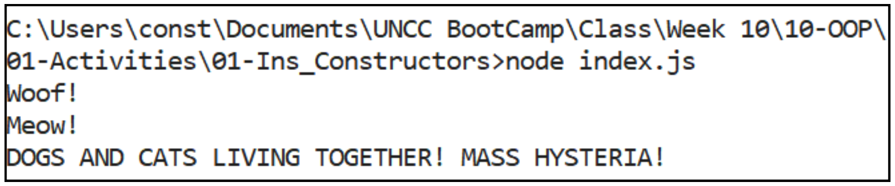
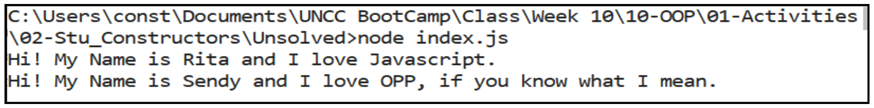
Constructors

```javascript
// Constructor function which can be used to create objects
            containing the properties "raining", "noise", and the "makeNoise()" function
function Animal(raining, noise) {
  this.raining = raining;
  this.noise = noise;
  this.makeNoise = () => {
    if (this.raining === true) {
      console.log(this.noise);
    }
  };
}

// Sets the variables "dogs" and "cats" to be animal objects and initializes them
            with raining and noise properties
const dogs = new Animal(true, "Woof!");
const cats = new Animal(false, "Meow!");

// Calling dogs and cats makeNoise methods
dogs.makeNoise();
cats.makeNoise();

// If we want, we can change an object's properties after they're created
cats.raining = true;
cats.makeNoise();

const massHysteria = (dogs, cats) => {
  if (dogs.raining === true && cats.raining === true) {
    console.log("DOGS AND CATS LIVING TOGETHER! MASS HYSTERIA!");
  }
};
massHysteria(dogs, cats);
```



```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP\
01-Activities\01-Ins_Constructors>node index.js
Woof!
Meow!
DOGS AND CATS LIVING TOGETHER! MASS HYSTERIA!
```

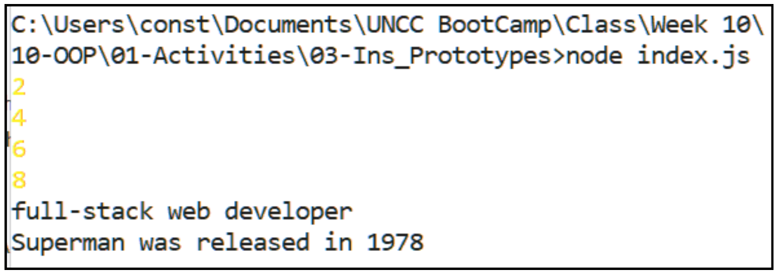
```javascript
// TODO: Create a constructor function called 'Developer'
            that takes in 'name' and 'tech'
function Developer(name, tech) {
  this.name = name;
  this.tech = tech;

  // TODO: Include a method called 'introduction()'
            that introduces the Developer with their name and favorite tech
  this.introduction = () => {
    console.log(`Hi! My Name is ${name} and I love ${tech}.`);
  };
}

// TODO: Create a new object using the 'Developer' constructor
const rita = new Developer("Rita", "Javascript");
const sendy = new Developer("Sendy", "OPP, if you know what I mean");

// TODO: Call the 'introduction()' method on the new object
rita.introduction();
sendy.introduction();
```



```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP\01-Activities
\02-Stu_Constructors\Unsolved>node index.js
Hi! My Name is Rita and I love Javascript.
Hi! My Name is Sendy and I love OPP, if you know what I mean.
```

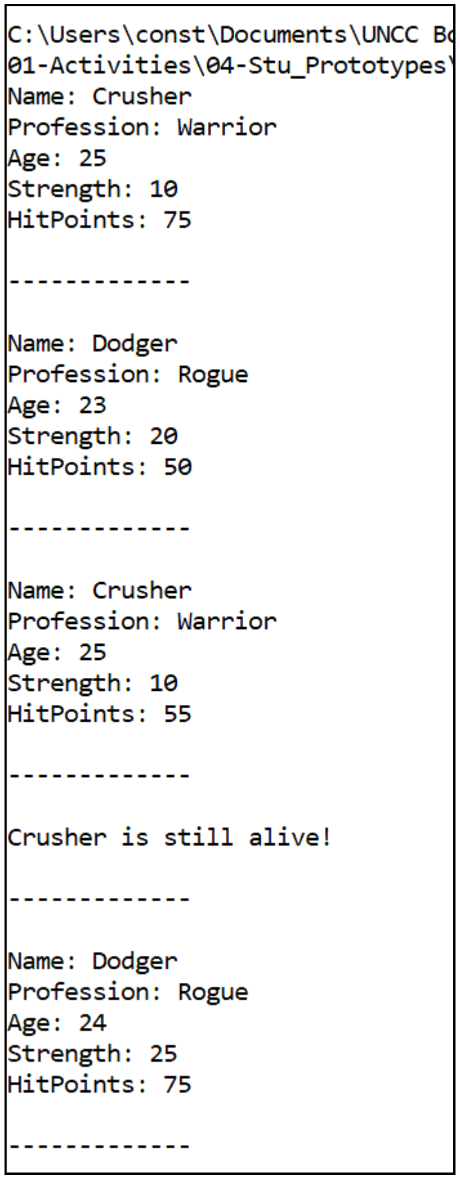
Prototype

```
// Array.prototype.forEach()
const myArray = [2, 4, 6, 8];
myArray.forEach((num) => console.log(num));

// String.prototype.toLowerCase()
const person = {
  name: "Eric",
  age: 28,
  occupation: "Full-Stack Web Developer",
};
console.log(person.occupation.toLowerCase());

// Prototype methods on constructor function
function Movie(title, releaseYear) {
  this.title = title;
  this.releaseYear = releaseYear;
}
const superman = new Movie("Superman", 1978);
Movie.prototype.logInfo = function () {
  console.log(`${this.title} was released in ${this.releaseYear}`);
};
superman.logInfo();
```



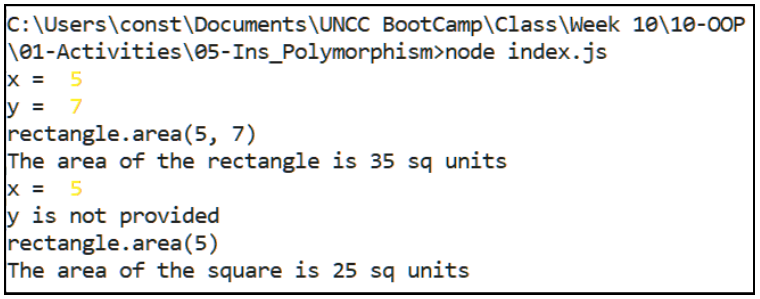
```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\
10-OOP\01-Activities\03-Ins_Prototypes>node index.js
2
4
6
8
full-stack web developer
Superman was released in 1978
```

```
// TODO: Add a comment describing what kind of function this is
function Character(name, type, age, strength, hitpoints) {
  this.name = name;
  this.type = type;
  this.age = age;
  this.strength = strength;
  this.hitpoints = hitpoints;
}

// TODO: Add a comment describing the purpose of `.prototype`
         in this method declaration
Character.prototype.printStats = function () {
  console.log(
    `Name: ${this.name}\nProfession: ${this.type}\nAge: ${this.age}
         \nStrength: ${this.strength}\nHitPoints: ${this.hitpoints}`
  );
  console.log('\n-------------\n');
};

// TODO: Add a comment describing the functionality of this method
Character.prototype.isAlive = function () {
  if (this.hitpoints > 0) {
    console.log(`${this.name} is still alive!`);
    console.log('\n-------------\n');
    return true;
  }
  console.log(`${this.name} has died!`);
  return false;
};

// TODO: Add a comment describing the functionality of this method
Character.prototype.attack = function (character2) {
  character2.hitpoints -= this.strength;
};

// TODO: Add a comment describing the functionality of this method
Character.prototype.levelUp = function () {
  this.age += 1;
  this.strength += 5;
  this.hitpoints += 25;
};
const warrior = new Character('Crusher', 'Warrior', 25, 10, 75);
const rogue = new Character('Dodger', 'Rogue', 23, 20, 50);
warrior.printStats();
rogue.printStats();
rogue.attack(warrior);

// TODO: Add a comment describing what you expect to see printed in the console
warrior.printStats();

// TODO: Add a comment describing what you expect to see printed in the console
warrior.isAlive();
rogue.levelUp();

// TODO: Add a comment describing what you expect to see printed in the console
rogue.printStats();
```



```
C:\Users\const\Documents\UNCC Bo
01-Activities\04-Stu_Prototypes\
Name: Crusher
Profession: Warrior
Age: 25
Strength: 10
HitPoints: 75

-------------


Name: Dodger
Profession: Rogue
Age: 23
Strength: 20
HitPoints: 50

-------------


Name: Crusher
Profession: Warrior
Age: 25
Strength: 10
HitPoints: 55

-------------


Crusher is still alive!

-------------


Name: Dodger
Profession: Rogue
Age: 24
Strength: 25
HitPoints: 75

-------------
```

