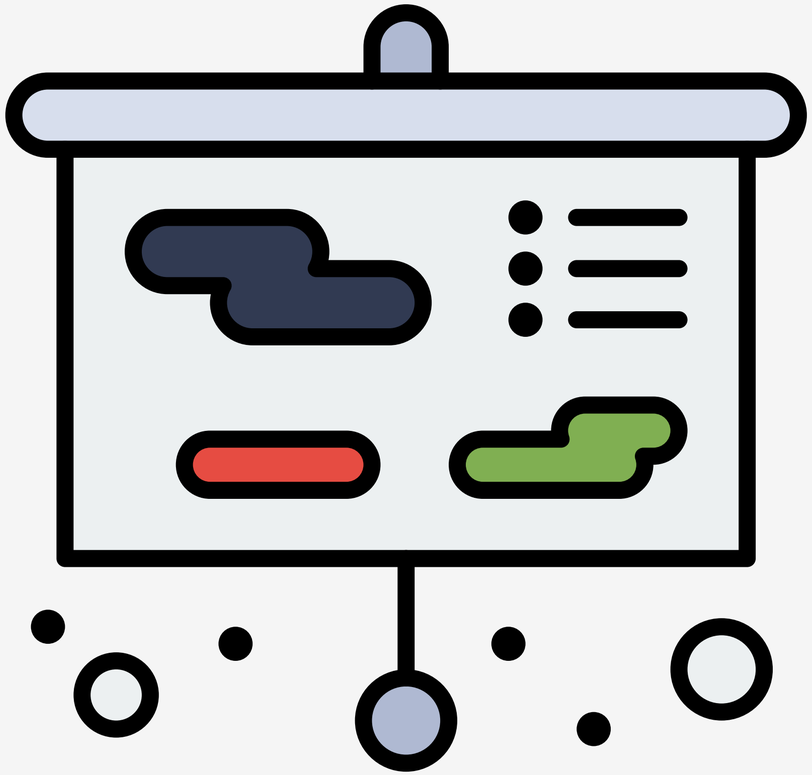
INTRODUCTION TO JAVASCRIPT

Lecture 5

TODAY'S TOPICS



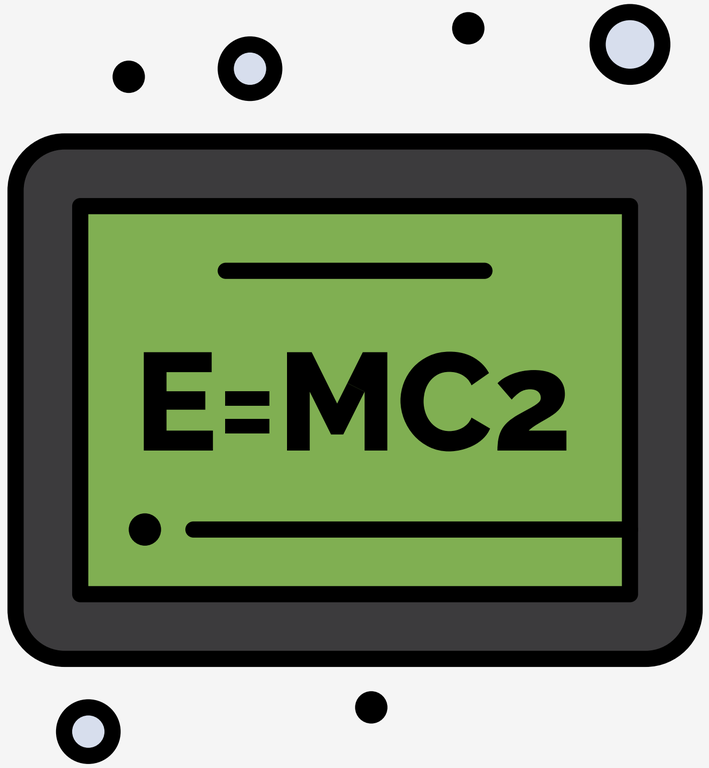
- Operators
- Conditional Statements
- Loops

