

# IE 5390 – Assignment - 10

Name: Simran Abhay Sinha

## Lab 19

1. Write code to iterate the first 10 numbers and in each iteration, print the sum of the current and previous number.

```
#lab19_1
# Initialize previous number
previous_num = 0

print("Printing sum of current and previous number in a range(10):")

# Iterate through the first 10 numbers
for i in range(10):
    # Calculate sum of current and previous number
    sum_nums = i + previous_num
    print("Current Number:", i, "Previous Number:", previous_num, "Sum:", sum_nums)

    # Update previous number to current number
    previous_num = i
```

The screenshot shows a Jupyter Notebook window with two code cells. Cell 2 contains the provided Python code for calculating the sum of current and previous numbers. Cell 3 contains a different piece of code related to substring counting.

```
[2]: #lab19_1
# Initialize previous number
previous_num = 0

print("Printing sum of current and previous number in a range(10):")

# Iterate through the first 10 numbers
for i in range(10):
    # Calculate sum of current and previous number
    sum_nums = i + previous_num
    print("Current Number:", i, "Previous Number:", previous_num, "Sum:", sum_nums)

    # Update previous number to current number
    previous_num = i

Printing sum of current and previous number in a range(10):
Current Number: 0 Previous Number: 0 Sum: 0
Current Number: 1 Previous Number: 0 Sum: 1
Current Number: 2 Previous Number: 1 Sum: 3
Current Number: 3 Previous Number: 2 Sum: 5
Current Number: 4 Previous Number: 3 Sum: 7
Current Number: 5 Previous Number: 4 Sum: 9
Current Number: 6 Previous Number: 5 Sum: 11
Current Number: 7 Previous Number: 6 Sum: 13
Current Number: 8 Previous Number: 7 Sum: 15
Current Number: 9 Previous Number: 8 Sum: 17
```

```
[3]: #lab19_2
# Given string
str_x = """Human health can be monitored using several types of data. The data collected from scans, labs, and procedures can be used for research purposes. The key to using data effectively is .... """

# Count occurrences of substring "data"
count_data = str_x.count("data")

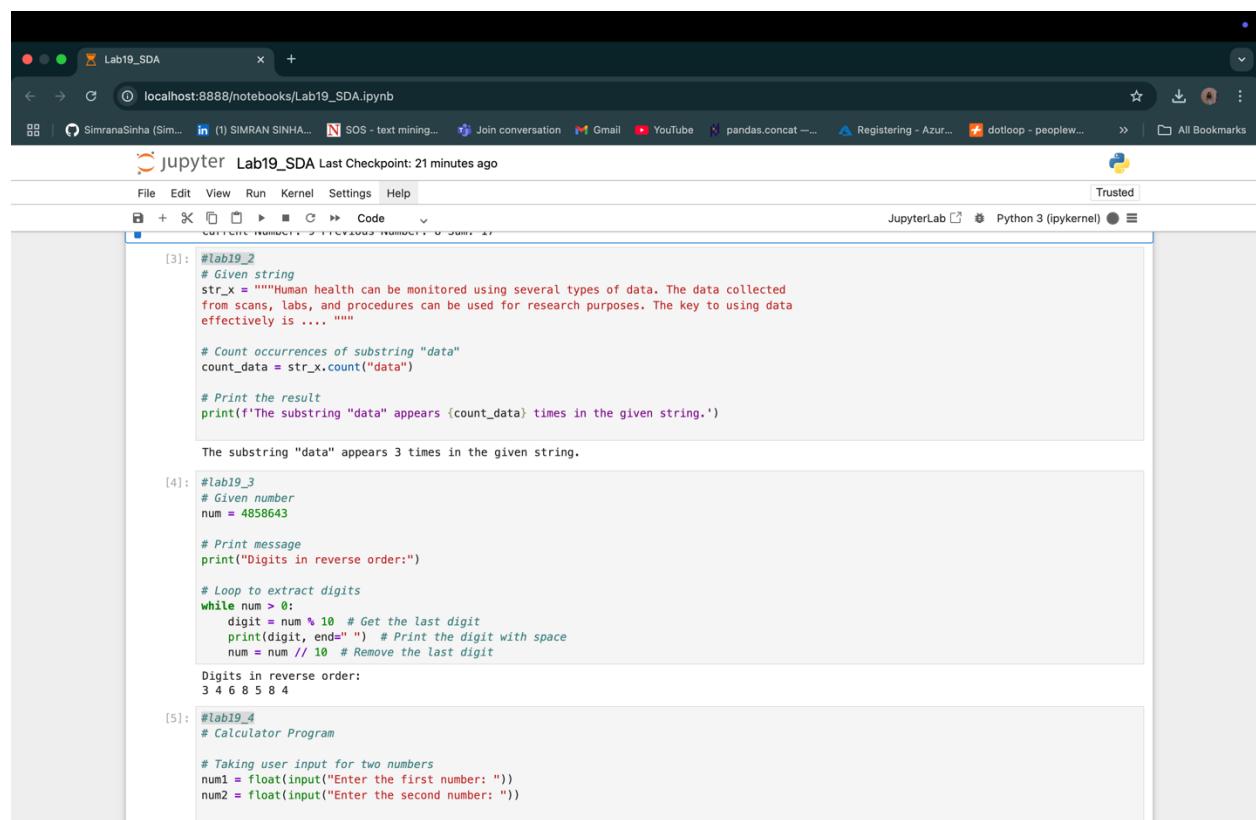
# Print the result
```

2. Write code to find how many times substring “data” appears in the given string:  
 str\_x = "Human health can be monitored using several types of data. The data collected from scans, labs, and procedures can be used for research purposes. The key to using data effectively is .... "  
 CURRENT NUMBER: 3 PREVIOUS NUMBER: 0 SAME: 17

```
[3]: #lab19_2
# Given string
str_x = """Human health can be monitored using several types of data. The data collected from scans, labs, and procedures can be used for research purposes. The key to using data effectively is .... """

# Count occurrences of substring "data"
count_data = str_x.count("data")

# Print the result
print(f'The substring "data" appears {count_data} times in the given string.')
```



```
[3]: #lab19_2
# Given string
str_x = """Human health can be monitored using several types of data. The data collected from scans, labs, and procedures can be used for research purposes. The key to using data effectively is .... """

# Count occurrences of substring "data"
count_data = str_x.count("data")

# Print the result
print(f'The substring "data" appears {count_data} times in the given string.')
The substring "data" appears 3 times in the given string.

[4]: #lab19_3
# Given number
num = 4858643

# Print message
print("Digits in reverse order:")

# Loop to extract digits
while num > 0:
    digit = num % 10 # Get the last digit
    print(digit, end=" ") # Print the digit with space
    num = num // 10 # Remove the last digit

Digits in reverse order:
3 4 6 8 5 8 4

[5]: #lab19_4
# Calculator Program

# Taking user input for two numbers
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))
```

3. Write python code to extract each digit from the integer 4858643 in the reverse order. The result should display “3 4 6 8 5 8 4” with a space separating the digits.

```
#lab19_3
# Given number
num = 4858643

# Print message
print("Digits in reverse order:")

# Loop to extract digits
while num > 0:
    digit = num % 10 # Get the last digit
    print(digit, end=" ") # Print the digit with space
    num = num // 10 # Remove the last digit
```

The screenshot shows a Jupyter Notebook interface with two code cells. Cell [4] displays code to count the occurrences of the substring "data" in a string "x". Cell [5] displays the provided Python code to print digits of a number in reverse order. The output of Cell [5] shows the digits 3 4 6 8 5 8 4 printed in reverse order.

```
[4]: # Count occurrences of substring "data"
count_data = str_x.count("data")

# Print the result
print(f'The substring "data" appears {count_data} times in the given string.')

The substring "data" appears 3 times in the given string.

[5]: #lab19_3
# Given number
num = 4858643

# Print message
print("Digits in reverse order:")

# Loop to extract digits
while num > 0:
    digit = num % 10 # Get the last digit
    print(digit, end=" ") # Print the digit with space
    num = num // 10 # Remove the last digit

Digits in reverse order:
3 4 6 8 5 8 4
```

4. Calculator program: Create a python application that takes user input for two numbers and performs basic calculations: addition, subtraction, division, and multiplication.

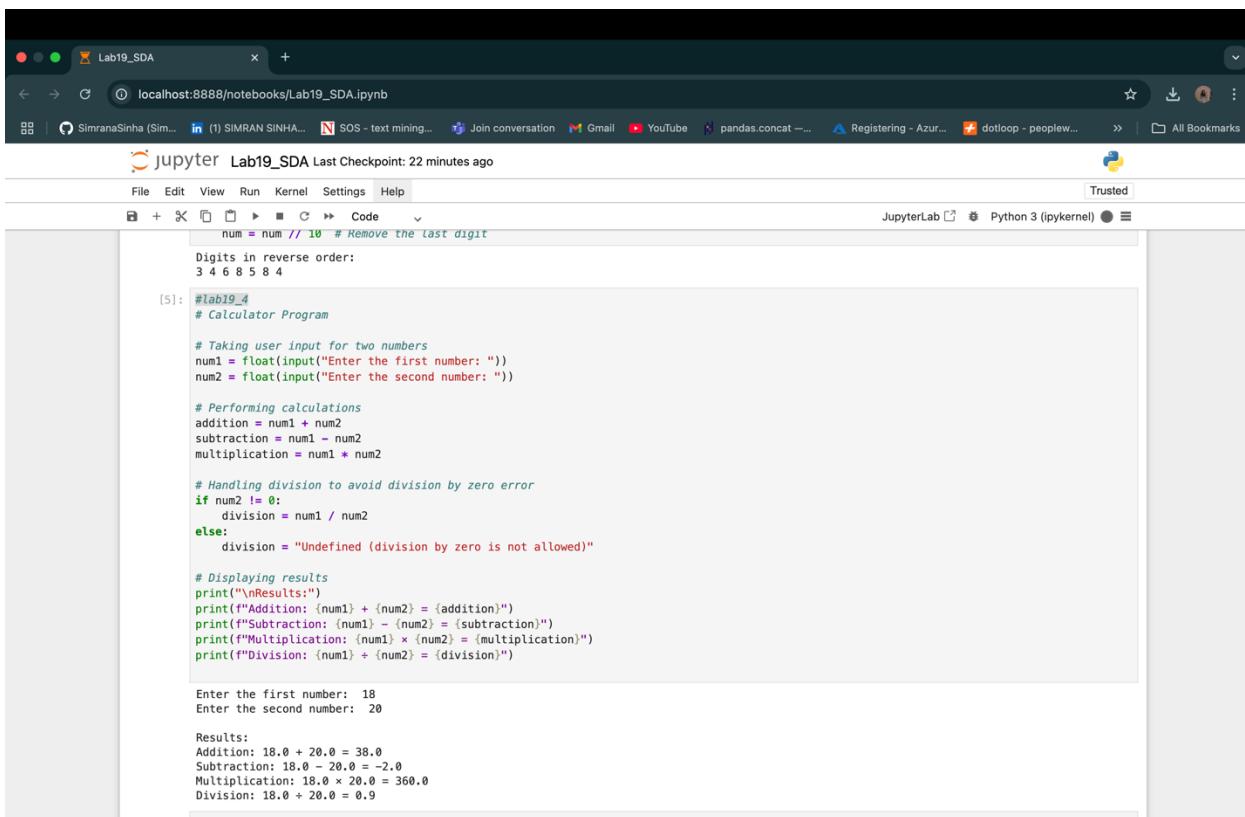
```
#lab19_4
# Calculator Program

# Taking user input for two numbers
num1 = float(input("Enter the first number: "))
num2 = float(input("Enter the second number: "))

# Performing calculations
addition = num1 + num2
subtraction = num1 - num2
multiplication = num1 * num2

# Handling division to avoid division by zero error
if num2 != 0:
    division = num1 / num2
else:
    division = "Undefined (division by zero is not allowed)"

# Displaying results
print("\nResults:")
print(f"Addition: {num1} + {num2} = {addition}")
print(f"Subtraction: {num1} - {num2} = {subtraction}")
print(f"Multiplication: {num1} × {num2} = {multiplication}")
print(f"Division: {num1} ÷ {num2} = {division}")
```