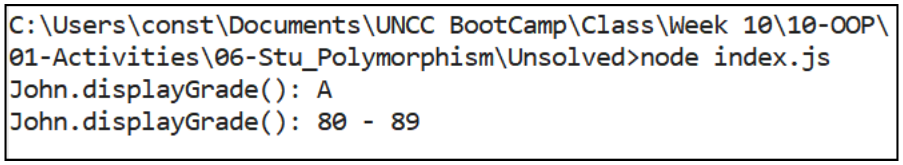
```
const OverloadDemo = function () {
  this.area = function (x, y) {
    console.log("x = ", x);
    if (!y) {
      console.log("y is not provided");
      return `\nThe area of the square is ${Math.pow(x, 2)} sq units`;
    }
    console.log("y = ", y);
    return `\nThe area of the rectangle is ${x * y} sq units`;
  };
};
const rectangle = new OverloadDemo();
console.log("rectangle.area(5, 7)", rectangle.area(5, 7));
console.log("rectangle.area(5)", rectangle.area(5, process.argv[2]));
```



```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP
\01-Activities\05-Ins_Polymorphism>node index.js
x =  5
y =  7
rectangle.area(5, 7)
The area of the rectangle is 35 sq units
x =  5
y is not provided
rectangle.area(5)
The area of the square is 25 sq units
```

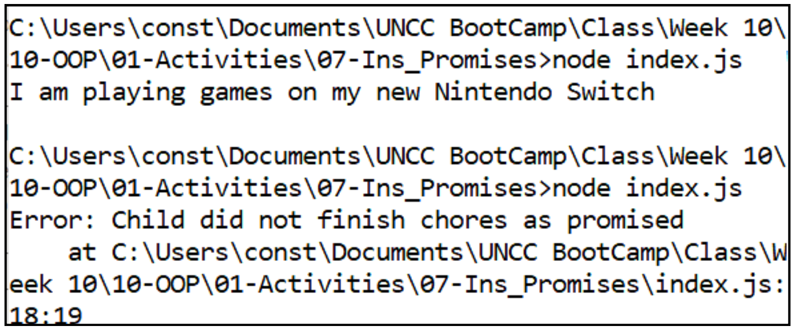
Polymorphism

```
// Helper function to evaluate if a number is within a given range
const inRange = (x, min, max) => (x - min) * (x - max) <= 0;
function Student(first, last, age) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;

  // Method that will simulate method overloading in JavaScript
  this.displayGrade = function (grade) {
    const input = grade;
    if (!input) {
      throw new Error("no grade provided");
    }
    let response;

    // Return a letter grade if a number grade was passed
    // Ex. 95 => 'A'
    if (typeof input === "number") {

     // TODO: Add logic here to return
            a single letter grade
     if (inRange(input, 90, 100)) {
       response = "A";
     }
     if (inRange(input, 80, 89)) {
       response = "B";
     }
     if (inRange(input, 70, 79)) {
       response = "C";
     }
     if (inRange(input, 60, 69)) {
       response = "D";
     }
     if (input < 60) {
       response = "F";
     }
     return response;
    }

  // Return a range if a letter grade was passed
  // Ex. 'A' => '90 - 100'
  if (typeof input === "string") {
    switch (input) {
      case "A":
        response = "90 - 100";
        break;
      case "B":
        response = "80 - 89";
        break;
      case "C":
        response = "70 - 79";
        break;
      case "D":
        response = "60 - 69";
        break;
      case "F":
        response = "0 - 59";
        break;
      default:
        response = "0";
        break;
    }
    return response;
    }
  };
}
const John = new Student("John", "Appleseed", "30");
console.log("John.displayGrade():", John.displayGrade(95));
console.log("John.displayGrade():", John.displayGrade("B"));
```



```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP\
01-Activities\06-Stu_Polymorphism\Unsolved>node index.js
John.displayGrade(): A
John.displayGrade(): 80 - 89
```

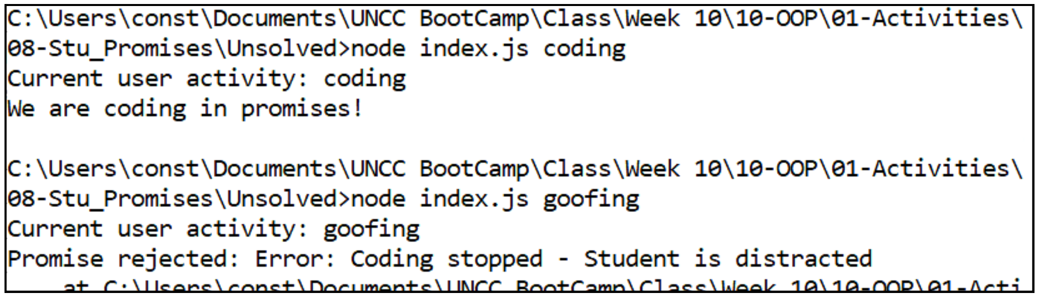
```
// Promise example using a real-life promise
// The real-life promise in this scenario is a child asking his parents
        for a Nintendo Switch
const choresDone = true;

// Promise
const willGetSwitch = new Promise((resolve, reject) => {

  // Check for a desireable outcome, if so resolve the promise
  if (choresDone) {
    const reward = {
      name: "Nintendo Switch",
      desired: true,
    };
    resolve(reward);

    // Otherwise reject the promise
  } else {
    const issue = new Error("Child did not finish chores as promised");
    reject(issue);
  }
});

// Another promise to call only if we get the reward
const playGames = (reward) => {
  const message = `I am playing games on my new ${reward.name}`;
  return Promise.resolve(message);
};
willGetSwitch
  .then(playGames)
  .then((resolved) => console.log(resolved))
  .catch((err) => console.error(err));
```



```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\
10-OOP\01-Activities\07-Ins_Promises>node index.js
I am playing games on my new Nintendo Switch

C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\
10-OOP\01-Activities\07-Ins_Promises>node index.js
Error: Child did not finish chores as promised
    at C:\Users\const\Documents\UNCC BootCamp\Class\W
eek 10\10-OOP\01-Activities\07-Ins_Promises\index.js:
18:19
```

Promises

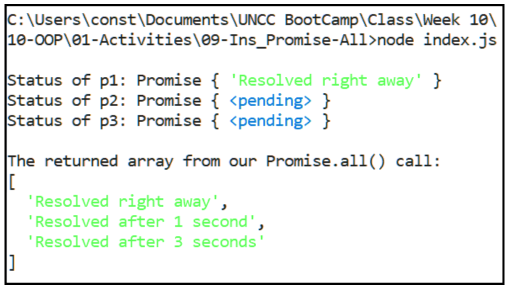
```
// Prompt the user to enter what they are currently doing
const userInput = process.argv[2];
console.log("Current user activity:", userInput);

// If the user does not enter anything, return an error message
if (!userInput) {
  console.error(
    "\nPlease enter your current activity\nUsage: `node index.js <activity>`"
  );
  process.exit();
}

// If the user enters anything other than the word 'coding',
        set 'studentDistracted' to 'true'
const studentDistracted = userInput !== "coding";

// TODO: Refactor the following to use promises
const practiceCoding = () =>
  new Promise((resolve, reject) => {
    if (studentDistracted) {
      reject(new Error("Coding stopped - Student is distracted"));
    }
    resolve("We are coding!");
  });

// TODO: Refactor to call 'practiceCoding()'
        and chain a 'then()' to log "We are coding in promises!" in the console
// TODO: Chain a 'catch()' to log "Promise rejected: " and the error
practiceCoding()
  .then(() => console.log("We are coding in promises!"))
  .catch((err) => console.error("Promise rejected:", err));
```



```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP\01-Activities\
08-Stu_Promises\Unsolved>node index.js coding
Current user activity: coding
We are coding in promises!

C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP\01-Activities\
08-Stu_Promises\Unsolved>node index.js goofing
Current user activity: goofing
Promise rejected: Error: Coding stopped - Student is distracted
    at C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP\01-Acti
```

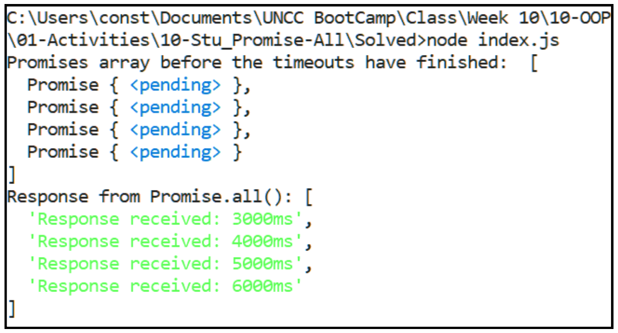
Promise All

```
// Promise resolved right away
const p1 = new Promise((resolve, reject) => {
  const time = 0;
  if (time < 0) {
    reject(new Error('Something went wrong'));
  } else {
    resolve('Resolved right away');
  }
});
console.log('Status of p1:', p1);

// Promise resolved after one second
const p2 = new Promise((resolve, reject) => {
  const time = 1000;
  if (time < 0) {
    reject(new Error('Something went wrong'));
  } else {
    setTimeout(() => {
      resolve('Resolved after 1 second');
    }, time);
  }
});
console.log('Status of p2:', p2);

// Promise resolved after three seconds
const p3 = new Promise((resolve, reject) => {
  const time = 3000;
  if (time < 0) {
    reject(new Error('Something went wrong'));
  } else {
    setTimeout(() => {
      resolve('Resolved after 3 seconds');
    }, time);
  }
});
console.log('Status of p3:', p3);

// After all three are resolved, Promise.all() returns the results
Promise.all([p1, p2, p3])
  .then((values) => {
    console.log('\nThe returned array from our Promise.all() call:');
    console.log(values);
  })
  .catch((err) => new Error(err));
```



```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\
10-OOP\01-Activities\09-Ins_Promise-All>node index.js

Status of p1: Promise { 'Resolved right away' }
Status of p2: Promise { <pending> }
Status of p3: Promise { <pending> }

The returned array from our Promise.all() call:
[
  'Resolved right away',
  'Resolved after 1 second',
  'Resolved after 3 seconds'
]
```

