

QUIZ

What does the `server` key on the options object passed to the `WebSocket.Server()` constructor accomplish?

```
const wsServer = new WebSocket.Server({ server: httpServer });
```

Specifies the HTTP client over which the WebSocket connection will be established.

Specifies the HTTP server over which the WebSocket connection will be established.



You got it!

Nothing, you should always use the `port` option instead.

Fill in the code to allow the client's `.send()` method to convert an object with the `.type` and `.payload` key to a String with the server then turning that String type back into an object.

```
// Client
const data = {
  type: 'NEW_USER'
  payload: { name: 'Lily' }
}

wsClient.send(✓ JSON.stringify (data));

// Server
wsServer.on('connection', (socket) => {
  socket.on('message', (message) => {
    const { type, payload } = ✓ JSON.parse (message);
  });
});
```




You got it!

Fill in the code for the server to log the message "New client connected!" when a new client has connected. Also, have the client log the message "WebSocket connection established!".


```
// Server
wsServer. (  , (  ) => {
  console.log('New client connected!');
});

// Client
wsClient. = (  ) => {
  console.log('WebSocket connection established!');
};
```

 You got it!

What is a benefit of using the `ws` Node.js package?

Provides a WebSocket server implementation for client-server WebSocket connections.

 Correct! The `ws` package does not provide a browser-compatible WebSocket client however.

Allows you to build a WebSocket server from scratch.


Enables modern browsers to provide a WebSocket client.

Fill in the code to send a message from the client to the server listening for the message over a WebSocket connection.

```
// Client
wsClient. ('This is the message that the server will receive.');
```



```
// Server
wsServer.on('connection', (socket) => {
  socket. (  , (  ) => {
    console.log('message received: ' + data);
  });
});
```

 You got it!

Fill in the blanks to outline the pattern of broadcasting:

A WebSocket server:

- ✓ Iterates through the complete list of its connected clients, and for each connected client:
 - ✓ Checks if the client's connection is still open:
 - ✓ If so, sends the broadcast message to that client
 - ✓ Continues to the next client



You got it!

Fill in the code to instantiate a `WebSocket` client object using the browser-native `WebSocket()` constructor.

```
const wsClient = new WebSocket ( 'ws://' + HOST_NAME );
```



You got it!

Fill in the code to send a message from the server to the client. Then, complete the client-side event handler such that it prints out each message received from the server.

```
// Server
wsServer.on('connection', (socket) => {
  socket. send ('Welcome to the server!');
});

// client
wsClient. onmessage = (messageEvent) => {
  console.log('Message received: ' + messageEvent.data );
};
```



You got it!

Which of the following is NOT TRUE of the implementation of the server broadcast pattern shown below:

```
function broadcast(data, socketToOmit) {  
  wsServer.clients.forEach((client) => {  
    if (client.readyState === WebSocket.OPEN && client !== socketToOmit) {  
      client.send(JSON.stringify(data));  
    }  
  });  
}
```

The `wsServer.clients` property holds an array of all client `websocket` objects connected to the server.

The constant value corresponding to the open ready state is `WebSocket.OPEN`.

The message will be sent to all clients.



Correct! `client !== socketToOmit` will omit a specific client.