

Area of a Triangle

Write a function that takes the base and height of a triangle and `return` its area.

Examples

`triArea(3, 2) → 3`



The screenshot shows a code editor on the left and a terminal on the right. The code editor contains the following JavaScript code:

```
1 - function triangle(x, y) {  
2   let area;  
3   area = 1 / 2 * (x * y)  
4   console.log("Area of triangle is:"+area)  
5 }  
6  
7 triangle(2, 3)
```

The terminal on the right displays the output: "Area of triangle is:3". It also includes a hint: "Hint: hit control+c anytime to enter REPL." and a prompt character ">".

`triArea(7, 4) → 14`



The screenshot shows a code editor on the left and a terminal on the right. The code editor contains the following JavaScript code:

```
1 - function triangle(x, y) {  
2   let area;  
3   area = 1 / 2 * (x * y)  
4   console.log("Area of triangle is:"+area)  
5 }  
6  
7 triangle(7, 4)
```

The terminal on the right displays the output: "Area of triangle is:14". It also includes a hint: "Hint: hit control+c anytime to enter REPL." and a prompt character ">".

`triArea(10, 10) → 50`



The screenshot shows a code editor on the left and a terminal on the right. The code editor contains the following JavaScript code:

```
1 - function triangle(x, y) {  
2   let area;  
3   area = 1 / 2 * (x * y)  
4   console.log("Area of triangle is:"+area)  
5 }  
6  
7 triangle(10, 10)
```

The terminal on the right displays the output: "Area of triangle is:50". It also includes a hint: "Hint: hit control+c anytime to enter REPL." and a prompt character ">".

Notes

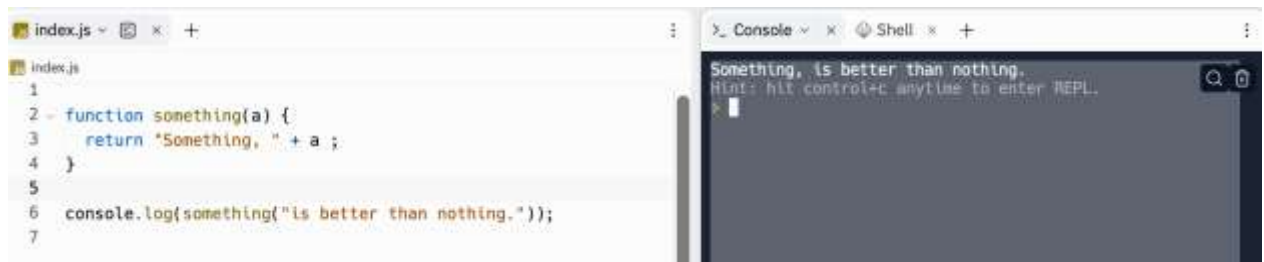
- The area of a triangle is: $(\text{base} * \text{height}) / 2$
- Don't forget to `return` the result.

Return Something to Me!

Write a function that returns the string `"something"` joined with a space `" "` and the given argument `a`.

