Introduction to Programming and Problem Solving

Week 13: Graphics Programming: Drawing with Turtle Graphics

Lecturer: Lemi Agrey Oliver

Department of Information Technology

Kumi University

codingissweet@gmail.com

Week 12 recap

- Before delving into our subject of discussion for this week let us do a quick recap of our last week lecture. Last week, we delved into the intriguing world of data structures and their functionalities. Our last lecture was a deep dive into three essential components: sets, tuples, and dictionaries, each offering unique capabilities in managing and manipulating data.

Sets:

- Sets have empowered us with the ability to handle collections of unique elements, allowing for efficient operations such as finding intersections, unions, differences, and symmetric differences. Their characteristic feature of exclusivity among elements has proven invaluable in various data-handling scenarios, enhancing our ability to work with distinct entities and streamline processes.

Week 12 recap+

Tuples:

- Tuples, with their immutability and ordered sequence, have shown us the power of structured data representation. Their unchangeable nature makes them reliable containers for preserving data integrity, especially in scenarios where preserving the sequence and contents is critical, like in coordinates, configuration settings, or representing fixed attributes of an entity.

Dictionaries:

- Diving into dictionaries was an exploration of key-value pairs, offering a versatile approach to managing data through associative arrays. The capability to access values via keys has unlocked a world of efficiency in handling diverse data types and structures. With its wide range of methods and functionalities, dictionaries have proven instrumental in applications ranging from data storage to web development.

Week 12 recap++

- Throughout last week's lecture, we not only explored the fundamental aspects of these data structures but also understood their practical implications in real-world applications. Those tools have expanded our capabilities in managing and organizing data, offering versatile solutions for an array of programming challenges.

Content

- Introduction to Graphics Programming
- Introduction to turtle
- Getting Started with the Turtle Module
- Moving the Turtle in any Direction
- Moving Turtle to Any Location
- The color, bgcolor, circle and Speed Method of Turtle
- Drawing with Colors
- Drawing Basic Shapes using Iteration
- Changing Color Dynamically Using List
- Turtles to Create Bar Charts

Introduction

- Graphic programming involves creating visual content, interactive user interfaces, and graphics for various applications, games, simulations, and more. It's a field that encompasses the creation and manipulation of visual elements through programming

Basics of Graphic Programming:

- Rendering Images: Graphic programming involves creating and displaying images, which can be as simple as drawing shapes or as complex as rendering detailed 3D models.
- User Interfaces: It deals with creating graphical user interfaces (GUIs) that enable users to interact with software visually, incorporating buttons, menus, windows, and other elements.

Basics of Graphic Programming:

- Graphics Libraries: Developers use graphics libraries or APIs (Application Programming Interfaces) like OpenGL, DirectX, Vulkan, WebGL, or graphics components in higher-level languages (e.g., Pygame in Python) to interact with the hardware and create graphics.
- 2D and 3D Graphics: Graphic programming covers both 2D and 3D graphics. In 2D, it's about rendering flat images, while 3D involves creating and manipulating 3D models, scenes, and environments.

Techniques and Concepts:

- Rasterization: The process of converting vector graphics into raster images (pixels) for display.
- Shaders: Code used to control the rendering pipeline for effects, textures, lighting, and more in modern graphics.
- Transformations: Manipulating objects by scaling, rotating, and translating them in a graphical space.
- Texture Mapping: Applying images or textures to 3D models to give them a realistic appearance.
- Animation: Creating movement and changes in graphics over time, used in games and simulations.

Applications of Graphic Programming:

- Games: Creating game environments, characters, effects, and interactions.
- Design and Animation: Crafting animations, 3D models, special effects, and visualizations.
- Data Visualization: Representing complex data in a visual form for easy comprehension.
- Simulations: Developing realistic simulations for training or scientific purposes.

