KINGSLAND UNIVERSITY

For-Loops

Repeating Blocks of Code

Table of Contents

Review

Conditional Statements Advanced

Nested Conditions

- An **`if...else`** statement can be **nested** within another **`if...else`** statement
  - Test one condition, followed by another

```
if (expression) {
  if (nested_expression)
    // Some code for execution
  else
    // Other code for execution
}
```

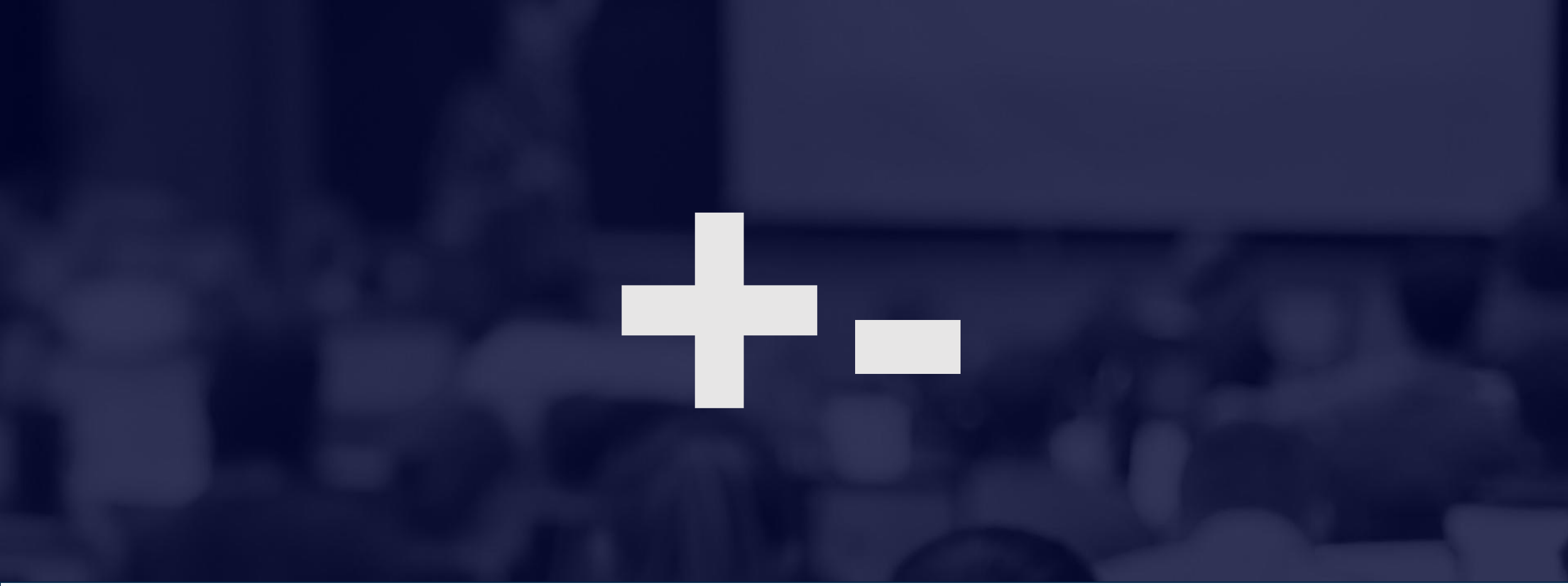
Conditional Operators

- **Logical operators** (such as **AND**, **OR**, **NOT**) are used to build complex logical conditions
- The logical operators in JavaScript are:
  - AND – **&&**
  - OR – **||**
  - Logical negation – **!**
  - Brackets – **( )**

Switch-Case

- Choosing among a list of possibilities
- Alternative to an **`if-else`** statement

```
switch (selector) {
  case someCase:
    statements;
    break;
  default:
    statements;
    break;
}
```



Increment and Decrement

Increment / Decrement Operators

- Increment (**++**) operator **increases** the value **by 1**
- Decrement (**--**) operator **decreases** the value **by 1**
- Can be used **prefix** and **postfix** form
  - Prefix: **`++i`**, **`--i`**
  - Postfix: **`i++`**, **`i--`**
- Both operators work only for numeric variables