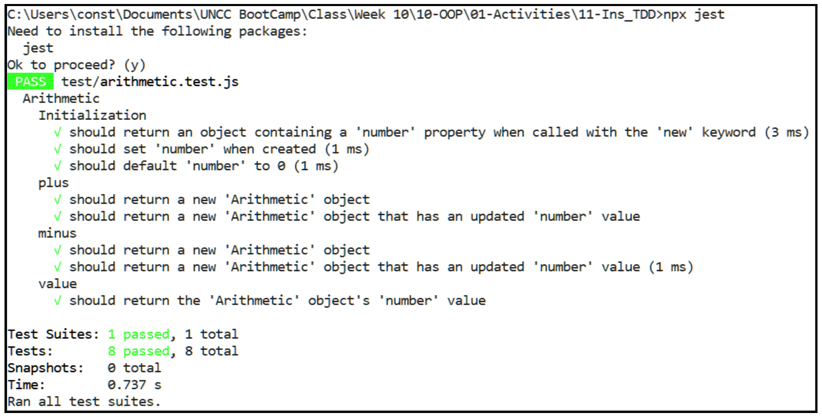
```
const apiCallDurations = [3000, 4000, 5000, 6000];
const maxDuration = 9999;
const callAPI = (duration) =>
  new Promise((resolve, reject) => {
    setTimeout(() => {

      // If the duration is longer than maxDuration, reject() the promise
      if (duration >= maxDuration) {
        reject(new Error(`This request timed out`));
      } else {

        // Otherwise resolve() the promise
        resolve(`Response received: ${duration}ms`);
      }
    }, duration);
  });

const promises = apiCallDurations.map((duration) => callAPI(duration));

console.log('Promises array before the timeouts have finished: ', promises);

// Promise.all will resolve when all promises in the array have been resolved
// Promise.all will reject if any of the promises in that array was rejected
Promise.all(promises)
  .then((response) => console.log('Response from Promise.all():', response))
  .catch((err) => console.error(err));
```



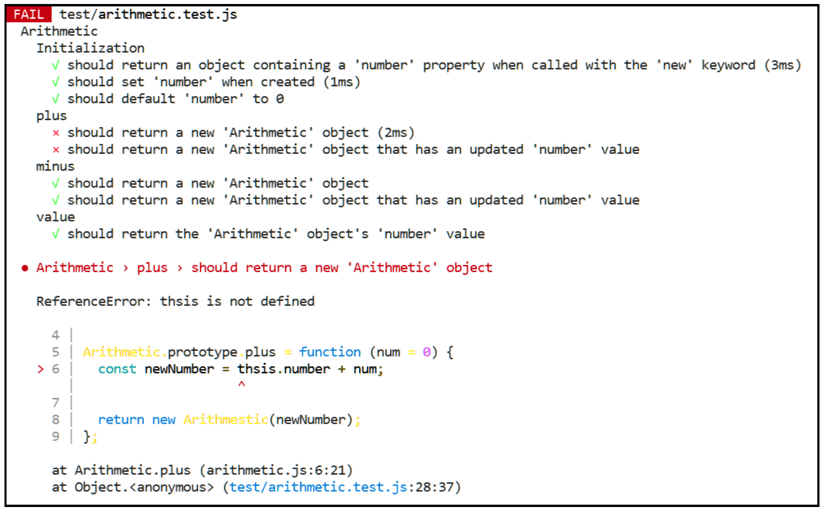
```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP
\01-Activities\10-Stu_Promise-All\Solved>node index.js
Promises array before the timeouts have finished:  [
  Promise { <pending> },
  Promise { <pending> },
  Promise { <pending> },
  Promise { <pending> }
]
Response from Promise.all(): [
  'Response received: 3000ms',
  'Response received: 4000ms',
  'Response received: 5000ms',
  'Response received: 6000ms'
]
```

```
const Arithmetic = require("../arithmetic");

describe("Arithmetic", () => {
 describe("Initialization", () => {
  it("should return an object containing a 'number' property when called with the 'new' keyword", () => {
   const obj = new Arithmetic();

   expect("number" in obj).toEqual(true);
  });

  it("should set 'number' when created", () => {
   const num = 108;
   const obj = new Arithmetic(num);

   expect(obj.number).toEqual(num);
  });

  it("should default 'number' to 0", () => {
   const obj = new Arithmetic();
   expect(obj.number).toEqual(0);
  });
 });

 describe("plus", () => {
  it("should return a new 'Arithmetic' object", () => {
   const obj = new Arithmetic(3).plus(3);

   expect(obj instanceof Arithmetic).toEqual(true);
  });

  it("should return a new 'Arithmetic' object that has an updated 'number' value", () => {
   const num = 8;
   const added = 7;
   const sum = num + added;
   const { number } = new Arithmetic(num).plus(added);

   expect(number).toEqual(sum);
  });
 });
 describe("minus", () => {
  it("should return a new 'Arithmetic' object", () => {
   const obj = new Arithmetic(9).minus(4);

   expect(obj instanceof Arithmetic).toEqual(true);
  });

  it("should return a new 'Arithmetic' object that has an updated 'number' value", () => {
   const num = 10;
   const subtracted = 17;
   const difference = num - subtracted;
   const { number } = new Arithmetic(num).minus(subtracted);

   expect(number).toEqual(difference);
  });
 });

 describe("value", () => {
  it("should return the 'Arithmetic' object's 'number' value", () => {
   const num = 10;
   const obj = new Arithmetic(num);
   const result = obj.value();

   expect(result).toEqual(num);
  });
 });
});
```

TDD



```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP\01-Activities\11-Ins_TDD>npx jest
Need to install the following packages:
  jest
Ok to proceed? (y)
 PASS  test/arithmetic.test.js
  Arithmetic
    Initialization
      √ should return an object containing a 'number' property when called with the 'new' keyword (3 ms)
      √ should set 'number' when created (1 ms)
      √ should default 'number' to 0 (1 ms)
    plus
      √ should return a new 'Arithmetic' object
      √ should return a new 'Arithmetic' object that has an updated 'number' value
    minus
      √ should return a new 'Arithmetic' object
      √ should return a new 'Arithmetic' object that has an updated 'number' value (1 ms)
    value
      √ should return the 'Arithmetic' object's 'number' value

Test Suites: 1 passed, 1 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        0.737 s
Ran all test suites.
```



```
 FAIL  test/arithmetic.test.js
  Arithmetic
    Initialization
      √ should return an object containing a 'number' property when called with the 'new' keyword (3ms)
      √ should set 'number' when created (1ms)
      √ should default 'number' to 0
    plus
      × should return a new 'Arithmetic' object (2ms)
      × should return a new 'Arithmetic' object that has an updated 'number' value
    minus
      √ should return a new 'Arithmetic' object
      √ should return a new 'Arithmetic' object that has an updated 'number' value
    value
      √ should return the 'Arithmetic' object's 'number' value

  ● Arithmetic › plus › should return a new 'Arithmetic' object

    ReferenceError: thsis is not defined

      4 |
      5 | Arithmetic.prototype.plus = function (num = 0) {
    > 6 |   const newNumber = thsis.number + num;
        |                     ^
      7 |
      8 |   return new Arithmestic(newNumber);
      9 | };

      at Arithmetic.plus (arithmetic.js:6:21)
      at Object.<anonymous> (test/arithmetic.test.js:28:37)
```

```
const Arithmetic = require('./arithmetic');

const value = new Arithmetic(4)
  .plus(8)
  .plus(15)
  .minus(16)
  .minus(23)
  .plus(42)
  .plus(108)
  .value();

console.log(value);
```