Basics of Turtle Graphics:

- Turtle Object: In the turtle module, a turtle object moves on the screen, leaving a trail as it moves.
- Drawing on a Canvas: The turtle moves in a two-dimensional plane, drawing lines as it moves. This creates shapes, patterns, and designs.

Basics of Turtle Graphics+

- Commands: You can control the turtle using simple commands like forward(), backward(), left(), right(), etc., which move the turtle and draw lines as it moves.
- Getting Started:
- To use the turtle module in Python, you need to import it:
- import turtle

Drawing with Colors

- Drawing with colors using the turtle module can add a lot of visual appeal to your creations. Here's how you can draw using different colors:
- Setting Pen Color:
- The pencolor() function changes the color of the pen that the turtle uses to draw.

```
• my_turtle.pencolor("red")  # Change pen color to red
```

Key Functions and Commands:

- forward(distance): Moves the turtle forward by the specified distance.
- backward(distance): Moves the turtle backward by the specified distance.
- left(angle): Turns the turtle left by the specified angle.
- right(angle): Turns the turtle right by the specified angle.
- penup(): Lifts the pen, allowing the turtle to move without drawing.
- pendown(): Lowers the pen, allowing the turtle to draw.
- speed(speed): Sets the drawing speed (0 to 10, with 0 being the fastest).
- color('color_name'): Changes the pen color.
- bgcolor('color_name'): Changes the background color of the screen.

Applications of Turtle Graphics

- **Education:** Turtle graphics is often used in educational settings to teach programming concepts to beginners.
- **Art and Design:** It can be used to create geometric shapes, patterns, and even simple artworks.
- **Basic Animation:** By moving the turtle with different patterns, simple animations can be created.
- Turtle graphics in Python is an excellent way to introduce programming logic, looping, and control structures in an interactive and visual manner, making it an engaging tool for learning the basics of programming and computer science.

Basic Movement Commands:

Moving Forward and Backward:

- forward(distance): Moves the turtle forward by the specified distance in pixels.
- backward(distance): Moves the turtle backward by the specified distance.

Turning Commands:

- left(angle): Turns the turtle left by the specified angle in degrees.
- right(angle): Turns the turtle right by the specified angle.

Basic Movement Commands+

Pen Control:

- penup(): Lifts the pen, allowing the turtle to move without drawing.
- pendown(): Lowers the pen, allowing the turtle to draw.

Drawing Shapes:

- Drawing Circles and Arcs:
- circle(radius): Draws a circle with the specified radius.
- circle(radius, extent=degrees): Draws an arc of a circle with a specified extent.

Appearance Control:

- Setting Speed and Color:
- speed(speed): Sets the drawing speed (0 to 10, with 0 being the fastest).
- color('color_name'): Changes the pen color.
- Stamping and Filling:
- stamp(): Stamps a copy of the turtle shape onto the screen.
- begin_fill() and end_fill(): Begin and end a shape filling operation.

Turtle functions

Positioning and Control:

- Getting and Setting Coordinates:
- pos(): Returns the turtle's current position.
- goto(x, y): Moves the turtle to the specified position (x, y).

Resetting and Clearing:

- reset(): Resets the turtle to its initial state.
- clear(): Clears the screen but keeps the turtle unchanged.

Event Handling:

- onclick(function): Calls a function when the turtle is clicked.
- onscreenclick(function): Calls a function when the screen is clicked.

Turtle functions

Exiting the Program:

- done(): Keeps the graphics window open until closed by the user.
- bye(): Closes the graphics window immediately.

Getting started with turtle

- Ensure you have Python installed. It usually comes pre-installed on many systems. To use the turtle module, you don't need to install anything separately.