Example: Increment / Decrement

- **Prefix** decrement

```
let a = 1;
console.log(--a); // 0
console.log(a);   // 0
```

Decreases the value and then prints it

- **Postfix** decrement

```
let a = 1;
console.log(a--); // 1
console.log(a);   // 0
```

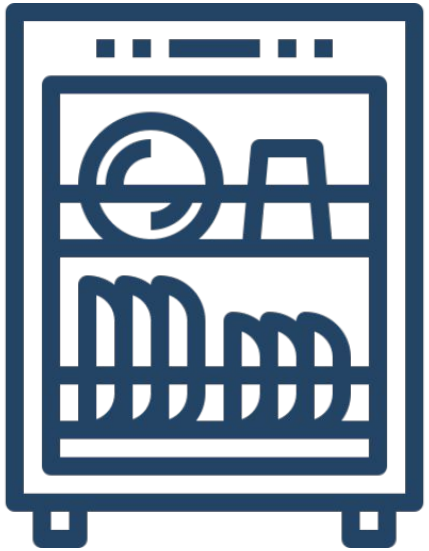
First prints the value and then decreases it

Loops: Introduction

For-Loops

Loops Example: Dishes

- Filling the dishwasher machine



For-Loop

Control Flow Statement

For-Loop: Example

- Print the numbers from 1 to 10:

Initial value

Condition

Step

```
for (let i = 1; i <= 10; i += 1) {
  console.log(i);
  console.log(i*i);
}
```

Loop body

For-Loop

- Allows code to be executed **repeatedly**
  - While certain **condition** is true

```
for (initialization; condition; step) {
    // Body of the for loop
}
```

- **Initialization** – initializes the loop variable
- **Condition** – logical exit condition
- **Step** – updates the loop variable

Problem: First N Numbers Sum

- Write a function, which **sums the numbers 1…N**:
  - Receive number **n** from the input
  - Sums all numbers from **1** to **n**
  - Prints the **sum** on the console as shown below:

5	→	**1+2+3+4+5=15**
6	→	**1+2+3+4+5+6=21**

Solution: First N Numbers Sum

```
function sumFirstNumbers(n) {
  let sum = 1;
  let result = '1';
  for (let i = 2; i <= n; i += 1) {
    result = result + '+' + i;
    sum += i;
  }
  result = result + '=' + sum;
  console.log(result);
}
```

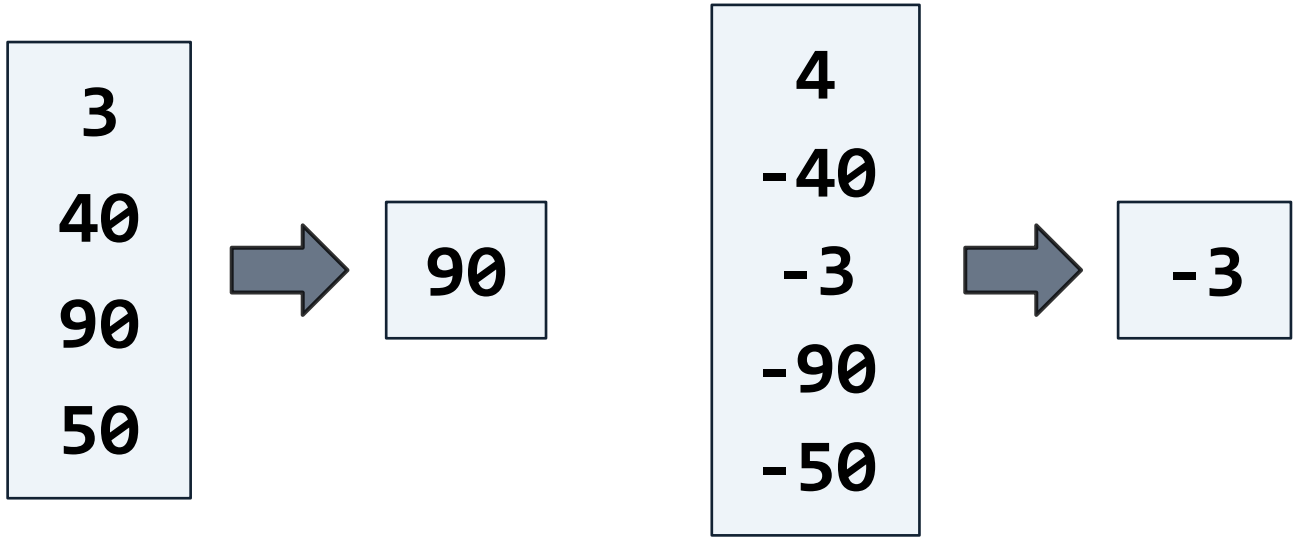
```
sumFirstNumbers(5);
```

```
sumFirstNumbers(7);
```

```
sumFirstNumbers(9);
```