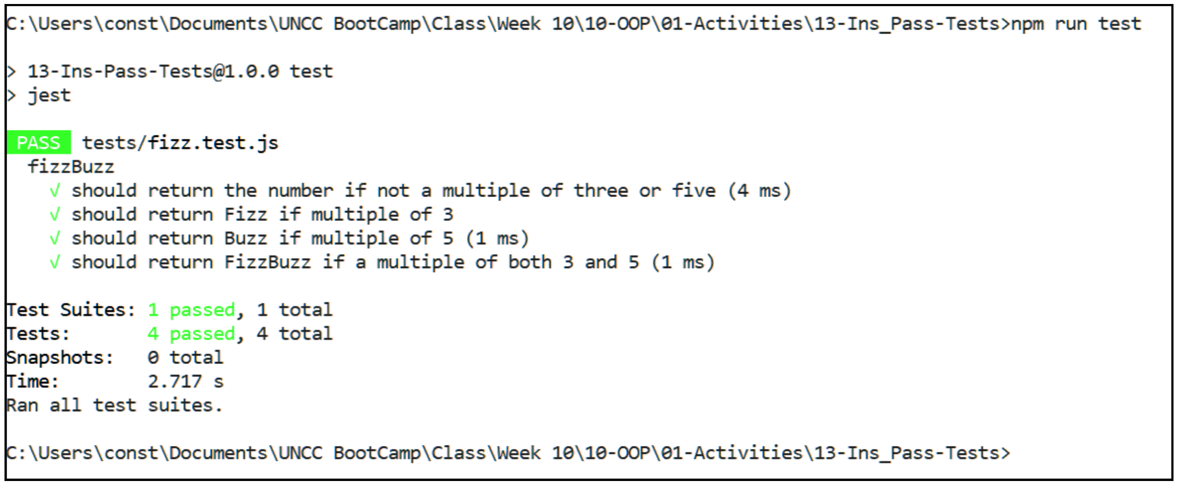
```
function Arithmetic(number = 0) {
  this.number = number;
}
Arithmetic.prototype.plus = function(num = 0) {
  const newNumber = this.number + num;
  return new Arithmetic(newNumber);
};
Arithmetic.prototype.minus = function(num = 0) {
  const newNumber = this.number - num;
  return new Arithmetic(newNumber);
};
Arithmetic.prototype.value = function() {
  return this.number;
};
module.exports = Arithmetic;
```

Pass– Tests

```
const fizzBuzz = (num) =>
// .map() will run a block of code on each element of the 'num' array
        and return a new array with modified values.
  num.map((singleNum) => {
    const multipleOf3 = singleNum % 3 === 0;
    const multipleOf5 = singleNum % 5 === 0;

   // Meets the requirement of the fourth test: 'should return FizzBuzz if a multiple of both 3 and 5'
    if (multipleOf3 && multipleOf5) {
      return 'FizzBuzz';
    }

   // Meets the requirement of the second test: 'should return Fizz if multiple of 3'
    if (multipleOf3) {
      return 'Fizz';
    }
   // Meets the requirement of the third test:'should return Buzz if multiple of 5'
    if (multipleOf5) {
      return 'Buzz';
    }
   // Meets the requirement of first test: 'should return the number if not a multiple of three or five'
    return singleNum;
   })
   .join(', ');

// Here, we export this module so that it can be imported by 'fizz.test.js'
module.exports = fizzBuzz;
```

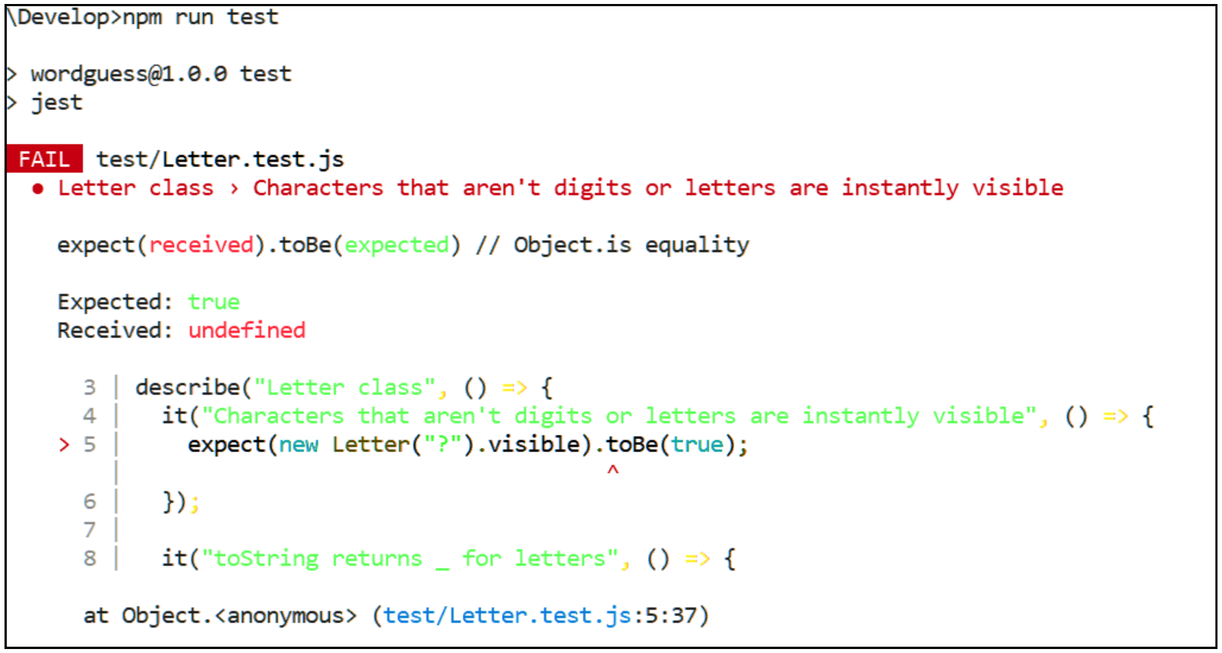
```
{
 "name": "13-Ins-Pass-Tests",
 "version": "1.0.0",
 "description": "",
 "main": "fizz.js",
 "directories": {
   "test": "tests"
 },
 "scripts": {
   "test": "jest"
 },
 "keywords": [],
 "author": "",
 "license": "ISC",
 "devDependencies": {
   "jest": "^26.5.2"
 }
}
```

```
const fizzBuzz = require('../fizz');

describe('fizzBuzz', () => {
  it('should return the number if not a multiple of three or five', () => {
    expect(fizzBuzz([1])).toBe('1');
    expect(fizzBuzz([1, 2])).toBe('1, 2');
  });

  it('should return Fizz if multiple of 3', () => {
    expect(fizzBuzz([3])).toBe('Fizz');
    expect(fizzBuzz([3, 6, 12])).toBe('Fizz, Fizz, Fizz');
  });

  it('should return Buzz if multiple of 5', () => {
    expect(fizzBuzz([10])).toBe('Buzz');
    expect(fizzBuzz([10, 20, 25])).toBe('Buzz, Buzz, Buzz');
  });

  it('should return FizzBuzz if a multiple of both 3 and 5', () => {
    expect(fizzBuzz([15])).toBe('FizzBuzz');
    expect(fizzBuzz([15, 30, 45])).toBe('FizzBuzz, FizzBuzz, FizzBuzz');
  });
});
```



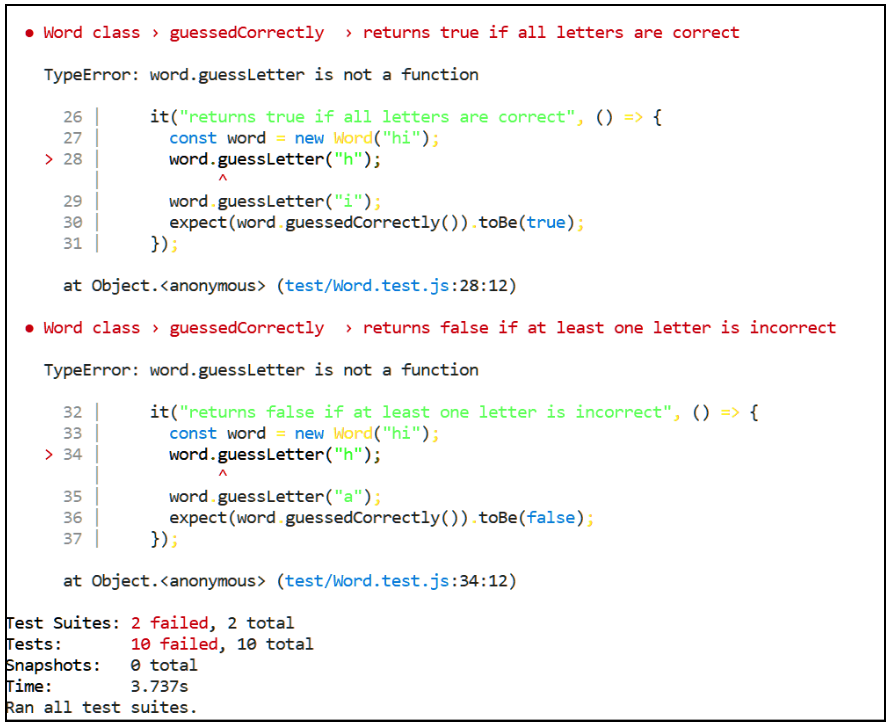
```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP\01-Activities\13-Ins_Pass-Tests>npm run test

> 13-Ins-Pass-Tests@1.0.0 test
> jest

 PASS  tests/fizz.test.js
  fizzBuzz
    √ should return the number if not a multiple of three or five (4 ms)
    √ should return Fizz if multiple of 3
    √ should return Buzz if multiple of 5 (1 ms)
    √ should return FizzBuzz if a multiple of both 3 and 5 (1 ms)

Test Suites: 1 passed, 1 total
Tests:       4 passed, 4 total
Snapshots:   0 total
Time:        2.717 s
Ran all test suites.

C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP\01-Activities\13-Ins_Pass-Tests>
```