Examples

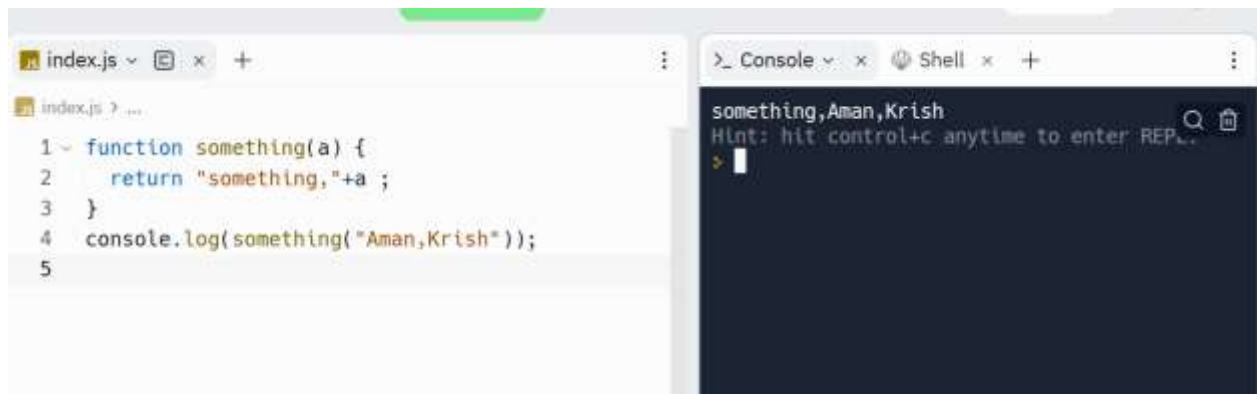
`giveMeSomething("is better than nothing") → "something is better than`



```
index.js
1
2 function something(a) {
3   return "Something, " + a ;
4 }
5
6 console.log(something("is better than nothing.));
7
```

```
Console
Something, is better than nothing.
Hint: hit control+c anytime to enter REPL.
```

nothing" giveMeSomething("Bob Jane") → "something Bob Jane"

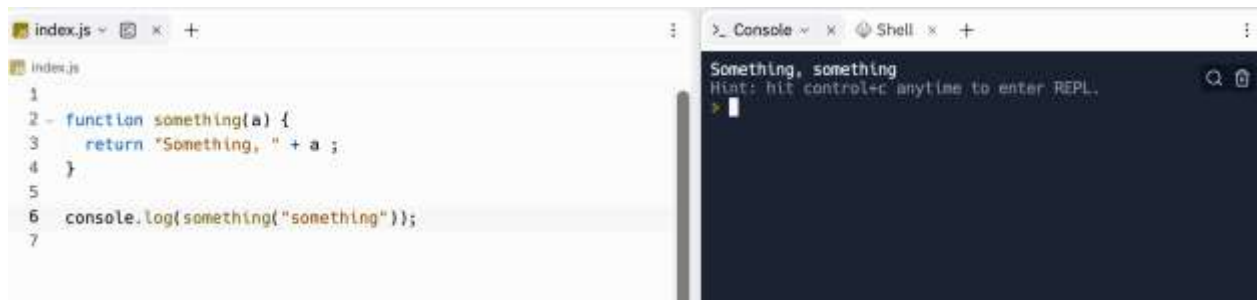


The screenshot shows a code editor with a file named `index.js`. The code defines a function `something(a)` that returns a string concatenation of `"something,"` and `a`. It then calls `console.log(something("Aman,Krish"))`. The console on the right shows the output: `something,Aman,Krish`.

```
index.js > ...  
1 ~ function something(a) {  
2   return "something,"+a ;  
3 }  
4 console.log(something("Aman,Krish"));  
5
```

Console: something,Aman,Krish
Hint: hit control+c anytime to enter REPL.

giveMeSomething("something") → "something something"



The screenshot shows a code editor with a file named `index.js`. The code defines a function `something(a)` that returns a string concatenation of `"Something, "` and `a`. It then calls `console.log(something("something"))`. The console on the right shows the output: `Something, something`.

```
index.js  
1  
2 ~ function something(a) {  
3   return "Something, " + a ;  
4 }  
5  
6 console.log(something("something"));  
7
```