Steps for getting started

- **Import turtle module**

```
1. import turtle
```

- **Create a screen and a turtle:**

```
1. screen = turtle.Screen()
2. my_turtle = turtle.Turtle()
```

Basic Commands

- **Moving the Turtle**

```
1. my_turtle.forward(100)  # Move the turtle forward by 100 units
2. my_turtle.left(90)  # Turn the turtle left by 90 degrees
```

Basic Commands+

➢ Changing Pen Color and Width

```
1. my_turtle.pencolor("blue")  # Change pen color
2. my_turtle.pensize(5)  # Change pen size
```

➢ Drawing Shapes

```
1. my_turtle.circle(50)  # Draw a circle with a radius of 50
2. my_turtle.dot(20, "red")  # Draw a dot with a radius of 20 in red
```

➢ Control the Turtle

```
1. my_turtle.penup()  # Lift the pen up
2. my_turtle.pendown()  # Put the pen down
```

Basic Commands++

Setting the Turtle's Heading:

- `my_turtle.setheading(angle) # Sets the direction of the turtle to a specific angle (0 is facing right)`

Going to a Specific Position:

- `my_turtle.goto(x, y)  # Moves the turtle to the specified coordinates (x, y)`

Setting the Turtle's Position Without Drawing:

```
1. my_turtle.penup()
2. # Lifts the pen up, allowing the turtle to move without drawing
3. my_turtle.setx(x)
4. # Set the turtle's x-coordinate without drawing a line
5. my_turtle.sety(y)
6. # Set the turtle's y-coordinate without drawing a line
7. my_turtle.pendown()
8. # Puts the pen down, allowing the turtle to draw again
```

Basic turtle examples

Drawing a Square

```
1.for _ in range(4):
2.    my_turtle.forward(100)
3.    my_turtle.left(90)
```

Shapes Example

- Drawing Polygons and Lines:
- Squares:

```
for _ in range(4):
    my_turtle.forward(100)
    my_turtle.left(90)
```

Regular Polygons:

```
sides = 6  # Change sides for different polygons
angle = 360 / sides
for _ in range(sides):
    my_turtle.forward(100)
    my_turtle.left(angle)
```

Basic turtle examples+

A simple example to draw a square using the turtle module:

```
1. import turtle
2. # Create a turtle screen
3. screen = turtle.Screen()
4. # Create a turtle
5. my_turtle = turtle.Turtle()
6. # Draw a square
7. for _ in range(4):
8. # Move the turtle forward by 100 units
9.     my_turtle.forward(100)
10.# Turn the turtle left by 90 degrees
11.    my_turtle.left(90)
12.# Close the turtle graphics window on click
13.screen.mainloop()
```

Basic Commands+

➢ **Finishing: To keep the window open after drawing, use:**

```
turtle.done()
```

Examples:

Moving Forward:

```
import turtle
# Create a turtle screen and turtle
screen = turtle.Screen()
my_turtle = turtle.Turtle()
# Move the turtle forward by 100 units
my_turtle.forward(100)
turtle.done()
```

This code will move the turtle 100 units forward from its current position.

Basic Commands++

➢Moving Backward:

```
1. import turtle
2. # Create a turtle screen and turtle
3. screen = turtle.Screen()
4. my_turtle = turtle.Turtle()
5. # Move the turtle backward by 100 units
6. my_turtle.backward(100)
7. turtle.done()
```

This code will move the turtle 100 units backward from its current position

Basic Commands++

➢ Turning Left and Right

➢ The below code moves the turtle forward, then turns left and moves forward again, finally turns right and moves forward once more in the new direction

```
1.  import turtle
2.  # Create a turtle screen and turtle
3.  screen = turtle.Screen()
4.  my_turtle = turtle.Turtle()
5.  # Move the turtle in different directions
6.  my_turtle.forward(100)  # Move forward
7.  my_turtle.left(90)  # Turn left by 90 degrees
8.  my_turtle.forward(100)# Move forward in the new direction
9.  my_turtle.right(45)  # Turn right by 45 degrees
10. turtle.done()
```

Moving Turtle to Any Location

- To move the turtle to any specific location on the canvas, you can use the goto() function or set its x and y coordinates using setx() and sety() functions. Here's how you can do it:

