it to uppercase using the toUpperCase() method, and then concatenating it with the rest of the word. The rest of the word is obtained by slicing the word from the second character onwards using the slice(1) method. The modified word is then assigned back to the words array at the same index (ind).
5. After iterating through all the words in the words array and modifying them to title case, the function returns the modified words joined back

into a single string using the space (' ') as the delimiter. This is done by calling the join(' ') method on the words array.

To understand how this function works, let's consider an example:

scss

Copy code

TITLE_CASE('hello world')

The input string is 'hello world'.

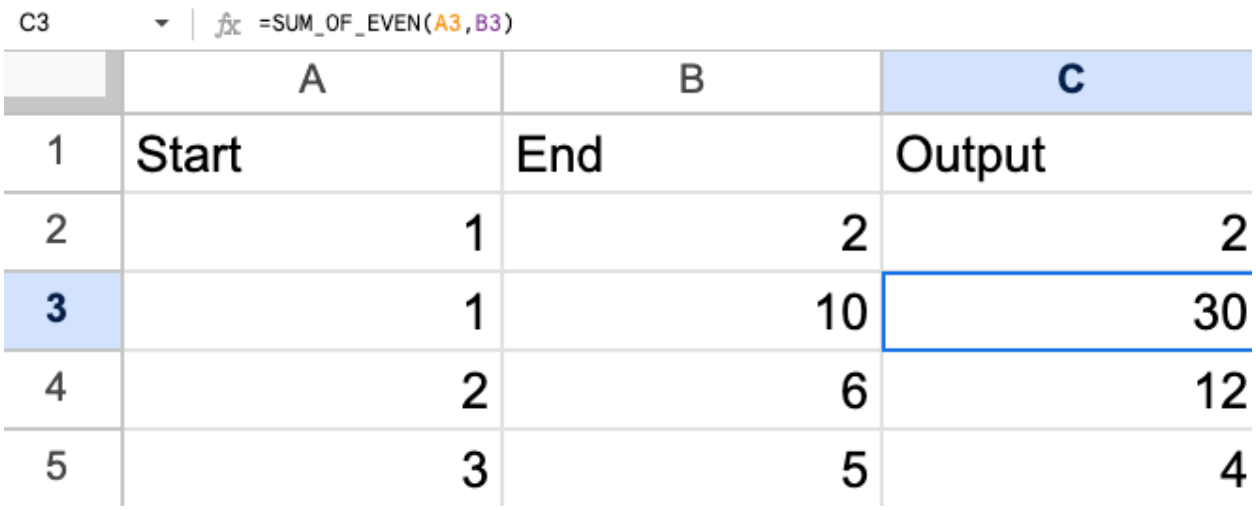
1. The toLowerCase() method is applied to the input string, resulting in 'hello world'.
2. The split(' ') method is applied to the lowercase string, resulting in the words array: ['hello', 'world'].
3. The forEach() method iterates over each word in the words array.
    - For the first iteration, the word is 'hello' at index 0.
        - The first character 'h' is capitalized using charAt(0).toUpperCase(), resulting in 'H'.
        - The rest of the word 'ello' is obtained using slice(1).
        - The modified word 'Hello' is assigned back to the words array at index 0.
    - For the second iteration, the word is 'world' at index 1.
        - The first character 'w' is capitalized using charAt(0).toUpperCase(), resulting in 'W'.
        - The rest of the word 'orld' is obtained using slice(1).
        - The modified word 'World' is assigned back to the words array at index 1.
4. After modifying all the words, the join(' ') method is called on the words array, resulting in the string 'Hello World'.
5. The modified string 'Hello World' is returned as the result of the function.

In summary, the TITLE_CASE function converts a given string into title case by converting the string to lowercase, splitting it into an array of words, capitalizing the first letter of each word, and then joining the modified words back into a single string.