Problem: Biggest Number

- Write a function to find **the biggest** among given **n** numbers:
  - Receives **n** (the **amount** of input numbers) and **n** numbers
  - Finds and prints the **biggest** number



Solution: Biggest Number

```
function biggestNumber(n, numbers) {
  let max = -Infinity;
  for (let i = 1; i <= n; i++) {
    let number = numbers.shift();
    if (number > max) {
      max = number;
    }
  }
  console.log(max);
}
```

```
biggestNumber(
  3, [40, 5, 90]);
```

```
biggestNumber(
  2, [-90, -40]);
```

Loops with a Step

For Loop with a Step

- The **increment** part in a for loop can either **increase** or **decrease** the value of a variable, even with a **step**

```
for (let i = 0; i < 10; i += 2)
  console.log(i);
```

Step = 2

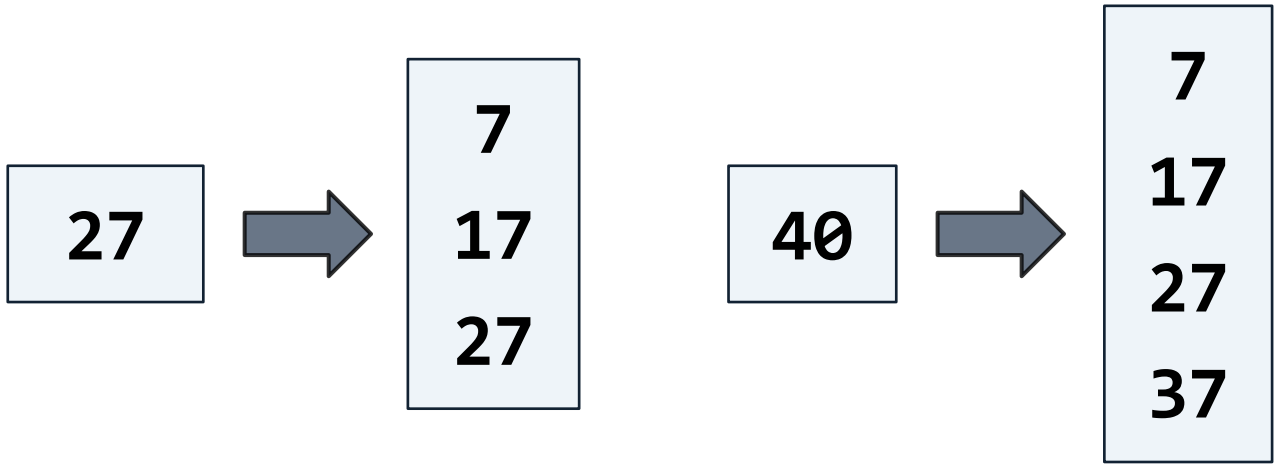
Always pay attention on the condition

```
for (let i = 10; i >= 0; i -= 2)
  console.log(i);
```

Step = -2

Problem: Numbers Ending in 7

- Write a function to print **numbers ending in 7** in given range:
  - Receives a number **n**
  - Prints all numbers from **7** to **n**, ending in 7



Solution: Numbers Ending in 7

```
function numbersEndingInSeven(n) {
  for (let i = 7; i <= n; i += 10) {
    console.log(i);
  }
}
```

```
numbersEndingInSeven(27);
numbersEndingInSeven(40);
```

Problem: Exam Countdown

- Write a function to print a "**countdown to an exam**":
  - Receives an integer **d**: the count of days before an exam
  - For each day **d**…**1** prints: "**{currentDay} days before the exam**"
  - At the end prints: "`The exam has come`"