Using goto():

- The goto() function moves the turtle to a specific (x, y) coordinate on the canvas.
- For instance, to move the turtle to the position (100, 50):
- my_turtle.goto(x, y)

```
1. my_turtle.goto(100, 50)
```

- Using setx() and sety():
- These functions allow you to set the turtle's x or y coordinate independently.

```
1. my_turtle.setx(x)  # Sets the turtle's x-coordinate
2. my_turtle.sety(y)  # Sets the turtle's y-coordinate
```

Moving Turtle to Any Location+

- For example, to move the turtle to the x-coordinate 150

```
my_turtle.setx(150)
```

- And to move it to the y-coordinate 75

```
my_turtle.sety(75)
```

- Combining Movements

```
import turtle
# Create a turtle screen and turtle
screen = turtle.Screen()
my_turtle = turtle.Turtle()
# Lift the pen up to avoid drawing a line
my_turtle.penup()
# Move to the specified (x, y) coordinates
my_turtle.goto(200, 100)
# Put the pen down to start drawing again (if needed)
my_turtle.pendown()
```

The color, bgcolor, circle and Speed Method of Turtle

- The turtle module provides methods to control the color, background color, drawing circles, and adjusting the speed of the turtle's movements.
- Changing Pen Color:
- pencolor():
- Changes the color of the pen that the turtle uses to draw.

```
my_turtle.pencolor("red")  # Change pen color to red
```

- Changing Background Color:
- bgcolor():

Sets the background color of the drawing canvas.

Set the background color to light blue

```
screen.bgcolor("lightblue")
```

The color, bgcolor, circle and Speed Method of Turtle

- **Drawing a Circle:**
- circle():
- Draws a circle with a specified radius.

```
1. my_turtle.circle(50)  # Draw a circle with a radius of 50
```

Adjusting the Speed of the Turtle:

- speed():
- Sets the speed of the turtle's movements. The speed can be a value between 0 and 10, with 0 being the fastest (no animation) and 10 being the slowest.

```
# Set the turtle's speed to 3 (moderate speed)
my_turtle.speed(3)
```

Filling Shapes with Color:

- You can also fill shapes drawn by the turtle with a specific color using the begin_fill() and end_fill() methods.
- my_turtle.begin_fill() # Start filling the shape
- # Draw your shape here
- my_turtle.end_fill() # End filling the shape
- my_turtle.fillcolor("yellow") # Set fill color to yellow

Filling Shapes with Color:

➢ **Example:**

➢ Here's an example that draws a red square with a blue fill:

```
1.  import turtle
2.  # Create a turtle screen and turtle
3.  screen = turtle.Screen()
4.  my_turtle = turtle.Turtle()
5.  #color
6.  my_turtle.pencolor("red")
7.  my_turtle.fillcolor("blue")
8.  my_turtle.begin_fill()
9.  for _ in range(4):
10.     my_turtle.forward(100)
11.     my_turtle.left(90)
12. my_turtle.end_fill()
13. turtle.done()
```

Drawing Basic Shapes using Iterations

- Using iterations, you can draw basic shapes easily. For example, to draw a series of shapes like squares, triangles, or other polygons, you can use loops to repeat the necessary commands. Here are examples of drawing squares and triangles using iterations:

Drawing Squares with Iterations:

```
1. import turtle
   screen = turtle.Screen()
2. my_turtle = turtle.Turtle()
   # Loop to draw four squares
3. for _ in range(4):
4.     for _ in range(4):  # Drawing each side of the square
5.         my_turtle.forward(100)
6.         my_turtle.left(90)
7.     my_turtle.left(10)  # Rotate for the next square
8. turtle.done()
```

Drawing Basic Shapes using Iterations+

- Drawing Triangles with Iterations:

- In the below code example The outer loop in this example repeats the process six times to draw six triangles. The inner loop draws each side of the triangle by moving forward and turning left 120 degrees.

```
1.  import turtle
    screen = turtle.Screen()
2.  my_turtle = turtle.Turtle()
3.  # Loop to draw six triangles
4.  for _ in range(6):
5.      for _ in range(3):  # Drawing each side of the triangle
6.          my_turtle.forward(100)
7.          my_turtle.left(120)
8.      my_turtle.left(60)  # Rotate for the next triangle
9.  turtle.done()
```

Changing Color Dynamically using List

- You can change the color of the turtle dynamically using a list to cycle through a range of colors. Here's an example of how you can do this:

```
1. import  turtle
2. screen = turtle.Screen()
3. my_turtle = turtle.Turtle()
4. # List of colors
5. colors = ['red', 'blue', 'green', 'orange', 'purple']
   # Loop to draw shapes with changing colors
6. for i in range(20):  # Drawing 20 shapes
7.     my_turtle.color(colors[i % len(colors)])  # Select color from the list using modulus
8.     for _ in range(4):  # Drawing each side of the shape (square in this case)
9.         my_turtle.forward(100)
10.        my_turtle.left(90)
11.    my_turtle.left(18)  # Rotate for the next shape
12.turtle.done()
```

Changing Color Dynamically using List+

- In this example, the list colors contains different color names. The loop cycles through these colors using the modulus operator (i % len(colors)) to ensure that the color index remains within the range of available colors.
- This code draws 20 shapes, each with a different color from the list. You can modify the number of shapes or colors in the colors list to suit your requirements.

Turtles to Create Bar Charts

Code Example: Creating Bar Charts with turtle

```
import turtle
# Function to create a bar
def draw_bar(t, height):
    t.begin_fill()
    t.left(90)
    t.forward(height)
    t.write(" " + str(height))  # Display the height next to the bar
    t.right(90)
    t.forward(40)
    t.right(90)
    t.forward(height)
    t.left(90)
    t.end_fill()
    t.forward(10)  # Space between bars
•
# # Set up the screen and turtle
screen = turtle.Screen()
screen.title("Bar Chart with Turtles")
my_turtle = turtle.Turtle()
my_turtle.color("blue", "cyan")  # Outline color, fill color
data = [48, 117, 200, 240, 160, 260, 220]
# Drawing the bar chart
for i in data:
    draw_bar(my_turtle, i)
screen.mainloop()
```

Code Example: Creating Bar Charts with turtle+

- This example uses a simple function draw_bar() to draw each bar of the chart. The function takes a turtle t and a height to create a bar of that specific height. The chart is created by iterating through the data list and drawing a bar for each value in the list.
- You can modify the data list to represent different data sets and adjust the colors, dimensions, and additional details to suit your visualization requirements.

Project example: Turtle Race Game:

- Develop a simple game where multiple turtles move across the screen, simulating a race. Users can bet on the winning turtle.

```
1.  import turtle
2.  import random
3.  # Set up the screen
4.  screen = turtle.Screen()
5.  screen.title("Turtle Race Game")
6.  screen.setup(width=800, height=400)
7.  # Create turtles for the race
8.  num_turtles = 5
9.  turtles = []
10. # Colors for each turtle
11. colors = ["red", "blue", "green", "orange", "purple"]
12. # Initialize turtles at starting positions
```

Project example: Turtle Race Game+

```
13. for i in range(num_turtles):
14.     new_turtle = turtle.Turtle()
15.     new_turtle.shape("turtle")
16.     new_turtle.color(colors[i])
17.     new_turtle.penup()
18.     new_turtle.goto(-350, -100 + i * 50)  # Starting position for each turtle
19.     turtles.append(new_turtle)
20. # Function to move each turtle randomly
21. def move_turtle(t):
22.     distance = random.randint(1, 10)
23.     t.forward(distance)
24. # Ask the user for a bet
25. user_bet = screen.textinput("Make your bet", "Which turtle will win the race? Enter a color: ")
26. # Race loop
27. winner = None
```

