CS-121: Week 4 / Lecture 2

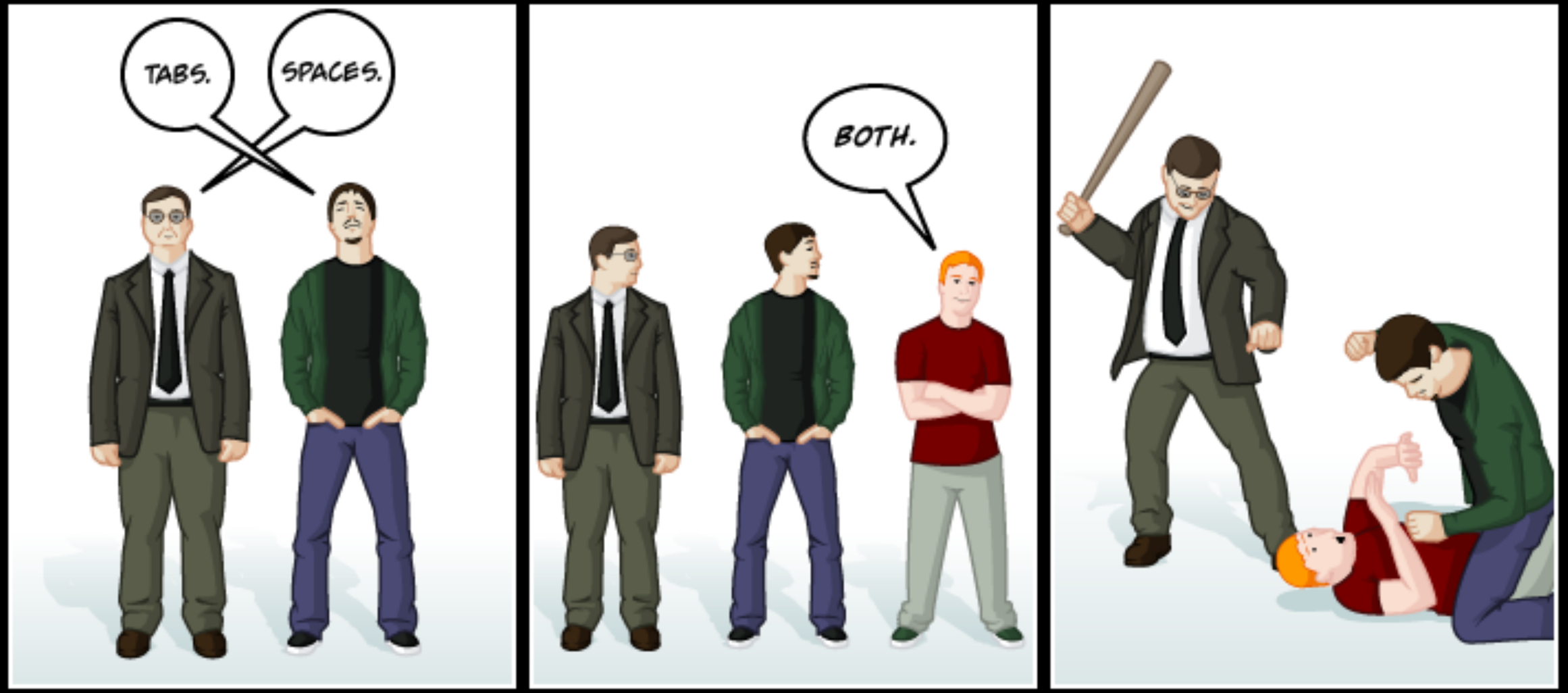
🌟 Lists & Loops

Upcoming

- **Today**: Lists & Loops + Next Project
- **Office Hours**: Tues/Thurs 4:30pm - 5:30pm
- **Next week**: Final week of unit 1!

 Code Blocks and Indentation

- Python uses indentation to define **blocks** of code.
- A block is a group of statements that execute together.
- Indentation is typically 4 spaces or one **Tab**.
- Ensures code is organized and readable.


TABS.
SPACES.
BOTH.


I DON'T KNOW IF
I USE TABS OR SPACES
VS CODE USER
AND AT THIS POINT
I'M TOO AFRAID TO ASK
imgflip.com

Example: Code Blocks with Indentation

```
if condition:
    # This is inside the block
    do_something()
    do_something_else()
# This is outside the block
do_another_thing()
```

- Lines indented after `if` are part of the code block.
- Unindented lines are outside the block.

