

Loops and Debugging



Overview

```
1  /*
2    - loops
3    - while loop
4    - for loop
5    - break, continue keywords
6
7    - debugging
8    - interpreting a stack trace/error message
9    - debugging a failing test
10   - using debugger
11  */
12
13
14
```



while loop

```
1  /* a while loop requires three elements:
2     1. the while keyword
3     2. a conditional expression that evaluates to a boolean value
4     3. a block of code
5
6     while (conditional) {
7         // block of code
8     }
9
10    the block of code will run over and over as long as the conditional
11    expression evaluates to true
12 */
13
14
```



count is 3
count is 2
count is 1

while loop

```
1  let count = 3;  
2  
3  while (count >= 1) {  
4    console.log('count is', count);  
5    count--;  
6  }  
7  
8  
9  
10  
11  
12  
13  
14
```



while loop

```
1  while (false) {  
2    console.log('this line of code will never run');  
3  }  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14
```

while loop

```
1 while (true) {  
2   console.log('this line will run forever');  
3   // (or until the machine running the code runs out of memory)  
4 }  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14
```

[illegible]

[illegible]



for loop

```
1  /* a for loop requires three elements:
2     1. the for keyword
3     2. three optional expressions
4     3. a block of code
5
6     for (initialization; condition; final-expression) {
7         // block of code
8     }
9
10    the block of code will run over and over until the condition
11    evaluates to false
12 */
13
14
```




i is: 1
i is: 2
i is: 3

for loop

```
1  /* the initialization is run first, and only once. it's often used to
2     define a counter variable */
3
4  /* then, before every iteration, the condition is checked to see if it's
5     true - if it is, the for loop will run another iteration */
6
7  /* then, after each iteration, the final expression is run */
8
9  for (let i = 1; i <= 3; i++) {
10     console.log('i is:', i);
11 }
12
13
14
```



for loop

i is: 5
i is: 4
i is: 3
i is: 2
i is: 1

```
1 // loop in either direction
2 for (let i = 5; i >= 1; i--) {
3   console.log('i is:', i);
4 }
```

5
6
7
8
9
10
11
12
13
14



for loop

i is: 100
i is: 200
i is: 300

```
1 // can increment by any number
2 for (let i = 100; i <= 300; i += 100) {
3   console.log('i is:', i);
4 }
```

5
6
7
8
9
10
11
12
13
14



for loop

a
b
c
d
e
f
g

```
1  // use for loops to iterate through a string
2  let letters = 'abcdefg';
3
4  for (let i = 0; i < letters.length; i++) {
5    let currentLetter = letters[i];
6    console.log(currentLetter);
7  }
8
9
10
11
12
13
14
```



continue keyword

i is: 1
i is: 2
i is: 4
i is: 5

```
1  // the continue keyword will cause the loop to skip to the next iteration
2  for (let i = 1; i <= 5; i++) {
3    if (i === 3) {
4      continue;
5    }
6
7    console.log('i is:', i);
8  }
9
10
11
12
13
14
```



count is 5
count is 3
count is 1

continue keyword

```
1 // the continue keyword also works in while loops
2 let count = 5;
3
4 while (count >= 1) {
5   if (count % 2 === 0) {
6     count--;
7     continue;
8   }
9
10  console.log('count is', count);
11  count--;
12 }
13
14
```



break keyword

```
1  // the break keyword breaks out of the loop permanently
2  let myGrade = 'A';
3
4  while (true) {
5      myGrade += '+';
6
7      if (myGrade.length === 3) {
8          break;
9      }
10 }
11
12 console.log(myGrade);
13
14
```



break keyword

i is 10
i is 9
i is 8
i is 7

```
1  // the break keyword also works in for loops
2  for (let i = 10; i >= 1; i--) {
3      console.log('i is', i);
4
5      if (i === 7) {
6          break;
7      }
8  }
9
10
11
12
13
14
```





Debugging

```
1  /* Developers spend way more time debugging code than writing it! */
2
3  /* Let's review some good debugging techniques to help you write better
4     code more quickly throughout the remaining workshops */
5
6
7
8
9
10
11
12
13
14
```



Debugging — error messages

```
1  /* Let's start by considering
2     bugs that come from writing
3     invalid JavaScript code. */
4
5  /* The testem page in your
6     browser passes helpful error
7     messages to you if it
8     couldn't run your code as
9     written */
10
11 /* This ReferenceError means the
12    the code tried to reference a
13    variable called sum that was
14    never defined */
```