5. Display Fibonacci series up to 10 terms. The Fibonacci Sequence is a series of numbers. The next number is found by adding up the two numbers before it. The first two numbers are 0 and 1.

```
#lab19_5
# Initialize the first two numbers
num1, num2 = 0, 1

print("Fibonacci Series up to 10 terms:")

# Loop to generate 10 terms
for _ in range(10):
    print(num1, end=" ") # Print the current term
    res = num1 + num2 # Calculate the next term
    num1, num2 = num2, res # Update values for the next iteration
```

The screenshot shows a Jupyter Notebook interface with several code cells and their outputs:

- Cell 1:** Contains Python code for basic arithmetic operations (Addition, Subtraction, Multiplication, Division) with inputs 18 and 20.
- Cell 2:** Contains the Fibonacci series code from the question, resulting in the output: Fibonacci Series up to 10 terms: 0 1 1 2 3 5 8 13 21 34
- Cell 3:** Contains Python code for a Fractal Tree Generator, defining a class `FractalTreeApp` with methods for initialization and root node setup.

**6. Gen AI Prompt Engineering:** Ask your AI Bot to create and run a cool Python App. Ask it to explain code

**Bot Used: Claude.Ai**

Bot:

The screenshot shows the Claude AI interface with a chat window and a code editor.

**Chat Window (Left):**

- Prompt: "Create and run a cool Python App and also explain the code using python"
- Response: "I'll create a Python app that generates a customizable fractal tree visualization. This is a fun way to demonstrate recursion and create interesting artwork."
- Panel: "Fractal Tree Generator" (with "Click to open code")
- Text: "This Python application creates an interactive fractal tree generator with a user-friendly GUI. Let me explain the key components:"
- Core Concepts** (List):
  - Fractals: Self-similar patterns that repeat at different scales - perfect for demonstrating recursion.
  - Turtle Graphics: Used for drawing the tree with simple forward/turn commands.
  - Tkinter: Provides the GUI components for user interaction.
- Code Breakdown** (List):
  - Class Structure: The `FractalTreeApp` class encapsulates all functionality.
  - GUI Components:
    - Left panel with sliders to control tree parameters
- Text: "tree should have flowers"
- Bottom: "Claude 3.7 Sonnet" and "Choose style"

```
← Fractal Tree Generator
import math
import turtle
import random
import tkinter as tk
from tkinter import ttk, colorchooser

class FractalTreeApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Fractal Tree Generator")
        self.root.geometry("800x600")

    # Setup main frames
    self.control_frame = ttk.Frame(root, padding="10")
    self.control_frame.pack(side=tk.LEFT, fill=tk.Y)

    self.canvas_frame = ttk.Frame(root)
    self.canvas_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

    # Create turtle canvas
    self.canvas = tk.Canvas(self.canvas_frame, bg="white")
    self.canvas.pack(fill=tk.BOTH, expand=True)

    # Create turtle screen
    self.screen = turtle.TurtleScreen(self.canvas)
    self.screen.tracer(0) # Turn off animation for speed
    self.turtle = turtle.RawTurtle(self.screen)

Last edited 19 minutes ago
```

The screenshot shows the Claude AI interface with a chat window and a code editor.

**Chat Window (Left):**

- Prompt: "Create and run a cool Python App and also explain the code using python"
- Response: "I'll create a Python app that generates a customizable fractal tree visualization. This is a fun way to demonstrate recursion and create interesting artwork."
- Panel: "Fractal Tree Generator" (with "Click to open code")
- Text: "This Python application creates an interactive fractal tree generator with a user-friendly GUI. Let me explain the key components:"
- Core Concepts** (List):
  - GUI Components:
    - Left panel with sliders to control tree parameters
    - Right panel with canvas for drawing the tree
    - Color picker and randomize option
  - Key Parameters:
    - Iterations: Controls the depth of recursion (how many branches)
    - Branch Length: Initial length of the trunk
    - Angle: Determines how much branches spread
    - Reduction: How much each sub-branch shrinks
  - The Drawing Algorithm:
    - `draw_tree()` is the recursive function that creates the fractal
    - Each branch splits into two smaller branches
    - The function calls itself for each new branch until it reaches the iteration limit
  - Special Features:
    - Branch thickness varies with iteration level
    - Color darkens for thicker branches
    - Option to randomize all parameters for creative exploration
- To Run This Code**
- Text: "tree should have flowers"
- Bottom: "Claude 3.7 Sonnet" and "Choose style"

```
← Fractal Tree Generator
self.screen.tracer(0) # Turn off animation for speed
self.turtle = turtle.RawTurtle(self.screen)
self.turtle.hideturtle()
self.turtle.speed(0)

# Control variables
self.iterations = tk.IntVar(value=8)
self.branch_length = tk.DoubleVar(value=100.0)
self.angle = tk.DoubleVar(value=30.0)
self.reduction = tk.DoubleVar(value=0.75)
self.branch_color = "#000000" # Default green
self.randomize_var = tk.BooleanVar(value=False)

# Create controls
self.create_controls()

def create_controls(self):
    ttk.Label(self.control_frame, text="Fractal Tree Controls", font=("Arial", 12)).pack(pady=10)

    # Iterations slider
    ttk.Label(self.control_frame, text="Iterations:").pack(anchor=tk.W, padx=(10, 0))
    iterations_slider = ttk.Scale(self.control_frame, from_=1, to=12,
                                 variable=self.iterations, orient=tk.HORIZONTAL,
                                 command=lambda value: self.update_branch_params())
    iterations_slider.pack(fill=tk.X, pady=5)

    # Branch length slider
    ttk.Label(self.control_frame, text="Initial Branch Length:").pack(anchor=tk.W, padx=(10, 0))
    branch_length_slider = ttk.Scale(self.control_frame, from_=10, to=100,
                                    variable=self.branch_length, orient=tk.HORIZONTAL,
                                    command=lambda value: self.update_branch_params())
    branch_length_slider.pack(fill=tk.X, pady=5)

    # Angle slider
    ttk.Label(self.control_frame, text="Angle:").pack(anchor=tk.W, padx=(10, 0))
    angle_slider = ttk.Scale(self.control_frame, from_=0, to=90,
                           variable=self.angle, orient=tk.HORIZONTAL,
                           command=lambda value: self.update_branch_params())
    angle_slider.pack(fill=tk.X, pady=5)

    # Reduction slider
    ttk.Label(self.control_frame, text="Reduction:").pack(anchor=tk.W, padx=(10, 0))
    reduction_slider = ttk.Scale(self.control_frame, from_=0.1, to=1.0,
                                variable=self.reduction, orient=tk.HORIZONTAL,
                                command=lambda value: self.update_branch_params())
    reduction_slider.pack(fill=tk.X, pady=5)

    # Randomize button
    randomize_button = ttk.Button(self.control_frame, text="Randomize", command=self.randomize_params)
    randomize_button.pack(pady=10)

    # Color picker
    color_picker = tk.colorchooser.ColorButton(self.control_frame, initialcolor="#000000", command=self.set_branch_color)
    color_picker.pack(pady=10)

    # Buttons
    self.generate_button = ttk.Button(self.control_frame, text="Generate", command=self.generate_fractal)
    self.generate_button.pack(pady=10)

    self.clear_button = ttk.Button(self.control_frame, text="Clear", command=self.clear_fractal)
    self.clear_button.pack(pady=10)

    self.quit_button = ttk.Button(self.control_frame, text="Quit", command=self.quit)
    self.quit_button.pack(pady=10)

    self.control_frame.pack(expand=True, fill=tk.BOTH)
```

The screenshot shows a web browser window with several tabs open. The active tab is titled "Customizable Fractal Tree Visualizer". The page content includes:

- Branch Length:** Initial length of the trunk.
- Angle:** Determines how much branches spread.
- Reduction:** How much each sub-branch shrinks.
- The Drawing Algorithm:**
  - `draw_tree()` is the recursive function that creates the fractal.
  - Each branch splits into two smaller branches.
  - The function calls itself for each new branch until it reaches the iteration limit.
- Special Features:**
  - Branch thickness varies with iteration level.
  - Color darkens for thicker branches.
  - Option to randomize all parameters for creative exploration.

**To Run This Code**

- Make sure you have Python installed with tkinter (included in standard Python).
- Copy the code to a .py file and run it.
- Adjust the sliders to customize your tree.
- Click "Generate Tree" to see your creation.

The application demonstrates principles of recursion, GUI development, and creative coding. Feel free to experiment with the parameters to create unique tree designs!

tree should have flowers

Claude 3.7 Sonnet Choose style

Fractal Tree Generator

```

length_slider.pack(fill=tk.X, pady=5)

# Angle slider
ttk.Label(self.control_frame, text="Branch Angle (degrees)").pack(anchor=tk.W)
angle_slider = ttk.Scale(self.control_frame, from_=5, to=60,
variable=self.angle, orient=tk.HORIZONTAL, length=200)
angle_slider.pack(fill=tk.X, pady=5)

# Reduction slider
ttk.Label(self.control_frame, text="Length Reduction").pack(anchor=tk.W)
reduction_slider = ttk.Scale(self.control_frame, from_=0.5, to=0.9,
variable=self.reduction, orient=tk.HORIZONTAL, length=200)
reduction_slider.pack(fill=tk.X, pady=5)

# Color picker
ttk.Label(self.control_frame, text="Branch Color").pack(anchor=tk.W, pady=5)
color_button = ttk.Button(self.control_frame, text="Choose Color",
command=self.choose_color)
color_button.pack(fill=tk.X, pady=5)

# Color indicator
self.color_indicator = tk.Canvas(self.control_frame, width=200, height=20,
background="white")
self.color_indicator.pack(pady=5)

# Randomize option
randomize_check = ttk.Checkbutton(self.control_frame, text="Randomize Parameters",
variable=self.randomize_var)
randomize_check.pack()