ANNOUNCEMENTS

- Sign-in Sheet
- Recordings

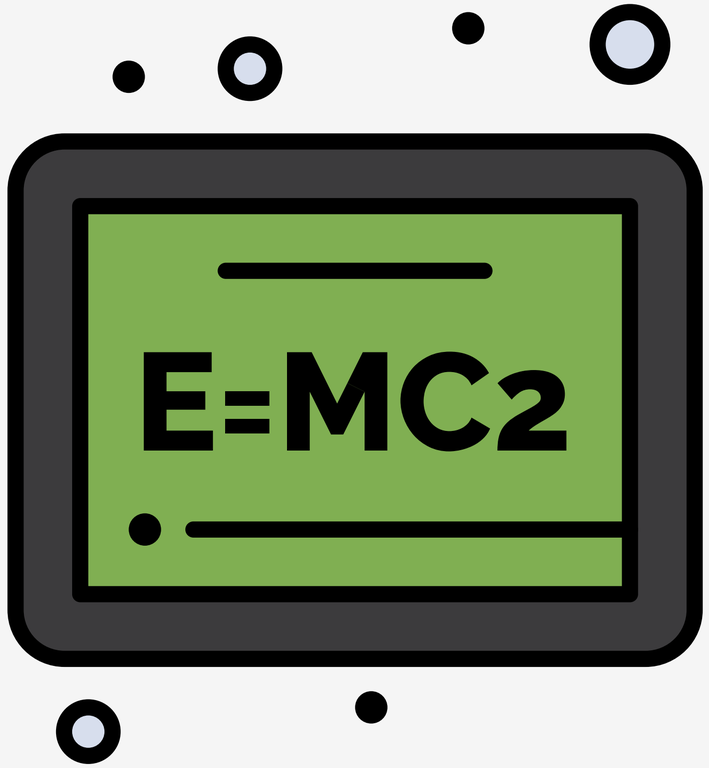
OPERATORS

JAVASCRIPT OPERATORS



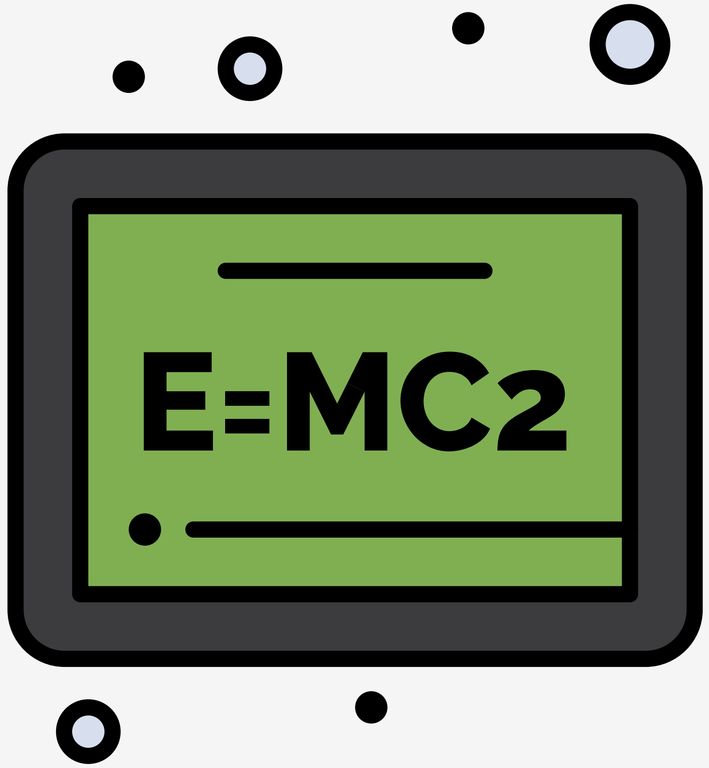
- Operators are words or symbols used to preform operations
- Operators with operands create expressions
- In JavaScript, there are unary, binary and ternary operators

JAVASCRIPT OPERATORS



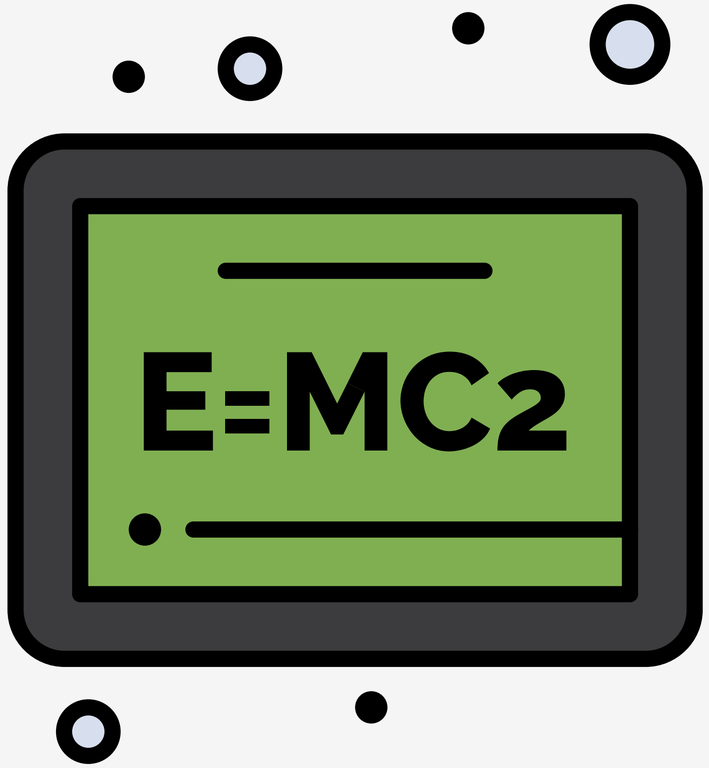
- Assignment Operators
- Comparison Operators
- Arithmetic Operators
- Logical Operators
- Conditional Operators
- Unary Operators