Extra: Test Fail Reading



```
\Develop>npm run test

> wordguess@1.0.0 test
> jest

 FAIL  test/Letter.test.js
  ● Letter class › Characters that aren't digits or letters are instantly visible

    expect(received).toBe(expected) // Object.is equality

    Expected: true
    Received: undefined

      3 | describe("Letter class", () => {
      4 |   it("Characters that aren't digits or letters are instantly visible", () => {
    > 5 |     expect(new Letter("?").visible).toBe(true);
        |                                     ^
      6 |   });
      7 |
      8 |   it("toString returns _ for letters", () => {

      at Object.<anonymous> (test/Letter.test.js:5:37)
```

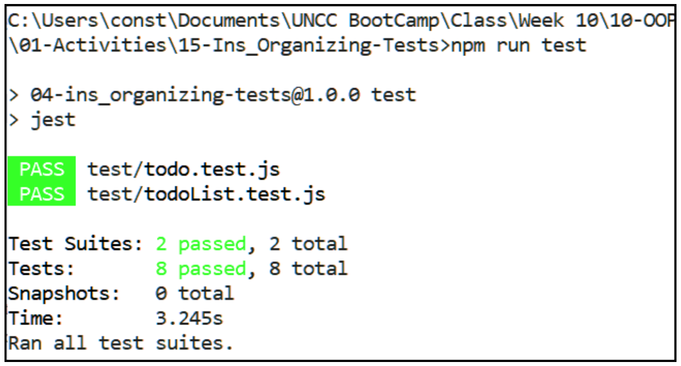
```
const Letter = require("../lib/Letter");

describe("Letter class", () => {
  it("Characters that aren't digits or letters are instantly visible", () => {
    expect(new Letter("?").visible).toBe(true);
  });

  it("toString returns _ for letters", () => {
    expect(new Letter("a").toString()).toBe("_");
  });

  describe("guess", () => {
    it("Correct guess returns true", () => {
      expect(new Letter("j").guess("j")).toBe(true);
    });

    it("Incorrect guess returns false", () => {
      expect(new Letter("j").guess("l")).toBe(false);
    });
  });

  describe("getSolution", () => {
    it("returns character", () => {
      expect(new Letter("l").getSolution()).toBe("l");
    });
  });
});
```



```
  ● Word class › guessedCorrectly  › returns true if all letters are correct

    TypeError: word.guessLetter is not a function

      26 |     it("returns true if all letters are correct", () => {
      27 |       const word = new Word("hi");
    > 28 |       word.guessLetter("h");
         |            ^
      29 |       word.guessLetter("i");
      30 |       expect(word.guessedCorrectly()).toBe(true);
      31 |     });

      at Object.<anonymous> (test/Word.test.js:28:12)

  ● Word class › guessedCorrectly  › returns false if at least one letter is incorrect

    TypeError: word.guessLetter is not a function

      32 |     it("returns false if at least one letter is incorrect", () => {
      33 |       const word = new Word("hi");
    > 34 |       word.guessLetter("h");
         |            ^
      35 |       word.guessLetter("a");
      36 |       expect(word.guessedCorrectly()).toBe(false);
      37 |     });

      at Object.<anonymous> (test/Word.test.js:34:12)

Test Suites: 2 failed, 2 total
Tests:       10 failed, 10 total
Snapshots:   0 total
Time:        3.737s
Ran all test suites.
```

Organizing Test

```
const TodoList = require("./todoList");

const todoList = new TodoList();

todoList.addTodo("Get Eggs");
todoList.addTodo("Get Milk");
todoList.addTodo("Get Bread");

console.log("Next todo:", todoList.getNextTodo());
todoList.completeNextTodo();

console.log("Next todo:", todoList.getNextTodo());
todoList.completeNextTodo();

console.log("Next todo:", todoList.getNextTodo());
todoList.completeNextTodo();
```



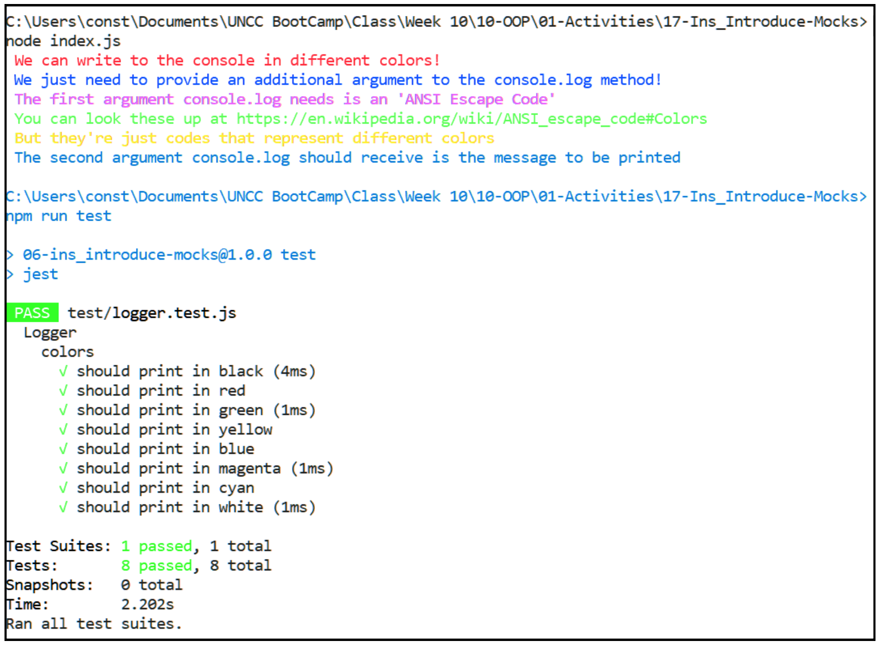
```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP
\01-Activities\15-Ins_Organizing-Tests>npm run test

> 04-ins_organizing-tests@1.0.0 test
> jest

 PASS  test/todo.test.js
 PASS  test/todoList.test.js

Test Suites: 2 passed, 2 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        3.245s
Ran all test suites.
```

```
const Todo = require('./todo');
function TodoList() {
  this.todos = [];
}
TodoList.prototype.addTodo = function (text) {
  this.todos.push(new Todo(text));
};
TodoList.prototype.getNextTodo = function () {
  return this.todos[0];
};
TodoList.prototype.completeNextTodo = function () {
  return this.todos.shift();
};
module.exports = TodoList;
```

```
const TodoList = require("../todoList");
const Todo = require("../todo");

describe("TodoList", () => {
  describe("Initialization", () => {

    // Positive test
    it("should create an object with a 'todos' property set to
            an empty array when called with the 'new' keyword", () => {
      // Arrange
      const todoList = new TodoList();
      // Assert
      expect(todoList.todos).toEqual([]);
    });
  });

  describe("addTodo", () => {
    // Positive Tests
    it("should add a new 'Todo' object to its 'todos' array", () => {

      // Arrange
      const todoList = new TodoList();
      const todoText = "Mow Lawn";

      // Act
      todoList.addTodo(todoText);

      // Assert
      expect(todoList.todos.length).toEqual(1);
      expect(todoList.todos[0] instanceof Todo).toEqual(true);
      expect(todoList.todos[0].text).toEqual(todoText);
    });

    // Exception test
    it("should throw an error if not provided text", () => {

      // Arrange
      const todoList = new TodoList();
      const err = new Error(
        "Expected parameter 'text' to be a non empty string"
      );
      const cb = () => todoList.addTodo();

      // Assert
      expect(cb).toThrowError(err);
    });
  });
  describe("getNextTodo", () => {

    // Positive test
    it("should return the 0th todo element in the 'todos' array
            without removing it", () => {

      // Arrange
      const todoList = new TodoList();
      const text1 = "Exercise";
      const text2 = "Wash Car";
      let nextTodo;

      // Act
      todoList.addTodo(text1);
      todoList.addTodo(text2);
      nextTodo = todoList.getNextTodo();

      // Assert
      expect(nextTodo instanceof Todo).toEqual(true);
      expect(nextTodo.text).toEqual(text1);
      expect(todoList.todos.length).toEqual(2);
    });

    // Negative test
    it("should return undefined if there are no todos", () => {

      // Arrange
      const todoList = new TodoList();
      let nextTodo;

      // Act
      nextTodo = todoList.getNextTodo();

      // Assert
      expect(typeof nextTodo).toEqual("undefined");
    });
  });
  describe("completeNextTodo", () => {

    // Positive test
    it("should return and remove the next todo in the list", () => {

      // Arrange
      const todoList = new TodoList();
      const text1 = "Make Dinner";
      const text2 = "Wash Dishes";
      let nextTodo;

      // Act
      todoList.addTodo(text1);
      todoList.addTodo(text2);
      nextTodo = todoList.completeNextTodo();

      // Assert
      expect(nextTodo instanceof Todo).toEqual(true);
      expect(nextTodo.text).toEqual(text1);
      expect(todoList.todos.length).toEqual(1);
      expect(todoList.getNextTodo().text).toEqual(text2);
    });
  });
});
```

Mocks - Intro

```javascript
const Logger = require("./logger");
const log = new Logger();

log.red("We can write to the console in different colors!");
log.blue("We just need to provide an additional argument to the console.log method!");
log.magenta("The first argument console.log needs is an 'ANSI Escape Code'");
log.green("You can look these up at https://en.wikipedia.org/wiki/
ANSI_escape_code#Colors");
log.yellow("But they're just codes that represent different colors");
log.cyan("The second argument console.log should receive is the message to be printed");
```



