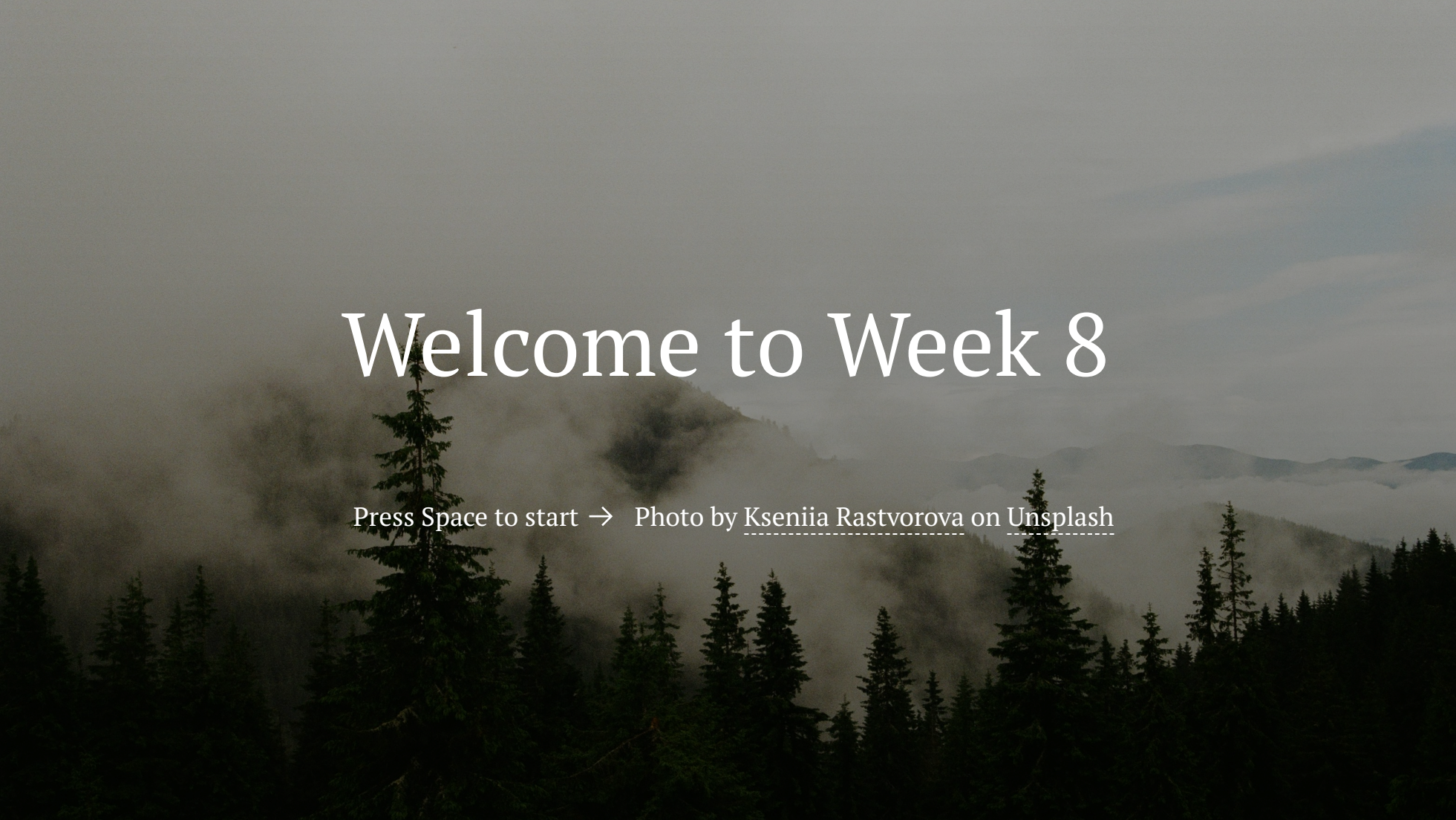
Welcome to Week 8

Press Space to start → 

Agenda for Today

- Announcements
- Quick comments on midterm
- `def`, `while`, `continue`, `break`
- bits and bytes
- Characters and colours using ASCII and Hexadecimal RGB

Announcements

- Putting more course content on GitHub going forward (readings, slides, due dates)
- Assignment 4 extension (now *due* Oct 30)