JAVASCRIPT OPERATORS



- Assignment Operators
- Comparison Operators
- Arithmetic Operators
- Logical Operators
- Conditional Operators
- Unary Operators

COMPARISON OPERATORS

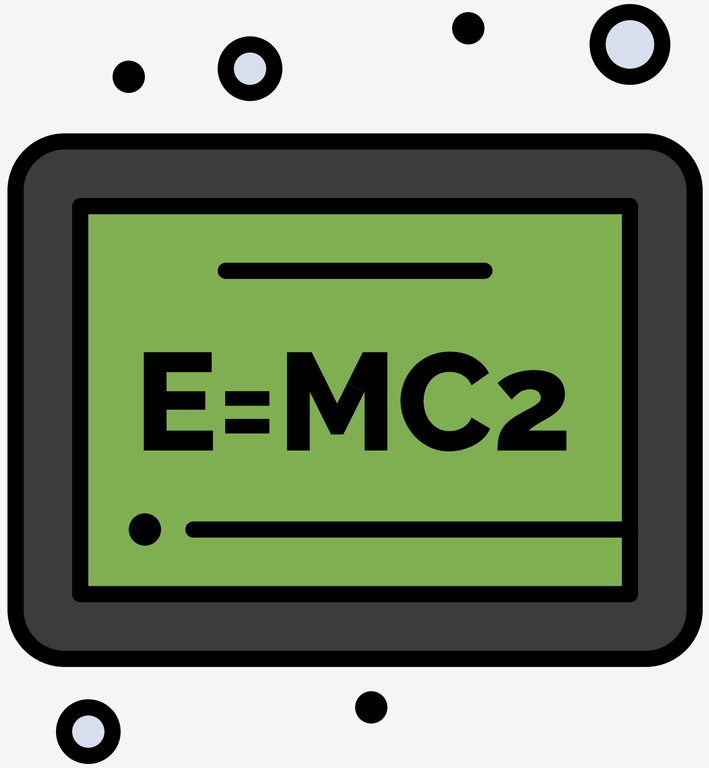


- Comparison operators are used to compare the values
- The equality operators are used to see if two equal or not equal
- The equality operators come in two flavours: strict and type-converting
- The type-converting (== and !=) will automatically convert values to be the same time
- The strict (=== and !==) will required value and data type to match

```
// equal

// type-converting
console.log(1 == 1)         // true
console.log('1' == 1)       // true

// strict Preferred!
console.log(1 === 1)        // true
console.log('1' === 1)      // false
```

```
// not equal

// type-converting
console.log(1 != 2)          // true
console.log(1 != '1')        // false

// strict Preferred!
console.log(1 !== 2)         // true
console.log(1 !== '1')       // true
```

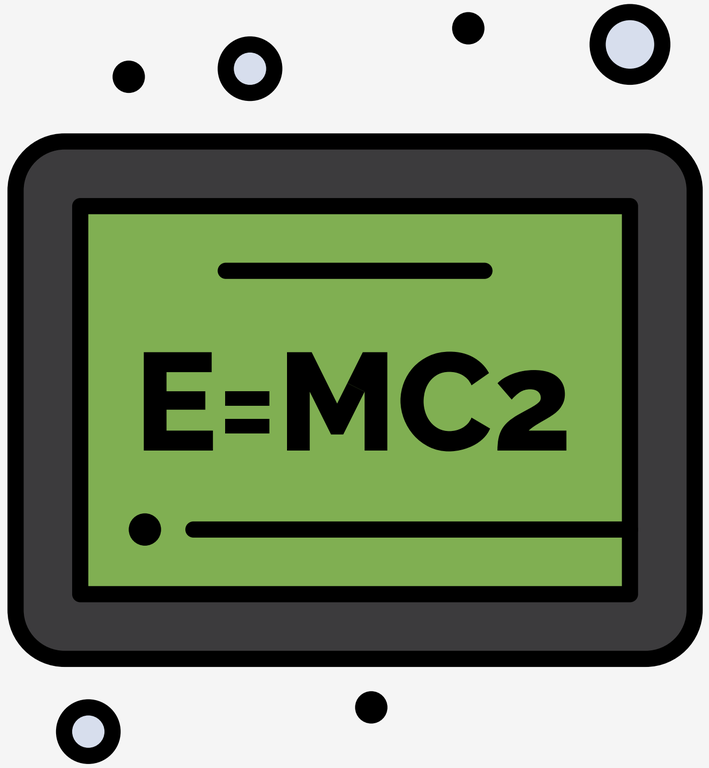
COMPARISON OPERATORS



- The relational operators are used to compare two values in relation to each other
- This includes:
  - less than (<)
  - greater than (>)
  - less than or equal to (<=)
  - greater than or equal to (>=)

```
// greater than
console.log(4 > 3)                  // true
console.log('banana' > 'apple')     // true

// less than
console.log(3 < 4)                  // true
console.log('apple' < 'banana')     // true

// less than or equal
console.log(3 <= 3)                 // true
```

LOGICAL OPERATORS



- Logical operators are used to alter or combine expression to create a complex condition
- Typically used with boolean values and will return boolean values
- There are three operators:
  - AND (&&)
  - OR (||)
  - NOT (!)