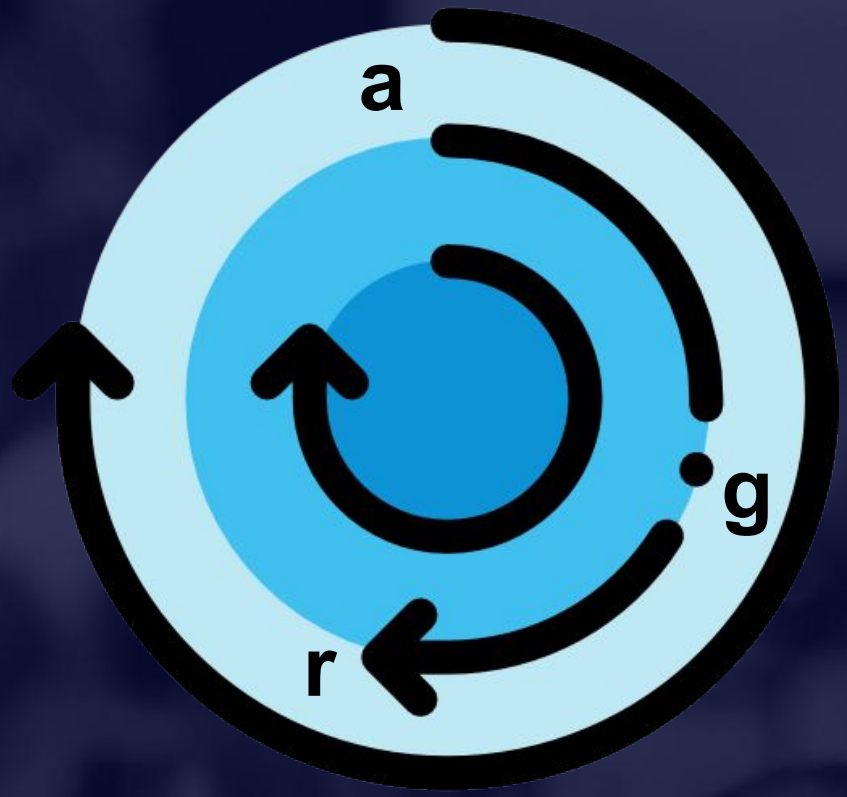
```
3
```

```
3 days before the exam
2 days before the exam
1 days before the exam
The exam has come
```

Solution: Exam Countdown

```
function examCountdown(days) {
  for (let i = days; i >= 1; i--) {
    console.log(`${i} days before the exam`);
  }
  console.log('The exam has come');
}
```

```
examCountdown(5);
examCountdown(10);
```



Iterating over Characters

The ASCII Table

- Computers can only understand numbers
  - **ASCII** code is the numerical representation of a character

Decimal	Hex	Html	Char
97	61	a	a
98	62	b	b

- **'a'** has the int value (ASCII code) of **97**
- **'b'** has the int value (ASCII code) of **98**
- Learn more at: https://ascii-code.com

- **Unicode** is more powerful character encoding standard: https://techterms.com/definition/unicode

Character Conversions

- Convert an **ASCII** / **Unicode** number into a character:

```
let letter = String.fromCharCode(65);
console.log(letter); // A
```

- Convert a character to its **ASCII** / **Unicode** code:

```
let letter = 'A';
let asciiValue =
  letter.charCodeAt(0);
console.log(asciiValue); // 65
```