Input	Output
hello world	Hello World

THE QUICK BROWN FOX	The Quick Brown Fox
cAn YoU cOnVeRt ThIs?	Can You Convert This?
tEStIng tiTle caSE	Testing Title Case
ALL CAPS	All Caps

Calculate the sum of the even numbers in a range



	A	B	C
1	Start	End	Output
2	1	2	2
3	1	10	30
4	2	6	12
5	3	5	4

```
function SUM_OF_EVEN(first,second) {
 let sum = 0;
 for (let i = first; i < second+1; i++) {
   if (i % 2 === 0) {
     sum += i;
   }
 }
 return sum;
}
```

The given code defines a function called SUM_OF_EVEN that calculates the sum of all even numbers within a given range. The function takes two parameters, first and second, representing the range of numbers (inclusive) between which the sum of even numbers is to be calculated. The function follows these steps:

1. It starts by declaring and initializing a variable called sum to 0. This variable will be used to store the cumulative sum of even numbers.
2. The function enters a for loop, starting from the value of first and iterating until second+1. The loop variable i represents each number within the range, including both first and second.
3. Inside the loop, an if statement is used to check if the current value of i is even. The condition i % 2 === 0 checks if i is divisible by 2 with no remainder, which indicates that i is an even number.
4. If the condition is true (i.e., i is even), the value of i is added to the sum variable using the += operator. This accumulates the even numbers and updates the sum.
5. Once the loop has iterated through all the numbers within the given range, the sum variable holds the total sum of even numbers.
6. Finally, the function returns the value of sum as the result.

To understand how this function works, let's consider an example:

scss

Copy code

```
SUM_OF_EVEN(1, 10)
```

The function will calculate the sum of even numbers between 1 and 10 (inclusive).

The for loop starts with i = 1 and iterates until i reaches 10+1, which means it iterates from 1 to 11 (inclusive).

Inside the loop:

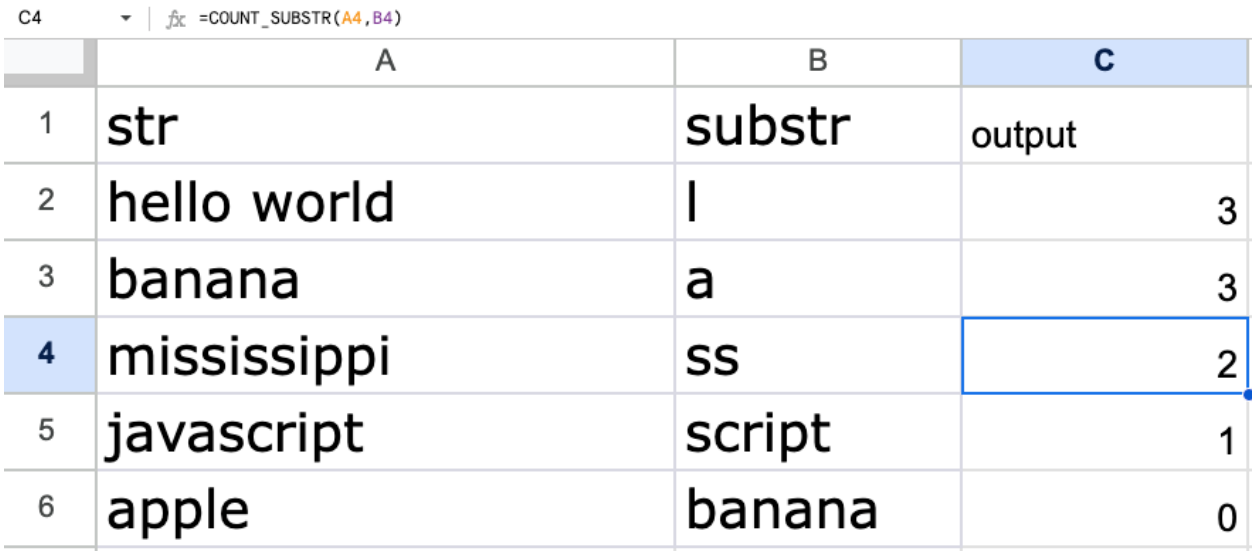
- At i = 1, the condition i % 2 === 0 is false since 1 is not an even number, so nothing happens.
- At i = 2, the condition is true, and 2 is added to the current value of sum, which is 0. So, sum becomes 2.
- At i = 3, the condition is false, so nothing happens.
- At i = 4, the condition is true, and 4 is added to sum, resulting in sum becoming 6.