Quick comments on midterm

- Midterm marks and grading details: as soon as possible!
- Feedback?
- Don’t stress too much: some kind of adjustment is likely
- I will look closely at patterns in midterm answers to inform final exam prep materials

Python Functions with `def`

- `def` : Used to define a function in Python

Syntax:

```
def function_name(parameters):
    # code block
    return True
```

Example:

```
def greet(name):
    return f"Hello, {name}!"
```

- Functions:
  - Encapsulate code for reuse
  - Can accept parameters
  - Can return values (or `None` if no return statement)

Three more examples

Ex1:

```
def foo1(a,b):
  return a+b
```

Ex2:

```
def foo2(n):
  return n % 2 == 0
```

Ex3:

```
def foo3(n):
  result = 1
  for i in range(1, n + 1):
    result *= i
  return result
```

What do these do? What should we call them?

Better names

Ex1:

```
def add_two_nums(a,b):
  return a+b
```

Ex2:

```
def is_even(n):
  return n % 2 == 0
```

Ex3:

```
def factorial(n):
  result = 1
  for i in range(1, n + 1):
    result *= i
  return result
```

Python `while` Loop

- `while` **loop**: Repeats a block of code as long as a condition is `True`

Syntax:

```
while condition:
    # code block
```

Example:

```
i = 0
while i < 5:
    print(i)
    i += 1
```

Why Use a `while` Loop (or not!)?

- **Good for: Flexible Condition**:
  - Use `while` when the number of iterations is unknown.
  - Example: Waiting for user input or external event.
  - When the loop condition is more complex than simple iteration (e.g., event-based, real-time updates).

Example:

```
while not user_input:
    user_input = input("Enter something: ")
```

- **Watch out for:**:
  - Infinite Loops: Easier to create loops that run indefinitely until a condition is met.
  - Probably don't need while loops when you can define a specific range or collection to iterate over.

Python `continue`

- `continue`: Used inside a loop. Skips the rest of the code inside the loop for the current iteration and moves to the next iteration.
- **Use Case**:
  - When you want to skip certain iterations based on a condition.

Syntax:

```
while condition:
    if some_check:
        continue
    # remaining code for other iterations
```

Example:

```
for i in range(5):
    if i == 2:
        continue
    print(i)
```

Output: `0, 1, 3, 4` (skips `2`)

- **Key Point**:
  - Useful for skipping specific cases without stopping the loop.
  - Sometime if you "found what you’re looking for" you can continue

More Practical Examples of `continue`

Skip Even Numbers:

```
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)
```

- Output: `1, 3, 5, 7, 9` (skips even numbers)

Skip Blank Input:

```
inputs = ["hello", "", "world", "", "python"]
for item in inputs:
    if item == "":
        continue
    print(item)
```

Output: `hello, world, python` (skips empty strings)

- **Key Takeaway**:
  - `continue` helps ignore unwanted cases, allowing the loop to focus on relevant data.

Python `break`

- `break`: Immediately exits the loop, stopping further iterations.
- **Use Case**:
  - When you need to stop the loop based on a condition.
- **Syntax**:

```
while condition:
    if some_check:
        break
    # remaining code
```

Example:

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

Practical Examples of `break`

Stop on User Input:

```
while True:
    user_input = input("Enter a number (or 'q' to quit): ")
    if user_input == 'q':
        break
    print(f"You entered: {user_input}")
```

- The loop stops when the user enters `q`.

Find First Match in a List:

```
numbers = [3, 5, 7, 8, 10, 12]
for num in numbers:
    if num % 2 == 0:
        print(f"First even number found: {num}")
        break
```

Summary of our control flow tools

- we already know `if`, `for`, and `range` really well
- now we know `def`, `while`, `continue`, and `break`
- These are our core control flow tools for CMPT 120
- If you think you'll write a lot of Python, consider just reading the offical docs now: https://docs.python.org/3/tutorial/controlflow.html
  - These are also useful if you just want to read the sections about if, for, range, def, while, continue, break

Connecting back to bits and bytes

Let’s put it all together and transition back to bits and bytes!

We’ll work through a longer code example that uses `def`, `while`, `continue`, and `break`.

