

# SER 321 C Session

**SI Session**

**Sunday, June 9th 2024**

*6:00 pm - 7:00 pm MST*

# Agenda



Server Socket Connection

Handling the Client Connection

OSI Model Review

Transport Layer Review

# SI Session Expectations

Thanks for coming to the **SER 321** SI session. We have a packed agenda and we are going to try to get through as many of our planned example problems as possible. This session will be recorded and shared with others.

- If after this you want to see additional examples, please visit the drop-in tutoring center.
- We will post the link in the chat now and at the end of the session.
  - [tutoring.asu.edu](https://tutoring.asu.edu)
- Please keep in mind we are recording this session and it will be made available for you to review 24-48 hours after this session concludes.
- Finally, please be respectful to each other during the session.

# Interact with us:

## Zoom Features



### Zoom Chat

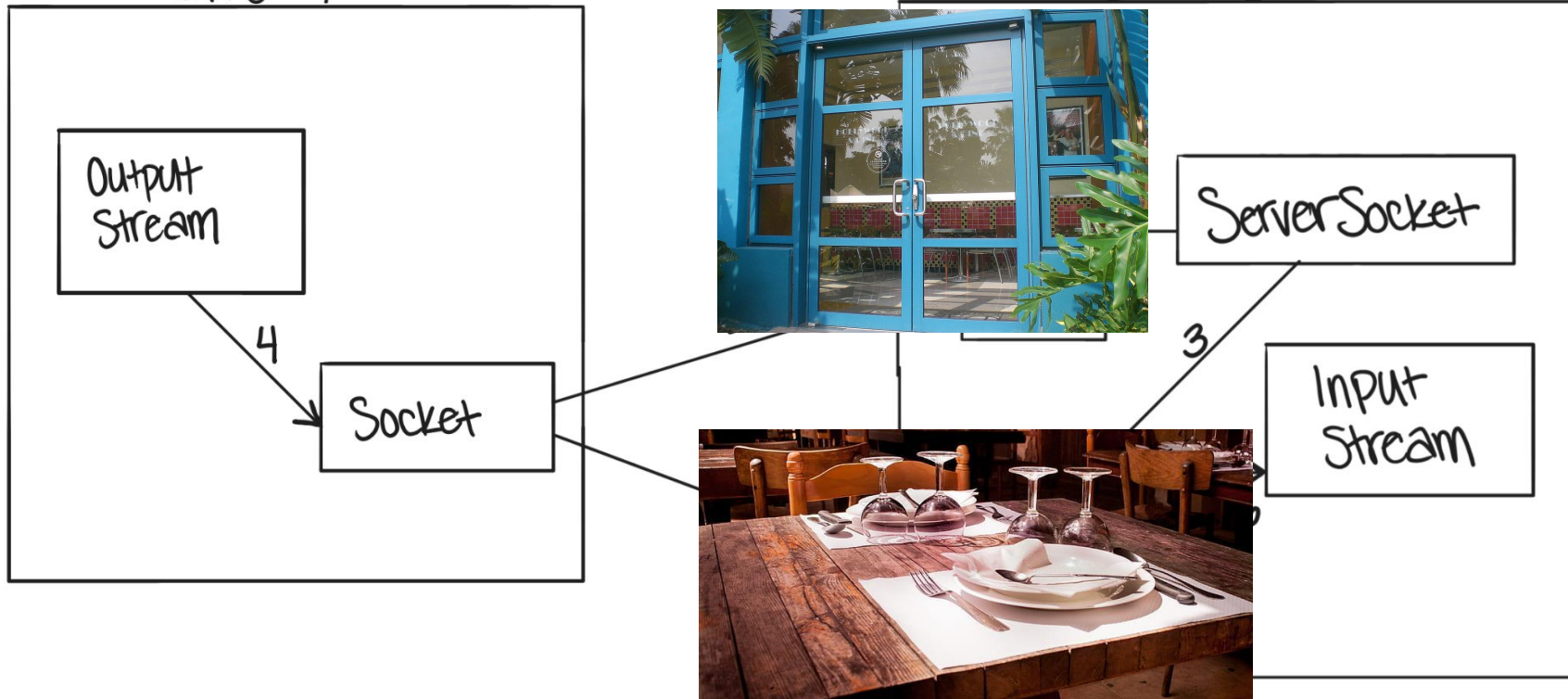
- Use the chat feature to interact with the presenter and respond to presenter's questions.
- Annotations are encouraged