🌐 Local vs. Global Variables

- **Global Variable**: Defined outside of all functions, accessible anywhere.
- **Local Variable**: Defined within a function or block, accessible only there.

Example: Variable Scope

```
x = "global"

def my_function():
    x = "local"
    print("Inside function:", x)

my_function()
print("Outside function:", x)
```

Output:

```
Inside function: local
Outside function: global
```

✨ New Tools

Loops and Friends of Loops

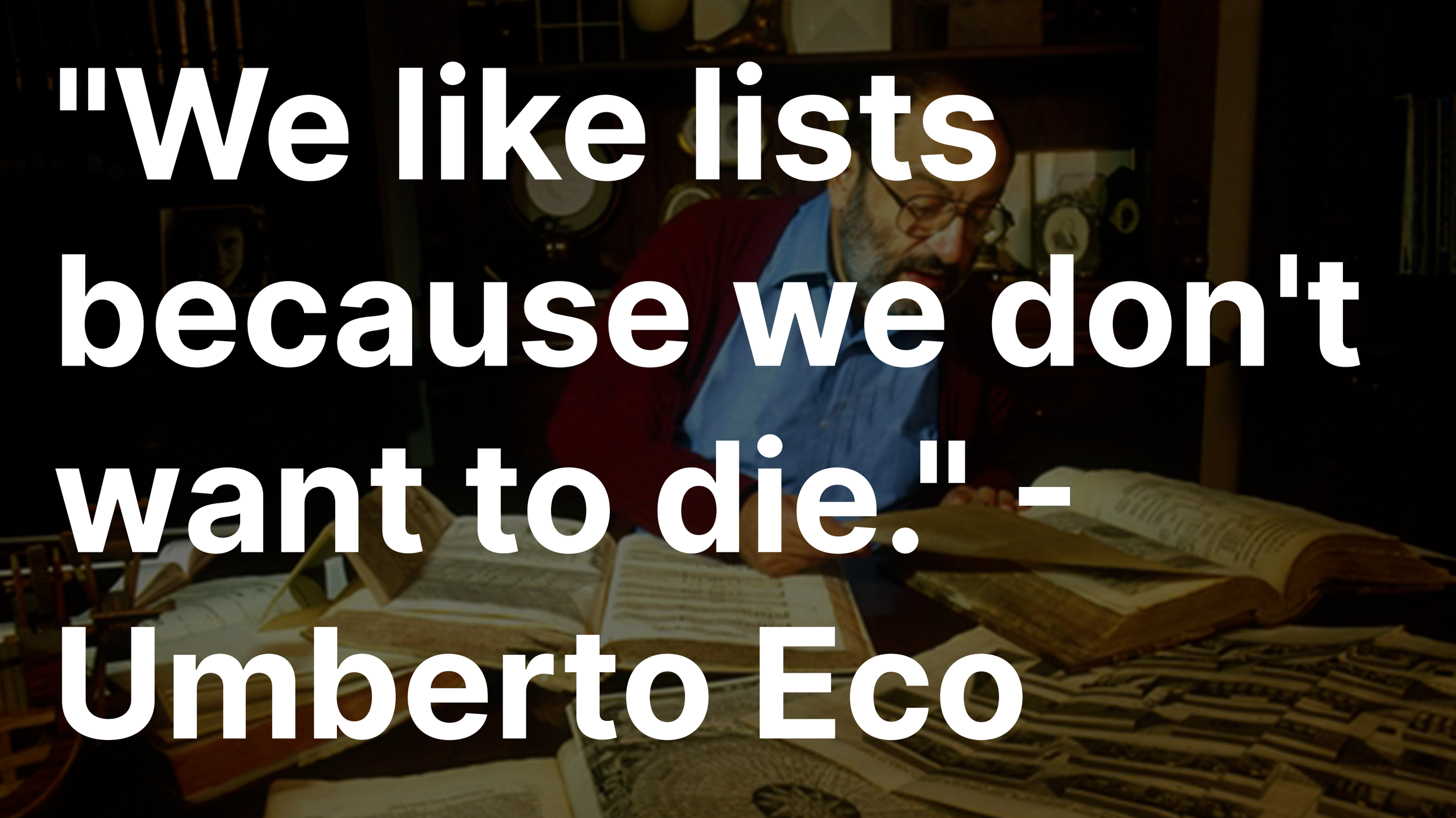
- Loops: `for` and `while`
- Lists
- `print()` with `end` parameter
- `range()` function
- Random numbers

Lists

- A **list** is a collection of items in a particular order.
- Lists are **mutable** (they can be changed).
- Defined using square brackets `[]`.