```

Last edited 19 minutes ago

Published

## Code:

The screenshot shows a Jupyter Notebook cell with the following Python code:

```

[*]: import math
import turtle
import random
import tkinter as tk
from tkinter import ttk, colorchooser

class FractalTreeApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Fractal Tree Generator")
        self.root.geometry("800x600")

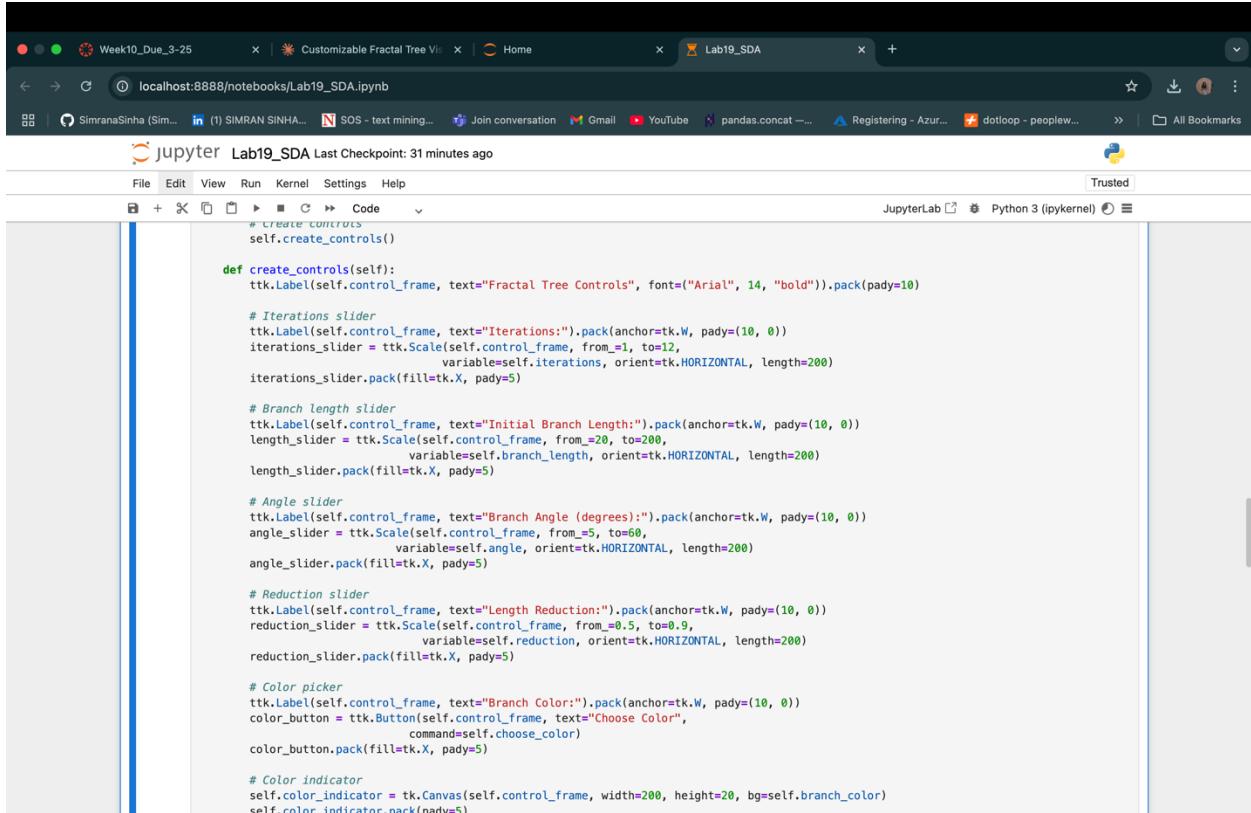
        # Setup main frames
        self.control_frame = ttk.Frame(root, padding="10")
        self.control_frame.pack(side=tk.LEFT, fill=tk.Y)

        self.canvas_frame = ttk.Frame(root)
        self.canvas_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

        # Create turtle canvas
        self.screen = turtle.TurtleScreen(self.canvas)
        self.screen.tracer(0) # Turn off animation for speed
        self.turtle = turtle.RawTurtle(self.screen)
        self.turtle.hideturtle()
        self.turtle.speed(0)

        # Control variables
        self.iterations = tk.IntVar(value=8)
        self.branch_length = tk.DoubleVar(value=100.0)
        self.angle = tk.DoubleVar(value=30.0)
        self.reduction = tk.DoubleVar(value=0.75)
        self.branch_color = "#008000" # Default green
        self.randomize_var = tk.BooleanVar(value=False)

```



```

jupyter Lab19_SDA Last Checkpoint: 31 minutes ago
File Edit View Run Kernel Settings Help
+ - C > Code
# Create controls
self.create_controls()

def create_controls(self):
    ttk.Label(self.control_frame, text="Fractal Tree Controls", font=("Arial", 14, "bold")).pack(pady=10)

    # Iterations slider
    ttk.Label(self.control_frame, text="Iterations:").pack(anchor=tk.W, pady=(10, 0))
    iterations_slider = ttk.Scale(self.control_frame, from_=1, to=12,
                                  variable=self.iterations, orient=tk.HORIZONTAL, length=200)
    iterations_slider.pack(fill=tk.X, pady=5)

    # Branch length slider
    ttk.Label(self.control_frame, text="Initial Branch Length:").pack(anchor=tk.W, pady=(10, 0))
    length_slider = ttk.Scale(self.control_frame, from_=20, to=200,
                              variable=self.branch_length, orient=tk.HORIZONTAL, length=200)
    length_slider.pack(fill=tk.X, pady=5)

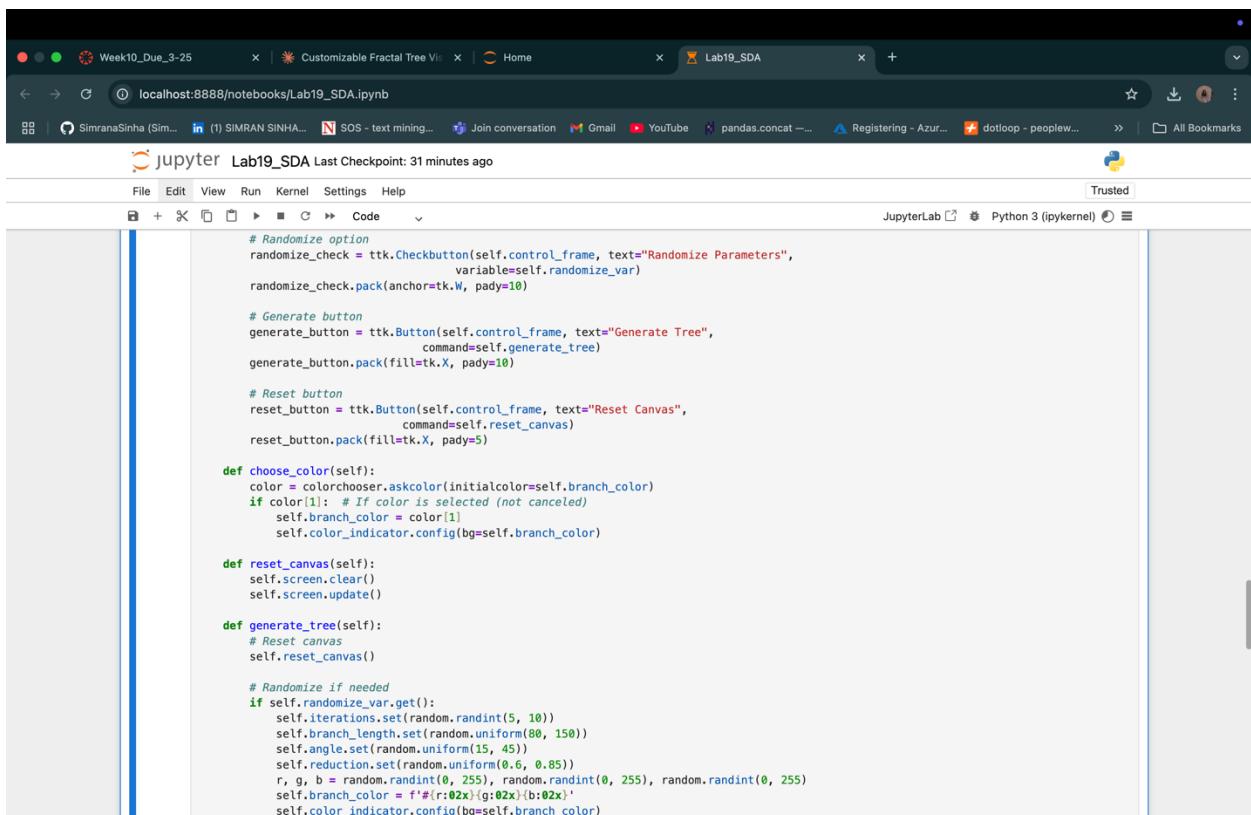
    # Angle slider
    ttk.Label(self.control_frame, text="Branch Angle (degrees):").pack(anchor=tk.W, pady=(10, 0))
    angle_slider = ttk.Scale(self.control_frame, from_=5, to=60,
                            variable=self.angle, orient=tk.HORIZONTAL, length=200)
    angle_slider.pack(fill=tk.X, pady=5)

    # Reduction slider
    ttk.Label(self.control_frame, text="Length Reduction:").pack(anchor=tk.W, pady=(10, 0))
    reduction_slider = ttk.Scale(self.control_frame, from_=0.5, to=0.9,
                                 variable=self.reduction, orient=tk.HORIZONTAL, length=200)
    reduction_slider.pack(fill=tk.X, pady=5)

    # Color picker
    ttk.Label(self.control_frame, text="Branch Color:").pack(anchor=tk.W, pady=(10, 0))
    color_button = ttk.Button(self.control_frame, text="Choose Color",
                             command=self.choose_color)
    color_button.pack(fill=tk.X, pady=5)

    # Color indicator
    self.color_indicator = tk.Canvas(self.control_frame, width=200, height=20, bg=self.branch_color)
    self.color_indicator.pack(fill=tk.X, pady=5)