# SER 321

## Sockets!

Client

Server



# SER 321

## Sockets!

> Task :runServer

Server ready for connections

Server is listening on port: 9099

-----

Values of the ServerSocket Object:

Inet Address: 0.0.0.0/0.0.0.0

Local Port: 9099

Server waiting for a connection

Server connected to client

-----

Values of the Client Socket Object after Connection:

Inet Address: /127.0.0.1

Local Address: /127.0.0.1

Local Port: 9099

Allocated Client Socket (Port): 60296

<=====----> 75% EXECUTING [1m 13s]

> :runServer



```
try {
    if (args.length > 0) {
        String host = args[0];
        int port = Integer.parseInt(args[1]);
        Socket server = new Socket(host, port);
        System.out.println("Connected to server at " + host + ":" + port);
        System.out.println("Values of the Socket Object for the Server:");
        System.out.println("\tHost: " + server.getLocalAddress());
        System.out.println("\tPort: " + server.getPort());
        System.out.println("\tLocal Port: " + server.getLocalPort());
    }
} catch (IOException e) {
    e.printStackTrace();
}

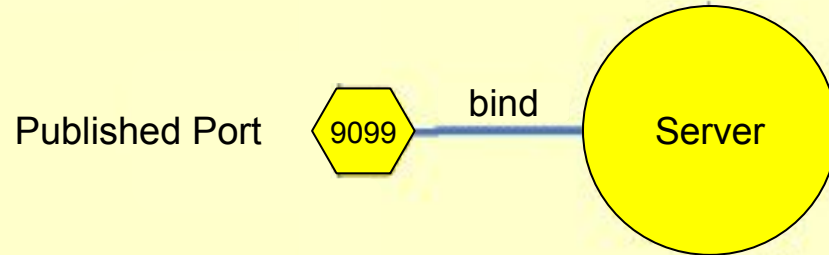
// Client side
try {
    String host = "localhost";
    int port = 9099;
    Socket clientSocket = new Socket(host, port);
    System.out.println("Connected to server at localhost:9099");
    System.out.println("Values of the Socket Object for the Server:");
    System.out.println("\tHost: /127.0.0.1");
    System.out.println("\tPort: 9099");
    System.out.println("\tLocal Port: 60296");
    String toSend = "75% EXECUTING [31s]";
    clientSocket.getOutputStream().write(toSend.getBytes());
    clientSocket.getOutputStream().flush();
    System.out.println("String to send: " + toSend);
    System.out.println("-----");
    System.out.println("Values of the Client Socket Object after Connection:");
    System.out.println("\tInet Address: " + clientSocket.getInetAddress());
    System.out.println("\tLocal Address: " + clientSocket.getLocalAddress());
    System.out.println("\tLocal Port: " + clientSocket.getLocalPort());
    System.out.println("\tAllocated Client Socket (Port): " + clientSocket.getPort());
} catch (IOException e) {
    e.printStackTrace();
}

// Server side
try {
    int numR = input.read(clientInput, off: 0, bufLen);
    System.out.println("Received from client: " + new String(clientInput, off: 0, numR));
} catch (IOException e) {
    e.printStackTrace();
}
```

# SER 321

## Sockets!

```
> Task :runServer
Server ready for connections
Server is listening on port: 9099
-----
Values of the ServerSocket Object:
Inet Address: 0.0.0.0/0.0.0.0
Local Port: 9099
Server waiting for a connection
Server connected to client
-----
Values of the Client Socket Object after Connection:
    Inet Address: /127.0.0.1
    Local Address: /127.0.0.1
    Local Port: 9099
    Allocated Client Socket (Port): 60296
<=====--> 75% EXECUTING [2m 36s]
> :runServer
```