- This process continues until i = 10, where 10 is added to sum, making sum equal to 30.

After the loop finishes, the function returns 30 as the sum of all even numbers between 1 and 10 (inclusive).
In summary, the SUM_OF_EVEN function calculates the sum of even numbers within a given range by iterating through each number in the range, checking if it's even, and adding it to the cumulative sum. The final sum is returned as the result of the function.

Start	End	Output
1	2	2
1	10	30
2	6	12
3	5	4

Count the number of occurrences of a given substring in a string



	A	B	C
1	str	substr	output
2	hello world	l	3
3	banana	a	3
4	mississippi	ss	2
5	javascript	script	1
6	apple	banana	0

```
function COUNT_SUBSTR(str,sbStr){
 let count = 0;
 let pos = str.indexOf(sbStr);
```

```
 while (pos !== -1){
   count++;
   pos = str.indexOf(sbStr,pos+1);
 }
 return count;
}
```

The provided code defines a function called COUNT_SUBSTR that takes two parameters: str and sbStr. The purpose of this function is to count the number of occurrences of a substring (sbStr) within a given string (str). Here is a detailed explanation of how the function works:

1. let count = 0; creates a variable called count and initializes it to 0. This variable will be used to keep track of the number of occurrences of sbStr in str.
2. let pos = str.indexOf(sbStr); initializes a variable called pos with the index of the first occurrence of sbStr in str. The indexOf() method returns the index of the first occurrence of the specified substring or -1 if it is not found.
3. The while loop is used to iterate over str and find all occurrences of sbStr. It continues as long as pos is not equal to -1, which means that there are still more occurrences of sbStr to be found.
4. Inside the while loop, count++ increments the count variable by 1, indicating that another occurrence of sbStr has been found.
5. pos = str.indexOf(sbStr, pos + 1); updates the value of pos to the index of the next occurrence of sbStr in str, starting from the position immediately after the previous occurrence. This ensures that the loop continues to find all occurrences of sbStr and doesn't get stuck in an infinite loop.
6. Once there are no more occurrences of sbStr in str, the indexOf() method returns -1, causing the while loop to exit.
7. Finally, the function returns the value of count, which represents the total number of occurrences of sbStr in str.

In summary, this function uses a loop and the indexOf() method to iterate over a string and count the number of occurrences of a specific substring within it.

str	substr	output
hello world	l	3
banana	a	3
mississippi	ss	2
javascript	script	1
apple	banana	0

Convert a number to its Roman numeral equivalent

D5 fx

	A	B
1	Input (num)	
2	1	I
3	3	III
4	4	IV
5	9	IX
6	10	X
7	14	XIV
8	50	L
9	44	XLIV
10	2023	MMXXIII
11	33	XXXIII

```
function TO_ROMAN(num){
 if(typeof num !== 'number') return NaN;
 let roman = '';
 const romanValues = {
```

```
  M: 1000,
  CM: 900,
  D: 500,
  CD: 400,
  C: 100,
  XC: 90,
  L: 50,
  XL: 40,
  X: 10,
  IX: 9,
  V: 5,
  IV: 4,
  I: 1
};
for(let key in romanValues){
  while(num >= romanValues[key]){
    roman += key;
    num -= romanValues[key];
  }
}
return roman;
}
```

The provided code defines a function called TO_ROMAN that takes a parameter num and converts it to a Roman numeral representation. Here is a detailed explanation of how the function works:

1. if(typeof num !== 'number') return NaN; checks if the num parameter is not a number. If it is not a number, the function immediately returns

NaN (Not a Number). This is a basic input validation step to ensure that the input is a valid number.