Loop over Characters

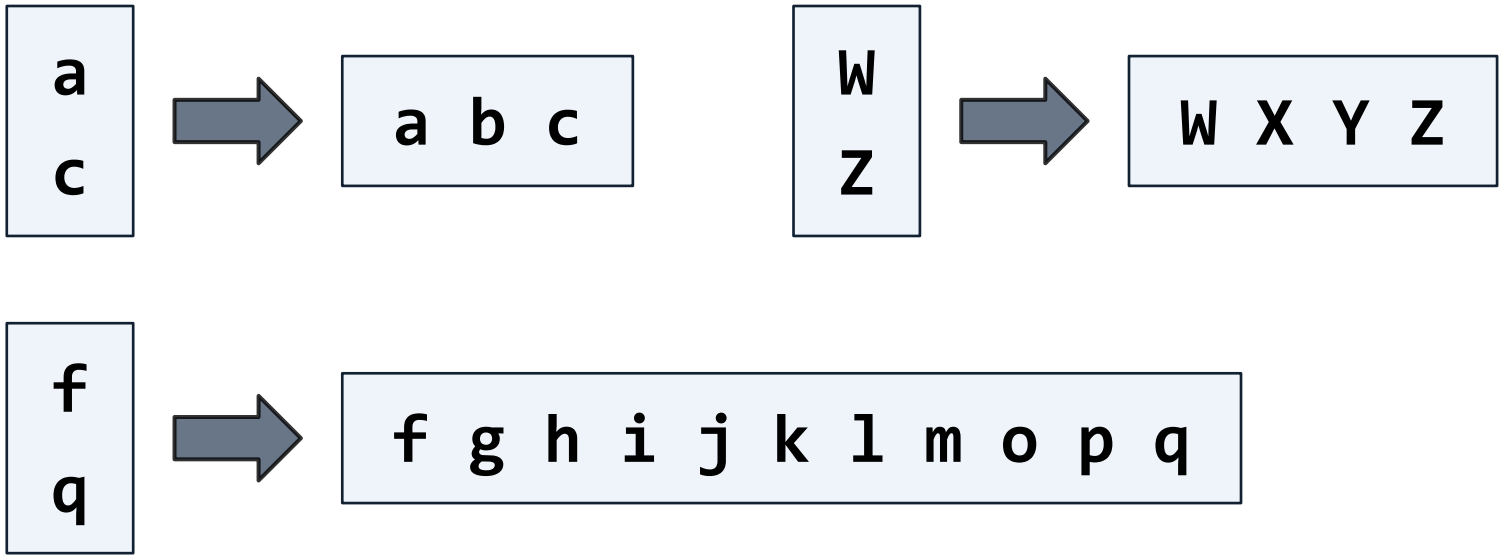
- We can **iterate over characters** using this code:

```
for (let i = 65; i <= 90; i++) {
  let letter = String.fromCharCode(i);
  console.log(letter);
}
```

```
A
B
C
D
...
Z
```

Problem: Latin Letters

- Write a function to print the **Latin letters in certain range**:
  - Receives **2 letters**, each on separate line
  - Prints all letters in the specified range



Solution: Latin Letters

```
function latinLetters(startChar, endChar) {
  let startValue = startChar.charCodeAt(0);
  let endValue = endChar.charCodeAt(0);
  let result = '';
  for (let i = startValue; i <= endValue; i++) {
    result += String.fromCharCode(i) + ' ';
  }
  console.log(result);
}
```

```
latinLetters('a', 'c');
```

```
latinLetters('b', 'e');
```

Infinite Loops

Repeating Code Infinitely

Infinite Loops

- Repeating a block of code an **infinite** number of times:

You can skip the initialization, condition and the increment

```
for ( ; ; ) {
  console.log("Infinite");
}
```

Infinite Loops in Programming

- Repeat certain logic **infinitely** (or until stopped)
- Used in **game development** for continuously drawing the game environment
- Used for **drawing animations**, frame after frame
- Used in **Web servers**: wait for clients and serve them infinitely

Problem: Sum Numbers Until 0

- Write a function to process **numbers** from the input parameters and **print their sum** until **0** is reached

```
5, 3, 2, 0
```

```
Sum = 5
Sum = 8
Sum = 10
Good bye
```

```
sumNumsUntil0([5, 3, 2, 0]);
Sum = 5
Sum = 8
Sum = 10
Good bye
```

Solution: Sum Numbers Until 0

```
function sumNumsUntil0(nums) {
  let sum = 0;
  for (;;) {
    let num = nums.shift();
    if (num == 0)
      break;
    sum += num;
    console.log(`Sum = ${sum}`);
  }
  console.log('Good bye');
}
```

```
sumNumsUntil0(
  [5, 3, 2, 0]);
```

Summary

- For-loops execute a block of code multiple times
- For-loop components:
  - Initialization
  - Condition
  - Step
  - Body

```
for (let i = 0; i < 9; i++) {
  console.log(i);
}
```

Questions?

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © Kingsland University – https://kingslanduniversity.com

KINGSLAND UNIVERSITY

THANK YOU