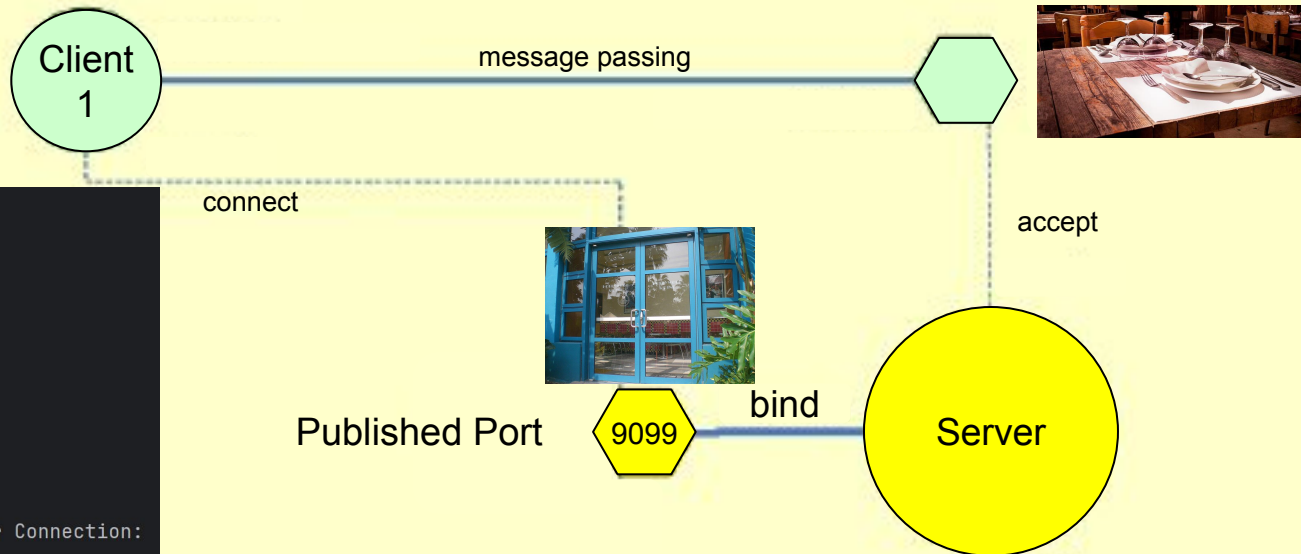


```
> Task :runClient
Connected to server at localhost:9099
Values of the Socket Object for the Server:
    Host: /127.0.0.1
    Port: 9099
    Local Port: 60296
String to send>
<=====--> 75% EXECUTING [2m 18s]s]
> :runClient
```

# SER 321

## Sockets!

```
> Task :runServer
Server ready for connections
Server is listening on port: 9099
-----
Values of the ServerSocket Object:
Inet Address: 0.0.0.0/0.0.0.0
Local Port: 9099
Server waiting for a connection
Server connected to client
-----
Values of the Client Socket Object after Connection:
Inet Address: /127.0.0.1
Local Address: /127.0.0.1
Local Port: 9099
Allocated Client Socket (Port): 60296
<=====--> 75% EXECUTING [2m 36s]
> :runServer
```



```
> Task :runClient
Connected to server at localhost:9099
Values of the Socket Object for the Server:
Host: /127.0.0.1
Port: 9099
Local Port: 60296
String to send>
<=====--> 75% EXECUTING [2m 18s]s]
> :runClient
```



**SER 321**

**Server Socket**

Java  
handles  
a few  
steps for  
us...