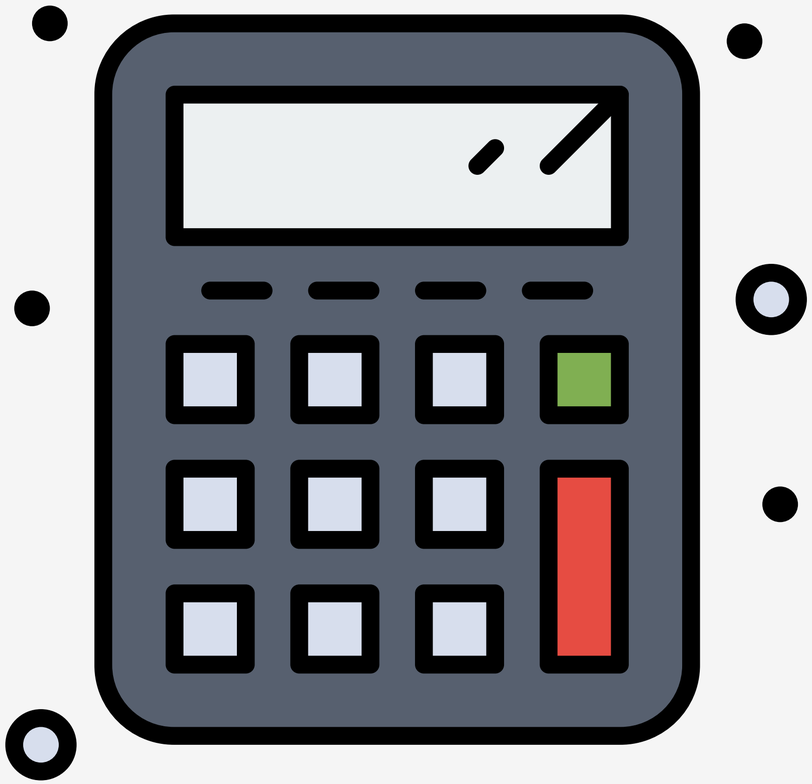
```
// AND Operator

// with booleans
console.log(true && true)      // true
console.log(true && false)     // false

// with strings and booleans
console.log('Cat' && true)     // true
console.log('Cat' && false)    // false
console.log('' && true)        // ''

// with strings
console.log('Cat' && 'Dog')    // 'Dog'
```

```
// OR Operator

// with booleans
console.log(true || true)      // true
console.log(true || false)     // true

// with strings and booleans
console.log('Cat' || true)     // 'Cat'
console.log(false || 'Dog')    // 'Dog'
console.log('' || true)        // true

// with strings
console.log('Cat' || 'Dog')    // 'Cat'
```

```
// NOT Operator

// with booleans
console.log(!true)        // false
console.log(!false)       // true

// with strings
console.log(!'Cat')       // false
console.log(!'')          // true
```