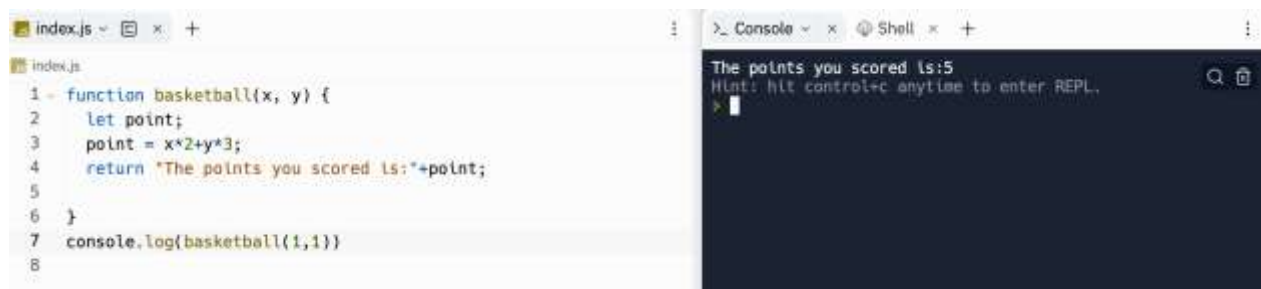
Console: Something, something
Hint: hit control+c anytime to enter REPL.

Basketball Points

You are counting points for a basketball game, given the amount of 2-pointers scored and 3-pointers scored, find the final points for the team and return that value.

Examples

`points(1, 1) → 5`

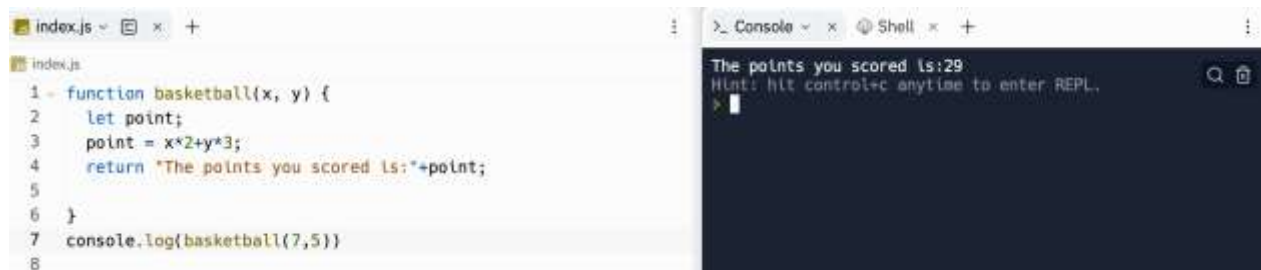


The screenshot shows a code editor with a file named `index.js`. The code defines a function `basketball(x, y)` that calculates the total points based on 2-pointers (`x`) and 3-pointers (`y`). The function returns a string: `"The points you scored is:" + point`. The console output shows the result of `console.log(basketball(1,1))` as `The points you scored is:5`.

```
1 function basketball(x, y) {  
2   let point;  
3   point = x*2+y*3;  
4   return "The points you scored is:" + point;  
5 }  
6  
7 console.log(basketball(1,1))  
8
```

The points you scored is:5
Hint: hit control+c anytime to enter REPL.

`points(7, 5) → 29`

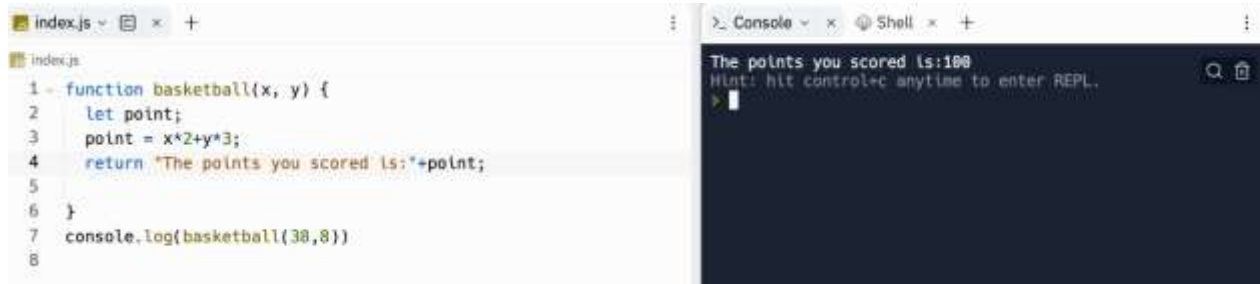


The screenshot shows the same code editor and console as the first example, but with the function `basketball` called with arguments `(7,5)`. The console output shows the result as `The points you scored is:29`.

```
1 function basketball(x, y) {  
2   let point;  
3   point = x*2+y*3;  
4   return "The points you scored is:" + point;  
5 }  
6  
7 console.log(basketball(7,5))  
8
```

