Loops and Debugging

Overview

```
- loops
  - while loop
- for loop
  - break, continue keywords
debugging
  - interpreting a stack trace/error message
  - debugging a failing test
  - using debugger
```

```
/* a while loop requires three elements:
     1. the while keyword
     2. a conditional expression that evaluates to a boolean value
     3. a block of code
   while (conditional) {
     // block of code
   the block of code will run over and over as long as the conditional
   expression evaluates to true
```

```
let count = 3;
while (count >= 1) {
  console.log('count is', count);
  count--;
```

```
while (false) {
  console.log('this line of code will never run');
```

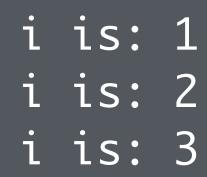
```
while (true) {
  console.log('this line will run forever');
  // (or until the machine running the code runs out o
}
```

```
this line will run forever
```

```
let count = 3;
   // have to make sure the conditional is eventually false!
   while (count >= 1) {
     console.log('count is', count);
14
```

```
count is 3
```

```
/* a for loop requires three elements:
     1. the for keyword
     2. three optional expressions
     3. a block of code
   for (initialization; condition; final-expression) {
     // block of code
   the block of code will run over and over until the condition
   evaluates to false
```

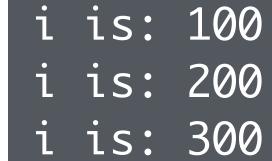


```
/* the initialization is run first, and only once. it's often used to
      define a counter variable */
   /* then, before every iteration, the condition is checked to see if it's
      true - if it is, the for loop will run another iteration */
   /* then, after each iteration, the final expression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



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

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



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

```
// use for loops to iterate through a string
let letters = 'abcdefg';
for (let i = 0; i < letters.length; i++) {
  let currentLetter = letters[i];
  console.log(currentLetter);
```



continue keyword

```
// the continue keyword will cause the loop to skip to the next iteration
for (let i = 1; i <= 5; i++) {
  if (i === 3) {
    continue;
  console.log('i is:', i);
```

continue keyword

```
// the continue keyword also works in while loops
let count = 5;
while (count >= 1) {
   if (count % 2 === 0) {
     count--;
     continue;
   console.log('count is', count);
   count--;
```

break keyword

```
// the break keyword breaks out of the loop permanently
let myGrade = 'A';
while (true) {
  myGrade += '+';
  if (myGrade.length === 3) {
    break;
console.log(myGrade);
```

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

break keyword

```
// the break keyword also works in for loops
for (let i = 10; i >= 1; i--) {
  console.log('i is', i);
 if (i === 7) \{
   break;
```



```
/* Developers spend way more time debugging code than writing it! */
/* Let's review some good debugging techniques to help you write better
   code more quickly throughout the remaining workshops */
```

Debugging — error messages

```
/* Let's start by considering
                                                                                    Options
                                            Jasmine 2.4.1
       bugs that come from writing
       invalid JavaScript code. */
                                            • X X X
                                           4 specs, 3 failures
                                                                             finished in 0.013s
   /* The testem page in your
       browser passes helpful error
                                           Spec List | Failures
       messages to you if it
       couldn't run your code as
                                           onlyOdds returns a number
       written */
                                            ReferenceError: sum is not defined
                                            ReferenceError: sum is not defined
   /* This ReferenceError means the
                                               at onlyOdds (http://localhost:7357/only-odds.js:11:7)
       the code tried to reference a
                                               at Object.it (http://localhost:7357/only-odds.spec.js:8:
       variable called sum that was
                                               at attemptSync (https://cdnjs.cloudflare.com/ajax/libs/
14
       never defined */
```

Debugging — error messages

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

```
Options
Samine 2.4.1
• X X X
4 specs, 3 failures
                                       finished in 0.013s
Spec List | Failures
onlyOdds returns a number
ReferenceError: sum is not defined
ReferenceError: sum is not defined
    at onlyOdds (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/
```

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

• • x •

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.

/* Note the plain-english explanation of what the test is looking for; that can help! */ • • × • 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.

/* If your test is failing with "undefined" outputs, makes sure you
are returning values from your functions, not console.logging them! */

```
• x x x

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
```

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

Debugging — debugger

```
/* To use debugger:
        1. add the debugger keyword in your function in VSCode
        2. invoke the function yourself */
   function onlyOdds(num) {
     debugger; // debugger keyword here
     let sum = 0;
     for (let i = num; i >= 1; i--) {
     if (i % 2 === 1) {
         sum += i;
     return sum;
14 onlyOdds(3); // invoking the function ourselves here
```

debugger: Chrome Dev Tools

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

debugger: VSCode

```
/* Click the debugging option in VSCode:
      followed by the green debug arrow at the
      top-left of the screen: DEBUG ▶ No C($
      Now, press 🔀 to see your code run line by line (press 🗖 to stop) */
   /* Note the Variables panel on the left side of the screen. This
      continuously updates the values assigned to every relevant variable. */
10
   /* Especially helpful to debug control-flow bugs (like loops) */
```

Recap

```
- loops
  - while loop
- for loop
  - break, continue keywords
debugging
  - interpreting a stack trace/error message
  - debugging a failing test
  - using debugger
```