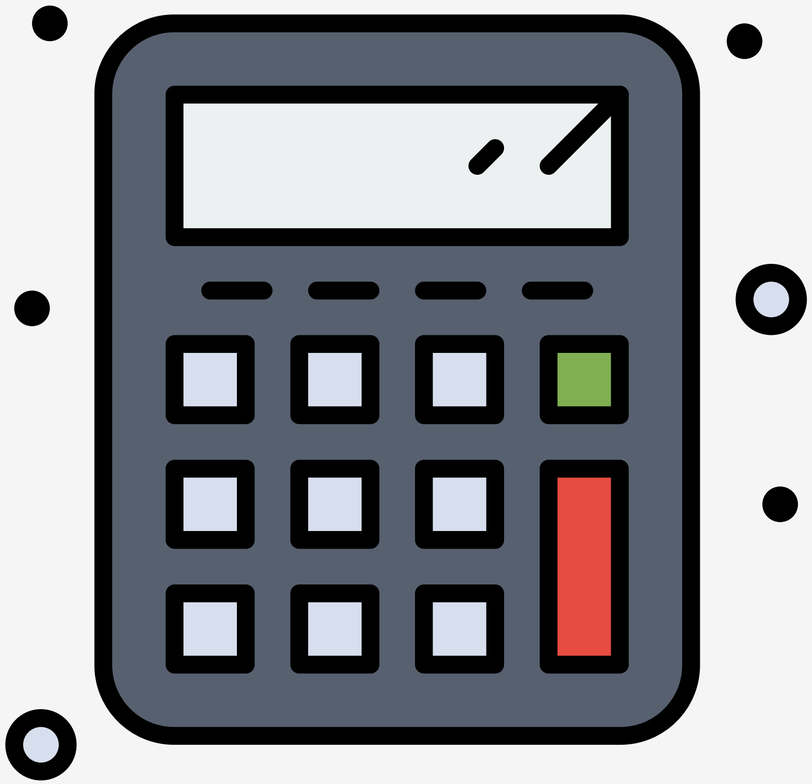
CONDITIONAL STATEMENTS

CONDITIONAL STATEMENTS



- Control the flow of a program by executing only when certain conditions are met.
- Two types: `if...else` statements and `switch` statements

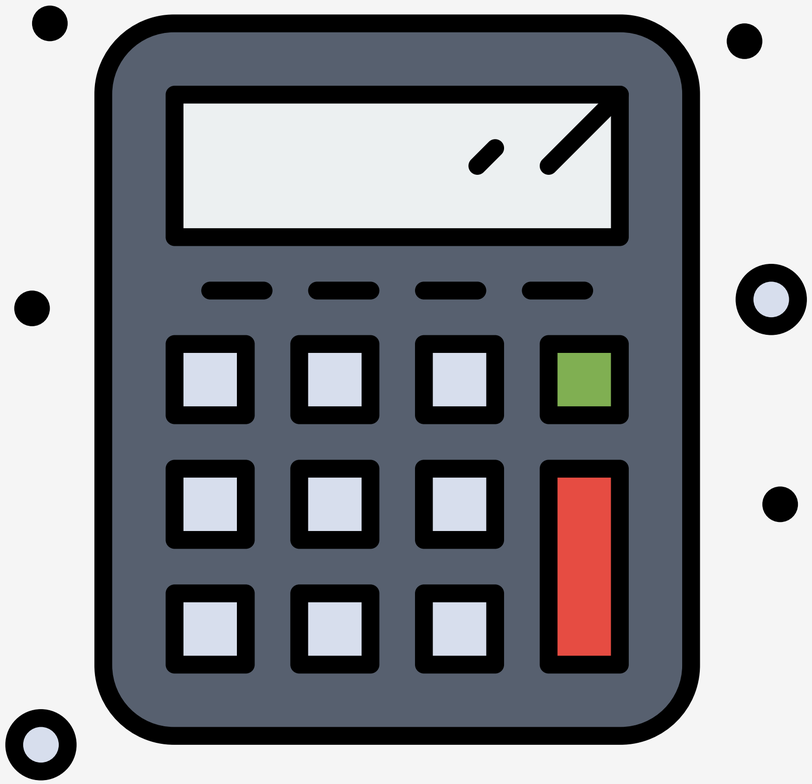
IF...ELSE STATEMENTS



- The `if` statement is the most basic conditional statement
- Consists of the `if` keyword, a condition, and a statement
- If the condition is truthy, the statement will execute

```
// declare the number variable
const number = 5

// Does number equal 5
if (number === 5) {
  // this block of code will execute
  console.log('Yes, number is equal to 5')
}

// Does number equal 6
if (number === 6) {
  // this block of code will not execute
  console.log('Yes, number is equal to 6')
}
```

