Conditionals

If statement

An ***if statement*** executes a group of statements if a condition is true. Braces { } surround the group of statements. Good practice is to indent statements in braces using a consistent number of spaces. This material indents 3 spaces.

Construct 6.3.1: if statement.

```
if (condition) {
   // Statements to execute when condition is true
}
```

If-else statement

An ***if-else statement*** executes a block of statements if the statement's condition is true, and executes another block of statements if the condition is false.

Construct 6.3.2: if-else statement.

```
if (condition) {
   // statements to execute when condition is true
}
else {
   // statements to execute when condition is false
}
```

Using { } around if and else blocks

JavaScript does not require braces { } around if or else blocks with a single statement. Good practice is to always use braces, which results in more readable code that is less susceptible to logic errors.

```
// Braces not required around single statements
if (vote == "M")
   memberCount++;
else
   nonMemberCount++;
```

Comparison operators

If and if-else statements commonly use comparison operators. A comparison operator compares two operands and evaluates to a Boolean value, meaning either true or false.

Table 6.3.1: Comparison operators.

Comparison operator	Name	Example
`==`	Equality	`2 == 2 // true` `"bat" == "bat" // true`
`!=`	Inequality	`2 != 3 // true` `"bat" != "zoo" // true`
`===`	Identity	`2 === 2 // true` `"2" === 2 // false`
`!==`	Non-identity	`2 !== 2 // false` `"2" !== 2 // true`
`<`	Less than	`2 < 3 // true` `"bat" < "zoo" // true`
`<=`	Less than or equal	`2 <= 3 // true` `"bat" <= "bat" // true`
`>`	Greater than	`3 > 2 // true` `"zoo" > "bat" // true`
`>=`	Greater than or equal	`3 >= 2 // true` `"zoo" >= "zoo" // true`

When the equality operator `==` and inequality `!=` operator compare a number and a string, the string is first converted to a number and then compared. Ex: `3 == "3"` is true because "3" is converted to 3 before the comparison, and 3 and 3 are the same.

The identity operator `===` performs strict equality. Two operands are strictly equal if the operands' data types and values are equal. Ex: `3 === 3` is true because both operands are numbers and the same value, but `"3" === 3` is false because "3" is a string, and 3 is a number. The non-identity

operator `!==` is the opposite of the identity operator. Ex: `"3" !== "3"` is false because both operands are the same type and value, but `"3" !== 3` is true because "3" is a string, and 3 is a number.

Other comparison operators also convert a string to a number when comparing a string with a number. Ex: `2 < "12"` is true because 2 is less than the number 12. When comparing two strings, JavaScript uses Unicode values to compare characters. Ex: `"cat" <= "dog"` is true because "c" has a smaller Unicode value than "d".

A common error when comparing two values for equality is to use a single = instead of == or ===. Ex: `if (name = "Sue")` assigns `name` with "Sue" instead of asking if `name` equals "Sue".

What is Unicode?

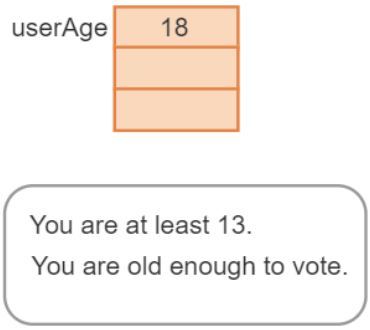
Unicode is a computing industry standard that assigns a unique number to characters in over one hundred different languages, including multiple symbol sets and emoji. The Unicode numbers for capital A-Z range from 65 to 90, and lowercase a-z range from 97 to 122.

Nested statements and else-if statement

If and else block statements can include any valid statements, including another if or if-else statement. An if or if-else statement that appears inside another if or if-else statement is called a nested statement.

```
let userAge = 18;
if (userAge <= 12) {
   console.log("Enjoy your early years.");
}
else {
   console.log("You are at least 13.");

   if (userAge >= 18) {
      console.log("You are old enough to vote.");
   }
   else {
      console.log("You are too young to vote.");
   }
}
```



A common situation is when several nested if-else statements are needed to execute one and only one block of statements. The else-if statement is an

alternative to nested if-else statements that produces an easier-to-read list of statement blocks.