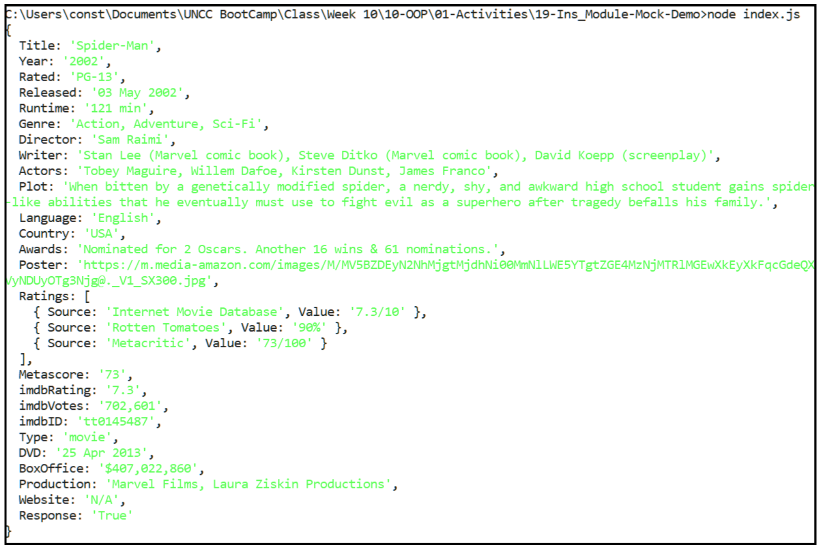
```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP\01-Activities\17-Ins_Introduce-Mocks>
node index.js
 We can write to the console in different colors!
 We just need to provide an additional argument to the console.log method!
 The first argument console.log needs is an 'ANSI Escape Code'
 You can look these up at https://en.wikipedia.org/wiki/ANSI_escape_code#Colors
 But they're just codes that represent different colors
 The second argument console.log should receive is the message to be printed

C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP\01-Activities\17-Ins_Introduce-Mocks>
npm run test

> 06-ins_introduce-mocks@1.0.0 test
> jest

 PASS  test/logger.test.js
  Logger
    colors
      √ should print in black (4ms)
      √ should print in red
      √ should print in green (1ms)
      √ should print in yellow
      √ should print in blue
      √ should print in magenta (1ms)
      √ should print in cyan
      √ should print in white (1ms)

Test Suites: 1 passed, 1 total
Tests:       8 passed, 8 total
Snapshots:   0 total
Time:        2.202s
Ran all test suites.
```

```javascript
const Logger = require("../logger");

const colors = {
  black: "\x1b[30m",
  red: "\x1b[31m",
  green: "\x1b[32m",
  yellow: "\x1b[33m",
  blue: "\x1b[34m",
  magenta: "\x1b[35m",
  cyan: "\x1b[36m",
  white: "\x1b[37m"
};

describe("Logger", () => {
  describe("colors", () => {
    it("should print in black", () => {
      const log = new Logger();
      const message = "Hello world!";
      const mock = jest.spyOn(console, "log");
      mock.mockImplementation(() => {});
      log.black(message);
      expect(mock).toBeCalledWith(colors.black, message);
      mock.mockRestore();
    });

    it("should print in red", () => {
      const log = new Logger();
      const message = "Hello world!";
      const mock = jest.spyOn(console, "log");
      mock.mockImplementation(() => {});
      log.red(message);
      expect(mock).toBeCalledWith(colors.red, message);
      mock.mockRestore();
    });

    it("should print in green", () => {
      const log = new Logger();
      const message = "Hello world!";
      const mock = jest.spyOn(console, "log");
      mock.mockImplementation(() => {});
      log.green(message);
      expect(mock).toBeCalledWith(colors.green, message);
      mock.mockRestore();
    });

    it("should print in yellow", () => {
      const log = new Logger();
      const message = "Hello world!";
      const mock = jest.spyOn(console, "log");
      mock.mockImplementation(() => {});
      log.yellow(message);
      expect(mock).toBeCalledWith(colors.yellow, message);
      mock.mockRestore();
    });

    it("should print in blue", () => {
      const log = new Logger();
      const message = "Hello world!";
      const mock = jest.spyOn(console, "log");
      mock.mockImplementation(() => {});
      log.blue(message);
      expect(mock).toBeCalledWith(colors.blue, message);
      mock.mockRestore();
    });

    it("should print in magenta", () => {
      const log = new Logger();
      const message = "Hello world!";
      const mock = jest.spyOn(console, "log");
      mock.mockImplementation(() => {});
      log.magenta(message);
      expect(mock).toBeCalledWith(colors.magenta, message);
      mock.mockRestore();
    });

    it("should print in cyan", () => {
      const log = new Logger();
      const message = "Hello world!";
      const mock = jest.spyOn(console, "log");
      mock.mockImplementation(() => {});
      log.cyan(message);
      expect(mock).toBeCalledWith(colors.cyan, message);
      mock.mockRestore();
    });

    it("should print in white", () => {
      const log = new Logger();
      const message = "Hello world!";
      const mock = jest.spyOn(console, "log");
      mock.mockImplementation(() => {});
      log.white(message);
      expect(mock).toBeCalledWith(colors.white, message);
      mock.mockRestore();
    });
  });
});
```

```javascript
function Logger() {
  // Here we add methods to the prototype once in a loop rather
          than writing them all out for each color
  if (!this.init) {
    Logger.prototype.init = true;

    const colors = {
      black: "\x1b[30m",
      red: "\x1b[31m",
      green: "\x1b[32m",
      yellow: "\x1b[33m",
      blue: "\x1b[34m",
      magenta: "\x1b[35m",
      cyan: "\x1b[36m",
      white: "\x1b[37m"
    };

    for (const key in colors) {
      // Each color method calls console.log with the color as
          the first argument, followed by any additional arguments.
      Logger.prototype[key] = function(...args) {
        console.log(colors[key], ...args);
      };
    }
  }
}

module.exports = Logger;
```

Mock - Module

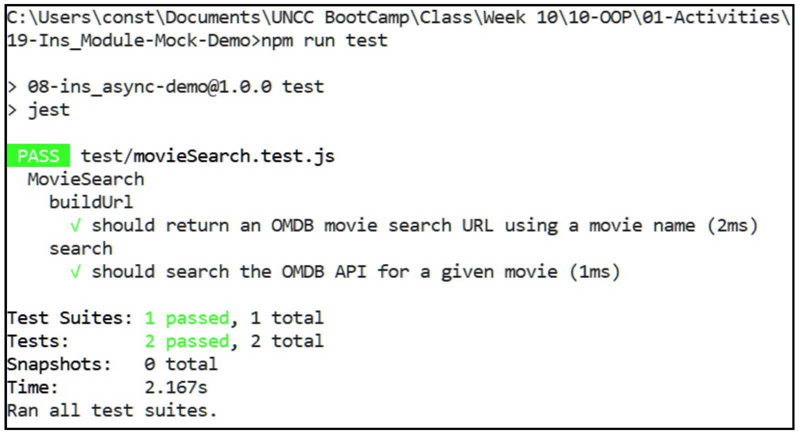
```
const MovieSearch = require("./movieSearch");

const movie = new MovieSearch();

movie.search("Spider-Man")
  .then(res => console.log(res.data));
```



```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP\01-Activities\19-Ins_Module-Mock-Demo>node index.js
{
  Title: 'Spider-Man',
  Year: '2002',
  Rated: 'PG-13',
  Released: '03 May 2002',
  Runtime: '121 min',
  Genre: 'Action, Adventure, Sci-Fi',
  Director: 'Sam Raimi',
  Writer: 'Stan Lee (Marvel comic book), Steve Ditko (Marvel comic book), David Koepp (screenplay)',
  Actors: 'Tobey Maguire, Willem Dafoe, Kirsten Dunst, James Franco',
  Plot: 'When bitten by a genetically modified spider, a nerdy, shy, and awkward high school student gains spider
-like abilities that he eventually must use to fight evil as a superhero after tragedy befalls his family.',
  Language: 'English',
  Country: 'USA',
  Awards: 'Nominated for 2 Oscars. Another 16 wins & 61 nominations.',
  Poster: 'https://m.media-amazon.com/images/M/MV5BZDEyN2NhMjgtMjdhNi00MmNlLWE5YTgtZGE4MzNjMTRlMGEwXkEyXkFqcGdeQX
VyNDUyOTg3Njg@._V1_SX300.jpg',
  Ratings: [
    { Source: 'Internet Movie Database', Value: '7.3/10' },
    { Source: 'Rotten Tomatoes', Value: '90%' },
    { Source: 'Metacritic', Value: '73/100' }
  ],
  Metascore: '73',
  imdbRating: '7.3',
  imdbVotes: '702,601',
  imdbID: 'tt0145487',
  Type: 'movie',
  DVD: '25 Apr 2013',
  BoxOffice: '$407,022,860',
  Production: 'Marvel Films, Laura Ziskin Productions',
  Website: 'N/A',
  Response: 'True'
}
```

```
const axios = require("axios");
const MovieSearch = require("../movieSearch");

jest.mock("axios");

describe("MovieSearch", () => {
  describe("buildUrl", () => {
    it("should return an OMDB movie search URL using a movie name", () => {
      const movie = new MovieSearch();
      const name = "Rocky";

      const url = movie.buildUrl(name);

      expect(url).toEqual(`https://www.omdbapi.com/?t=${name}&apikey=trilogy`);
    });
  });

  describe("search", () => {
    it("should search the OMDB API for a given movie", () => {
      const movie = new MovieSearch();
      const name = "Rocky";

      axios.get.mockReturnValue(
        new Promise(function(resolve) {
          resolve({ data: {} });
        })
      );

      expect(movie.search(name)).resolves.toEqual({ data: {} });
      expect(axios.get).lastCalledWith(movie.buildUrl(name));
    });
  });
});
```