Task: write a program that takes an 8-bit binary number from user and prints out the answer in decimal. Now, are program will continue to run until user enters 'exit' (while loop + break) and will ask the user to fix their input if it is invalid (continue).

Step 1: Write a function to convert binary to decimal

```
def binary_to_decimal(binary_str):
    """Convert a binary string to a decimal number."""
    decimal = 0
    for digit in binary_str:
        decimal = decimal * 2 + int(digit)
    return decimal
```

Step 2: Write a function to convert decimal to ascii

(Turns out we can just use a Python built-in: effectively we're just giving it a new name)

```
def decimal_to_ascii(decimal):
    """Convert a decimal number to its ASCII character."""
    return chr(decimal)
```

Step 3: Use a while loop with break and continue

```
def process_binary_input():
    """Process binary input and handle conversion to decimal and ASCII."""
    while True:
        binary_input = input("Enter an 8-bit binary number (or 'exit' to quit): ")

        if binary_input.lower() == 'exit':
            print("Exiting...")
            break

        # Ensure input is 8 bits and binary
        if len(binary_input) != 8 or not all(bit in '01' for bit in binary_input):
            print("Invalid input! Please enter an 8-bit binary number.")
            continue

        # Convert binary to decimal
        decimal_value = binary_to_decimal(binary_input)

        # Convert decimal to ASCII
        ascii_char = decimal_to_ascii(decimal_value)

        print(f"Binary: {binary_input} -> Decimal: {decimal_value} -> ASCII: {ascii_char}")

# Call the function to start the loop
process_binary_input()
```

Put it together (live narrated example in VS Code)

How to Read Binary Code

Converting Binary to Decimal

- **Binary Representation**:
  - Binary is a base-2 system (only `0` and `1`).
  - Each binary digit (bit) represents a power of 2.
  - Rightmost digit is 1.
  - Second from right is 2.
  - Third from right 4. And so on.

Example:

Binary: 1011

```
1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 1 * 2^0
# 11
```

Step-by-Step Guide

- **Start from the rightmost bit**:
  - Assign powers of 2 (starting from 2^0).
- **Multiply each bit** by its corresponding power of 2.
- **Sum the results** to get the decimal number.

- Binary: `1011` - Decimal: `11`

Reading Binary

Power of 2	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
Value	128	64	32	16	8	4	2	1
Binary	1	0	1	1	0	1	0	1

Total: 128 + 0 + 32 + 16 + 0 + 4 + 0 + 1 = 181

What is ASCII?

- American Standard Code for Information Interchange
- ASCII is a table that maps decimal numbers to characters

ASCII , some random examples

Character	Binary	Decimal	Hexadecimal	Description
`A`	0100 0001	65	0x41	Uppercase A
`a`	0110 0001	97	0x61	Lowercase a
`0`	0011 0000	48	0x30	Number zero
` `	0010 0000	32	0x20	Space
`!`	0010 0001	33	0x21	Exclamation
`\n`	0000 1010	10	0x0A	New line

Note: ASCII uses 7 bits, allowing for 128 characters (0-127).
Extended ASCII uses 8 bits, allowing for 256 characters (0-255).

ASCII Table Sequence

Dec	Hex	Char	Description
32	0x20	` `	Space
33	0x21	`!`	Exclamation
34	0x22	`"`	Double quote
35	0x23	`#`	Hash
36	0x24	`$`	Dollar
37	0x25	`%`	Percent
38	0x26	`&`	Ampersand
39	0x27	`'`	Single quote
40	0x28	`(`	Open paren
41	0x29	`)`	Close paren

Dec	Hex	Char	Description
42	0x2A	`*`	Asterisk
43	0x2B	`+`	Plus
44	0x2C	`,`	Comma
45	0x2D	`-`	Hyphen
46	0x2E	`.`	Period
47	0x2F	`/`	Forward slash
48	0x30	`0`	Zero
49	0x31	`1`	One
50	0x32	`2`	Two
51	0x33	`3`	Three

Python Examples

```
# Get ASCII code from character
print(ord('A'))  # Output: 65
print(ord('!'))  # Output: 33

# Get character from ASCII code
print(chr(65))   # Output: A
print(chr(33))   # Output: !

# String of ASCII characters
print(''.join(chr(i) for i in range(65, 91)))  # A-Z
```

Decode the Binary Message! 🔍

```
01001000 01101001 00100001 00100000 01011001
01101111 01110101 00100000 01110010 01101111
01100011 01101011 00100001
```

💡 Hint: Each character is 8 bits (1 byte)

Could try grabbing scratch paper and looking up the ASCII table or...