In the example below, the `grade` variable is assigned with A, B, C, D, or F depending on the `score` variable. The code segment on the left uses nested if-else statements. The code segment on the right performs the same logic with else-if statements.

Figure 6.3.1: Nested if-else statements vs. else-if statements.

```
// Nested if-else statements
if (score >= 90) {
   grade = "A";
}
else {
   if (score >= 80) {
      grade = "B";
   }
   else {
      if (score >= 70) {
         grade = "C";
      }
      else {
         if (score >= 60) {
            grade = "D";
         }
         else {
            grade = "F";
         }
      }
   }
}
```

```
// else-if statements
if (score >= 90) {
   grade = "A";
}
else if (score >= 80) {
   grade = "B";
}
else if (score >= 70) {
   grade = "C";
}
else if (score >= 60) {
   grade = "D";
}
else {
   grade = "F";
}
```

Logical operators

JavaScript logical operators perform AND, OR, and NOT logic.

Table 6.3.2: Logical operators.

Logical operator	Name	Description	Example
`&&`	And	True if both sides are true	`(1 < 2 && 2 < 3)  // true`
`||`	Or	True if either side is true	`(1 < 2 || 2 < 0)  // true`
`!`	Not	True if expression is not true	`!(2 == 2)  // false`

Multiple `&&` and `||` conditions may be combined into a single complex condition. Ex: `(1 < 2 && 2 < 3 || 3 < 4)`. Complex conditions are evaluated from left to right, but `&&` has higher precedence than `||`, so `&&` is evaluated before `||`. Good practice is to use parentheses `()` around conditions that use `&&` and `||` to explicitly indicate the order of evaluation. Ex: `(a < 0 || a > 1 && b > 2)` is better expressed as: `(a < 0 || (a > 1 && b > 2))`.

Logic involving "not" can be difficult for humans to correctly read or understand. Ex: "Are you not hungry?" is more difficult for a human to understand than the equivalent "Are you satisfied?" Good practice is to avoid using the not operator when possible. Ex: `!(score > 10)` is better expressed as: `score <= 10`.

More conditionals

Truthy and falsy

A truthy value is a non-Boolean value that evaluates to `true` in a Boolean context. Ex: `if (18)` evaluates to `true` because non-zero numbers are truthy values. A falsy value is a non-Boolean value that evaluates to `false` in a Boolean context. Ex: `if (null)` evaluates to `false` because `null` is a falsy value.

Table Truthy values.

Example	Description
`if (32)`	Non-zero number
`if ("cat")`	Non-empty string

`if (myObject)`	Object variable
`if (myArray)`	Array variable

Falsy values.

Example	Description
`if (0)`	Zero
`if ("")`	Empty string
`if (NaN)`	Not a number
`if (undefined)`	Variable that has not been assigned a value
`if (null)`	No object value

Conditional (ternary) operator

The conditional operator allows developers to write concise conditional statements.
The ***conditional operator*** (or ***ternary operator***) has three operands separated by a question mark (`?`) and colon (`:`). If the `condition` evaluates to `true`, then the value of `expression1` is returned, otherwise the value of `expression2` is returned.

Construct 6.4.1: Conditional (ternary) operator.

```
condition ? expression1 : expression2
```

Switch statement

The switch statement is an alternative to writing multiple else-if statements. A switch statement compares an expression's value to several cases using strict equality (===) and executes the first matching case's statements. If no case matches, an optional default case's statements execute.

The break statement stops executing a case's statements and causes the statement immediately following the switch statement to execute. Omitting

the break statement causes the next case's statements to execute, even though the case does not match.

Construct 6.4.2: switch statement.

```
switch (expression) {
   case value1:
      // Statements executed when expression's value matches value1
      break;   // optional
   case value2:
      // Statements executed when expression's value matches value2
      break;   // optional

   // ...

   default:
      // Statements executed when no cases match
}
```

```
change = 10;
switch (change) {
   case 1:
      coin = "penny";
      break;
   case 5:
      coin = "nickel";
      break;
   case 10:
      coin = "dime";
      break;
   case 25:
      coin = "quarter";
      break;
   default:
      coin = "unknown";
}

console.log(coin);
```