```



```

jupyter Lab19_SDA Last Checkpoint: 31 minutes ago
File Edit View Run Kernel Settings Help
+ - C > Code
# Randomize option
randomize_check = ttk.Checkbutton(self.control_frame, text="Randomize Parameters",
                                   variable=self.randomize_var)
randomize_check.pack(anchor=tk.W, pady=10)

# Generate button
generate_button = ttk.Button(self.control_frame, text="Generate Tree",
                             command=self.generate_tree)
generate_button.pack(fill=tk.X, pady=10)

# Reset button
reset_button = ttk.Button(self.control_frame, text="Reset Canvas",
                           command=self.reset_canvas)
reset_button.pack(fill=tk.X, pady=5)

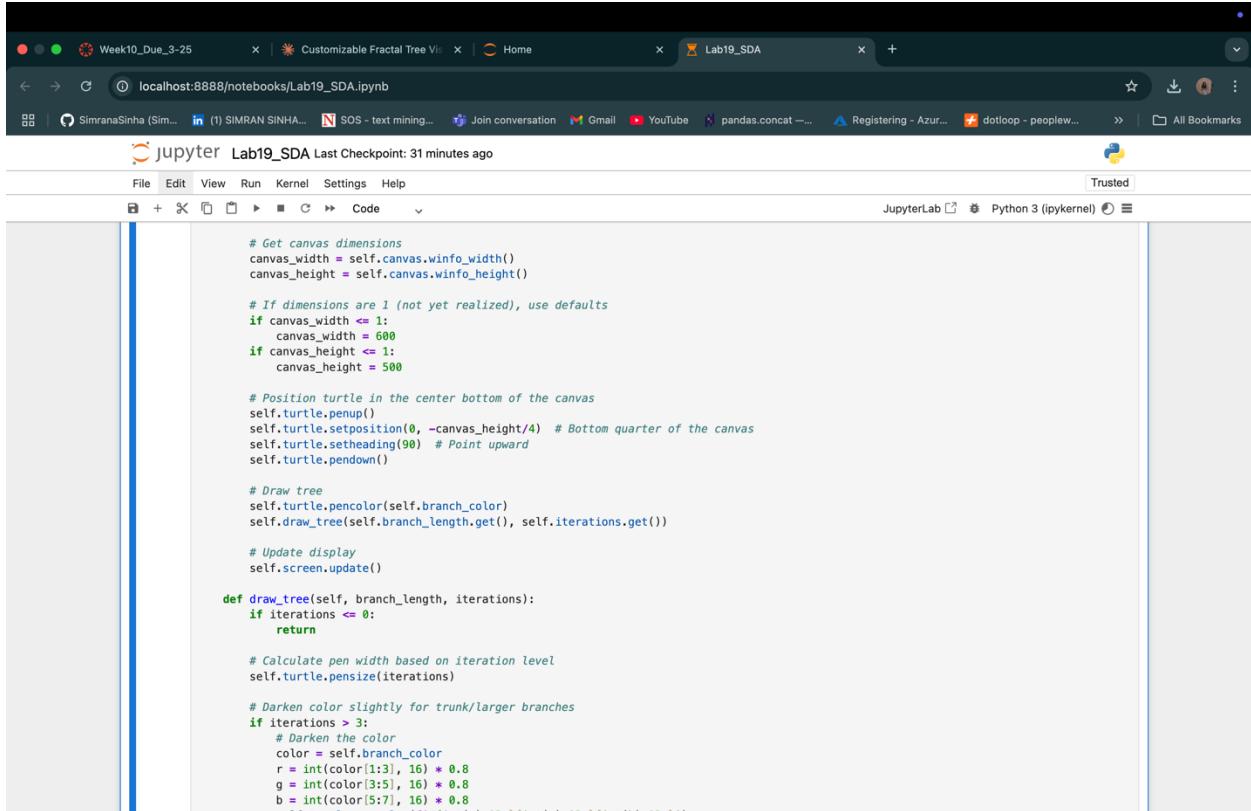
def choose_color(self):
    color = colorchooser.askcolor(initialcolor=self.branch_color)
    if color[1]: # If color is selected (not canceled)
        self.branch_color = color[1]
        self.color_indicator.config(bg=self.branch_color)

def reset_canvas(self):
    self.screen.clear()
    self.screen.update()

def generate_tree(self):
    # Reset canvas
    self.reset_canvas()

    # Randomize if needed
    if self.randomize_var.get():
        self.iterations.set(random.randint(5, 10))
        self.branch_length.set(random.uniform(80, 150))
        self.angle.set(random.uniform(15, 45))
        self.reduction.set(random.uniform(0.6, 0.85))
        r, g, b = random.randint(0, 255), random.randint(0, 255), random.randint(0, 255)
        self.branch_color = f'#({r:02x}){({g:02x})}{({b:02x})}'
        self.color_indicator.config(bg=self.branch_color)

```



```
# Get canvas dimensions
canvas_width = self.canvas.winfo_width()
canvas_height = self.canvas.winfo_height()

# If dimensions are 1 (not yet realized), use defaults
if canvas_width <= 1:
    canvas_width = 600
if canvas_height <= 1:
    canvas_height = 500

# Position turtle in the center bottom of the canvas
self.turtle.penup()
self.turtle.setposition(0, -canvas_height/4) # Bottom quarter of the canvas
self.turtle.setheading(90) # Point upward
self.turtle.pendown()

# Draw tree
self.turtle.pencolor(self.branch_color)
self.draw_tree(self.branch_length.get(), self.iterations.get())

# Update display
self.screen.update()

def draw_tree(self, branch_length, iterations):
    if iterations <= 0:
        return

    # Calculate pen width based on iteration level
    self.turtle.pensize(iterations)

    # Darken color slightly for trunk/larger branches
    if iterations > 3:
        # Darken the color
        color = self.branch_color
        r = int(color[1:3], 16) * 0.8
        g = int(color[3:5], 16) * 0.8
        b = int(color[5:7], 16) * 0.8
        self.turtle.pencolor(r, g, b)

    # Draw branch
    self.turtle.forward(branch_length)

    # Save position and heading
    pos = self.turtle.position()
    heading = self.turtle.heading()

    # Right branch
    self.turtle.right(self.angle.get())
    self.draw_tree(branch_length * self.reduction.get(), iterations - 1)

    # Left branch
    self.turtle.penup()
    self.turtle.setposition(pos)
    self.turtle.setheading(heading)
    self.turtle.pendown()
    self.turtle.left(self.angle.get())
    self.draw_tree(branch_length * self.reduction.get(), iterations - 1)

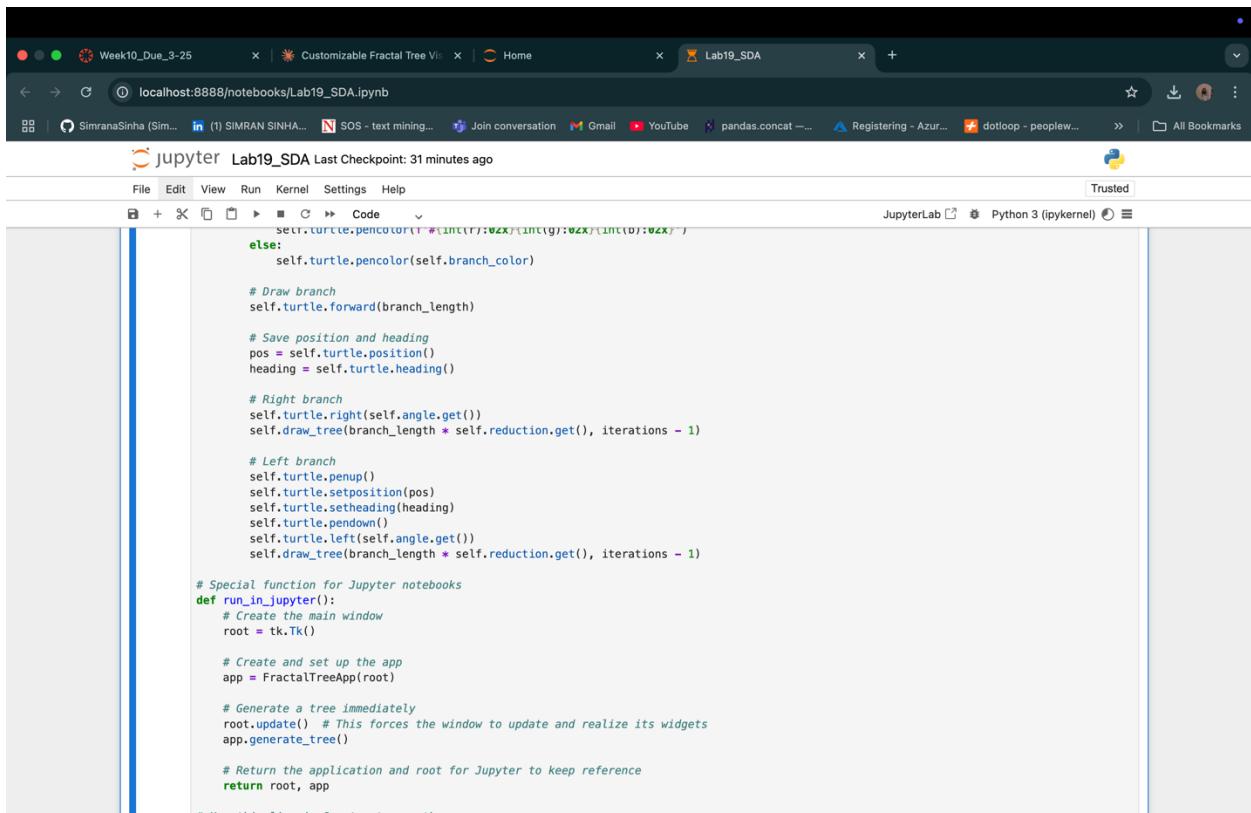
# Special function for Jupyter notebooks
def run_in_jupyter():
    # Create the main window
    root = tk.Tk()

    # Create and set up the app
    app = FractalTreeApp(root)

    # Generate a tree immediately
    root.update() # This forces the window to update and realize its widgets
    app.generate_tree()

    # Return the application and root for Jupyter to keep reference
    return root, app

# Use this line in Jupyter to run the app
# run_in_jupyter()
```



```
else:
    self.turtle.pencolor(int(r)*255,int(g)*255,int(b)*255)

# Draw branch
self.turtle.forward(branch_length)

# Save position and heading
pos = self.turtle.position()
heading = self.turtle.heading()

# Right branch
self.turtle.right(self.angle.get())
self.draw_tree(branch_length * self.reduction.get(), iterations - 1)

# Left branch
self.turtle.penup()
self.turtle.setposition(pos)
self.turtle.setheading(heading)
self.turtle.pendown()
self.turtle.left(self.angle.get())
self.draw_tree(branch_length * self.reduction.get(), iterations - 1)

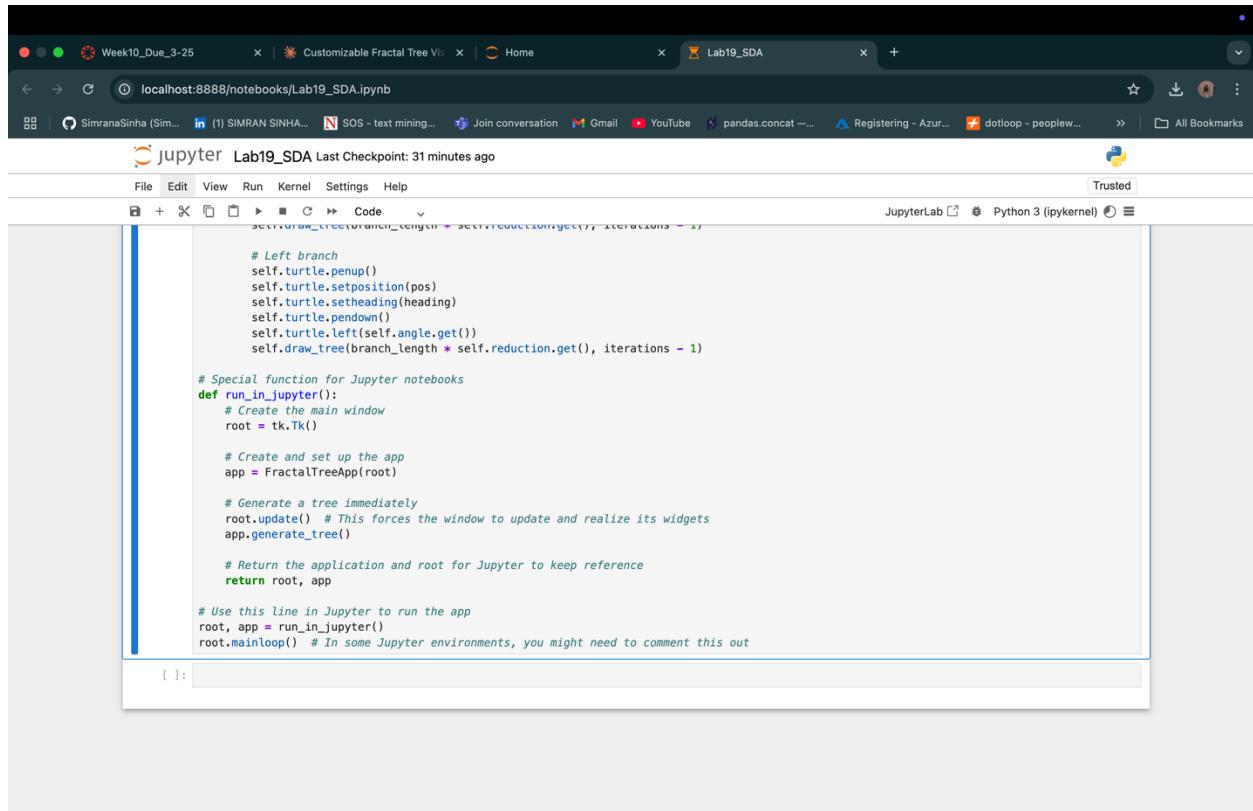
# Special function for Jupyter notebooks
def run_in_jupyter():
    # Create the main window
    root = tk.Tk()

    # Create and set up the app
    app = FractalTreeApp(root)

    # Generate a tree immediately
    root.update() # This forces the window to update and realize its widgets
    app.generate_tree()

    # Return the application and root for Jupyter to keep reference
    return root, app

# Use this line in Jupyter to run the app
# run_in_jupyter()
```