Or: convert 01001000 -> 72 -> H.

```
def binary_to_decimal(binary_str):
    """Convert a binary string to a decimal number."""
    decimal = 0
    for digit in binary_str:
        decimal = decimal * 2 + int(digit)
    return decimal

binary_to_decimal("01001000")
chr(72)
```

Solution: Hi! You rock!

Hexadecimal

Hexadecimal is a numbering system that's often used for RGB colors.

- Binary uses powers of 2, so we only need two characters (0 and 1)
- Decimal uses power of 10, so we need ten characters
- Hexadecimal uses power of 16…

But we only have 10 digits?

Taking 6 extra characters from the alphabet

In hexadecimal, there are 16 digits (we use A - F for numbers 11-16).

- 0000 (binary) <> 0 (decimal) <> 0 (hexadecimal)
- 0001 (binary) <> 1 (decimal) <> 1 (hexadecimal)
- 0010 (binary) <> 2 (decimal) <> 2 (hexadecimal)

Ok, it’s all the same until…

- 1010 (binary) <> 10 (decimal) <> A (hexadecimal)
- 1011 (binary) <> 11 (decimal) <> B (hexadecimal)
- 1100 (binary) <> 12 (decimal) <> C (hexadecimal)
- 1101 (binary) <> 13 (decimal) <> D (hexadecimal)
- 1110 (binary) <> 14 (decimal) <> E (hexadecimal)
- 1111 (binary) <> 15 (decimal) <> F (hexadecimal)

(TLDR: A = 10, B = 11, and so on)

Example of hexadecimal

00FFAA is a green-blue color

00 means "0 red"

FF means "255 Green"

AA means "170 blue"

Why is FF 255?

FF -> 15 * (16 ** 1) + 15 (16 ** 0) = 225 + 16 = 240 + 15 = 255

Importance of Grouping

If you just know that 1 byte is 8 bits, that binary codes used powers of 2, and that and hexadecimal uses powers of 16 and the characters A-F, you can solve a lot of problems from "first principles".

However we also need to learn common practices around grouping:

- when storing text using an 8-bit character set like ASCII, bits are grouped into 8 bits so a string like "01001000 01101001" can be converted to two characters
- when storing colors as RGB hexadecimal strings, we only need 4 bits per character (only 16 possible characters)
- How many bytes for the string "AB"? 8 times 2 equals 16
- How many bytes for the color 00FFAA? 4 times 6 equals 24.

Some additional topics of interest: unicode, mojibake

About Unicode

- Unicode is a superset of ASCII.
- A is a super of B means A includes B, but also other stuff.
- ASCII (American Standard Code for Information Interchange) defines 128 characters, including English letters, digits, and control characters, using 7 bits per character.
- Unicode, on the other hand, was designed to encompass more characters from more writing systems worldwide, using up to 32 bits per character. The first 128 characters in Unicode are identical to ASCII, ensuring backward compatibility.

What happens if we have a text file encoded as Unicode, but we tell our computer it's ASCII?

Garbled text: called Mojibake!

For fun: see Wikipedia on this https://en.wikipedia.org/wiki/Mojibake

One more example connecting back to our coding skills

```
dec = ["65", "66", "67"]
chars = ["A", "B", "C"]
characters_as_dec = ["65", "65", "65", "67", "67", "67"]
for i in range(len(characters_as_dec)):
    for j in range(len(dec)):
        if characters_as_dec[i] == dec[j]:
            print(chars[j])
```

Outcomes

What do we need to know here?

1. Knows what a bit represents in a computer
2. Convert a given number of bytes (or kb, mb) to bits
3. Convert binary <> decimal
4. Understand the goal of hexadecimal
5. Know that binary numbers convert to hexadecimal by grouping of 4 bits
6. Know the purpose of ASCII
7. Know relationship of Unicode to ASCII
8. Know how to store RGB numbers (3 byte aka 24 bits aka 6 hexadecimal digits)