1. **High-level: what is TDD?**

   Test-drive development!

   TDD is a quick repetitive cycle that revolves around first determining what a piece of code should do and writing tests for that behavior before actually writing any code.

   Test-driven development dictates that tests, not application code, should be written first, and then application code should only be written to pass the already written tests.

2. **How could we trigger a syntax error?**

   - a misspelled function keyword (`funtion broken() {...}`)
   - incorrect number of curly braces

3. **How does TDD make sure your code isn't bloated?**

   It's a bitch to write tests! So, because writing tests sucks, you only do it when it's really needed -> that means you don't do it for shit code.

4. **How do I create a function to wait for the result of a promise, returning the number of seconds it waited?**
   This is based off of Alissa's promise.js extra promise practice.

   ```
   function wait(seconds) {
   . return new Promise (resolve =>
   setTimeout(resolve, seconds *
   1000)
   }
   ```

5. **How do we use async await functions to wait for a number of seconds, then print a message? Message and seconds being the parameters.**

   ```
   async function
   waitAndPrint(message, seconds)
   {
   . await new Promise(resolve =>
   setTimeOut(resolve,
   . seconds * 1000) )
   . console.log(message)
   }
   ```

6. **How is the syntax of using 'assert' different from 'expect'?**

   With assert, we immediately chain a function on, in the parameters of that condition or check we then reference our result and expectation.
   With expect we immediately put in our result, then chain on our conditions or checks.

   ```
   it("should reverse the input string", function() {
   . let test = reverseString("hello");
   . let result = "olleh";
   . // the line below is where the actual test is!
   . assert.strictEqual(test, result);
   })
   ```

7. **How many times is 'before' run in mocha?**

   Once before all tests in a given block

   ```
   describe("animalMakers", function() {
   . before(function() {
   . . console.log("after");
   . });
   . describe("penguinMaker", function() {
   . . it("should make penguins", () => {});
   . });
   . describe("catMaker", function() {
   . . it("should make cats", () => {});
   . });
   });
   ```

8. **HTTP Headers: Accept**

   What the client can receive. May be expansive to accept all kinds of data, or limited, such as only accepting `application/json`

9. **HTTP Headers: Content-***

   Defines details about the body, indicating what format it is in, such as `application/json`or application/x-www.form-urlencoded

10. **HTTP Headers: Expires**

    When the response should be considered stale and needs to be re fetched. Often used to cache a response so that the subsequent requests can load directly from the cache instead of hitting the server.

11. **HTTP Headers: Host**

    Root path of our URI (typically a domain like appacademy.io, could also be an IP address)

12. **Http Headers: Location**

    A server typically adds this to a response so that the client can perform a redirection

13. **HTTP Headers: Referrer**

    The URL that you're coming from (such as when you click a link to a new site)

14. **HTTP Headers: Set-cookie**

    The server is telling the client to create/update a key/value pair in its cookies

15. **HTTP Headers: User-Agent**	Information about which browser the request originated from.
16. **HTTP Verbs: DELETE**	Destroy a resource on the server, such as removing a product, or logging out a user (destroying their session)
17. **HTTP Verbs: GET**	Direct Requests. Do not contain a body, simply asking for data
18. **HTTP Verbs: PATCH**	Update a resource on the server. Does not need to have the whole resource, usually just the identifier and what fields are being updated.
19. **HTTP Verbs: POST**	Creating a new resource on the server. Usually what is generated when we submit a form, with the form's data being passed in the body of the request
20. **HTTP Verbs: PUT**	Update a resource on the server. Contain the whole resource to be updated.

21. **If I wanted to make a specific error message any time a TypeError is thrown, how would I do that?**	use instanceof TypeError:
	} catch (error) {
	. if (error instanceof TypeError) {
	. console.error(`Wrong Type: ${error.message}`);
	. } else {
	. console.error(error.message);
	. }
	}

