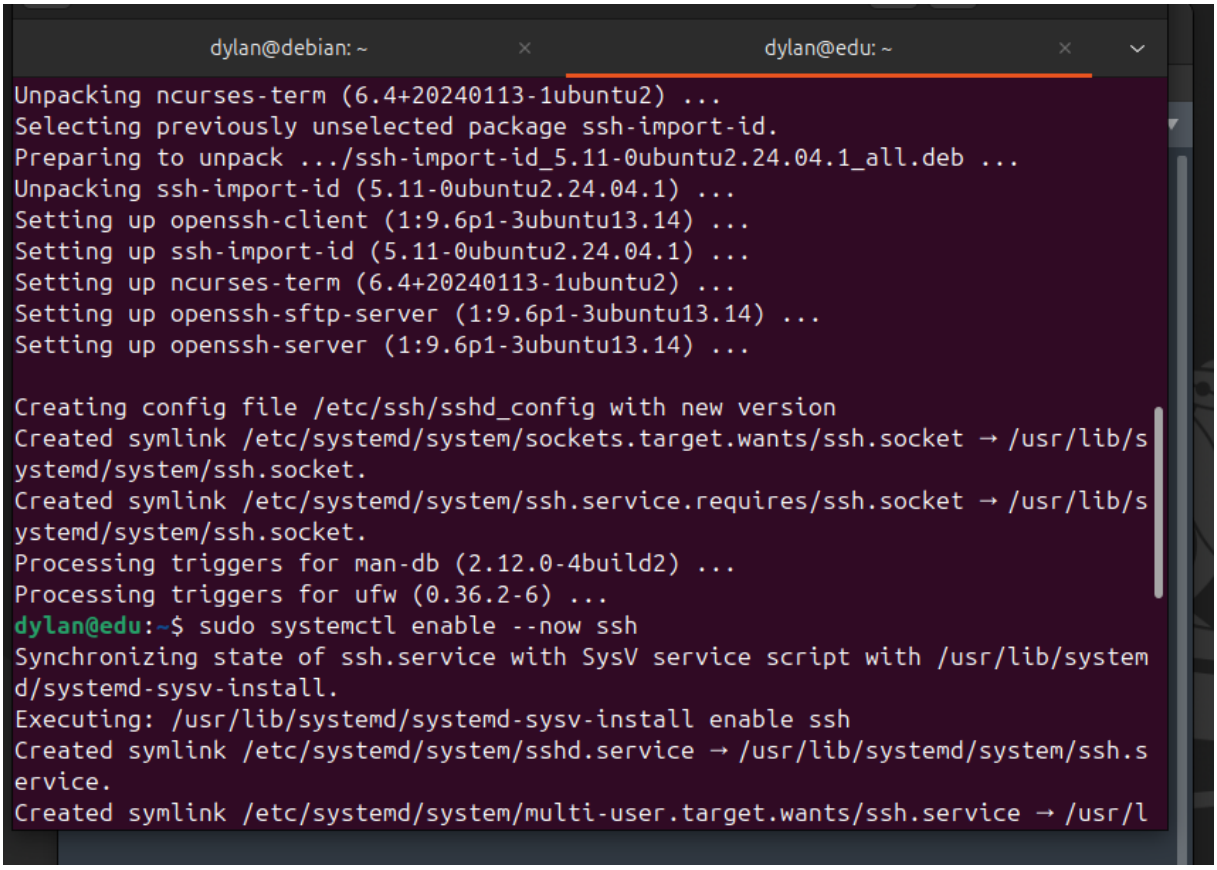
Template Week 6 – Networking

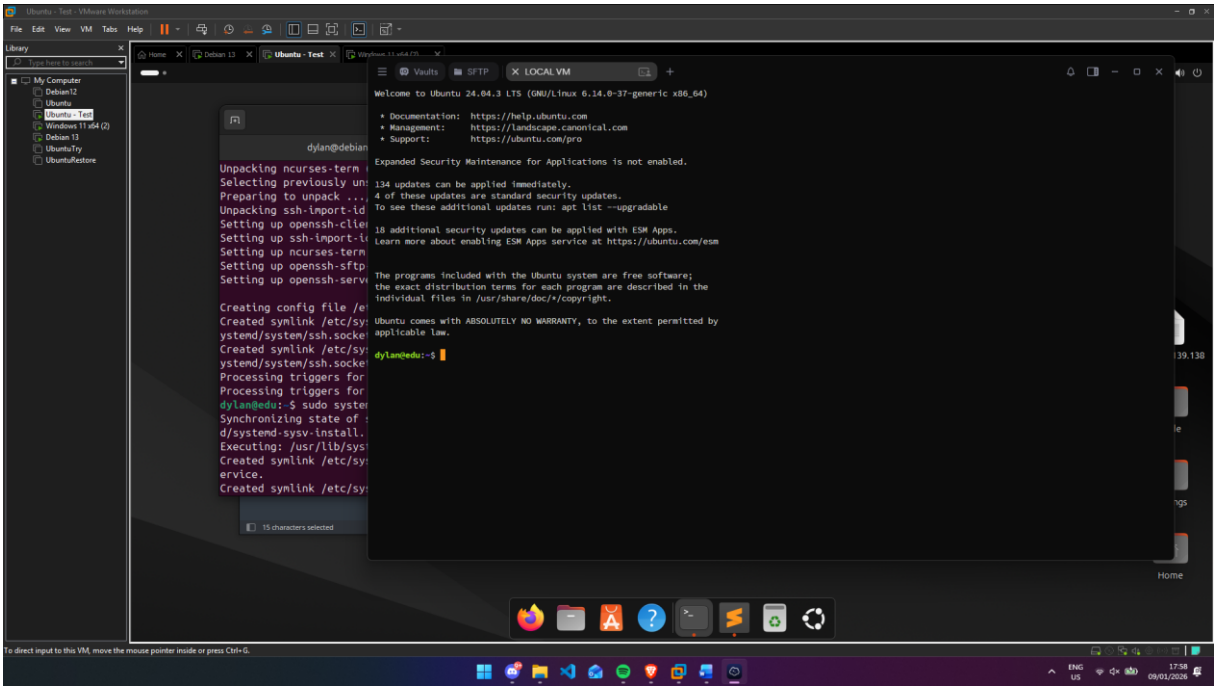
Student number:

Assignment 6.1: Working from home