✅ Lists in the Real World?

- Shopping list
- To-do list
- List of friends
- Class list
- Book list
- Christmas list
- …


"We like lists because we don't want to die." - Umberto Eco

Lists in Apps

- **Spotify**: List of songs in a playlist.
- **Twitter**: List of tweets in a thread.
- **Instagram**: List of photos in a post.
- **Netflix**: List of movies in a category.
- **Amazon**: List of products in a cart.
- **Google Maps**: List of places in a route.
- **Pathwright**: List of steps.

Creating and Accessing Lists in Python

```
# Creating a list
fruits = ["apple", "banana", "cherry"]

# Accessing elements by index
print(fruits[0])  # Outputs: apple
print(fruits[1])  # Outputs: banana
```

Loops in Programming

- **Loops** allow us to repeat a block of code multiple times.
- Two main types in Python:
  - `for` loops
  - `while` loops

`for` Loops

- Used to iterate over a **sequence** (like a list, tuple, string).
- Syntax:

```
for variable in sequence:
    # Code block to execute
```

Example: `for` Loop

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(f"I like {fruit}")
```

Output:

```
I like apple
I like banana
I like cherry
```

`range()` Function

- Generates a sequence of numbers.
- Commonly used with `for` loops.

```
for i in range(5):
    print(i)
```

Output:

```
0
1
2
3
4
```

Using `range()` with Start and End

```
for i in range(1, 6):
    print(i)
```

Output:

1
2
3
4
5

`while` Loops

- Keep executing a block of code as long as a condition is `True`.
- Syntax:

```
while condition:
    # Code block to execute
```

Example: `while` Loop

```
count = 1
while count <= 5:
    print(f"Count is {count}")
    count += 1
```

Output:

```
Count is 1
Count is 2
Count is 3
Count is 4
Count is 5
```

Loop Control Statements

- `break`: Exit the loop immediately.
- `continue`: Skip to the next iteration.

Breaking Out of a Loop with `break`

```
for num in range(1, 10):
    if num == 5:
        break
    print(num)
```

Output:

1
2
3
4

Skipping Iterations with `continue`

```
for num in range(1, 6):
    if num == 3:
        continue
    print(num)
```

Output:

```
1
2
4
5
```

🛑 Avoiding Infinite Loops

- Ensure your loop has a condition that will eventually be `False`.
- Be cautious with `while True` loops.
- Use `break` statements wisely.

```python
count = 0
while True:
    count += 1
    print(f"Loop {count}")
```

```python
count = 0
max_count = 1000
while count < max_count: # Add a condition to stop the loop
    count += 1
    print(f"Loop {count}")
```

```python
count = 0
max_count = 1000
while True:
    count += 1
    print(f"Loop {count}")
    if count >= max_count:
        break # Exit the loop when count reaches max_count
```

Real-World Examples of Loops

- Automate repetitive tasks.
- Process data structures like lists.
- Control program flow dynamically.

Example: Summing Numbers

```
numbers = [1, 2, 3, 4, 5]
total = 0
for num in numbers:
    total += num
print(f"Total sum: {total}")
```

Output:

```
Total sum: 15
```

Example: Capitalizing Names

```
names = ["alice", "bob", "charlie"]
capitalized_names = []
for name in names:
    capitalized_names.append(name.title())
print(capitalized_names)
```

Output:

```
['Alice', 'Bob', 'Charlie']
```

Example: Validating User Input

Prompt the user to enter a number between 1 and 10. Keep prompting until they provide a valid number.

```
number = int(input("Enter a number between 1 and 10: "))
while number < 1 or number > 10:
    print("Invalid input.")
    number = int(input("Please enter a number between 1 and 10: "))
print(f"You entered {number}")
```

Explanation:

- Uses a `while` loop to keep asking the user until they provide a number between 1 and 10.
- Demonstrates input validation using loops.

🖨️ Print with `end` Parameter

- By default, `print()` adds a newline character at the end.
- You can change this behavior using the `end` parameter.