The points you scored is:29
Hint: hit control+c anytime to enter REPL.

points (38, 8) → 100



The screenshot shows a code editor with a file named 'index.js'. The code defines a function 'basketball(x, y)' that calculates a point value based on x and y, and then logs the result. The console shows the output of the function call.

```
index.js
1 function basketball(x, y) {
2   let point;
3   point = x*2+y*3;
4   return "The points you scored is:"+point;
5 }
6
7 console.log(basketball(38,8))
8
```

Console: The points you scored is:100
Hint: hit control+c anytime to enter REPL.

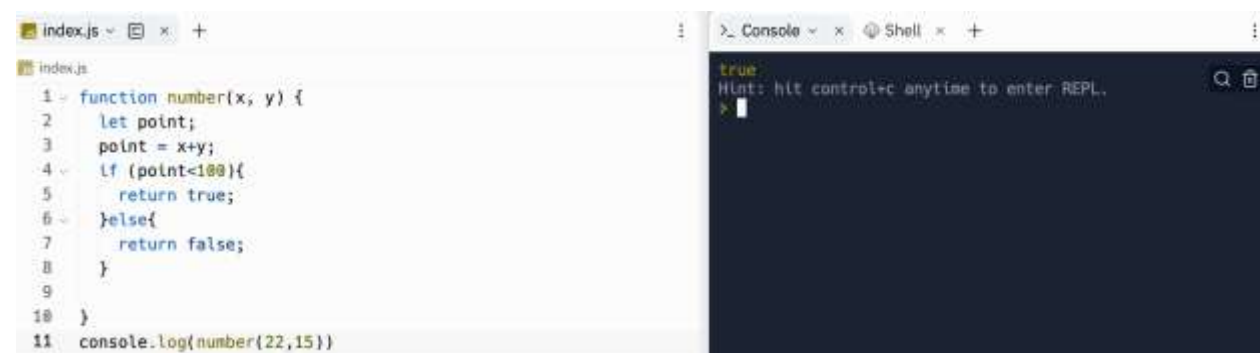
Less Than 100?

Given two numbers, return `true` if the sum of both numbers is less than 100.

Otherwise return `false`. **Examples**

lessThan100(22, 15) → true

// 22 + 15 = 37



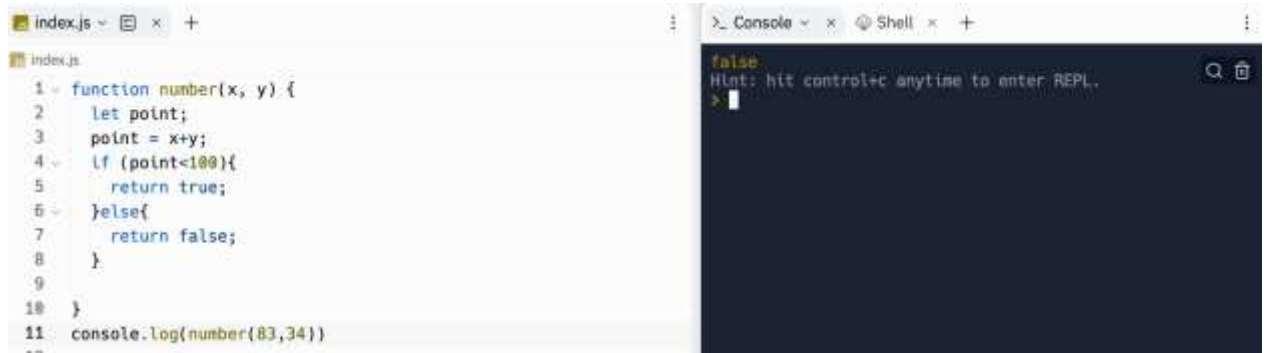
The screenshot shows a code editor with a file named 'index.js'. The code defines a function 'number(x, y)' that checks if the sum of x and y is less than 100 and returns a boolean value. The console shows the output of the function call.

```
index.js
1 function number(x, y) {
2   let point;
3   point = x+y;
4   if (point<100){
5     return true;
6   }else{
7     return false;
8   }
9 }
10
11 console.log(number(22,15))
```