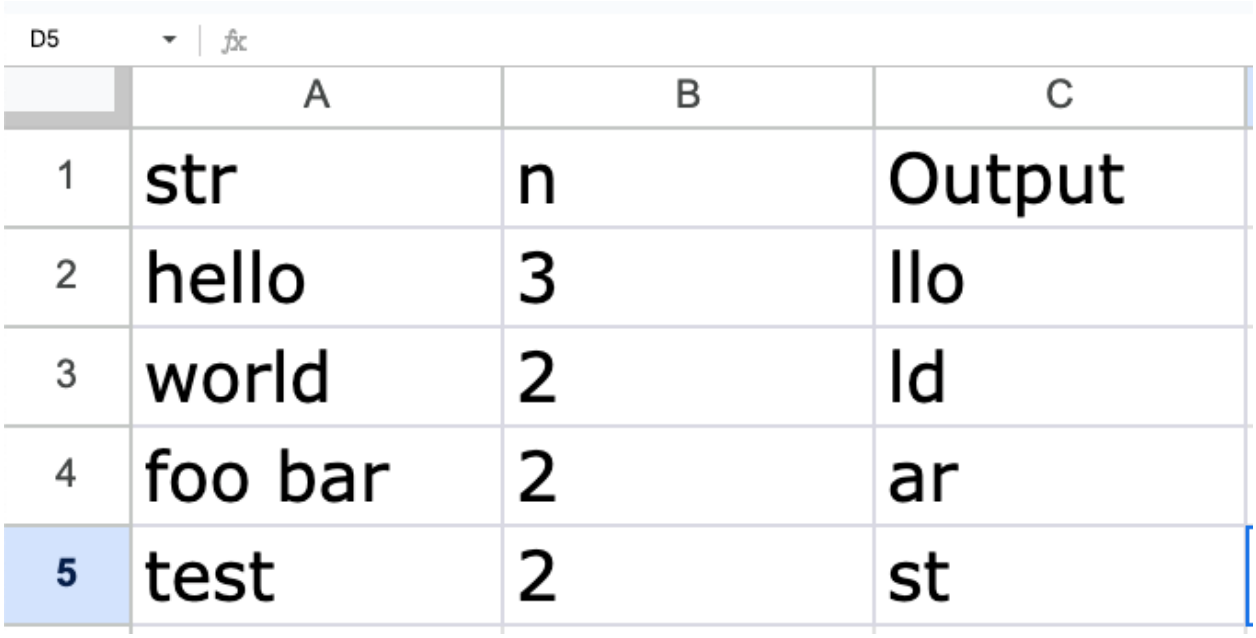
2. let roman = ''; creates an empty string variable called roman, which will store the Roman numeral representation of the input number.
3. const romanValues = {...}; declares a constant object called romanValues that maps Roman numerals to their corresponding decimal values. This mapping is used to convert the input number to its Roman numeral representation. The object stores the Roman numerals as keys and their corresponding decimal values as values.
4. The for...in loop iterates over each key in the romanValues object.
5. Inside the loop, the while loop is used to check if the input number (num) is greater than or equal to the current Roman numeral's decimal value (romanValues[key]).
6. If the condition is true, it means that the current Roman numeral should be added to the roman string. So, roman += key; concatenates the current Roman numeral (key) to the roman string.
7. Additionally, num -= romanValues[key]; subtracts the decimal value of the current Roman numeral from the num variable. This ensures that we keep track of the remaining value that needs to be converted into Roman numerals.
8. The loop continues until the num variable is less than the decimal value of the current Roman numeral, at which point it moves to the next Roman numeral in the romanValues object.
9. Once all the Roman numerals have been processed and appended to the roman string, the function exits the loop.
10. Finally, the function returns the roman string, which represents the Roman numeral representation of the input number.

In summary, this function converts a decimal number into its Roman numeral representation by iteratively subtracting the decimal values of Roman numerals from the input number and appending the corresponding Roman numerals to a string.

