



UNIT 01

LESSON 01.03



resolving nested quotes

string interpolation

const

resolving conflicting nested quotes

Strings can go in either double or single quotes, but nested quotes can cause problems.

1. Declare a string in single quotes with an internal single quote (apostrophe). Check the console. There's an error:

```
let song = 'Won't Get Fooled Again';  
// Unexpected identifier 't'
```

unexpected identified

An **unexpected identifier**, also known as an **unexpected token**, is a character that doesn't belong. The problem is, the apostrophe was mistaken for the end of the string, which makes 'Won' the entire string--and everything else 'unexpected'.

mixing-and-matching quotes

If you have an inner single quote (apostrophe), avoid conflicts by wrapping the string in double quotes.

2. Resolve the nested quote conflict by switching to double quotes:

```
let song = "Won't Get Fooled Again";  
console.log(song); // Won't Get Fooled Again
```

escaping quotes

Instead of mixing-and-matching quotes, you can "escape quotes" with a backslash, which causes the punctuation to be ignored.

3. Declare another variable with single quotes (apostrophes) nested inside single quotes, and then fix the error by escaping the apostrophes:

```
// let song2 = 'Don't Stop Believin''; // error
let song2 = 'Don\'t Stop Believin\'';
console.log(song2);
```

string interpolation

Long string concatenation can get hard to read, with all its quote and plus-signs. A cleaner alternative is **string interpolation**, which doesn't have any quotes or plus signs. To convert concatenation to interpolation, follow these steps:

- Delete all quotes and plus signs.
- Wrap everything in a pair of backticks ````.
- Put variables in curly braces, preceded by a dollar sign: **`${variable}`**

4. Do a string concatenation and then switch to string interpolation:

```
// string concatenation
let firstName = 'Bob';
let greeting = 'Hi, ' + firstName + '! How are you?';
console.log(greeting); // Hi, Bob! How are you?

// string interpolation
greeting = Hi, ${firstName}! How are you?;
console.log(greeting); // Hi, Bob! How are you?
```

The advantage of string interpolation is more apparent when there are multiple variables and substrings.

5. Declare a few more variables and make a longer, more complex concatenation:

```
let petName = "Fluffy";
let age = 3;
let food = "tuna";

let catGreeting = "Meow! My name is " + petName + "! I am " + age + "
years old. My favorite food is " + food + ".";

console.log(catGreeting);
// Meow! My name is Fluffy! I am 3 years old. My favorite food is
tuna.
```

This concatenation has four pairs of quotes and six plus signs, making it hard to read and write.

6. Refactor the above as string interpolation:

```
catGreeting = `Meow! My name is ${petName}. I am ${age} years old. My
favorite food is ${food}`;
```

```
console.log(catGreeting);  
// Meow! My name is Fluffy! I am 3 years old. My favorite food is  
tuna.
```

line breaks and tabs

With string concatenation, hitting enter to get a line break is ignored, as are tabs.

7. Do a string concatenation with line breaks and tabs:

```
let item1 = 'eggs';  
let item2 = 'bread';  
let item3 = 'milk';  
let shoppingList = 'Shopping List:' +  
    item1 +  
    item2 +  
    item3;  
console.log(shoppingList);  
// Shopping List:eggsbreadmilk
```

8. Try it with string interpolation. Now, hitting enter gives you a line break in the output. Tab works too:

```
shoppingList = `Shopping List:  
    ${item1}  
    ${item2}  
    ${item3}`;  
console.log(shoppingList);  
/*  
Shopping List:  
    eggs  
    bread  
    milk  
*/
```

The line breaks and tabs can be done with regular concatenation:

- `\n` in string concatenation gives a line break.
- `\t` in string concatenation gives a tabs.

9. Try the shopping list with regular string concatenation, including the line breaks and tabs. It can be done, but is a pain to write, so feel free to copy-paste this one:

```
shoppingList = "Shopping List: \n\t" + item1 + "\n\t" + item2 + "\n\t"  
+ item3;  
console.log(shoppingList);  
/*
```

```
Shopping List:
  eggs
  bread
  milk
*/
```

const means constant

- A **variable** is a named value that can *vary* (change).
- A **constant** is a named value that *cannot* be changed.

Constants, by convention, have UPPERCASE names, but an all-caps name alone does not make for a constant. Below, **let MOON_DIAMETER** is uppercase, but you can still change the value.

10. Declare an uppercase variable with **let**:

```
let MOON_DIAMETER = 2159.1;
console.log(MOON_DIAMETER); // 2159.1
MOON_DIAMETER *= 10; // multiply by 10
console.log(MOON_DIAMETER); // 21591
```

To declare a true constant, use the **const** keyword. As its name implies, **const** makes true constants. If you try to change it, you get an error.

11. Declare a true constant with **const**:

```
const DOZEN = 12;
DOZEN = 13; // Error: Assignment to constant variable
```

12. You cannot change a const, so it gives you an error. Comment it out:

```
// DOZEN = 13;
```

13. Try redeclaring DOZEN as a **var**:

```
var DOZEN = 13; // SyntaxError: Identifier DOZEN has already been
declared.
```

14. It gives an error, so comment it out.

```
// var DOZEN = 13;
```

- **End Lesson 01.03**
- **Next: 01.03 Lab**
- **Lesson 01.04**