Console: true
Hint: hit control+c anytime to enter REPL.

`lessThan100(83, 34) → false`

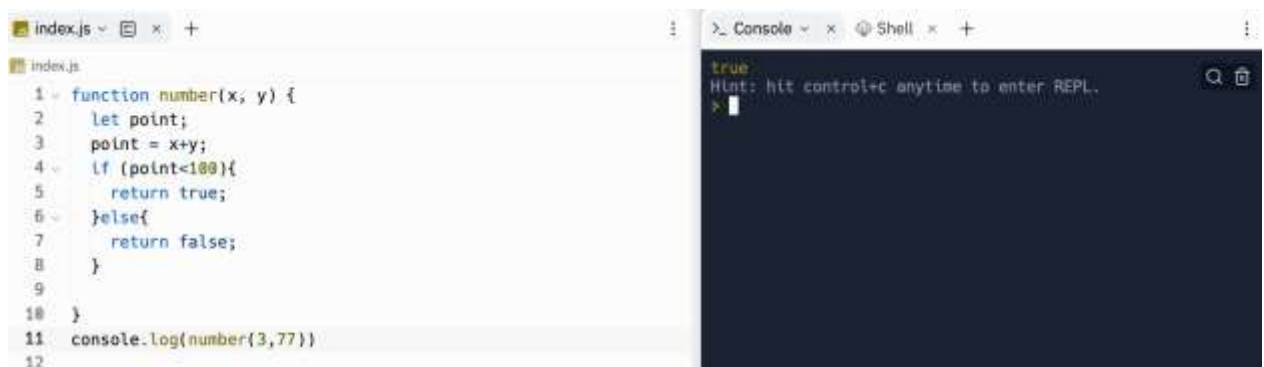
`// 83 + 34 = 117`



The screenshot shows a code editor with a file named `index.js`. The code defines a function `number(x, y)` that calculates the sum of `x` and `y` into a variable `point`. It then checks if `point` is less than 100. If true, it returns `true`; otherwise, it returns `false`. The function is called with `number(83, 34)` and the result is logged to the console. The console output shows `false`, indicating that the sum of 83 and 34 (which is 117) is not less than 100.

```
1 function number(x, y) {  
2   let point;  
3   point = x+y;  
4   if (point<100){  
5     return true;  
6   }else{  
7     return false;  
8   }  
9 }  
10  
11 console.log(number(83,34))
```

`lessThan100(3, 77) → true`



The screenshot shows the same code editor and console as the previous one, but with the function called with `number(3, 77)`. The console output now shows `true`, indicating that the sum of 3 and 77 (which is 80) is less than 100.