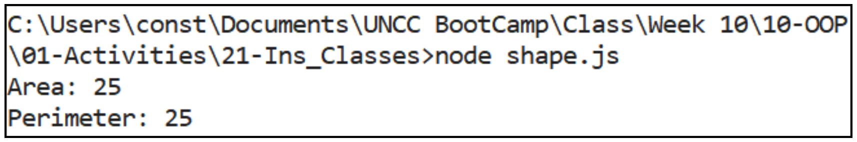
```
const axios = require("axios");

function MovieSearch() {}

MovieSearch.prototype.buildUrl = function(movie) {
  return `https://www.omdbapi.com/?t=${movie}&apikey=trilogy`;
};

MovieSearch.prototype.search = function(movie) {
  return axios.get(this.buildUrl(movie));
};

module.exports = MovieSearch;
```



```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP\01-Activities\
19-Ins_Module-Mock-Demo>npm run test

> 08-ins_async-demo@1.0.0 test
> jest

 PASS  test/movieSearch.test.js
  MovieSearch
    buildUrl
      √ should return an OMDB movie search URL using a movie name (2ms)
    search
      √ should search the OMDB API for a given movie (1ms)

Test Suites: 1 passed, 1 total
Tests:       2 passed, 2 total
Snapshots:   0 total
Time:        2.167s
Ran all test suites.
```

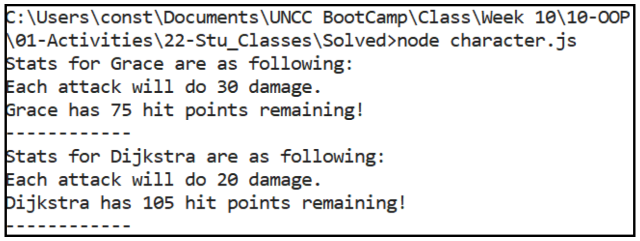
Classes

```
class Shape {
  // Just like constructor functions, classes can accept arguments
  constructor(area, perimeter) {
    this.area = area;
    this.perimeter = perimeter;
  }

  printInfo() {
    console.log(`Area: ${this.area}`);
    console.log(`Perimeter: ${this.perimeter}`);
  }
}

const shape = new Shape(25, 25);
shape.printInfo();
```

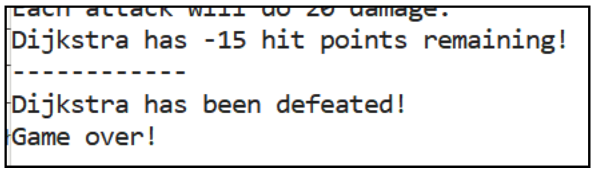


```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP
\01-Activities\21-Ins_Classes>node shape.js
Area: 25
Perimeter: 25
```

```
class Character {
  constructor(name, strength, hitPoints) {
    this.name = name;
    this.strength = strength;
    this.hitPoints = hitPoints;
  }

  // Method which prints all of the stats for a character
  printStats() {
    console.log(`Stats for ${this.name} are as following:`);
    console.log(`Each attack will do ${this.strength} damage.`);
    console.log(`${this.name} has ${this.hitPoints} hit points remaining!`);
    console.log('------------');
  }

  // Method which determines whether or not a character's "hitpoints" are less then zero
  // Returns true or false depending upon the outcome
  isAlive() {
    if (this.hitPoints <= 0) {
      console.log(`${this.name} has been defeated!`);
      return false;
    }
    return true;
  }

  // Method which takes in a second object and decreases their "hitPoints"
                by this character's strength
  attack(opponent) {
    console.log(`${this.name} hit ${opponent.name} for ${this.strength}`);
    opponent.hitPoints -= this.strength;
  }
}

// Creates two unique characters using the "character" constructor
const grace = new Character('Grace', 30, 75);
const dijkstra = new Character('Dijkstra', 20, 105);

// This keeps track of whose turn it is
let graceTurn = true;
grace.printStats();
dijkstra.printStats();
const turnInterval = setInterval(() => {

  // If either character is not alive, end the game
  if (!grace.isAlive() || !dijkstra.isAlive()) {
    clearInterval(turnInterval);
    console.log('Game over!');
  } else if (graceTurn) {
    grace.attack(dijkstra);
    dijkstra.printStats();
  } else {
    dijkstra.attack(grace);
    grace.printStats();
  }

  // Switch turns
  graceTurn = !graceTurn;
}, 2000);
```

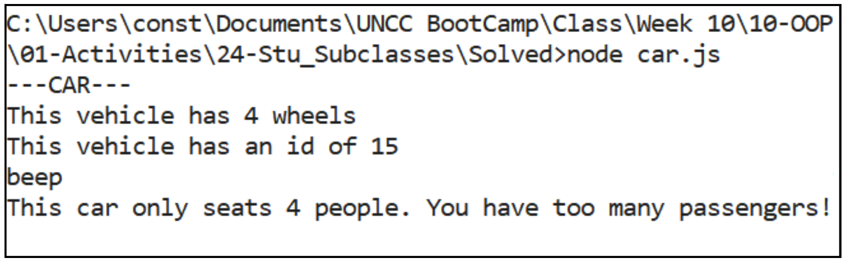


```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP
\01-Activities\22-Stu_Classes\Solved>node character.js
Stats for Grace are as following:
Each attack will do 30 damage.
Grace has 75 hit points remaining!
------------
Stats for Dijkstra are as following:
Each attack will do 20 damage.
Dijkstra has 105 hit points remaining!
------------
```

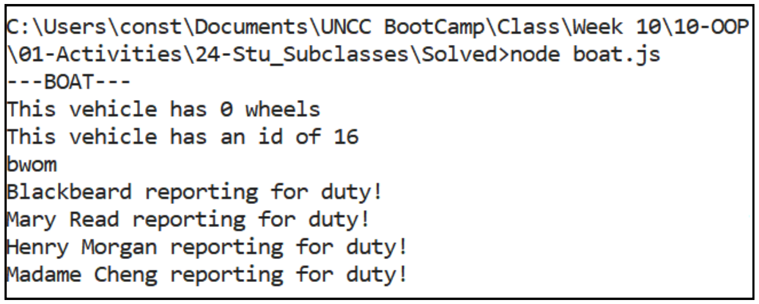


```
Each attack will do 20 damage.
Dijkstra has -15 hit points remaining!
------------
Dijkstra has been defeated!
Game over!
```

Sub-Classes



```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP
\01-Activities\24-Stu_Subclasses\Solved>node car.js
---CAR---
This vehicle has 4 wheels
This vehicle has an id of 15
beep
This car only seats 4 people. You have too many passengers!
```



```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP
\01-Activities\24-Stu_Subclasses\Solved>node boat.js
---BOAT---
This vehicle has 0 wheels
This vehicle has an id of 16
bwom
Blackbeard reporting for duty!
Mary Read reporting for duty!
Henry Morgan reporting for duty!
Madame Cheng reporting for duty!
```

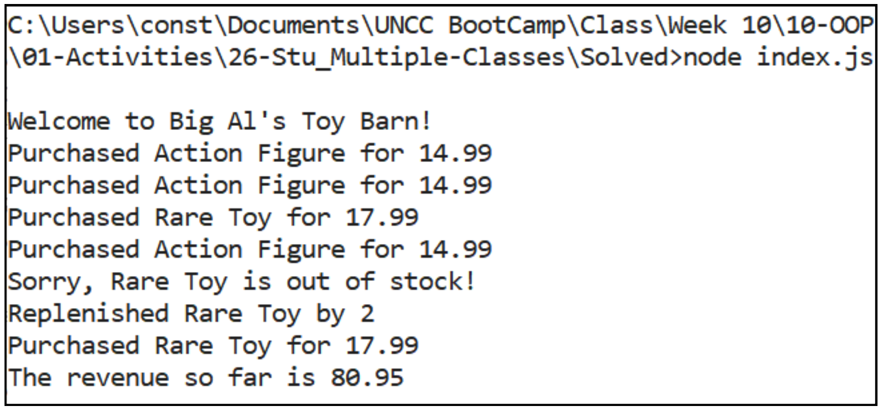
```
const Vehicle = require('./vehicle');
class Car extends Vehicle {
  constructor(id, color, passengers) {
    super(id, 4, 'beep');
    this.color = color;
    this.passengers = passengers;
  }

  useHorn() {
    console.log(this.sound);
  }

  checkPassengers() {
    if (this.passengers.length > 4) {
      console.log(
        'This car only seats 4 people. You have too many passengers!'
      );
    } else {
      console.log(`You have room for ${4 - this.passengers.length} people.`);
    }
  }
}

const carPassengers = [
  'Aristotle',
  'Confucius',
  'Socrates',
  'Lao-Tzu',
  'Plato',
];
const car = new Car(15, 'blue', carPassengers);

console.log('---CAR---');

car.printInfo();
car.useHorn();
car.checkPassengers();
```

```
class Vehicle {
  constructor(id, numberOfWheels, sound) {
    this.id = id;
    this.numberOfWheels = numberOfWheels;
    this.sound = sound;
  }

  printInfo() {
    console.log(`This vehicle has ${this.numberOfWheels} wheels`);
    console.log(`This vehicle has an id of ${this.id}`);
  }
}

module.exports = Vehicle;
```

```
const Vehicle = require('./vehicle');

class Boat extends Vehicle {
  constructor(id, type, crew) {
    super(id, 0, 'bwom');
    this.type = type;
    this.crew = crew;
  }
  Use
Horn() {
    console.log(this.sound);
  }

  CrewSoundOff() {
    this.crew.forEach((member) => {
      console.log(`${member} reporting for duty!`);
    });
  }
}

const boatPassengers = [
  'Blackbeard',
  'Mary Read',
  'Henry Morgan',
  'Madame Cheng',
];

const boat = new Boat(16, 'sailboat', boatPassengers);

console.log('---BOAT---');
boat.printInfo();
boat.useHorn();
boat.crewSoundOff();
```