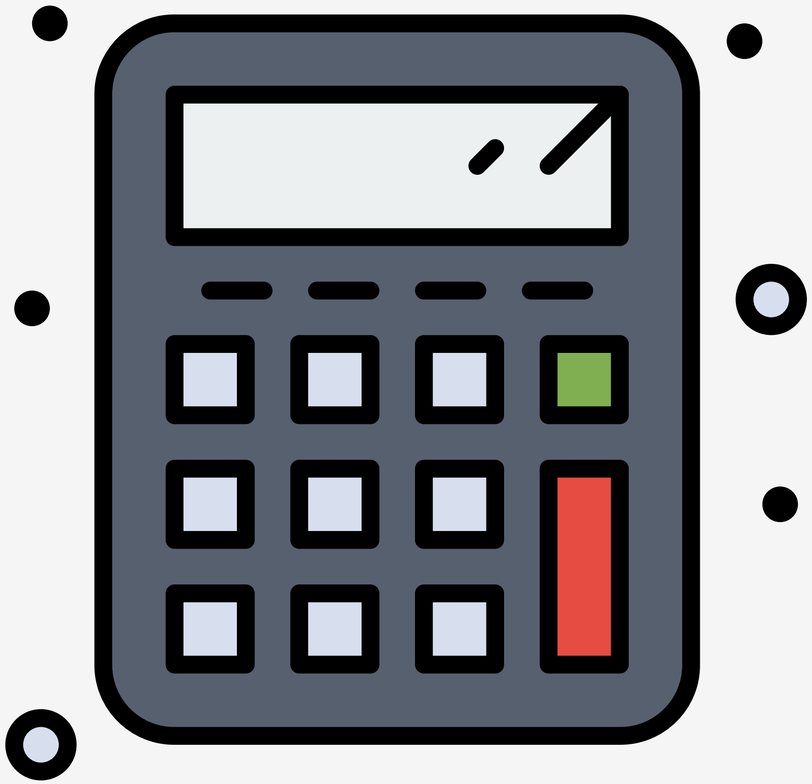
IF...ELSE STATEMENTS



- The `else` statement is use to execute an optional statement if the condition of the previous `if` statement is falsy
- It is not possible to have an `else` statement on its own

```
// declare the number variable
const number = 6

if (number === 5) {
  // this block of code will NOT execute
  console.log('Yes, number is equal to 5')
} else {
  // this block of code will execute
  console.log('No, number is NOT equal to 5')
}
```

IF...ELSE STATEMENTS

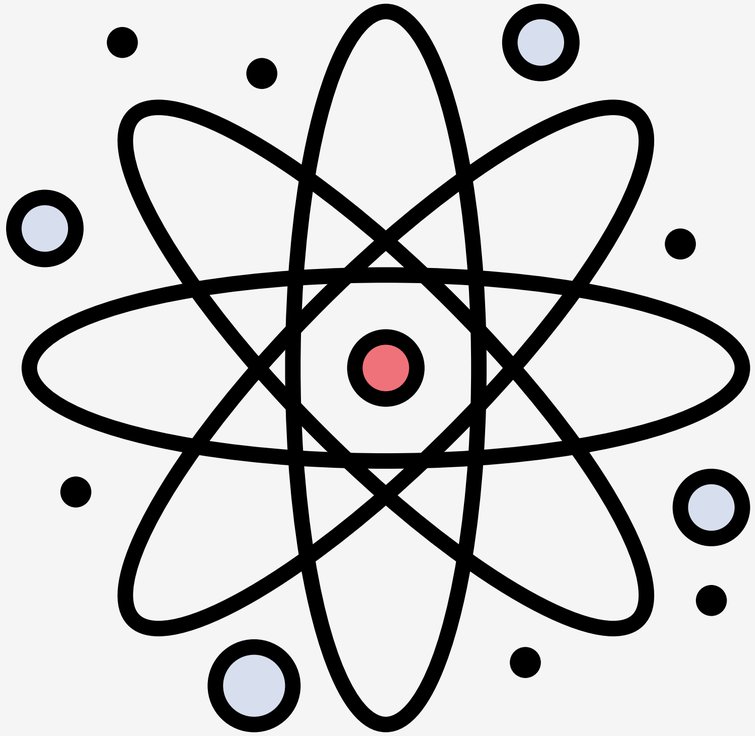


- Nested if...else statements can be use to check for multiple conditions
- The `else if` clause can be used as a shorthand

```
// declare the number variable
const number = 6

if (number === 5) {
  // this block of code will NOT execute
  console.log('Yes, number is equal to 5')
} else if (number === 6) {
  // this block of code will execute
  console.log('Yes, number is equal to 6')
} else if (number === 7) {
  // this block of code will NOT execute
  console.log('Yes, number is equal to 7')
}
```

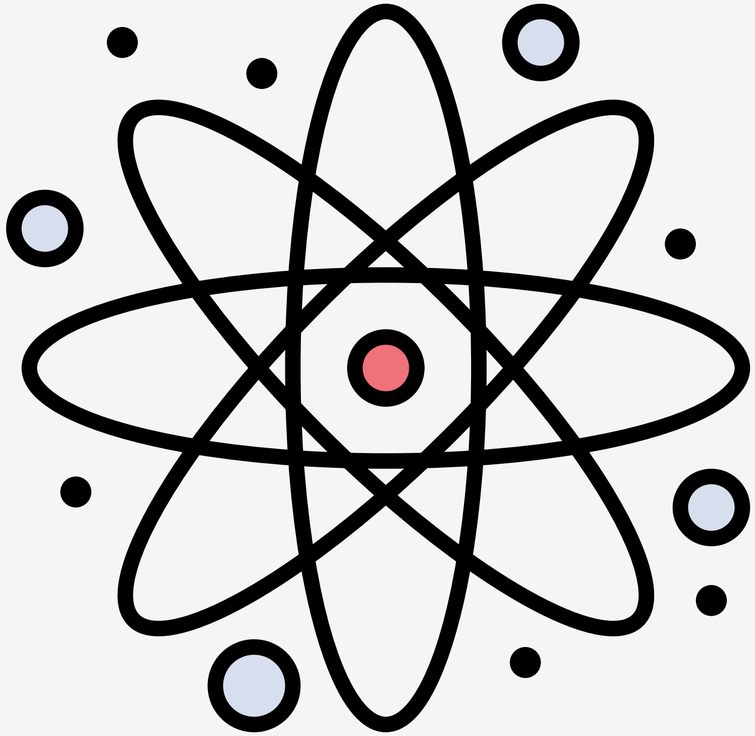
LOOPS

LOOPS



- Loops are statements that are used to repeat a block of code until a specified condition is met.
- JavaScript has following loops:
  - `for`
  - `for...of`
  - `for...in`
  - `while`