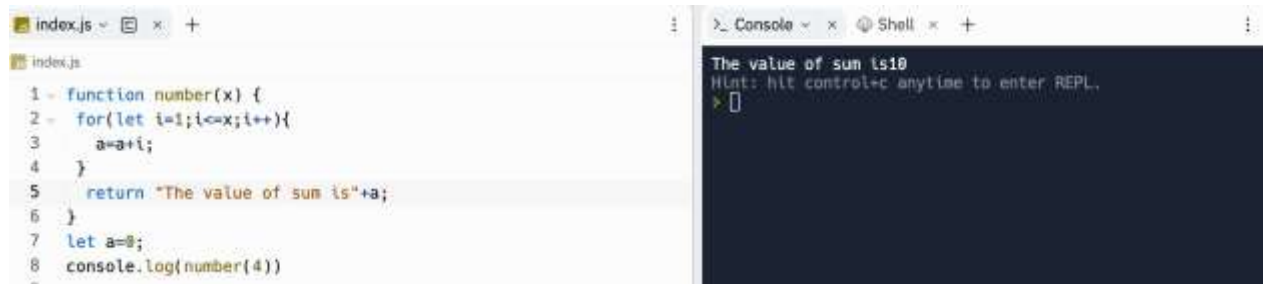
```
1 function number(x, y) {  
2   let point;  
3   point = x+y;  
4   if (point<100){  
5     return true;  
6   }else{  
7     return false;  
8   }  
9 }  
10  
11 console.log(number(3,77))
```

Add up the Numbers from a Single Number

Create a function that takes a number as an argument. Add up all the numbers from 1 to the number you passed to the function. For example, if the input is 4 then your function should return 10 because $1 + 2 + 3 + 4 = 10$.

Examples

`addUp(4)` → 10

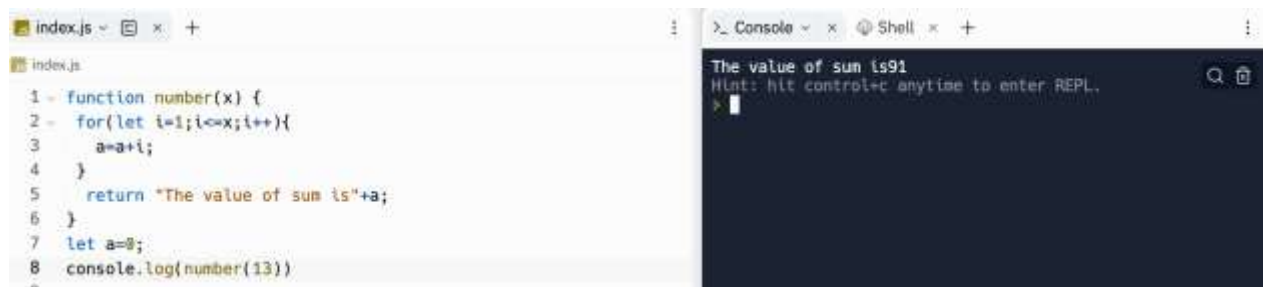


The screenshot shows a code editor with a file named `index.js`. The code defines a function `number(x)` that calculates the sum of integers from 1 to `x` using a `for` loop. It then calls `console.log(number(4))`. The adjacent console window displays the output: "The value of sum is 10".

```
1 - function number(x) {  
2 -   for(let i=1; i<=x; i++){  
3 -     a=a+i;  
4 -   }  
5 -   return "The value of sum is "+a;  
6 - }  
7 - let a=0;  
8 - console.log(number(4))
```