 **Jasmine** 2.4.1 Options

● × × ×

4 specs, 3 failures finished in 0.013s

Spec List | Failures

only0dds returns a number

ReferenceError: sum is not defined

ReferenceError: sum is not defined

at only0dds (http://localhost:7357/only-odds.js:11:7)


at Object.it (http://localhost:7357/only-odds.spec.js:8:)

at attemptSync (https://cdnjs.cloudflare.com/ajax/libs/)



Debugging — error messages

```
1  /* Note the stack trace below the
2     error */
3
4  /* The first at... line gives the
5     location where the error
6     occurred in only-odds.js: it
7     looks like the error happened
8     on line 11. */
9
10 /* This line number may not
11     always be accurate, but its
12     often a good place to start */
13
14 /* Google unfamiliar errors */
```

 **Jasmine** 2.4.1 Options

● × × ×

4 specs, 3 failures finished in 0.013s

Spec List | Failures

only0dds returns a number

ReferenceError: sum is not defined

ReferenceError: sum is not defined

at only0dds (http://localhost:7357/only-odds.js:11:7)

at Object.it (http://localhost:7357/only-odds.spec.js:8:)

at attemptSync (https://cdnjs.cloudflare.com/ajax/libs/)



Debugging — failing tests

1 */* When your test is failing, you'll get an output that compares the value*
2 *your function returned against the expected value. */*
3
4



5
6
7 4 specs, 1 failure

finished in 0.007s

8 Spec List | Failures
9

10 onlyOdds returns the sum of all odd nums between the provided argument
11 and 0

12
13 Expected 55 to equal 25.

14 Error: Expected 55 to equal 25.



Debugging — failing tests

```
1  /* Note the plain-english explanation of what the test is looking for;  
2   that can help! */  
3  
4
```



4 specs, 1 failure

finished in 0.007s

Spec List | Failures

onlyOdds returns the sum of all odd nums between the provided argument
and 0

Expected 55 to equal 25.

Error: Expected 55 to equal 25.



Debugging — failing tests

1 /* If your test is failing with "undefined" outputs, makes sure you
2 are returning values from your functions, not console.logging them! */
3
4
5
6
7
8
9
10
11
12
13
14

● × × ×

4 specs, 3 failures

Spec List | Failures

onlyOdds returns a number

Expected 'undefined' to equal 'number'.

stack@http://cdnjs.cloudflare.com/ajax/libs/jasmine/2.4.1/jasmine.js:1577:26



Debugging — failing tests

```
1  /* It can also help to look directly at the code that defines how the test
2     is supposed to work. Look for a test in the *.spec.js file with the
3     same plain-language explanation, and with the same expect value. */
4
5     it ('returns the sum of all odd nums between the provided argument and 0', () => {
6         let returnedValue = onlyOdds(10);
7         expect(returnedValue).toEqual(25);
8     });
9
10 /* Now you can see that the test is passing in the number 10 to your
11    function. This can help you debug! */
12
13 /* All of the code inside of the tests, besides the line that starts with
14    expect, is plain-old JavaScript */
```




Debugging — debugger

```
1  /* To use debugger:
2     1. add the debugger keyword in your function in VSCode
3     2. invoke the function yourself */
4  function onlyOdds(num) {
5     debugger; // debugger keyword here
6     let sum = 0;
7     for (let i = num; i >= 1; i--) {
8         if (i % 2 === 1) {
9             sum += i;
10        }
11    }
12    return sum;
13 }
14 onlyOdds(3); // invoking the function ourselves here
```




debugger: Chrome Dev Tools

```
1  /* Open Chrome Dev Tools (right-click > Inspect > Console) */
2
3  /* Run your code in CodePen */
4
5  /* Your code will pause when it hits the debugger keyword */
6
7  /* Note the Scope panel on the right side of the screen. This
8     continuously updates the values assigned to every relevant variable. */
9
10 /* Select the    button to move your code forward one line */
11    
12 /* It's sometimes easier to use debugger when you call the function
13    directly through the console instead of letting tests call your code */
14
```



debugger: VSCode

1 /* Click the debugging option in VSCode:



2
3 followed by the green debug arrow at the
4 top-left of the screen:



5
6 Now, press  to see your code run line by line (press  to stop) */

7
8 /* Note the Variables panel on the left side of the screen. This
9 continuously updates the values assigned to every relevant variable. */

10
11 /* Especially helpful to debug control-flow bugs (like loops) */



Recap

```
1  /*
2    - loops
3    - while loop
4    - for loop
5    - break, continue keywords
6
7    - debugging
8    - interpreting a stack trace/error message
9    - debugging a failing test
10   - using debugger
11  */
12
13
14
```