22. **If I want to invoke a function before every 'it' block, what function do I use? Where do I put that?**	I would use the beforeEach('description', callback), and I would put that within the parent describe block that holds all the 'it' blocks.
23. **If we invoke an undefined variable, what error type do we get?**	TypeError
24. **If we're using the assert testing library, what are the most common functions we use? How are they different?**	deepStrictEqual & strictEqual Deep will compare the values, which equal will compare that they are literally the same thing, i.e. if we compare two arrays with equal that have the same value, they'll return false, because though they have the same stuff, they're not literally the same 'thing'.
25. **If we try to redefine a const variable, what error type do we get?**	TypeError
26. **REVIEW: What is the typical npm workflow for a new project?**	1. mkdir new folder / cd to the folder 2. npm init --y (makes our package.json) 3. npm install PACKAGESyouWANT (starts our package-lock.json) 4. probably gonna do testing, so: npm install mocha npm install chai npm install chai-spies 5. update our 'scripts' { 'test': 'mocha' } always forget this one bc I have it globally installed - not strictly necessary, but good practice in case I need to use another testing framework. Don't want to use mocha as muscle memory.
27. **What are, in order, the three parts of a good test?**	1. Arrange - define your variables for use in the remainder of your test 2. Act - perform the action who's result will be tested, this isn't necessary when checking for a thrown error (normally) 3. Assert - this is where we'll actually assert that a value is going to equal, contain, etc.
28. **What are integration tests?**	Once you have your unit tests in place you know each piece works in isolation - but what about when those pieces interact with each other? Integration tests are the next level up, they will test the interactions between two pieces of your application. Integration tests will ensure the units you've written work coherently together.
29. **What are the 3 things a Promise can return?**	1. Pending - it's not resolved or rejected.. yet 2. Resolved - no error was thrown 3. Rejected - there was an issue with the promise

30.	**What are the 4 big advantages of TDD?**	1. Writing tests before code ensures that the code written works. 2. Only required code is written, because writing tests is hard. 3. Modular code: to test it, it's gotta be able to be broken down into small testable chunks. 4. Understanding what code should be doing - Writing tests for a piece of code ensures that the developer writing that code knows what the piece of code is trying to achieve.
31.	**What are the arguments for chia spies? What's the full syntax?**	const spy = chai.spy.on(Object, 'functionWeAreTracking') Object.functionWeAreTracking(1) expect(spy).to.have.been.called.once (or '.exactly(1)') expect(spy).not. to.have.been.called()
32.	**What are the HTTP verbs?**	POST - Create GET - Read PUT - Update/Replace PATCH - Update/Modify DELETE - Delete
33.	**What are the levels of the testing pyramid?**	Bottom: Unit testing Middle: Integration testing Top: End-to-end testing
34.	**What are the most common errors in JS?**	1. SyntaxError - error in syntax of code. 2. ReferenceError - an invalid reference is made. 3. TypeError - a variable or parameter is not of a valid type. Other errors: 4. RangeError -when a numeric variable or parameter is outside of its valid range. 5. InternalError - represents an error in the internal JavaScript engine. 6. EvalError - represents an error with the global eval function. 7. URIError - represents an error that occurs when encodeURI() or decodeURI() are passed invalid parameters.
35.	**What are the parts of HTTP request?**	1. The method, i.e. GET 2. the path, normally root, '/' 3. protocol version, 'HTTP/1.1' 4. headers GET / HTTP/1.1 headers
36.	**What are the pre- & post-test hooks we need to know?**	before('description', callback) - cb is invoked before the block of code is run beforeEach('description', callback) - cb is invoked before each 'it' statement after('description', callback) - cb is invoked after the block of code is run afterEach('description', callback) - cb is invoked after each 'it' statement
37.	**What are the steps of TDD?**	Red: Write the tests and watch them fail (a failing test is red). It's important to ensure the tests initially fail so that you don't have false positives. Green: Write the minimum amount of code to ensure the tests pass (a passing test will be green). Refactor: Refactor the code you just wrote. Your job is not over when the tests pass! One of the most important things you do as a software developer is to ensure the code you write is easy to maintain and read.
38.	**What do error codes in the 100-199 range mean?**	Informational
39.	**What do error codes in the 200-299 range mean?**	Successful - 200 OK: received and fulfilled, typically with a body that has the requested data
40.	**What do error codes in the 300-399 range mean?**	Redirection - 302 Found: the resource has moved. We usually see this with a Location header, where a browser will automatically redirect the request to the new location.