1. Define Params

2. Create Socket

3. **C ONLY** Create a struct for the address

3-5. Mark Socket to Listen

5. Mark Socket to Listen for Connections

6. Wait for Connection

7. Handle Client Connection

8. Close Client Connection

9. Continue Listening for Connections

# SER 321

## Server Socket

1. Define Params

2. Create Socket

3-5. Mark Socket to Listen

6. Wait for Connection

7. Handle Client Connection

8. Close Client Connection

9. Continue Listening

1

2 & 3-5

9

6

```
public static void main (String args[]) {  
  
    if (args.length != 1) {  
        System.out.println("Expected arguments: <port(int)>");  
        System.exit( status: 1);  
    }  
  
    try {  
        port = Integer.parseInt(args[0]);  
    } catch (NumberFormatException nfe) {  
        System.out.println("[Port|sleepDelay] must be an integer");  
        System.exit( status: 2);  
    }  
  
    try {  
        //open socket  
        ServerSocket serv = new ServerSocket(port);  
        System.out.println("Server ready for connections");  
  
        while (true){  
            System.out.println("Server waiting for a connection");  
            sock = serv.accept(); // blocking wait  
            System.out.println("Client connected");  
            // ...  
        }  
    }  
}
```

# SER 321

## Server Socket

What needs to be done here?

1. Define Params

2. Create Socket

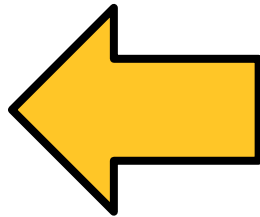
3-5. Mark Socket to Listen

6. Wait for Connection

7. Handle Client Connection

8. Close Client Connection

9. Continue Listening



1

2

3

4

5

**SER 321****Server Socket**

What needs to be done here?

Is input  
*from the client*  
or  
*to the client* ?

1. Define Params

```
// setup the object reading channel  
in = new ObjectInputStream(sock.getInputStream());
```

```
// get output channel  
OutputStream out = sock.getOutputStream();
```

```
// create an object output writer (Java only)  
os = new DataOutputStream(out);
```

1

2

3

4

5

```
clientSock = sock.accept(); // blocking wait  
PrintWriter out = new PrintWriter(clientSock.getOutputStream(), autoFlush: true);  
InputStream input = clientSock.getInputStream();  
System.out.println("Server connected to client");
```

# SER 321

## Server Socket

What needs to be done here?

```
static void overandout() {  
    try {  
        os.close();  
        in.close();  
        sock.close();  
    } catch (Exception e) {e.printStackTrace();}  
}  
  
try {  
    s = (String) in.readObject();  
} catch (Exception e) {  
    System.out.println("Client disconnect");  
    connected = false;  
    continue;  
}
```

1 Create input/output streams

2

3

4

5

# SER 321

## Server Socket

What needs to be done here?

```
JSONObject res = isValid(s);

if (res.has(key: "ok")) {
    writeOut(res);
    continue;
}

JSONObject req = new JSONObject(s);

res = testField(req, key: "type");
if (!res.getBoolean(key: "ok")) {
    res = noType(req);
    writeOut(res);
    continue;
}
```

```
public static JSONObject isValid(String json) {
    try {
        static JSONObject testField(JSONObject req, String key){
            JSONObject res = new JSONObject();

            // field does not exist
            if (!req.has(key)){
                res.put("ok", false);
                res.put("message", "Field " + key + " does not exist in request");
                return res;
            }

            return res.put("ok", true);
        }
    }
    return res;
}

return new JSONObject();
}
```

**SER 321****Server Socket**

What needs to be done here?

```
int numr = input.read(clientInput, off: 0, buflen);
```

```
String received = new String(clientInput, offset: 0, numr);  
System.out.println("read from client: " + received);  
out.println(received);
```

```
if (req.getString(key: "type").equals("echo")) {  
    res = echo(req);  
} else if (req.getString(key: "type").equals("add")) {  
    res = add(req);  
} else if (req.getString(key: "type").equals("addmany")) {  
    res = addmany(req);  
} else {  
    res = wrongType(req);  
}  
writeOut(res);
```

1 Create input/output streams

2 Check for disconnect

3 Check Protocol

4

5

# SER 321

## Server Socket

What needs to be done here?

1. Define Params

2. Create Socket

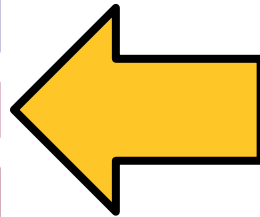
3-5. Mark Socket to Listen

6. Wait for Connection

7. Handle Client Connection

8. Close Client Connection

9. Continue Listening



1 Create input/output streams

2 Check for disconnect

3 Check Protocol

4 Read Headers

5 Handle Accordingly



# SER 321

## OSI Model

Unit

Layer

What we are *really*  
talking about

Data	Application	
Data	Presentation	
Data	Session	
Segment	Transport	
Packet	Network	
Frame	Data Link	
Bits	Physical	

# SER 321

## OSI Model

Unit

Layer

What we are *really*  
talking about

Data	Application	HTTP(s), SMTP, FTP, IMAP, POP, etc.	Content/Payload Info
Data	Presentation	Translation, compression, encryption	
Data	Session	AuthN, authZ, session mgmt	
Segment	Transport	TCP/UDP	
Packet	Network	IP address, routing and delivery	Transmission Info
Frame	Data Link	LLC, MAC, data transmission in LAN	
Bits	Physical	Signal, Binary transmission	

**SER 321**

**TCP vs. UDP Matching**

Unreliable

TCP

OR

UDP

**SER 321**

**TCP vs. UDP Matching**

Connection-Oriented

TCP

Reliable

OR

UDP

Unreliable

# SER 321

## TCP vs. UDP Matching

Uses Streams

TCP

Reliable

Connection-Oriented

OR

UDP

Unreliable

Connectionless

# SER 321

## TCP vs. UDP Matching

Has Less Overhead

TCP

Reliable

Connection-Oriented

Uses Streams

OR

UDP

Unreliable

Connectionless

Uses Datagrams

# SER 321

## TCP vs. UDP Matching

Has Less Overhead

TCP

Reliable

Connection-Oriented

Uses Streams

Has More Overhead

OR

UDP

Unreliable

Connectionless

Uses Datagrams

Has Less Overhead

**SER 321**

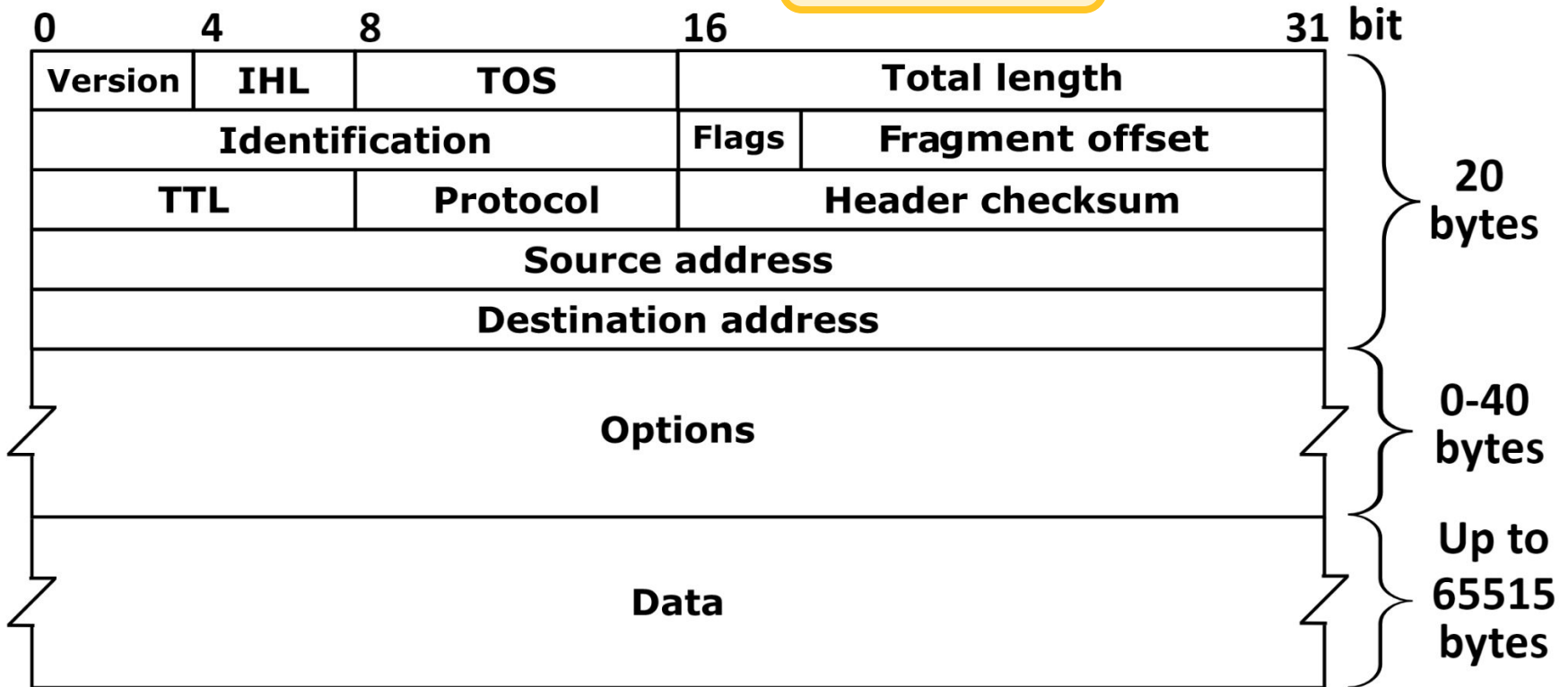
**TCP vs. UDP Matching**

TCP

OR

UDP

IP Header





## TCP vs. UDP Matching

OR

# UDP

### TCP segment header

Offsets		0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset	Reserved 0 0 0 0				C W R	E C E	U R G	A C K	P S H	R S T	S Y N	F I N	Window Size																		
16	128	Checksum																Urgent pointer (if URG set)															
20	160	Options (if <i>data offset</i> > 5. Padded at the end with "0" bits if necessary.)																															