Screenshot installation openssh-server:

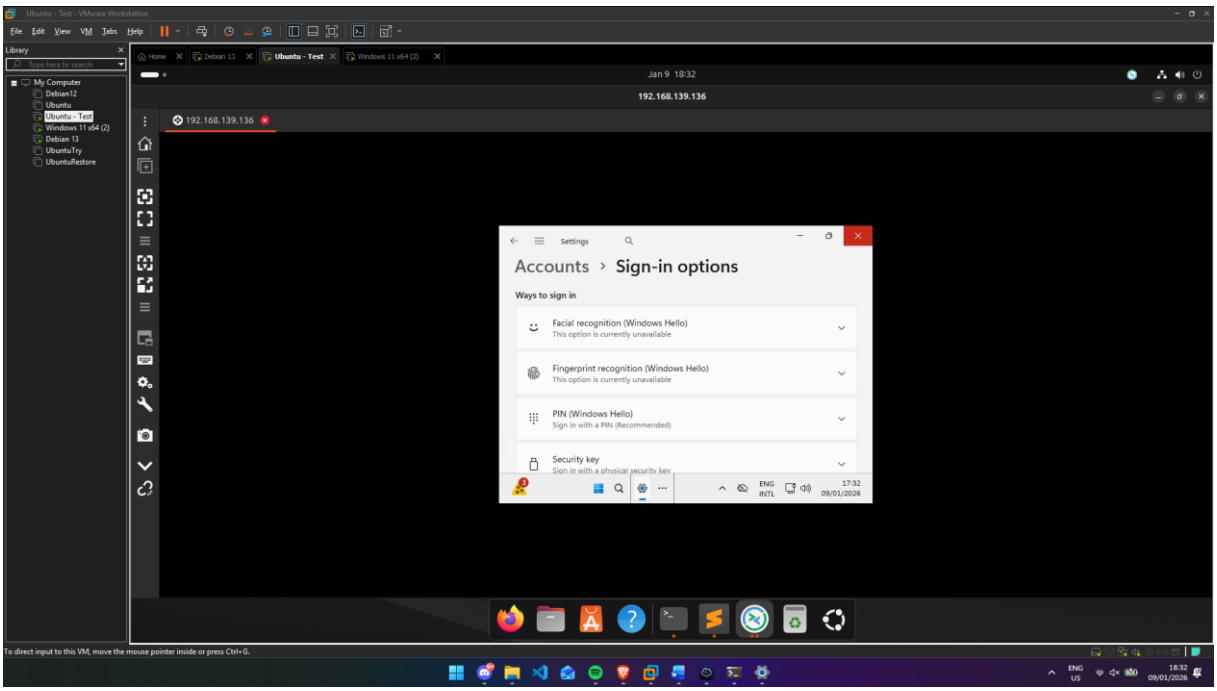


Screenshot successful SSH command execution:



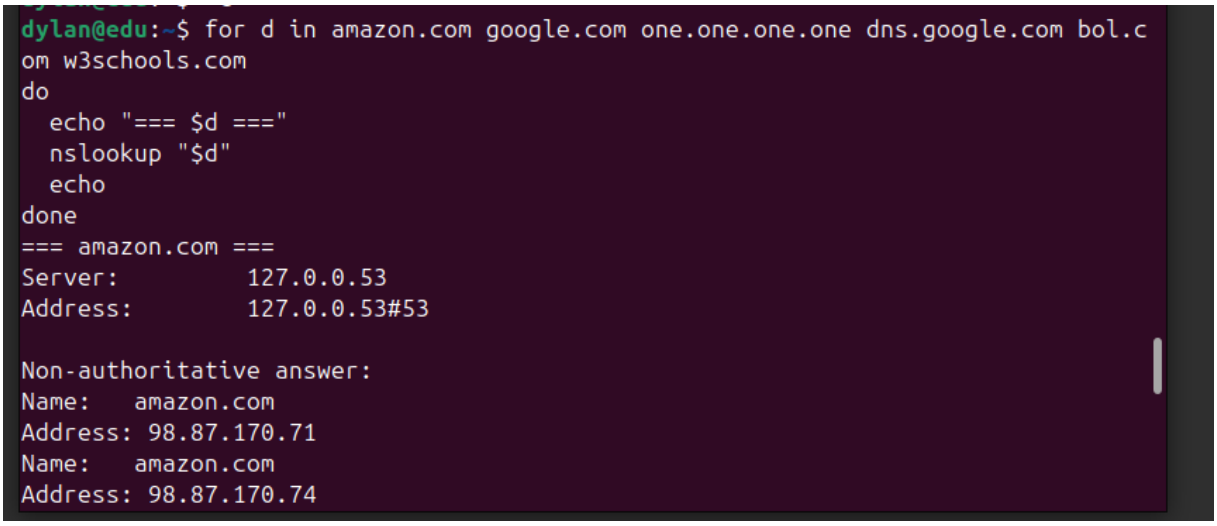
Screenshot successful execution SCP command:

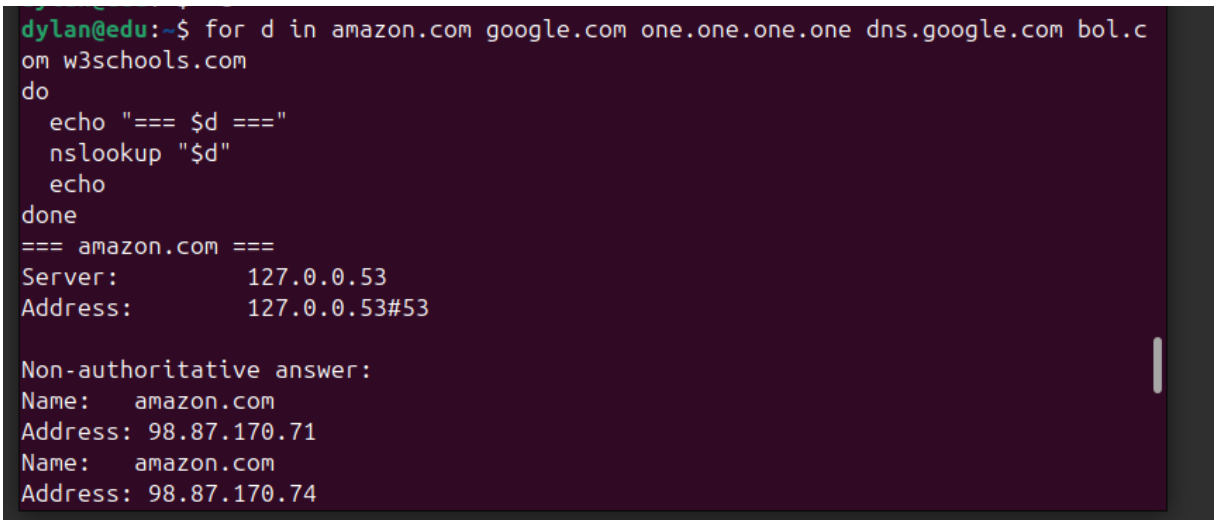
Screenshot remmina:



Assignment 6.2: IP addresses websites

Relevant screenshots nslookup command:





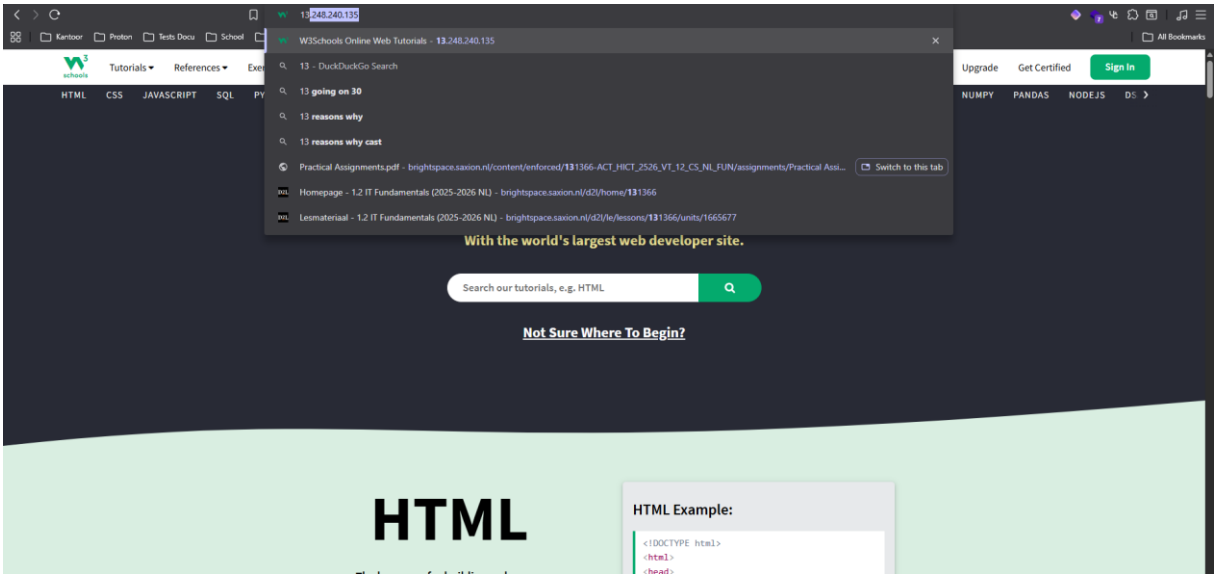
```
=== bol.com ===
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   bol.com
Address: 79.170.100.62


=== w3schools.com ===
Server:         127.0.0.53
Address:        127.0.0.53#53

Non-authoritative answer:
Name:   w3schools.com
Address: 76.223.115.82
Name:   w3schools.com
Address: 13.248.240.135
```

Screenshot website visit via IP address:



When I visited http://13.248.240.135/ it opened the W3Schools website. This happens because many websites use shared hosting/CDNs where multiple domains can share the same IP address, so opening the IP directly can show the default website configured for that server.

Assignment 6.3: subnetting

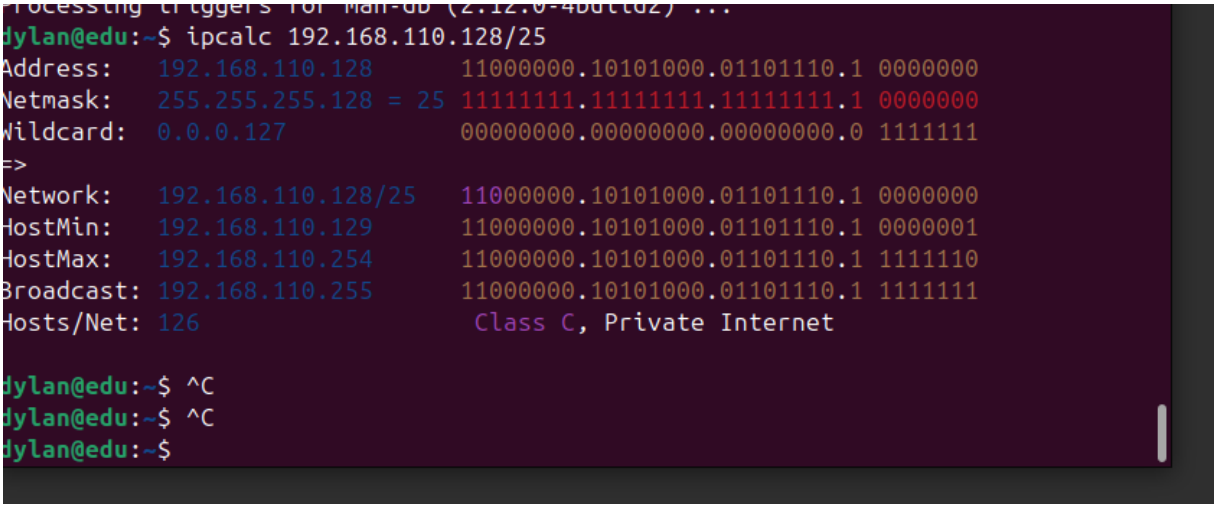
How many IP addresses are in this network configuration 192.168.110.128/25?
A **/25** leaves **7 host bits** (because 32 − 25 = 7).
So total addresses = $2^7 = 128$.

What is the usable IP range to hand out to the connected computers?
For a /25 subnet:

- **Network address**: 192.168.110.128
- **Broadcast address**: 192.168.110.255
- Usable hosts are everything in between:

Usable range: 192.168.110.129 → 192.168.110.254

Check your two previous answers with this Linux command: `ipcalc 192.168.110.128/25`



Explain the above calculation in your own words.