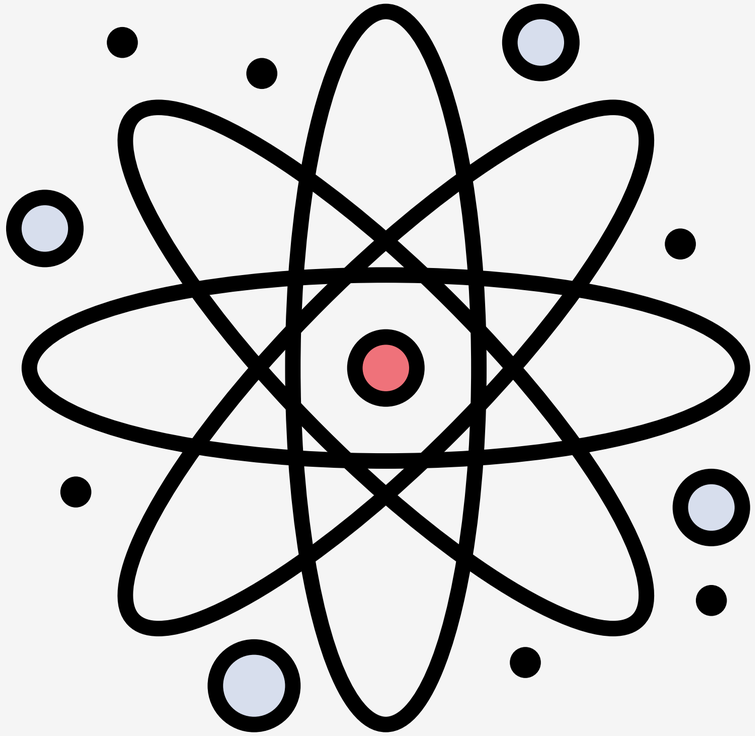
FOR LOOP



- The `for` loop is used when the number of iterations is knowable
- Consists of three expressions separated by semi-colons
    1. The initialization of the iterator
    2. The condition that is check before each loop to see if the loop should continue
    3. The iteration of the iterator

```
// will loop 5 times
for (let i = 0; i < 5; i++) {
  console.log(i) // Logs 0 to 4
}

// iterating over an array
const animals = ['cat', 'dog', 'mouse']

for (let i = 0; i < animals.length; i++) {
  // Logs all the animals in the array
  console.log(animals[i])
}
```

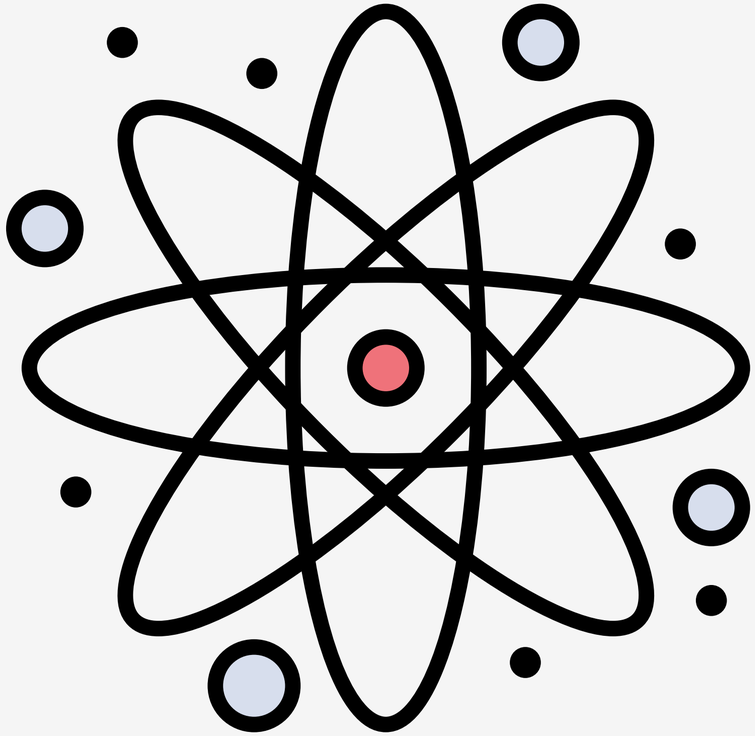
FOR...OF LOOP



- The `for...of` loop is used iterate over iterable objects, like strings and arrays
- The expression starts with the initialization of a variable, which hold the value of each item
- This followed by the `of` keyword
- The expression ends with the iterable object

```
// iterating over an array
const animals = ['cat', 'dog', 'mouse']

for (const animal of animals) {
  // Logs all the animals in the array
  console.log(animal)
}

// iterate over a string
const name = 'Ted Mosby'

for (const char of name) {
  // Logs each character of the name
  console.log(char)
}
```