The screenshot shows a Jupyter Notebook interface with the title "Lab19\_SDA". The code cell contains Python code for generating a fractal tree using a turtle graphics library. The code includes comments explaining the steps: setting up the turtle, defining a special function for Jupyter notebooks, creating the main window, generating the tree immediately, and returning the application and root for Jupyter to keep reference. It also includes a note to use a specific line in Jupyter to run the app.

```

# Left branch
self.turtle.penup()
self.turtle.setposition(pos)
self.turtle.setheading(heading)
self.turtle.pendown()
self.turtle.left(self.angle.get())
self.draw_tree(branch_length * self.reduction.get(), iterations - 1)

# Special function for Jupyter notebooks
def run_in_jupyter():
    # Create the main window
    root = tk.Tk()

    # Create and set up the app
    app = FractalTreeApp(root)

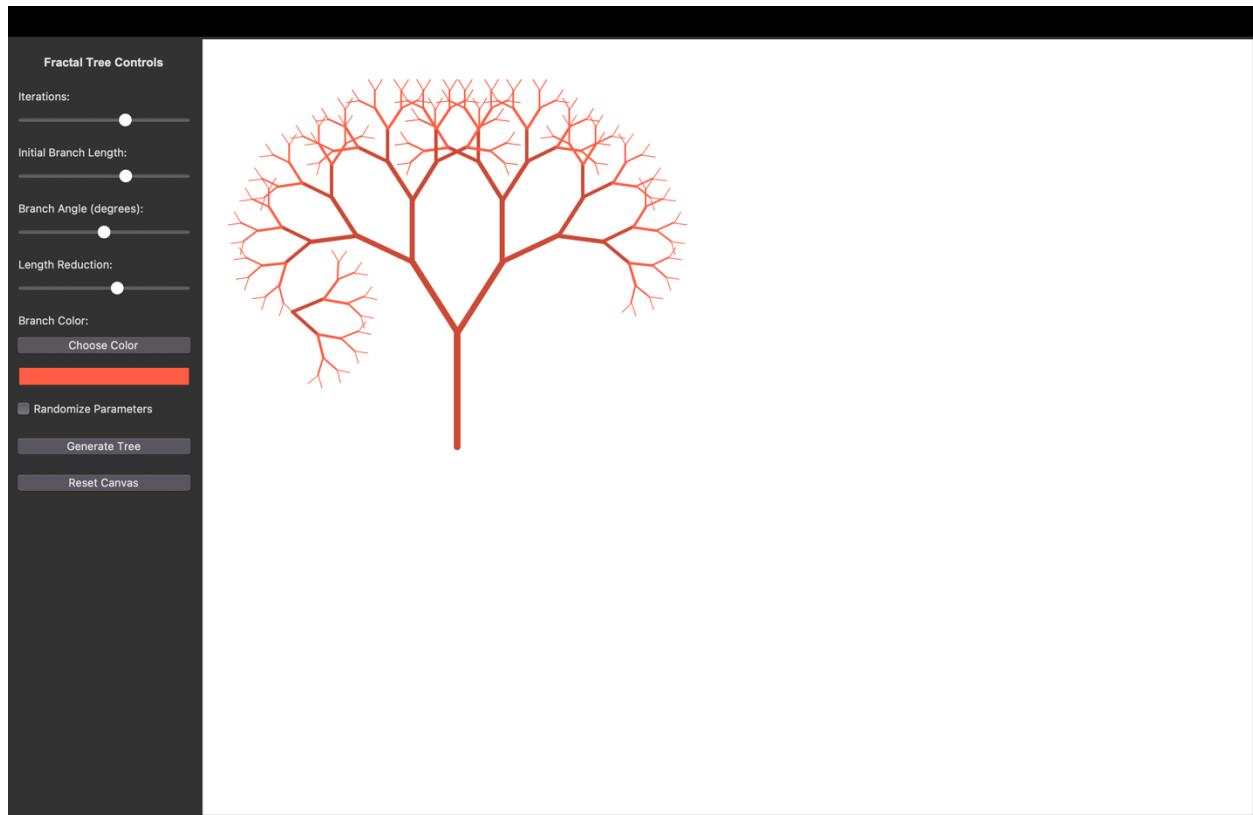
    # Generate a tree immediately
    root.update() # This forces the window to update and realize its widgets
    app.generate_tree()

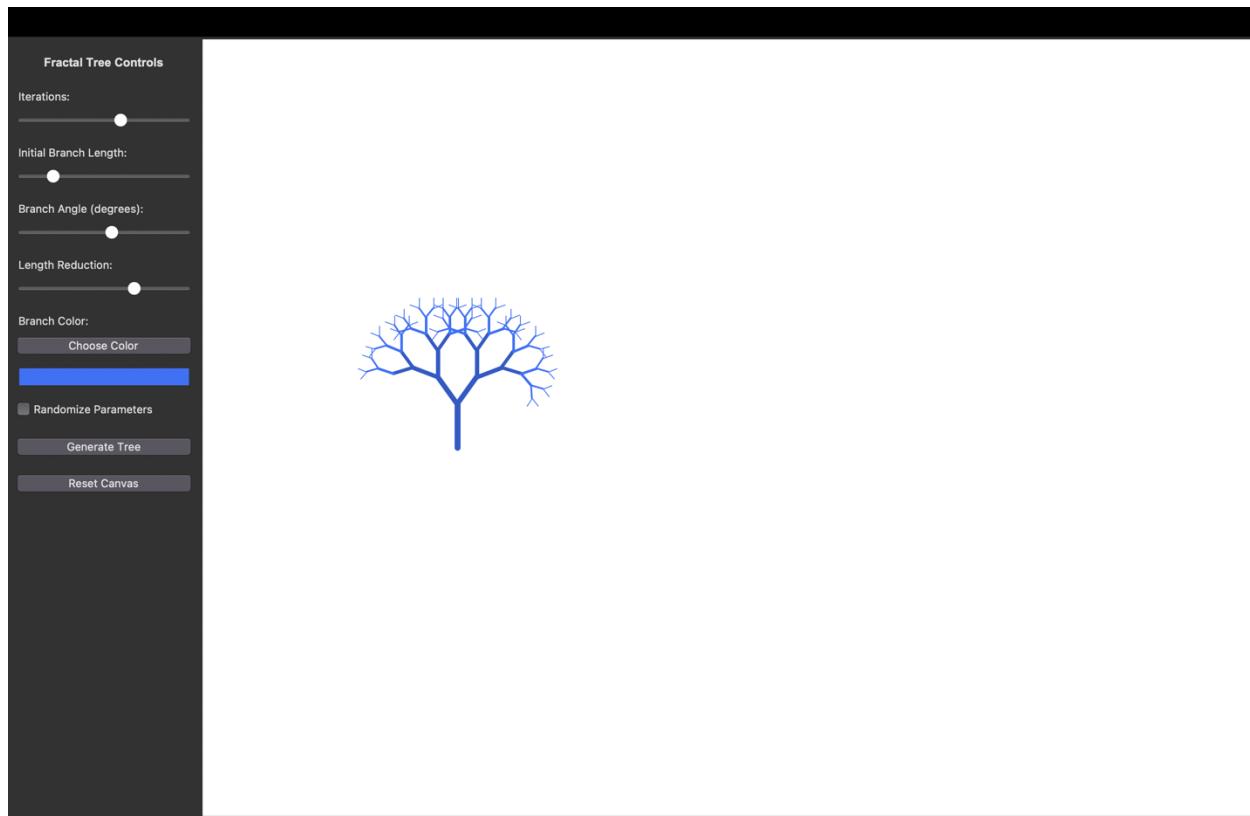
    # Return the application and root for Jupyter to keep reference
    return root, app

# Use this line in Jupyter to run the app
root, app = run_in_jupyter()
root.mainloop() # In some Jupyter environments, you might need to comment this out

```

**Output:**





## Lab 20

1. Write a program to prompt the user for hours and rate per hour to compute gross pay. Consider that the factory gives the employee 1.5 times the hourly rate for hours worked above 40 hours.

Code:

```
# Lab20_1
def compute_pay(hours, rate):
    if hours > 40:
        overtime_hours = hours - 40
        overtime_pay = overtime_hours * (rate * 1.5)
        regular_pay = 40 * rate
        total_pay = regular_pay + overtime_pay
    else:
        total_pay = hours * rate
    return total_pay

# Prompt user for input
try:
    hours = float(input("Enter Hours: "))
    rate = float(input("Enter Rate: "))
    pay = compute_pay(hours, rate)
    print("Pay:", pay)
except ValueError:
    print("Error: Please enter numeric values for hours and rate.")
```

Output:

```
File Edit View Run Kernel Settings Help
File + % > < & C >> Code Trusted
[1]: hours = float(input("Enter Hours: "))
rate = float(input("Enter Rate: "))
pay = compute_pay(hours, rate)
print("Pay:", pay)
except ValueError:
    print("Error: Please enter numeric values for hours and rate.")

Enter Hours: 45
Enter Rate: 10
Pay: 475.0
```

```
[2]: #Lab20_2
sample_dict = {
    "name": "Ryan",
    "age": 21,
    "salary": 60000,
    "city": "Boston"
}

# Keys to extract
keys = ["name", "salary"]

# Extract the required keys
new_dict = {key: sample_dict[key] for key in keys}

# Print the result
print(new_dict)

{'name': 'Ryan', 'salary': 60000}
```

```
[3]: #Lab20_3
list1 = ["M", "na", "i", "Ke"]
list2 = ["y", "me", "s", "lly"]

# Using list comprehension with zip()
result = [a + b for a, b in zip(list1, list2)]

print(result)
['Mv', 'name', 'is', 'Kelly']
```

2. Write a Python program to create a new dictionary by extracting the mentioned keys from the below dictionary.

Given dictionary:

```
sample_dict = {
    "name": "Ryan",
    "age": 21,
    "salary": 60000,
    "city": "Boston"}
```

Code:

```
#Lab20_2
sample_dict = {
    "name": "Ryan",
    "age": 21,
    "salary": 60000,
    "city": "Boston"
}

# Keys to extract
keys = ["name", "salary"]

# Extract the required keys
new_dict = {key: sample_dict[key] for key in keys}

# Print the result
print(new_dict)
```

Output:

```
jupyter Week10 Last Checkpoint: 3 days ago
File Edit View Run Kernel Settings Help
JupyterLab Trusted Python 3 (ipykernel) Trusted
```

```
[2]: #Lab20_2
sample_dict = {
    "name": "Ryan",
    "age": 21,
    "salary": 60000,
    "city": "Boston"
}

# Keys to extract
keys = ["name", "salary"]

# Extract the required keys
new_dict = {key: sample_dict[key] for key in keys}

# Print the result
print(new_dict)
{'name': 'Ryan', 'salary': 60000}
```

```
[3]: #Lab20_3
list1 = ["M", "na", "i", "Ke"]
list2 = ["y", "me", "s", "lly"]

# Using list comprehension with zip()
result = [a + b for a, b in zip(list1, list2)]
```

3. Write a program to add two lists index-wise. Create a new list that contains the 0th index item from both the list, then the 1st index item, and so on till the last element. any leftover items will get added at the end of the new list.