Multiple Classes

```
const Store = require('./store');
const { toys } = require('./toy');

const store = new Store("Big Al's Toy Barn", toys);

store.welcome();
store.processProductSale('Action Figure');
store.processProductSale('Action Figure');
store.processProductSale('Rare Toy');
store.processProductSale('Action Figure');

store.processProductSale('Rare Toy'); // <= Should be out of stock

store.replenishStock('Rare Toy', 2); // <= Should be restocked
store.processProductSale('Rare Toy');

store.printRevenue(); // <= Should be 80.95
```

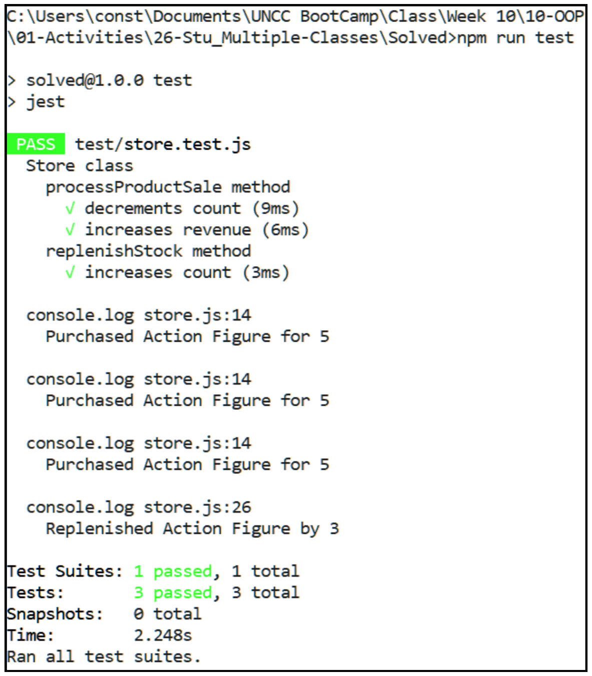


```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP
\01-Activities\26-Stu_Multiple-Classes\Solved>node index.js

Welcome to Big Al's Toy Barn!
Purchased Action Figure for 14.99
Purchased Action Figure for 14.99
Purchased Rare Toy for 17.99
Purchased Action Figure for 14.99
Sorry, Rare Toy is out of stock!
Replenished Rare Toy by 2
Purchased Rare Toy for 17.99
The revenue so far is 80.95
```

```
class Toy {
  constructor(name, price, count) {
    this.name = name;
      this.price = price;
      this.count = count;
    }
  }
  const toys = [
    new Toy("Action Figure", 14.99, 5),
    new Toy("Rare Toy", 17.99, 1)
  ];
  module.exports = {
    Toy,
    toys
  };
```

```
class Store {
  constructor(name, stock) {
    this.name = name;
    this.stock = stock;
    this.revenue = 0;
  }

  processProductSale(name) {
    this.stock.forEach(item => {
      if (item.name === name) {
        if (item.count > 0) {
          item.count--;
          this.revenue += item.price;
          console.log(`Purchased ${item.name} for ${item.price}`);
        } else {
          console.log(`Sorry, ${item.name} is out of stock!`);
        }
      }
    });
  }

  replenishStock(name, count) {
    this.stock.forEach(item => {
      if (item.name === name) {
        item.count += count;
        console.log(`Replenished ${item.name} by ${item.count}`);
      }
    });
  }

  printRevenue() {
    console.log(`The revenue so far is ${this.revenue}`);
  }

  welcome() {
    console.log(`Welcome to ${this.name}!`);
  }
}

module.exports = Store;
```

```
const Store = require("../store");

describe("Store class", () => {
  describe("processProductSale method", () => {
    it("decrements count", () => {
      const store = new Store("Big Al's Toy Barn", [
        { name: "Action Figure", price: 5.0, count: 1 }
      ]);
      store.processProductSale("Action Figure");
      expect(store.stock[0].count).toBe(0);
    });

    it("increases revenue", () => {
      const store = new Store("Big Al's Toy Barn", [
        { name: "Action Figure", price: 5.0, count: 2 }
      ]);
      store.processProductSale("Action Figure");
      store.processProductSale("Action Figure");
      expect(store.revenue).toBe(10);
    });
  });

  describe("replenishStock method", () => {
    it("increases count", () => {
      const store = new Store("Big Al's Toy Barn", [
        { name: "Action Figure", price: 5.0, count: 1 }
      ]);
      store.replenishStock("Action Figure", 2);
      expect(store.stock[0].count).toBe(3);
    });
  });
});
```



```
C:\Users\const\Documents\UNCC BootCamp\Class\Week 10\10-OOP
\01-Activities\26-Stu_Multiple-Classes\Solved>npm run test

> solved@1.0.0 test
> jest

 PASS  test/store.test.js
  Store class
    processProductSale method
      √ decrements count (9ms)
      √ increases revenue (6ms)
    replenishStock method
      √ increases count (3ms)

  console.log store.js:14
    Purchased Action Figure for 5

  console.log store.js:14
    Purchased Action Figure for 5

  console.log store.js:14
    Purchased Action Figure for 5

  console.log store.js:26
    Replenished Action Figure by 3

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        2.248s
Ran all test suites.
```

# Git Stash

You've been using Git for some time now, adding new tools to your Git skills each week. Now you'll learn how to take advantage of the useful `git stash` feature.

Imagine that you're busy implementing a sign-up component on your team's web application when Your manager asks you to stop what you're doing and add a notification to the homepage. You'll need to checkout to a different feature branch to change tasks, but you don't want to commit your half-completed work on the sign-up page. You also don't want to cause any errors by trying to checkout to a different branch with unsaved changes.

That's where `git stash` comes in! As the name suggests, this command adds your uncommited changes to a stash that you can access later. This stash is stored locally on your machine and doesn't get pushed up to Github or any other version control system.

## Use git stash in a Demo

Important: If you haven't done so already, change your default branch name to `main` with the following command:

```
git config --global init.defaultBranch main
```

Create a new repository called `stashPop` using the following commands:

```
mkdir stashPop
cd stashPop
git init
```

Create and commit a new `README.md` file with the following commands:

```
touch README.md
git add -A
git commit -m "initial commit"
```

Checkout a new branch called `dev` and create a new file with the following commands:

```
git checkout -b dev
touch newFile.js
```

Add and commit the new file with the following commands:

```
git add newFile.js
git commit -m "newFile is tracked in dev"
```

Now that we have a `newFile.js` file that is being tracked in the `dev` branch, let's checkout to `main` and verify the files with the following commands:

```
git checkout main
ls
```

No-
tice that we don't have a `newFile.js` file in `main`, so let's create one with the following command:

```
touch newFile.js
```

Let's say that we're not ready to commit this file to ` main` yet, but we need to switch to the `dev` branch again. Try to do so with the following command:

```
git checkout dev
```

Git will prevent you from switching branches, displaying the following error message:

```
error: The following untracked working tree files
would be overwritten by checkout: newFile.js
Please move or remove them before you switch bran
ches.
Aborting
```

This happens because we have a conflicting file (`newFile.js`) that would be overwritten by the check out. If we were to commit `newFile.js`, then Git would know to swap them out, but we're not ready to commit yet.

We will use `git stash` to put away our changes so that we can checkout the `dev` branch. Run the following commands to discover what happens:

```
git stash -u
ls
```

The `-u` flag tells Git to include untracked files in our stash. `newFile.js` has now been put away, so we can safely checkout to `dev` with the following command:

```
git checkout dev
```

Don't worry, the stash isn't lost. We can use additional commands to bring those changes back. Run the following commands to see this in action:

```
git checkout main
git stash pop
```

The command `git stash pop` will pull the latest stash off the stack and reapply its changes. Other useful `stash` commands include the following:

* `git stash drop`&mdash;
  Delete the latest stash from the stack.

* `git stash apply`&mdash;
  Apply your stashed changes,
  but also keep a copy in the stack.

* `git stash clear`&mdash;
  Clear all stashed entries from the stack.