FOR...IN LOOP

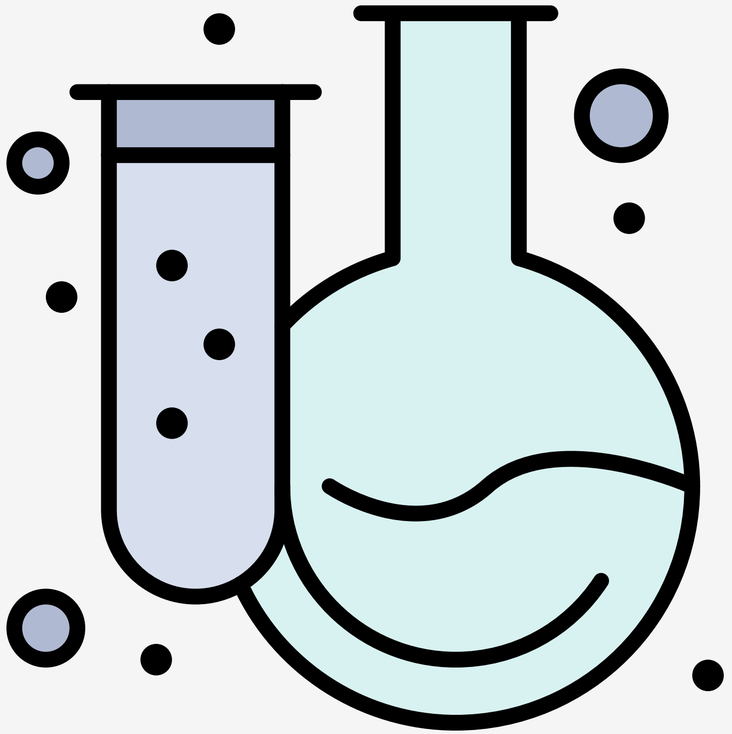


- The `for...in` loop is used iterate over properties of an object
- The expression starts with the initialization of a variable, which hold the key of each property
- This followed by the `in` keyword
- The expression ends with the object
- When retrieving values, bracket notation *MUST* be used

```
// iterate over properties
const sounds = {
  cow: 'moo',
  duck: 'quack',
  horse: 'nay'
}

for (const animal in sounds) {
  // Logs each animals' sound
  console.log(sounds[animal])
}
```

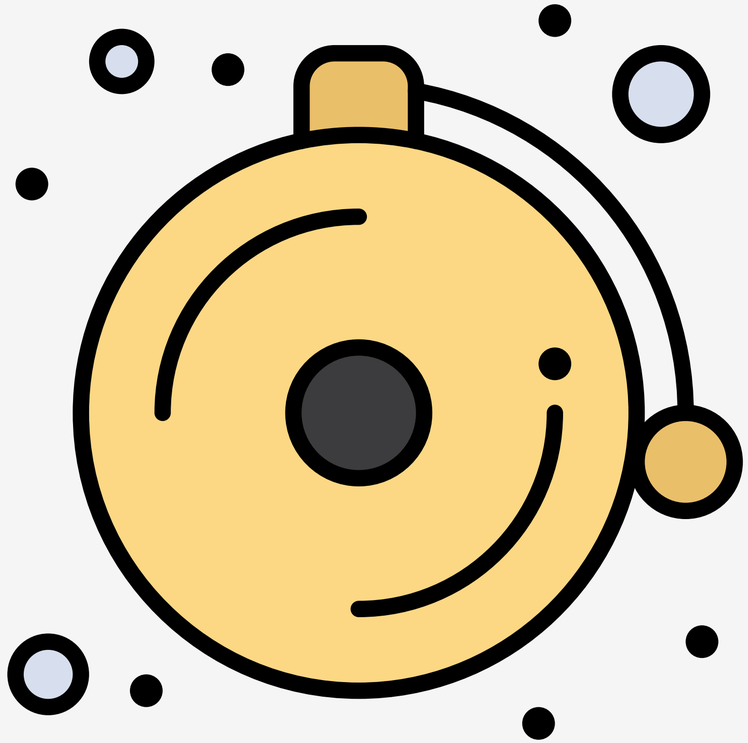
PRACTICE

LOOPING LIZARDS



- *FORK THE PEN!*
- Use loops and the `lizards()` function to create lizards of different colors
- Pass the color as a string to the `lizards()` function.
  Example: `lizards('red')`
- Do *NOT* alter the `colors` array
- Submit the URL to your pen
- *DUE:* **Thu. Sep. 26 @ 11:59 PM**

NEXT TIME...



- **Hands-on:** Loops
- **Exercise:** Looping Lyrics