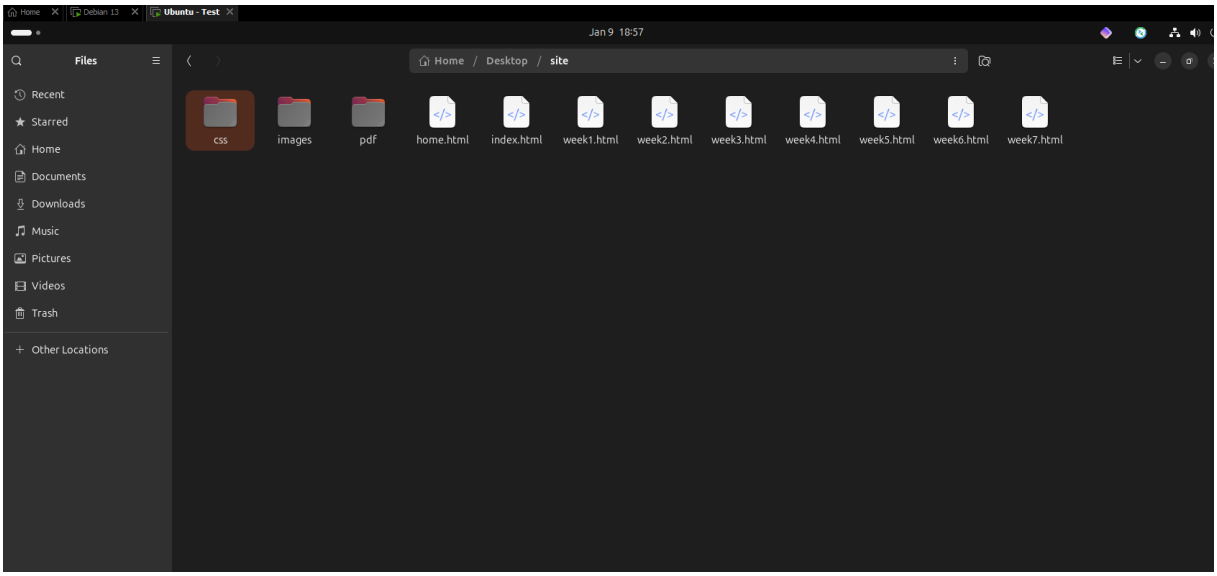
A /25 subnet means the first 25 bits are the network part and the remaining 7 bits are for hosts. With 7 host bits you can make 128 combinations, so the subnet contains 128 addresses. The first address is reserved as the network address and the last as the broadcast address, so you can hand out the addresses in between to devices: 192.168.110.129 to 192.168.110.254.

Assignment 6.4: HTML

Screenshot IP address Ubuntu VM:

```
dylan@edu:~$ ip  a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host noprefixroute
       valid_lft forever preferred_lft forever
2: ens33: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:49:93:d7 brd ff:ff:ff:ff:ff:ff
    altname enp2s1
    inet 192.168.139.134/24 brd 192.168.139.255 scope global dynamic noprefixroute ens33
       valid_lft 1326sec preferred_lft 1326sec
    inet6 fe80::20c:29ff:fe49:93d7/64 scope link
       valid_lft forever preferred_lft forever
dylan@edu:~$
```

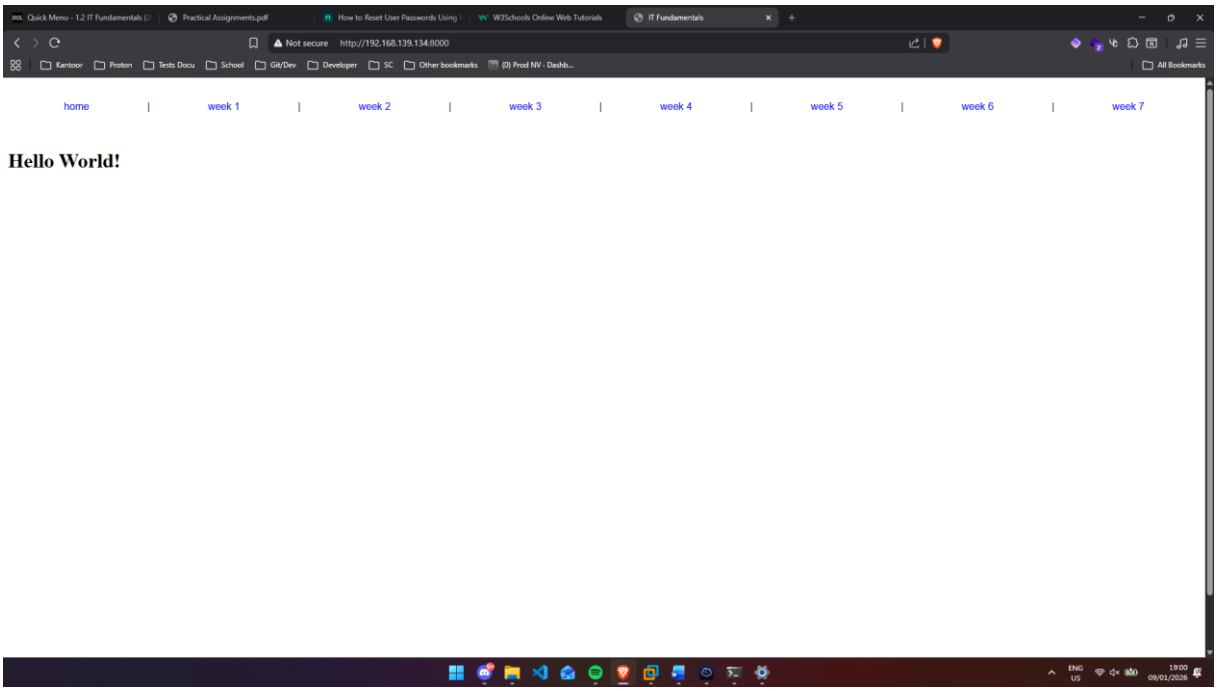
Screenshot of Site directory contents:



Screenshot python3 webserver command:

```
dylan@edu:~/Desktop/site$ python3 -m http.server 8000
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.139.1 - - [09/Jan/2026 18:59:14] "GET / HTTP/1.1" 200 -
192.168.139.1 - - [09/Jan/2026 18:59:15] "GET /css/mypdfstyle.css HTTP/1.1" 200 -
192.168.139.1 - - [09/Jan/2026 18:59:15] "GET /home.html HTTP/1.1" 200 -
192.168.139.1 - - [09/Jan/2026 18:59:15] code 404, message File not found
192.168.139.1 - - [09/Jan/2026 18:59:15] "GET /favicon.ico HTTP/1.1" 404 -
```

Screenshot web browser visits your site



Assignment 6.5: Network segment

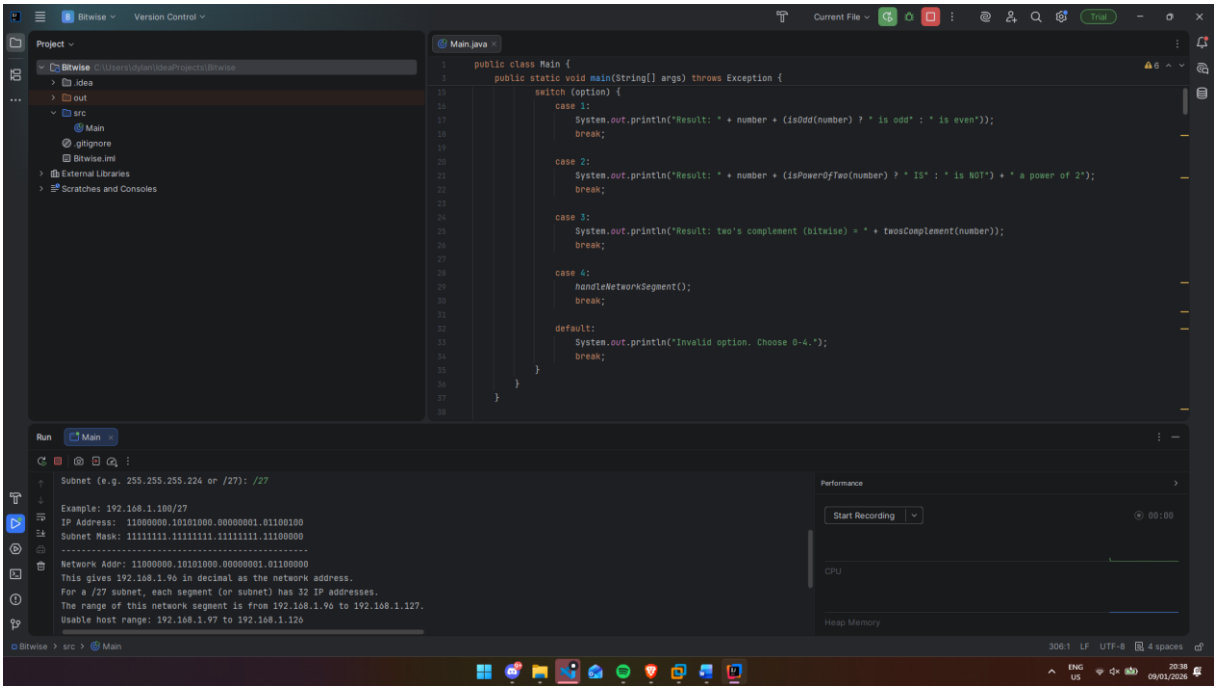
Remember that bitwise java application you've made in week 2? Expand that application so that you can also calculate a network segment as explained in the PowerPoint slides of week 6. Use the bitwise & AND operator. You need to be able to input two Strings. An IP address and a subnet.