Given: list1 = ["M", "na", "i", "Ke"]

list2 = ["y", "me", "s", "lly"]

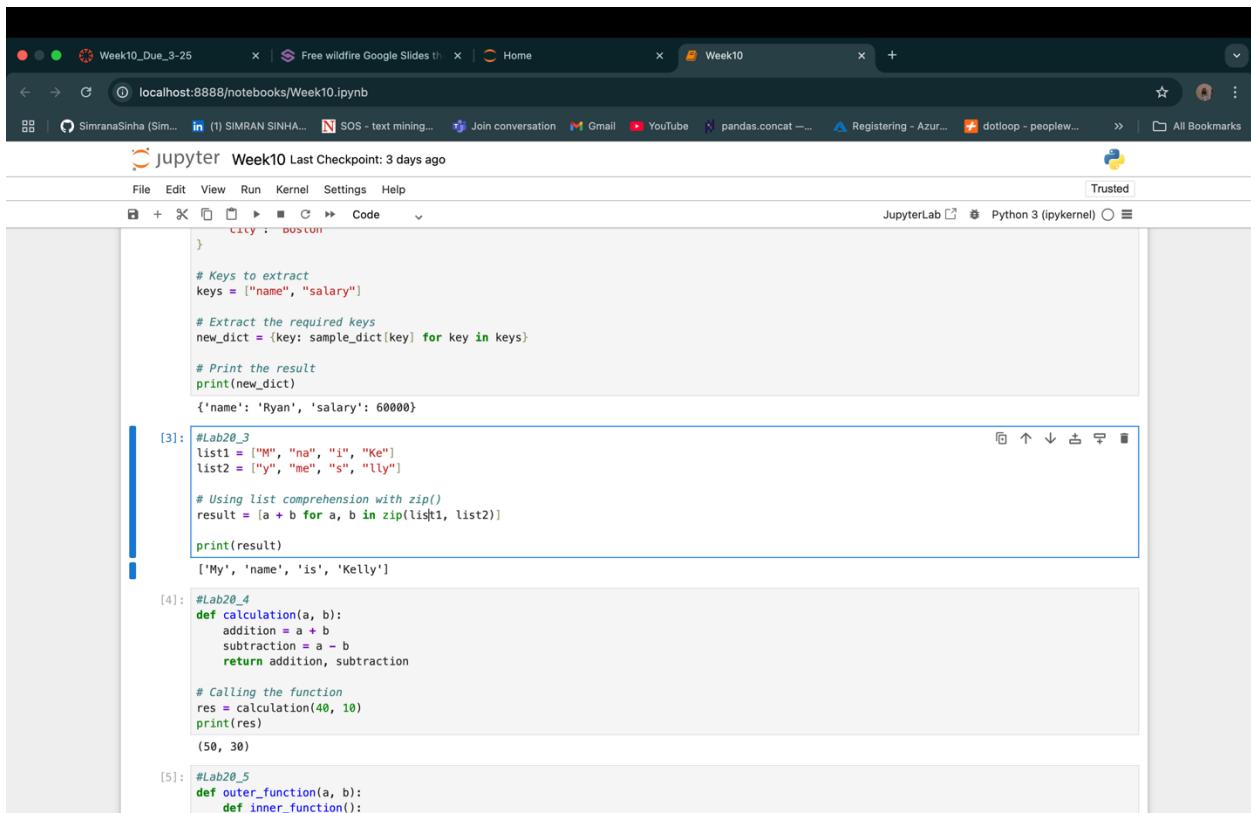
Code:

```
#Lab20_3
list1 = ["M", "na", "i", "Ke"]
list2 = ["y", "me", "s", "lly"]

# Using list comprehension with zip()
result = [a + b for a, b in zip(list1, list2)]

print(result)
```

Output:



The screenshot shows a Jupyter Notebook interface with the following details:

- Header:** Week10\_Due\_3-25, Free wildfire Google Slides th, Home, Week10.
- Toolbar:** File, Edit, View, Run, Kernel, Settings, Help, Trusted, JupyterLab, Python 3 (ipykernel).
- Code Cells:**
  - Cell 3: Contains code to extract specific keys from a dictionary and print the result. Output: {'name': 'Ryan', 'salary': 60000}.
  - Cell 3: Contains the provided code snippet for adding two lists index-wise. Output: ['My', 'name', 'is', 'Kelly'].
  - Cell 4: Contains a function definition for addition and subtraction. Output: (50, 30).
  - Cell 5: Contains a function definition for outer\_function(a, b) which calls inner\_function().

4. Write a program to create function calculation() such that it can accept two variables and calculate addition and subtraction. Also, it must return both addition and subtraction in a single return call.

Given: def calculation(a, b):

# Your Code

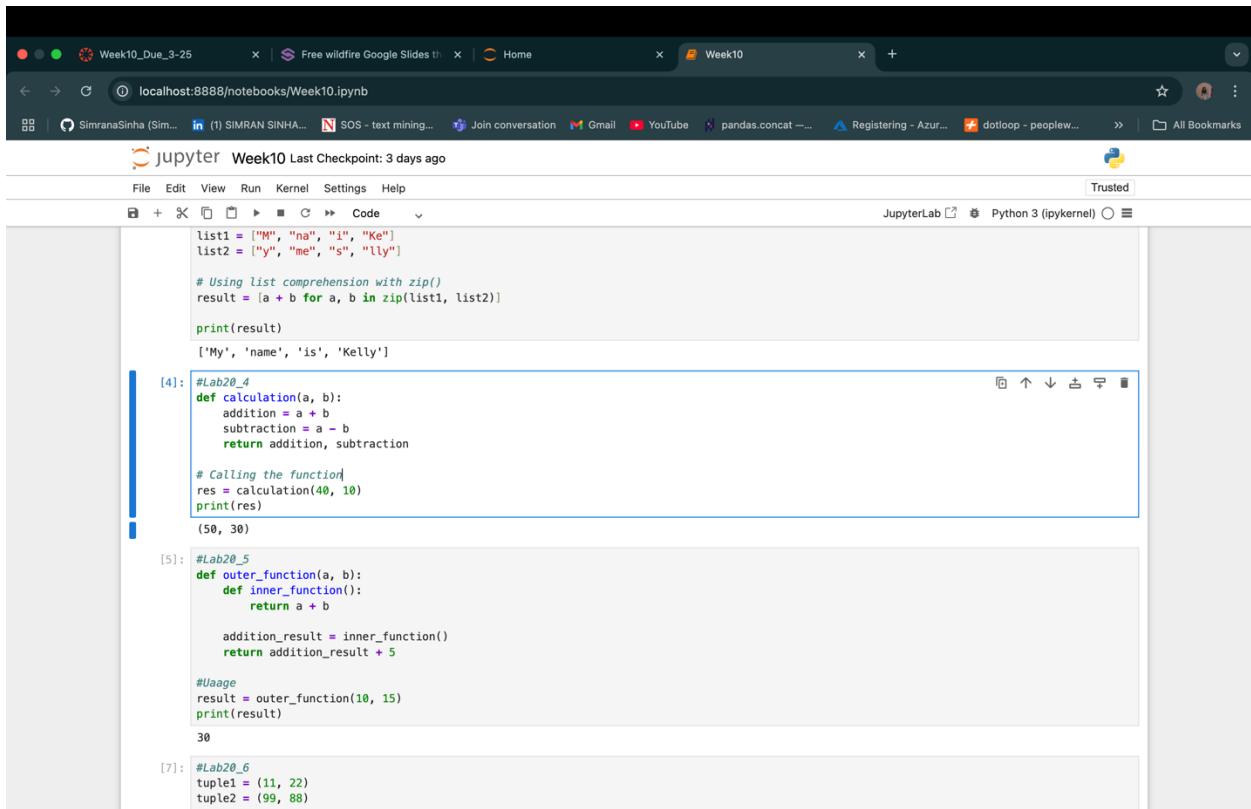
```
res = calculation(40, 10)
print(res)
```

Code:

```
#Lab20_4
def calculation(a, b):
    addition = a + b
    subtraction = a - b
    return addition, subtraction

# Calling the function
res = calculation(40, 10)
print(res)
```

Output:



The screenshot shows a Jupyter Notebook interface with several code cells and their outputs. The notebook is titled "Week10.ipynb".

- Cell 1:** Contains Python code for concatenating two lists using list comprehension and zip(). The output is: ['My', 'name', 'is', 'Kelly']
- Cell 2:** Contains the provided code for the calculation function. The output is: (50, 30)
- Cell 3:** Contains code for a function that returns the sum of two numbers. The output is: 30
- Cell 4:** Contains code for a function that adds 5 to the result of another function. The output is: 30
- Cell 5:** Contains code for a function that adds 5 to the result of another function. The output is: 30
- Cell 6:** Contains code for a function that adds 5 to the result of another function. The output is: 30
- Cell 7:** Contains code for a function that adds 5 to the result of another function. The output is: 30

5. In Python, we can create a nested function inside a function. We can use the nested function to perform complex tasks multiple times within another function or avoid loop and code duplication.