Project example: Turtle Race Game++

```
28. while True:
29.     for turtle in turtles:
30.         move_turtle(turtle)
31.         x = turtle.xcor()
32.         if x >= 350:  # Check if a turtle has reached the finish line
33.             winner = turtle.pencolor()
34.             break
35.     if winner:
36.         break
37. # Display the winner
38. if user_bet.lower() == winner.lower():
39.     result = "Congratulations! You won! The {} turtle is the winner!".format(winner)
40. else:
41.     result = "Sorry, you lost. The {} turtle won the race.".format(winner)
42. screen.textinput("Race Over", result)  # Display the race result
43. turtle.done()
```

Explanation of the Turtle Race game project

- Import and Initialization: The code imports the turtle module and creates a screen for the game. It also initializes an empty list, turtles, to store the turtle participants.
- Color Assignment: Colors are assigned to each turtle from the colors list.
- Turtle Initialization: Five turtles are created, each given a color, shape, and positioned at the starting line.
- User Input: The program prompts the user to place a bet by selecting a color (representing a turtle) to win the race.
- Race Simulation: The game enters a loop where each turtle moves forward until one of them crosses the finish line (x >= 350). When a turtle reaches the finish line, it's recognized as the winner.
- Outcome Determination: Compares the user's bet with the winning turtle and displays the result via a message box using textinput().

A program to display the multiplication table from 1 to 10 in the Turtle graphics window

```
import turtle as t
t.penup()
x = -100
y = 100
t.goto(x,y) #Move pen at location x and y
t.penup()
for i in range(1,11,1): # value of i varies from 1 to 10
    y = y - 20
    for j in range(1,11,1): # Value of j varies from 1 to 10
        t.penup()
        t.speed(1)
        t.forward(20)
        t.write(i*j)
    t.goto(x, y)
t.done()[1]
```

Code Explanation:

- import turtle as t: Imports the turtle module and uses the alias t for convenience.
- t.penup(): Lifts the pen, preventing the turtle from drawing while moving.
- x = -100 and y = 100: Initializes the starting coordinates for the grid.
- t.goto(x, y): Moves the turtle's pen to the specified initial coordinates.
- for i in range(1, 11, 1):: Begins the outer loop that iterates from 1 to 10 (inclusive).
- y = y - 20: Decrements the y-coordinate by 20 to position the next row.

Code Explanation:

- for j in range(1, 11, 1):: Initiates the inner loop that also iterates from 1 to 10 (inclusive).
- t.penup(): Lifts the pen before moving.
- t.speed(1): Sets the speed of the turtle to 1 (slowest speed) for a more controlled display.
- t.forward(20): Moves the turtle forward by 20 units without drawing a line.
- t.write(i * j): Writes the product of i * j at the current location of the turtle.

Code Explanation:

- t.goto(x, y): Moves the turtle to the beginning of the next row.
- t.done(): Ends the turtle graphics and closes the window.
- Summary:
- The code uses nested loops to create a 10x10 grid, where each cell contains the product of i * j for i and j ranging from 1 to 10. The turtle moves across the grid, writing the multiplication table values without drawing lines, creating a visual representation of the multiplication table.

Reference

[1] Kamthane, A. N., & Kamthane , A. A. (2018). PROGRAMMING AND PROBLEM SOLVING WITH PYTHON (p. 346). McGraw Hill Education (India) Private Limited.

[2] (2020, October 20). *Draw Colorful Spiral Web Using Turtle Graphics in Python*. GeeksforGeeks. Retrieved November 14, 2023, from https://www.geeksforgeeks.org/draw-colorful-spiral-web-using-turtle-graphics-in-python/

[3] (n.d.). *Turtle - Turtle graphics*. Python Documentation. Retrieved November 14, 2023, from https://docs.python.org/3/library/turtle.html