:	:																																
56	448																																

### UDP datagram header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port															Destination port																
4	32	Length															Checksum																

**SER 321**

**Scratch Space**

# Questions?



## Survey:

<http://bit.ly/ASN2324>



## Upcoming Events

### SI Sessions:

- Monday, June 10th at 6:00 pm MST
- Thursday, June 13th at 6:00 pm MST
- Sunday, June 16th at 6:00 pm MST

### Review Sessions:

- Review Session - **Wednesday**, July 3rd at 6:00 pm MST (2 hr Session)
- Q&A Session - Sunday, July 7th at 6:00 pm MST (Final Session)

# More Questions?

Check out our other resources!

tutoring.asu.edu



## Academic Support

Academic Support Network (ASN) provides a variety of free services in-person and online to help currently enrolled ASU students succeed academically.

### Services



#### Subject Area Tutoring

Need in-person or online help with math, science, business, or engineering courses? Just hop into our Zoom room or drop into a center for small group tutoring. We'll take it from there.

[Need help using Zoom?](#)

[View the tutoring schedule](#)

[View digital resources](#)

Go to Zoom



#### Writing Tutoring

Need help with undergraduate or graduate writing assignments? Schedule an in-person or online appointment, access your appointment link, or wait in our drop-in queue.

[Access your appointment link](#)

[Access the drop-in queue](#)

Schedule Appointment



#### Online Study Hub

Join our online peer communities to connect with your fellow Sun Devils. Engage with our tools to search our bank of resources, videos, and previously asked questions. Or, ask our Tutorbot questions.

Now supporting courses in Math, Science, Business, Engineering, and Writing.

Online Study Hub

1-

Go to Zoom

2-

[Need help using Zoom?](#)

[View the tutoring schedule](#)

[View digital resources](#)



1. Click on 'Go to Zoom' to log onto our Online Tutoring Center.
2. Click on 'View the tutoring schedule' to see when tutors are available for specific courses.