- Create an outer function that will accept two parameters, a and b
- Create an inner function inside an outer function that will calculate the addition of a and b
- At last, an outer function will add 5 into addition and return it

Code:

```
5]: #Lab20_5
def outer_function(a, b):
    def inner_function():
        return a + b

    addition_result = inner_function()
    return addition_result + 5

#Usage
result = outer_function(10, 15)
print(result)
```

Output:

```
File Edit View Run Kernel Settings Help
JupyterLab Trusted Python 3 (ipykernel)
subtraction = a - b
return addition, subtraction

# Calling the function
res = calculation(40, 10)
print(res)
(50, 30)

[5]: #Lab20_5
def outer_function(a, b):
    def inner_function():
        return a + b

    addition_result = inner_function()
    return addition_result + 5

#Usage
result = outer_function(10, 15)
print(result)
30

[7]: #Lab20_6
tuple1 = (11, 22)
tuple2 = (99, 88)

# Swapping tuples
tuple1, tuple2 = tuple2, tuple1

print("tuple1:", tuple1)
print("tuple2:", tuple2)
tuple1: (99, 88)
tuple2: (11, 22)

[8]: #Lab20_7
tuple1 = (11, [22, 33], 44, 55)

# Modifying the first item of the list inside the tuple
```

## 6. Swap two tuples in Python

Given: tuple1 = (11, 22)

tuple2 = (99, 88)

Code:

```
#Lab20_6
tuple1 = (11, 22)
tuple2 = (99, 88)

# Swapping tuples
tuple1, tuple2 = tuple2, tuple1

print("tuple1:", tuple1)
print("tuple2:", tuple2)
```

Output:

The screenshot shows a Jupyter Notebook interface with a single code cell highlighted in blue. The code cell contains the following Python code:

```
#Lab20_6
tuple1 = (11, 22)
tuple2 = (99, 88)

# Swapping tuples
tuple1, tuple2 = tuple2, tuple1

print("tuple1:", tuple1)
print("tuple2:", tuple2)
```

The output of the code is displayed below the cell, showing the swapped tuples:

```
tuple1: (99, 88)
tuple2: (11, 22)
```

The notebook has tabs for 'Week10\_Due\_3-25', 'Free wildfire Google Slides th', 'Home', and 'Week10'. The 'Week10' tab is active. The browser address bar shows 'localhost:8888/notebooks/Week10.ipynb'. The JupyterLab interface includes a toolbar with File, Edit, View, Run, Kernel, Settings, Help, and a Trusted status indicator.

7. Given is a nested tuple. Write a program to modify the first item (22) of a list inside a following tuple to 222

Given: tuple1 = (11, [22, 33], 44, 55)

Code:

```
#Lab20_7
tuple1 = (11, [22, 33], 44, 55)

# Modifying the first item of the list inside the tuple
tuple1[1][0] = 222

print("tuple1:", tuple1)
```

Output:

```
File Edit View Run Kernel Settings Help
File + % < > >> Code Trusted
print(result)
30
[7]: #Lab20_6
tuple1 = (11, 22)
tuple2 = (99, 88)

# Swapping tuples
tuple1, tuple2 = tuple2, tuple1

print("tuple1:", tuple1)
print("tuple2:", tuple2)

tuple1: (99, 88)
tuple2: (11, 22)

[8]: #Lab20_7
tuple1 = (11, [22, 33], 44, 55)

# Modifying the first item of the list inside the tuple
tuple1[1][0] = 222

print("tuple1:", tuple1)

tuple1: (11, [222, 33], 44, 55)

[9]: #Lab20_8
sample_dict = {
    'Physics': 82,
    'Math': 65,
    'History': 75
}

# Get the key with the minimum value
min_key = min(sample_dict, key=sample_dict.get)

print(min_key)
Math
```

### 8. Get the key of a minimum value from the following dictionary

Given: sample\_dict = {

'Physics': 82,

Math': 65,

'history': 75

}

Code:

```
tuple2: (11, 22)
```

```
#Lab20_8
sample_dict = {
    'Physics': 82,
    'Math': 65,
    'History': 75
}

# Get the key with the minimum value
min_key = min(sample_dict, key=sample_dict.get)

print(min_key)
```

Output:

The screenshot shows a Jupyter Notebook interface with several cells of code and their outputs.

- Cell 8:** Prints tuple2: (11, 22).
- Cell 9:** Prints tuple1: (11, [22, 33], 44, 55). It contains code to modify the first item of the list inside the tuple.
- Cell 9 (continued):** Prints tuple1: (11, [222, 33], 44, 55). It contains code to get the key with the minimum value.
- Cell 10:** Prints 200 present in a dict. It contains code to check if 200 exists in the dictionary values.
- Cell 11 (empty):** Prints #Lab20\_10 Gen AI.

9. Write a Python program to check if value 200 exists in the following dictionary.

Given: sample\_dict = {'a': 100, 'b': 200, 'c': 300}

Expected Output: 200 present in a dict

**Hint:**

- Get all values of a dict in a list using the values() method.
  - Next, use the if condition to check if 200 is present in the given list

Code:

```
#Lab20_9
sample_dict = {'a': 100, 'b': 200, 'c': 300}

# Check if 200 exists in the dictionary values
if 200 in sample_dict.values():
    print("200 present in a dict")
else:
    print("200 not found in a dict")
```

## Output:

The screenshot shows a Jupyter Notebook interface with several tabs open at the top: "Week10\_Due\_3-25", "Free wildfire Google Slides th", "Home", and "Week10". The main window displays a code cell with Python code related to dictionaries. The code includes a sample dictionary with subjects and marks, and logic to find the minimum value and check for a specific key ('200'). The output cell shows the result '200 present in a dict'. Below this, another cell starts with "#Lab20\_10 Gen AI" and begins defining a class for a movie recommender application.

```
'Physics': 82,
'Math': 65,
'History': 75

}

# Get the key with the minimum value
min_key = min(sample_dict, key=sample_dict.get)

print(min_key)
Math

[10]: #Lab20_9
sample_dict = {'a': 100, 'b': 200, 'c': 300}

# Check if 200 exists in the dictionary values
if 200 in sample_dict.values():
    print("200 present in a dict")
else:
    print("200 not found in a dict")

200 present in a dict

[ ]: #Lab20_10 Gen AI
import tkinter as tk
from tkinter import ttk, messagebox
import random

# Movie Recommendation System using HashMap
class MovieRecommenderApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Movie Recommender")
        self.root.geometry("400x500")
        self.root.configure(bg="#2C3E50") # Dark background color

        # HashMap (Dictionary) storing movie genres as keys and movie lists as values
        self.movie_db = {
            "Action": ["Mad Max: Fury Road", "John Wick", "Gladiator", "Die Hard"],
```

## 10. Gen AI Prompt Engineering:

there are many other types of data structure: stack, queue, linked list, graph, and hash Map. Select one. Ask you AI bot to explain it, when to use, and provide cool app

**Bot Used: ChatGPT**

Prompt:

The screenshot shows a dark-themed ChatGPT interface. In the top right corner, there is a "Share" button and a small profile picture. The main area contains a user message and a generated response.

**User Message:**

There are many other types of data structure: stack, queue, linked list, graph, and hash Map.  
Use HashMap  
Explain Hash map and also provide me a cool app and explain the code

**Generated Response:**

**What is a HashMap?**  
A HashMap (or Dictionary in Python) is a key-value data structure that allows fast retrieval of values based on unique keys. It uses a hashing function to map keys to an index in an underlying array, making lookups, insertions, and deletions extremely fast ( $O(1)$  in most cases).

**Key Features of HashMap:**

- **Fast Lookups:** Accessing elements is very efficient (on average,  $O(1)$  time complexity).
- **Unique Keys:** Each key maps to a single value, ensuring no duplicates.
- **Dynamic Sizing:** It can grow dynamically without performance degradation.
- **Unordered Storage:** Unlike lists, HashMaps do not maintain order.

**Cool App: "Real-Time Movie Recommendation System"** 🎬🍿  
We will build a real-time movie recommendation system using a HashMap (Dictionary in Python). The app will allow users to search movies by genre, and it will return recommendations instantly.

GenAI:

The screenshot shows a dark-themed ChatGPT interface. On the left, there is a sidebar with various recent conversations and a "Upgrade plan" section. The main area displays a detailed explanation of a HashMap and a cool app idea.

**What is a HashMap?**  
A HashMap (or Dictionary in Python) is a key-value data structure that allows fast retrieval of values based on unique keys. It uses a hashing function to map keys to an index in an underlying array, making lookups, insertions, and deletions extremely fast ( $O(1)$  in most cases).

**Key Features of HashMap:**

- **Fast Lookups:** Accessing elements is very efficient (on average,  $O(1)$  time complexity).
- **Unique Keys:** Each key maps to a single value, ensuring no duplicates.
- **Dynamic Sizing:** It can grow dynamically without performance degradation.
- **Unordered Storage:** Unlike lists, HashMaps do not maintain order.

**Cool App: "Real-Time Movie Recommendation System"** 🎬🍿  
We will build a real-time movie recommendation system using a HashMap (Dictionary in Python). The app will allow users to search movies by genre, and it will return recommendations instantly.

**Python Code:**

```
python
```

Ask anything

ChatGPT can make mistakes. Check important info.

The screenshot shows a ChatGPT interface with a dark theme. The user has pasted a Python script into the code editor. The script is a class named `MovieRecommenderApp` that uses `tkinter` to create a GUI for a movie recommendation system. It includes a dictionary `self.movie_db` mapping genres to lists of movies and a `self.title_label` for displaying the title. A large downward arrow indicates the code continues below.

```
import tkinter as tk
from tkinter import ttk, messagebox
import random

# Movie Recommendation System using HashMap
class MovieRecommenderApp:
    def __init__(self, root):
        self.root = root
        self.root.title("🎬 Movie Recommender 🎬")
        self.root.geometry("400x500")
        self.root.configure(bg="#2C3E50") # Dark background color

        # HashMap (Dictionary) storing movie genres as keys and movie lists as values
        self.movie_db = {
            "Action": ["Mad Max: Fury Road", "John Wick", "Gladiator", "Die Hard"],
            "Comedy": ["Step Brothers", "Superbad", "The Hangover", "Dumb and Dumber"],
            "Sci-Fi": ["Interstellar", "Inception", "The Matrix", "Blade Runner 2049"],
            "Horror": ["The Conjuring", "A Nightmare on Elm Street", "The Exorcist", "The Shining"],
            "Drama": ["The Shawshank Redemption", "Forrest Gump", "The Green Mile", "The Godfather"]
        }

        # Title Label
        self.title_label = tk.Label(root, text="🎬 Movie Recommender 🎬", font=("Arial", 16))
        self.title_label.pack(pady=20)

    # Genre Selection Label
```

The screenshot shows a ChatGPT interface with a Python script pasted into the code editor. The script creates a Tkinter window with a genre dropdown menu, a 'Get Recommendation' button, and a result label. It includes a 'get\_recommendation' function that selects a random movie from a database based on the chosen genre. If no genre is selected, it shows an error message.