IP: 192.168.1.100 and subnet: 255.255.255.224 for /27

```
Example: 192.168.1.100/27
Calculate the network segment
IP Address:   11000000.10101000.00000001.01100100
Subnet Mask:  11111111.11111111.11111111.11100000
-----------------------------------------------
Network Addr: 11000000.10101000.00000001.01100000

This gives 192.168.1.96 in decimal as the network address.
For a /27 subnet, each segment (or subnet) has 32 IP addresses (2^5).
The range of this network segment is from 192.168.1.96 to 192.168.1.127.
```

Paste source code here, with a screenshot of a working application.



```
public class Main {

    public static void main(String[] args) throws Exception {
        int number = readInt("Enter an integer: ");

        while (true) {
            printMenu();
            int option = readInt("Choose option: ");

            if (option == 0) {
                System.out.println("Bye!");
                break;
            }

            switch (option) {
                case 1:
                    System.out.println("Result: " + number + (isOdd(number) ? " is odd" : " is even"));
                    break;

                case 2:
                    System.out.println("Result: " + number + (isPowerOfTwo(number) ? " IS" : " is NOT") + " a power of 2");
                    break;

                case 3:
                    System.out.println("Result: two's complement (bitwise) = " + twosComplement(number));
                    break;

                case 4:
                    handleNetworkSegment();
```

```
                break;

            default:
                System.out.println("Invalid option. Choose 0-4.");
                break;
        }
    }
}

private static void printMenu() {
    System.out.println();
    System.out.println("Menu:");
    System.out.println("1) Is number odd?");
    System.out.println("2) Is number a power of 2?");
    System.out.println("3) Two's complement of number?");
    System.out.println("4) Calculate network segment (IP & Subnet)");
    System.out.println("0) Exit");
}

// Reads an integer from stdin (no Scanner).
private static int readInt(String prompt) throws Exception {
    while (true) {
        System.out.print(prompt);
        String line = readLine();
        line = trim(line);

        if (line.length() == 0) continue;

        boolean negative = false;
        int i = 0;

        if (line.charAt(0) == '-') {
            negative = true;
            i = 1;
            if (line.length() == 1) continue;
        }

        int value = 0;
        boolean ok = true;

        while (i < line.length()) {
            char c = line.charAt(i);
            if (c < '0' || c > '9') {
                ok = false;
                break;
            }
            int digit = c - '0';
            value = value * 10 + digit;
```

```
            i++;
        }

        if (!ok) {
            System.out.println("Please enter a valid integer.");
            continue;
        }

        return negative ? -value : value;
    }
  }

  private static String readString(String prompt) throws Exception {
    while (true) {
        System.out.print(prompt);
        String line = trim(readLine());
        if (line.length() == 0) continue;
        return line;
    }
  }

  // Reads a line from System.in using System.in.read().
  private static String readLine() throws Exception {
    StringBuilder sb = new StringBuilder();
    int ch;

    while ((ch = System.in.read()) != -1) {
        if (ch == '\r') continue;
        if (ch == '\n') break;
        sb.append((char) ch);
    }

    return sb.toString();
  }

  // Trim spaces/tabs without using imports
  private static String trim(String s) {
    int start = 0;
    int end = s.length() - 1;

    while (start <= end && isWhitespace(s.charAt(start))) start++;
    while (end >= start && isWhitespace(s.charAt(end))) end--;

    if (start > end) return "";
    return s.substring(start, end + 1);
  }

  private static boolean isWhitespace(char c) {
```

```
        return c == ' ' || c == '\t';
    }

    // 1) Odd if LSB is 1
    private static boolean isOdd(int n) {
        return (n & 1) != 0;
    }

    // 2) Power of 2: n > 0 and only one bit set
    private static boolean isPowerOfTwo(int n) {
        return n > 0 && (n & (n - 1)) == 0;
    }

    // 3) Two's complement using bitwise ops: ~n + 1
    private static int twosComplement(int n) {
        return (~n) + 1;
    }

    // -------------------------
    // Week 6: Network segment
    // -------------------------
    private static void handleNetworkSegment() throws Exception {
        String ipStr = readString("IP address (e.g. 192.168.1.100): ");
        String subnetStr = readString("Subnet (e.g. 255.255.255.224 or /27): ");

        int ip = parseIPv4(ipStr);
        int mask = parseSubnetToMask(subnetStr);
        int prefix = maskToPrefixStrict(mask);

        int network = ip & mask;              // bitwise AND as required
        int broadcast = network | (~mask);

        int hostBits = 32 - prefix;
        long totalAddresses = 1L << hostBits;

        System.out.println();
        System.out.println("Example: " + toDotted(ip) + "/" + prefix);
        System.out.println("IP Address:  " + toBinaryDotted(ip));
        System.out.println("Subnet Mask: " + toBinaryDotted(mask));
        System.out.println("-----------------------------------------------");
        System.out.println("Network Addr: " + toBinaryDotted(network));
        System.out.println("This gives " + toDotted(network) + " in decimal as the network address.");
        System.out.println("For a /" + prefix + " subnet, each segment (or subnet) has " + totalAddresses
+ " IP addresses.");

        System.out.println("The range of this network segment is from " + toDotted(network) + " to " +
toDotted(broadcast) + ".");
```

```
        if (prefix <= 30) {
            int firstHost = network + 1;
            int lastHost = broadcast - 1;
            System.out.println("Usable host range: " + toDotted(firstHost) + " to " + toDotted(lastHost));
        } else if (prefix == 31) {
            System.out.println("Usable host range: /31 is typically used for point-to-point (2 usable IPs).");
        } else {
            System.out.println("Usable host range: /32 is a single IP.");
        }

        System.out.println();
    }

    private static int parseSubnetToMask(String subnetStr) throws Exception {
        subnetStr = trim(subnetStr);

        if (subnetStr.length() == 0) throw new Exception("Subnet is empty.");

        if (subnetStr.charAt(0) == '/') {
            int prefix = parsePrefix(subnetStr.substring(1));
            return prefixToMask(prefix);
        }

        boolean onlyDigits = true;
        for (int i = 0; i < subnetStr.length(); i++) {
            char c = subnetStr.charAt(i);
            if (c < '0' || c > '9') {
                onlyDigits = false;
                break;
            }
        }

        if (onlyDigits && subnetStr.length() <= 2) {
            int prefix = parsePrefix(subnetStr);
            return prefixToMask(prefix);
        }

        return parseIPv4(subnetStr);
    }

    private static int parsePrefix(String s) throws Exception {
        s = trim(s);
        if (s.length() == 0) throw new Exception("Prefix is empty.");

        int value = 0;
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if (c < '0' || c > '9') throw new Exception("Invalid prefix: " + s);
```

```
            value = value * 10 + (c - '0');
        }

        if (value < 0 || value > 32) throw new Exception("Prefix must be 0..32");
        return value;
    }

    private static int prefixToMask(int prefix) {
        if (prefix == 0) return 0;
        int mask = (int) (0xFFFFFFFFL << (32 - prefix));
        return mask;
    }

    // Validate mask is contiguous 1s then 0s, return prefix length
    private static int maskToPrefixStrict(int mask) throws Exception {
        int ones = 0;
        boolean zeroSeen = false;

        for (int bit = 31; bit >= 0; bit--) {
            int b = (mask >>> bit) & 1;
            if (b == 1) {
                if (zeroSeen) throw new Exception("Subnet mask is not contiguous (invalid mask).");
                ones++;
            } else {
                zeroSeen = true;
            }
        }

        return ones;
    }

    private static int parseIPv4(String ip) throws Exception {
        ip = trim(ip);
        int part = 0;
        int dots = 0;
        int value = 0;
        boolean hasDigit = false;

        for (int i = 0; i < ip.length(); i++) {
            char c = ip.charAt(i);

            if (c >= '0' && c <= '9') {
                hasDigit = true;
                part = part * 10 + (c - '0');
                if (part > 255) throw new Exception("Octet out of range (>255).");
            } else if (c == '.') {
                if (!hasDigit) throw new Exception("Invalid IP format.");
                value = (value << 8) | part;
```

```
                dots++;
                part = 0;
                hasDigit = false;
            } else {
                throw new Exception("Invalid character in IP: " + c);
            }
        }

        if (dots != 3 || !hasDigit) throw new Exception("Invalid IP format.");
        value = (value << 8) | part;

        return value;
    }

    private static String toDotted(int v) {
        int a = (v >>> 24) & 0xFF;
        int b = (v >>> 16) & 0xFF;
        int c = (v >>> 8) & 0xFF;
        int d = v & 0xFF;
        return a + "." + b + "." + c + "." + d;
    }

    private static String toBinaryDotted(int v) {
        int a = (v >>> 24) & 0xFF;
        int b = (v >>> 16) & 0xFF;
        int c = (v >>> 8) & 0xFF;
        int d = v & 0xFF;
        return toBinary8(a) + "." + toBinary8(b) + "." + toBinary8(c) + "." + toBinary8(d);
    }

    private static String toBinary8(int n) {
        String s = Integer.toBinaryString(n & 0xFF);
        StringBuilder sb = new StringBuilder();
        for (int i = s.length(); i < 8; i++) sb.append('0');
        sb.append(s);
        return sb.toString();
    }
}
```

Ready? Save this file and export it as a pdf file with the name: **week6.pdf**