```
print("Hello", end=" ")
print("World")
```

Output: `Hello World`

Example: Printing a Grid

```
size = 3
for _ in range(size):
    for _ in range(size):
        print("*", end=" ")
    print()
```

Output:

```
* * *
* * *
* * *
```

 Using Random Numbers

- Random numbers can make programs more dynamic.
- Python's `random` module allows us to generate random numbers.

Importing the `random` Module

- To use the `random` module, you need to import it:
  `import random`

Generating Random Numbers

- **`random.randint(a, b)`**: Returns a random integer N such that `a <= N <= b`.

```
import random
num = random.randint(1, 10)
print(num)
```

Using Random Numbers in Loops

- Random numbers can control loop execution or generate random data in loops.
- Example: Generating a list of random numbers.

```
import random
random_numbers = []
for _ in range(5):
    num = random.randint(1, 100)
    random_numbers.append(num)
print(random_numbers)
```

Example Output:

```
[23, 85, 67, 42, 19]
```

🧩 Code Reading Puzzles

Puzzle 1: Guess the Output

```
for i in range(1, 6):
    if i == 3:
        break
    print(i)
```

- **What is the output of this code?**

Puzzle 1: Explanation

Output:

```
1
2
```

- **Explanation**:
  - The loop runs from `i = 1` to `5`.
  - When `i == 3`, `break` exits the loop.
  - So only `1` and `2` are printed.

Puzzle 2: Guess the Output

```
i = 0
while i < 5:
    i += 1
    if i % 2 == 0:
        continue
    print(i)
```

- **What is the output of this code?**

Puzzle 2: Explanation

Output:

```
1
3
5
```

- **Explanation**:
  - The loop increments `i` from `1` to `5`.
  - When `i` is even (`i % 2 == 0`), `continue` skips the `print` statement.
  - So only odd numbers are printed.

Puzzle 3: Guess the Output (Variable Scope)

```
def outer_function():
    x = 'outer'
    def inner_function():
        x = 'inner'
        print(x)
    inner_function()
    print(x)

x = 'global'
outer_function()
print(x)
```

- **What is the output of this code?**

Puzzle 3: Explanation

Output:

```
inner
outer
global
```

```
def outer_function():
    x = 'outer'
    def inner_function():
        x = 'inner'
        print(x)
    inner_function()
    print(x)

x = 'global'
outer_function()
print(x)
```

- **Explanation**:
  - `x` in `inner_function` is a local variable, separate from `x` in `outer_function`.
  - `x` in `outer_function` retains its value `'outer'`.
  - `x` in the global scope remains `'global'`.

🚀 Let's Code!

- Open your code notebook.
- Try the practice exercises together:
  - **Even/Odd Numbers**
  - **Guess the Number Game**

✅ Challenge: Even/Odd Numbers

Write a program that prints the numbers from 1 to 20.

- For each number, if it is even, print `"X is even"`.
- If it is odd, print `"X is odd"`, where X is the number.

Hint: Use a for loop, `range`, `if-else` conditions, and the modulo operator `%`.

📝 Even/Odd Solution Step-by-Step

```
for num in range(1, 21):
    if num % 2 == 0:
        print(f"{num} is even")
    else:
        print(f"{num} is odd")
```

- **Step 1**: Loop from 1 to 20 using `range(1, 21)`.
- **Step 2**: Use `if` to check if `num` is divisible by 2.
- **Step 3**: Print the appropriate message depending on whether `num` is even or odd.

🎮 Challenge: Guess the Number Game

Create a guessing game where the computer selects a random number between 1 and 10, and the user tries to guess it.

- Generate a random number between 1 and 10 using `random.randint(1, 10)`.
- Use a `while` loop to allow the user to keep guessing until they get it right.
- Provide feedback: Tell the user if their guess is too low or too high.
- When the user guesses correctly, exit the loop and

📝 Guess the Number Solution Step-by-Step

```
import random
number_to_guess = random.randint(1, 10)
guess = int(input("Guess a number between 1 and 10: "))
while guess != number_to_guess:
    if guess < number_to_guess:
        print("Too low.")
    else:
        print("Too high.")
    guess = int(input("Guess again: "))
print("Congratulations! You guessed the number.")
```

- **Step 1**: Import the `random` module and generate a random number.
- **Step 2**: Ask the user for their first guess.
- **Step 3**: Use a `while` loop to keep asking until the guess matches the number.
- **Step 4**: Provide feedback after each incorrect guess.
- **Step 5**: Congratulate the user upon the correct guess.

🐶 Project 4: Pet.py v2

- A real game loop
- Actions + choices
- Random events
- Refactor and iterate

Questions?

👋 See you next time!