# More Questions?

## Check out our other resources!

[tutoring.asu.edu/online-study-hub](https://tutoring.asu.edu/online-study-hub)

 **Academic Support Network**

 [Services](#)  [Faculty and Staff Resources](#) [About Us](#) 

[University College](#)

## Online Study Hub

Online peer communities for students and tutors, YouTube channels, and Tutorbots.



### What are online peer communities?

Individual courses have an online peer community that allows you to connect with your peers to post and answer questions and to develop study groups.



### How can tutoring center videos help?

Videos can help supplement the learning you're doing in and outside of class and include step-by-step methods for how to understand concepts.



### How does the Tutorbot work?

You can ask the Tutorbot questions about course concepts and the Tutorbot will recommend additional resources and examples to help address your questions.

Select a subject

- Any -

[Apply](#)



Academic Support Network



[Services](#) 

[Faculty and Staff Resources](#)

[About Us](#) 

[University College](#)

Select a subject

- Any -

[Apply](#)

Business

### ACC 231

Uses of Accounting Info I

 [Peer Community](#)

### ACC 241

Uses of Accounting Info II

 [Peer Community](#)

### CIS 105

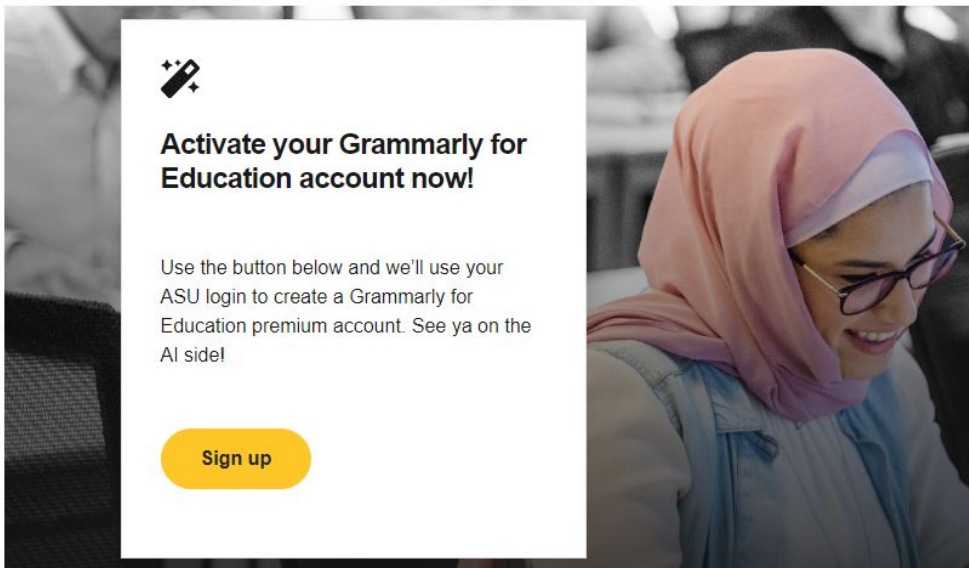
Computer Applications and Information Technology


 [Peer Community](#)

Don't forget to check out the Online Study Hub for additional resources!

# Expanded Writing Support Available

Including Grammarly for Education, at no cost!





**Activate your Grammarly for Education account now!**

Use the button below and we'll use your ASU login to create a Grammarly for Education premium account. See ya on the AI side!

[Sign up](#)



[tutoring.asu.edu/expanded-writing-support](https://tutoring.asu.edu/expanded-writing-support)

\*Available slots for this pilot are limited

## Additional Resources

- [Course Repo](#)
- [Gradle Documentation](#)
- [GitHub SSH Help](#)
- [Linux Man Pages](#)
- [OSI Interactive](#)
- [MDN HTTP Docs](#)
  - [Requests](#)
  - [Responses](#)
- [JSON Guide](#)
- [org.json Docs](#)
- [javax.swing package API](#)
- [Swing Tutorials](#)