The value of sum is 10
Hint: hit control+c anytime to enter REPL.
>

`addUp(13)` → 91

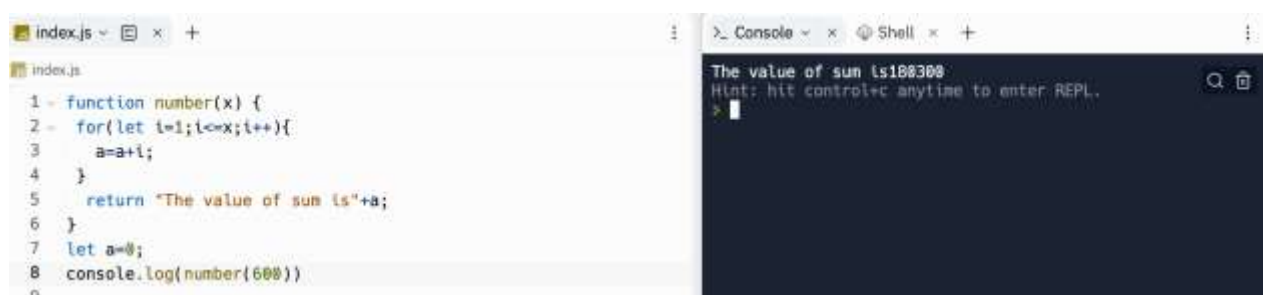


The screenshot shows the same code editor with `index.js`, but the function call is now `console.log(number(13))`. The console window displays the output: "The value of sum is 91".

```
1 - function number(x) {  
2 -   for(let i=1; i<=x; i++){  
3 -     a=a+i;  
4 -   }  
5 -   return "The value of sum is "+a;  
6 - }  
7 - let a=0;  
8 - console.log(number(13))
```

The value of sum is 91
Hint: hit control+c anytime to enter REPL.
>

`addUp(600)` → 180300



The screenshot shows the code editor with `index.js` where the function call is `console.log(number(600))`. The console window displays the output: "The value of sum is 180300".

```
1 - function number(x) {  
2 -   for(let i=1; i<=x; i++){  
3 -     a=a+i;  
4 -   }  
5 -   return "The value of sum is "+a;  
6 - }  
7 - let a=0;  
8 - console.log(number(600))
```

The value of sum is 180300
Hint: hit control+c anytime to enter REPL.
>

Notes

Expect any positive number between 1 and 1000.

Oddish vs. Evenish

Create a function that determines whether a number is **Oddish** or **Evenish**. A number is **Oddish** if the sum of all of its digits is odd, and a number is **Evenish** if the sum of all of its digits is even.

If a number is **Oddish**, return `"Oddish"`. Otherwise, return `"Evenish"`.

For example, `oddishOrEvenish(121)` should return `"Evenish"`, since $1 + 2 + 1 =$

4. `oddishOrEvenish(41)` should return `"Oddish"`, since $4 + 1 = 5$.

Examples

```
oddishOrEvenish(43) → "Oddish"  
// 4 + 3 = 7
```

```
// 7 % 2 = 1
```

```
oddishOrEvenish(373) → "Oddish"
```

```
// 3 + 7 + 3 = 13
```

```
// 13 % 2 = 1
```

```
oddishOrEvenish(4433) → "Evenish"
```

```
// 4 + 4 + 3 + 3 = 14
```

```
// 14 % 2 = 0
```




Any Prime Number in Range

Create a function that returns `true` if there's at least one prime number in the given range

(`n1` to `n2` (inclusive)), `false` otherwise. **Examples**

```
primeInRange(10, 15) → true //
```

Prime numbers in range: 11, 13

```
primeInRange(62, 66) → false
```

```
// No prime numbers in range.
```

```
primeInRange(3, 5) → true
```

```
// Prime numbers in range: 3, 5
```

Notes

- `n2` is always greater than `n1`.
- `n1` and `n2` are always positive.
- 0 and 1 aren't prime numbers.

Left Shift by Powers of Two

The left shift operation is similar to multiplication by powers of two.

Sample calculation using the left shift operator (\ll):

$$10 \ll 3 = 10 * 2^3 = 10 * 8 = 80$$

$$-32 \ll 2 = -32 * 2^2 = -32 * 4 = -128$$

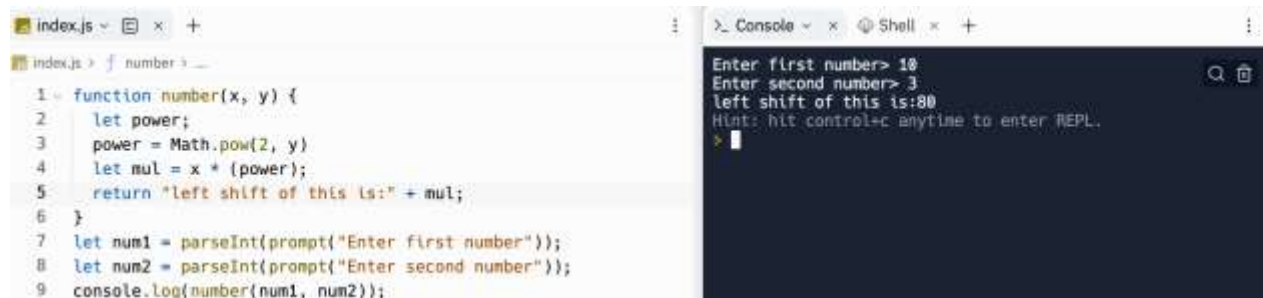
$$5 \ll 2 = 5 * 2^2 = 5 * 4 = 20$$

Write a function that mimics (without the use of \ll) the left shift operator and returns the result from the two given integers.

Examples

`shiftToLeft(5, 2) → 20`

`shiftToLeft(10, 3) → 80`



`shiftToLeft(-32, 2) → -128`

`shiftToLeft(-6, 5) → -192`

`shiftToLeft(12, 4) → 192`

`shiftToLeft(46, 6) → 2944`

Notes

- There will be no negative values for the second parameter `y`.
- This challenge is more like recreating the left shift operation, thus, the use of the operator directly is prohibited.
- Alternatively, you can solve this challenge via recursion.



Convert a Number to Base-2

Create a function that returns a base-2 (binary) representation of a base-10 (decimal) string number. To convert is simple: ((2) means base-2 and (10) means base-10) $010101001_2 = 1 + 8 + 32 + 128$.

Going from right to left, the value of the most right bit is 1, now from that every bit to the left will be $\times 2$. The values of an 8 bit binary number are (256, 128, 64, 32, 16, 8, 4, 2, 1).

Examples

`binary(1) → "1"`

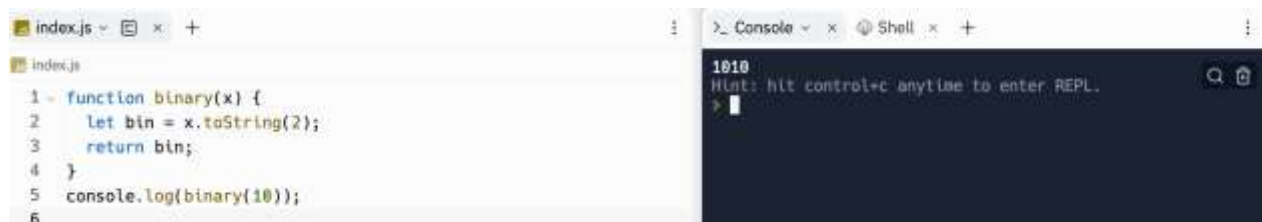
`// 1*1 = 1`

`binary(5) → "101"`

`// 1*1 + 1*4 = 5`

`binary(10) → "1010"`

`// 1*2 + 1*8 = 10`

A screenshot of a web browser's developer console. On the left, the 'index.js' file is open, showing a function `binary(x)` that converts a decimal number to a binary string using `x.toString(2)`. It also includes a call to `console.log(binary(10))`. On the right, the 'Console' tab is active, displaying the output `1010` and a hint: 'Hint: hit control+c anytime to enter REPL.'

```
index.js
1 - function binary(x) {
2   let bin = x.toString(2);
3   return bin;
4 }
5 console.log(binary(10));
6
```

Console

1010
Hint: hit control+c anytime to enter REPL.

Notes

- Numbers will always be below 1024 (not including 1024).
- The `&&` operator could be useful.
- The strings will always go to the length at which the most left bit's value gets bigger than the number in decimal.
- If a binary conversion for 0 is attempted, return "0".