Input (num)	
1	I
3	III
4	IV

9	IX
10	X
14	XIV
50	L
44	XLIV
2023	MMXXIII
33	XXXIII

Return the last number of characters of a string



	A	B	C
1	str	n	Output
2	hello	3	llo
3	world	2	ld
4	foo bar	2	ar
5	test	2	st

```
function LAST_VALS(str,num){
 if(num >= str.length){
   return str;
 }
 return str.slice(str.length - num);
```

```
}
```

The provided code defines a function called LAST_VALS that takes two parameters: str and num. The function returns the last num characters from the str string. Here is a detailed explanation of how the function works:

1. The function begins with an if statement: if(num >= str.length). This condition checks if the value of num is greater than or equal to the length of the str string. If this condition is true, it means that the function should return the entire str string because there are not enough characters to extract the last num characters. In this case, the function returns the str string as it is.
2. If the condition in the if statement is false, the function proceeds to the next line: return str.slice(str.length - num);. The slice() method is used to extract a portion of the str string.
3. The argument passed to slice() is str.length - num. This calculates the starting index from which the last num characters should be extracted. By subtracting num from str.length, we get the starting index of the desired portion.
4. When slice() is called with a single argument, it extracts characters starting from the specified index until the end of the string. Therefore, str.slice(str.length - num) returns the last num characters from the str string.
5. Finally, the extracted portion of the string is returned by the function.

In summary, the LAST_VALS function extracts the last num characters from the str string and returns them. If the value of num is greater than or equal to the length of the string, the entire str string is returned.

str	n	Output
hello	3	llo
world	2	ld
foo bar	2	ar
test	2	st

Compound Interest

D3 =COMPOUND_INTEREST(A3,B3,C3)

	A	B	C	D
1	Principal	Rate	Time	Compound Interest
2	1000	5	1	50.00
3	5000	3.5	3	543.59
4	2500	7	5	1006.38
5	10000	2	10	2189.94
6	3000	4.5	2	276.07

code for the COMPOUND_INTEREST function and explain how it works.

```
function COMPOUND_INTEREST(principal, rate, time) {
  var interest = principal * (Math.pow((1 + (rate / 100)), time)
- 1);
  return interest.toFixed(2);
}
```

This code defines a function called COMPOUND_INTEREST that takes three parameters: principal, rate, and time. The function calculates the compound interest based on these parameters and returns the result rounded to two decimal places.
Let's go through the code step by step:

1. var interest = principal * (Math.pow((1 + (rate / 100)), time) - 1);
   This line calculates the compound interest using the formula: `interest = principal * (Math.pow((1 + (rate / 100)), time) - 1);`
   - principal: The initial amount of money or investment.
   - rate: The annual interest rate (percentage).
   - time: The time period (in years) for which the interest is compounded.
2. The formula (1 + (rate / 100)) calculates the multiplication factor for each time period based on the interest rate. The Math.pow() function

is used to raise this factor to the power of time, representing the number of compounding periods. Subtracting 1 from this value accounts for the initial principal amount.
The result of this calculation is assigned to the interest variable.

3. return interest.toFixed(2);
   This line returns the calculated interest value rounded to two decimal places using the toFixed() method. The toFixed() method converts the number into a string representation with the specified number of decimal places.

Now, let's consider an example table of data from Google Sheets:

Principal	Rate	Time	Compound Interest
1000	5%	1	
5000	3.5%	3	
2500	7%	5	
10000	2%	10	
3000	4.5%	2	

To calculate the compound interest for each row, you can call the COMPOUND_INTEREST function for each row's Principal, Rate, and Time values.
For example, to calculate the compound interest for the first row (Principal: 1000, Rate: 5%, Time: 1), you would call the function like this: COMPOUND_INTEREST(1000, 5, 1);
The function will return the calculated compound interest value rounded to two decimal places, which you can then fill into the "Compound Interest" column in the corresponding row of the table.
You can repeat this process for each row to calculate the compound interest for the entire table.

```
function COMPOUND_INTEREST(principal, rate, time) {
```

```
 var interest = principal * (Math.pow((1 + (rate / 100)), time)
- 1);
 return interest.toFixed(2);
}
```

Principal	Rate	Time	Compound Interest
1000	5	1	50.00
5000	3.5	3	543.59
2500	7	5	1006.38
10000	2	10	2189.94
3000	4.5	2	276.07