41. **What do error codes in the 400-499 range mean?**	Client Error - 400 Bad Request: General response that the server couldn't understand your request. Often seen with typos, if a more specific 404 is not issued. - 401 Unauthorized: The resource may exist, but you're not allowed to see it unless you are authorized. (Try logging in with valid credentials before sending the request again.) - 403 Forbidden: The resource may exist, but you're not allowed to see it, even if you are logged in. Can also be seen if you're trying to perform an action that is not allowed (such as creating a duplicate record). Maybe this is a resource that you need special permissions for, like admin access. - 404 Not Found: The resource doesn't exist. It may be that it hasn't been created, or that you just had a typo in what you were requesting.
42. **What do error codes in the 500-599 range mean?**	Server Error - 500 Internal Server Error: The server tried to process your request, but something went wrong, typically there was some kind of runtime error in the server code due to your request.
43. **What does "Content-Type" mean in HTTP requests and responses?**	In responses, a Content-Type header tells the client what the content type of the returned content actually is. In requests, (such as POST or PUT), the client tells the server what type of data is actually sent.
44. **What does HTTP mean?**	Hypertext Transfer Protocol
45. **What error do we get when we reference a non-existent variable?**	ReferenceError const puppy = "puppy"; console.log(pupy); // mistyped variable name
46. **What is a 200 response code?**	- 200 OK: received and fulfilled, typically with a body that has the requested data
47. **What is a 302 response code?**	- 302 Found: the resource has moved. We usually see this with a Location header, where a browser will automatically redirect the request to the new location.
48. **What is a 400 error code?**	- 400 Bad Request: General response that the server couldn't understand your request. Often seen with typos, if a more specific 404 is not issued.
49. **What is a 401 error code?**	- 401 Unauthorized: The resource may exist, but you're not allowed to see it unless you are authorized. (Try logging in with valid credentials before sending the request again.)
50. **What is a 403 error code?**	- 403 Forbidden: The resource may exist, but you're not allowed to see it, even if you are logged in. Can also be seen if you're trying to perform an action that is not allowed (such as creating a duplicate record). Maybe this is a resource that you need special permissions for, like admin access.
51. **What is a 404 error code?**	- 404 Not Found: The resource doesn't exist. It may be that it hasn't been created, or that you just had a typo in what you were requesting.
52. **What is a 500 error code?**	Internal Server Error: The server tried to process your request, but something went wrong, typically there was some kind of runtime error in the server code due to your request.
53. **What is end-to-end testing?**	End-to-end tests are the highest level of testing - these will test the whole of your application. End-to-end tests are the closest automated tests come to testing the an actual user experience of your application. These are generally the slowest tests to write and run.
54. **What is the issue with defining the same variables twice in multiple tests?**	There isn't one! Trick question. That's common practice - we need to test multiple parts of a contructor, meaning we'll need to define our new object in multiple tests. Caveat: It's not DRY, so use hooks when possible.
55. **What is unit testing?**	The smallest unit of testing - used to test the smallest pieces of your application in isolation to ensure each piece works before you attempt to put those pieces together. Each unit test should focus on testing one thing. These are generally the fastest tests to write and run.

56. **What kind of error does this throw? Why?** **const puppy = "puppy";** **puppy = "apple";**	TypeError! Because it's a type of constant, meaning once defined it can't be changed, so JS says that type can't be changed.
57. **What's wrong with this?** **assert.throws(sandwichMaker(14), TypeError);**	We didn't wrap the function we expect to error out in an anonymous fat arrow! So it'll halt code & won't actually assert anything. assert.throws(() => sandwichMaker(14)), TypeError);
58. **What tests do we write most often?**	Unit tests - they're fast & easy to write, there should be quite a few of them.
59. **What type of error can't be caught in a try/catch block?**	SyntaxError - it runs at compile time, ie the code cannot be parsed to determine the instructions
60. **What type of error do we get if we reference a variable that is out of scope?**	ReferenceError
61. **What type of HTTP request never has a body?**	GET
62. **When testing for error output in mocha, how do we ensure that we get a pass for throwing an error?**	First, we need to invoke our function with a value we know will fail, so we can force the error. Second, we have to wrap that error causing function in another function - otherwise our code will actually fail & throw the error we're testing for. (confusing, I know).
63. **When using promises, what are the two common methods we use?**	.then & .catch.
64. **Which function is invoked before every test within the same describe block?**	beforeEach('description', callback)
65. **Which type of HTTP request normally has a body?**	POST
66. **Why can't you catch a syntax error?**	It happens at compile time, not run time, so it's thrown before the try/catch, or anything else.
67. **Why do we test?**	1. Ensure code works 2. Code is flexible & easy to refactor without fear of breaking everything 3. Collaboration is easy(easier?) 4. Documentation is implicit in testing