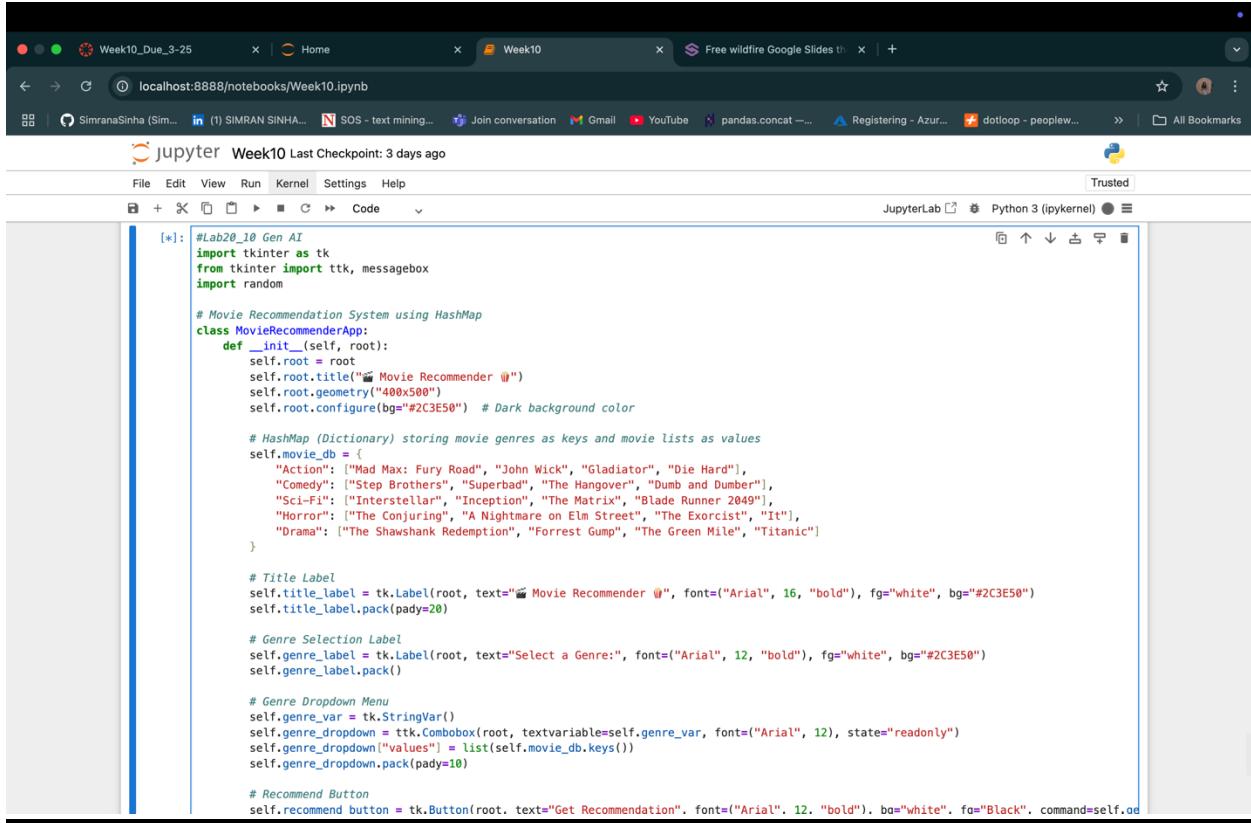
```
# Genre Dropdown Menu
self.genre_var = tk.StringVar()
self.genre_dropdown = ttk.Combobox(root, textvariable=self.genre_var, font=("Arial", 12, "bold"))
self.genre_dropdown["values"] = list(self.movie_db.keys())
self.genre_dropdown.pack(pady=10)

# Recommend Button (Updated colors)
self.recommend_button = tk.Button(
    root, text="Get Recommendation",
    font=("Arial", 12, "bold"), bg="#F39C12", fg="black",
    activebackground="#D68910", activeforeground="white",
    command=self.get_recommendation
)
self.recommend_button.pack(pady=20)

# Result Label
self.result_label = tk.Label(root, text="", font=("Arial", 14, "bold"), fg="#D68910")
self.result_label.pack(pady=20)

def get_recommendation(self):
    genre = self.genre_var.get()
    if genre in self.movie_db:
        movie = random.choice(self.movie_db[genre])
        self.result_label.config(text=f"\ud83d\udcbb {movie}")
    else:
        messagebox.showerror("Error", "Please select a valid genre.")
```

Code:



```

[*]: #Lab20_10 Gen AI
import tkinter as tk
from tkinter import ttk, messagebox
import random

# Movie Recommendation System using HashMap
class MovieRecommenderApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Movie Recommender")
        self.root.geometry("400x500")
        self.root.configure(bg="#2C3E50") # Dark background color

    # HashMap (Dictionary) storing movie genres as keys and movie lists as values
    self.movie_db = {
        "Action": ["Mad Max: Fury Road", "John Wick", "Gladiator", "Die Hard"],
        "Comedy": ["Step Brothers", "Superbad", "The Hangover", "Dumb and Dumber"],
        "Sci-Fi": ["Interstellar", "Inception", "The Matrix", "Blade Runner 2049"],
        "Horror": ["The Conjuring", "A Nightmare on Elm Street", "The Exorcist", "It"],
        "Drama": ["The Shawshank Redemption", "Forrest Gump", "The Green Mile", "Titanic"]
    }

    # Title Label
    self.title_label = tk.Label(root, text="Movie Recommender", font=("Arial", 16, "bold"), fg="white", bg="#2C3E50")
    self.title_label.pack(pady=20)

    # Genre Selection Label
    self.genre_label = tk.Label(root, text="Select a Genre:", font=("Arial", 12, "bold"), fg="white", bg="#2C3E50")
    self.genre_label.pack()

    # Genre Dropdown Menu
    self.genre_var = tk.StringVar()
    self.genre_dropdown = ttk.Combobox(root, textvariable=self.genre_var, font=("Arial", 12), state="readonly")
    self.genre_dropdown["values"] = list(self.movie_db.keys())
    self.genre_dropdown.pack(pady=10)

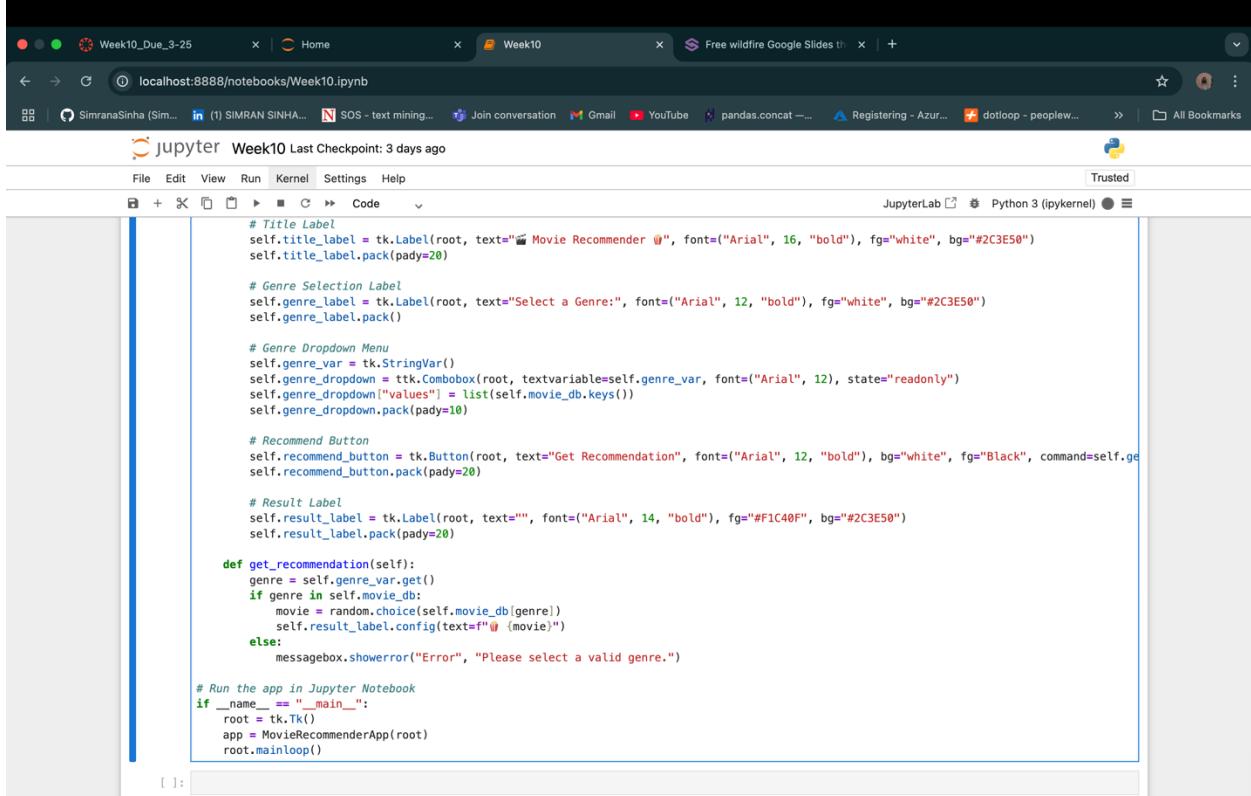
    # Recommend Button
    self.recommend_button = tk.Button(root, text="Get Recommendation", font=("Arial", 12, "bold"), bg="white", fg="Black", command=self.get_recommendation)
    self.recommend_button.pack(pady=20)

    # Result Label
    self.result_label = tk.Label(root, text="", font=("Arial", 14, "bold"), fg="#F1C40F", bg="#2C3E50")
    self.result_label.pack(pady=20)

    def get_recommendation(self):
        genre = self.genre_var.get()
        if genre in self.movie_db:
            movie = random.choice(self.movie_db[genre])
            self.result_label.config(text=f"\u263a {movie}")
        else:
            messagebox.showerror("Error", "Please select a valid genre.")

    # Run the app in Jupyter Notebook
if __name__ == "__main__":
    root = tk.Tk()
    app = MovieRecommenderApp(root)
    root.mainloop()

```



```

[*]: #Lab20_10 Gen AI
import tkinter as tk
from tkinter import ttk, messagebox
import random

# Movie Recommendation System using HashMap
class MovieRecommenderApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Movie Recommender")
        self.root.geometry("400x500")
        self.root.configure(bg="#2C3E50") # Dark background color

    # HashMap (Dictionary) storing movie genres as keys and movie lists as values
    self.movie_db = {
        "Action": ["Mad Max: Fury Road", "John Wick", "Gladiator", "Die Hard"],
        "Comedy": ["Step Brothers", "Superbad", "The Hangover", "Dumb and Dumber"],
        "Sci-Fi": ["Interstellar", "Inception", "The Matrix", "Blade Runner 2049"],
        "Horror": ["The Conjuring", "A Nightmare on Elm Street", "The Exorcist", "It"],
        "Drama": ["The Shawshank Redemption", "Forrest Gump", "The Green Mile", "Titanic"]
    }

    # Title Label
    self.title_label = tk.Label(root, text="Movie Recommender", font=("Arial", 16, "bold"), fg="white", bg="#2C3E50")
    self.title_label.pack(pady=20)

    # Genre Selection Label
    self.genre_label = tk.Label(root, text="Select a Genre:", font=("Arial", 12, "bold"), fg="white", bg="#2C3E50")
    self.genre_label.pack()

    # Genre Dropdown Menu
    self.genre_var = tk.StringVar()
    self.genre_dropdown = ttk.Combobox(root, textvariable=self.genre_var, font=("Arial", 12), state="readonly")
    self.genre_dropdown["values"] = list(self.movie_db.keys())
    self.genre_dropdown.pack(pady=10)

    # Recommend Button
    self.recommend_button = tk.Button(root, text="Get Recommendation", font=("Arial", 12, "bold"), bg="white", fg="Black", command=self.get_recommendation)
    self.recommend_button.pack(pady=20)

    # Result Label
    self.result_label = tk.Label(root, text="", font=("Arial", 14, "bold"), fg="#F1C40F", bg="#2C3E50")
    self.result_label.pack(pady=20)

    def get_recommendation(self):
        genre = self.genre_var.get()
        if genre in self.movie_db:
            movie = random.choice(self.movie_db[genre])
            self.result_label.config(text=f"\u263a {movie}")
        else:
            messagebox.showerror("Error", "Please select a valid genre.")

    # Run the app in Jupyter Notebook
if __name__ == "__main__":
    root = tk.Tk()
    app = MovieRecommenderApp(root)
    root.mainloop()

```

## Output:

