EECS 12 Project 2: Python Library Turtle

1. **Overview of Turtle**

Turtle is an inbuilt Python library that allows the user to create drawings in a graphics canvas. It is a very simple way for programmers to create programs with graphical interfaces. The nice part about turtle is that its nearly 100% compatible with previous versions up to 2.5. Also, it uses tkinter for the underlying graphics.

1. **Model (Classes)**

\*\*\* To use Turtle classes, all you have to do is use the following code: \*\*\*\*

from turtle import \*

Turtle Motion:

Using the motion class of turtle is very much like appending a list.

For example:

Appending a list requires you to ensure that the list is defined and then to use an append statement.

>>> new\_ list = []

>>> new\_list.append( 10)

>>> print(new\_list)

>>> 10

Similarly:

Turtle simply needs you to specify the directional command of the cursor.

>>> from turtle import \*

>>> orign = turtle.position()

>>> origin

(0,0)

>>> origin.forward(25)

>>> origin

(25, 0)

>>> origin.backward(25)

>>> origin

(0, 0)

**- Attributes**

Distance: (float) movement for specific distances

Coordinates: (pair of coordinates) used to find specific distances and establish starting and ending points

Input: (float) used to establish specific coordinates

**- Example use**

>>> import turtle

>>> origin = turtle.position()

>>> origin.forward(25)

>>> origin.backward(25)

>>> origin.right(50)

>>> origin.left(50)

>>> point = turtle.setposition(10, 10)

>>> point

(10,10)

>>> point = turtle.home()

>>> point

(0,0)

>>> point.setheading()

(0,0)

>>> circle = turtle.circle(10, 15, 20)

>>> origin.undo()

>>> tutle.speed(‘fast’)

>>> x\_cord = turtle.setx(10)

>>> y\_cord = turtle.sety(10)

Turtle Canvas:

The Turtle Canvas Class is an extremely versatile class because it allows for quick and easy accessibility of the canvas. The canvas is the working station of the turtle graphics window and includes everything from color to cartesian coordinate systems.

It is implemented the same way as the Turtle Motion Class:

>>> import turtle

>>> canvas = turtle.position() #initializes the canvas at point (0,0); no input is read as (0,0)

>>> canvas.stamp() #creates a turtle stamp

>>> canvas.clearstamp() #removes the turtle stamp

**- Attributes**

Coordinates = (float) initialize and establish beginning and ends of drawings

Speed = (string/fast) establishes the speed at which cursor moves

**- Example Use**

>>> import turtle

>>> canvas = turtle.position()

>>> canvas.stamp()

>>> canvas.clearstamp()

>>> canvas.position()

>>> canvas

(0, 0)

>>> canvas.xcor()

>>> 0.00

>>> canvas.ycor()

>>> 0.00

>>> canvas.heading()

>>> canvas.color(‘blue’)

>>> canvas.filling()

True

>>> canvas.reset()

>>> canvas.clear()

>>> canvas.showturtle() #shows the cursor

Turtle Appearance:

Since the turtle library is a fun and interactive part of the python language, it, also, allows for the user to change the cursor! The cursor can change from anything like a triangle or an arrow to an actual turtle. The cursor class also includes colors to accommodate the cursor styling with.

It is implemented in the same way as the Turtle Motion Class:

For example,

>>> import turtle

>>> cursor = turtle.shape(‘triangle’)

>>> cursor.resizemode(10, 20) #image drawn by the cursor accommodates this restriction

**- Attributes**

Shape = (string) changes shape of the cursor

Boundaries = (float) changes the boundaries of a shape and the cursor

**- Examples**

>>> import turtle

>>> cursor = turtle.shape(‘triangle’)

>>> cursor.resizemode(10, 20)

>>> shape = turtle.circle(5, 10, 20)

>>> shape.rmode(1, 2, 4)

>>> shape.shapesize(3)

>>> cursor.turtlesize(5)

Turtle Inputs:

The Turtle library allows multiple ways for a user to input values when a function is run. Instead of the conventional input() call, the library allows for mouse clicks, and the pressing/release of a key.

It is implemented in the same way as the Turtle Motion Class:

>>> import turtle

>>> user\_in = turtle.onclicks()

>>> user\_in = turtle.release()

>>> user\_in = turtle.ondrag()

**- Attributes**

User\_ in = mouse click or pressing a key

**- Examples**

>>> import turtle

>>> user\_in = turtle.onclicks()

>>> user\_in = turtle.release()

>>> user\_in = turtle.ondrag()

Special Turtle Methods

The Special Turtle methods is a class of methods that establishes faster ways to do tasks that are reoccurring when programming in Turtle. It’s basically a class of shortcuts that simplify coding for the programmer and makes the code more readable.

It is implemented in the same way as the Turtle Motion Class:

>>> import turtle

>>> circle = circle(5, 10, 20)

>>> poly = circle.begin\_poly(‘circle’)

>>> poly.end\_poly()

**- Attributes**

Coordinates = (float) Begins the function from the given coordinate

**- Examples**

>>> import turtle

>>> circle = circle(5, 10, 20)

>>> poly = circle.begin\_poly(‘circle’)

>>> poly.end\_poly()

>>> poly.get\_poly()

True

>>> poly.clone()

>>> poly.getturtle() #gets the cursor direction

>>> poly.getscreen()

1. **Functions**

Turtle Motion

forward

backward

right

left

setposition

home

setheading

circle

undo

speed

setx

sety

Turtle Canvas

stamp

clearstamp

position

xcor

ycor

heading

color

filling

reset

clear

showturtle

Turtle Appearance

shape

resizemode

shapesize

turtlesize

Turtle Inputs

onclicks

onrelease

ondrag

Special Turtle Methods

begin\_poly

end\_poly

get\_poly

clone

getturtle

getscreen

Turtle Motion

1. forward(distance)

Moves cursor forward

**- Attributes**

Distance: (float) Cursors moved by this distance

**- Examples**

>>> import turtle

>>> origin = turtle.position()

>>> origin.forward(25)

>>> origin

(25, 0)

1. backward(distance)

Moves cursor backwards

**- Attributes**

Distance: (float) Cursors moved by this distance

**- Examples**

>>> import turtle

>>> origin = turtle.position()

>>> origin.backward(25)

>>> origin

(-25, 0)

1. right(angle)

Moves cursor by the angle to the right

**- Attributes**

Angle: (float) Moves cursor by the given angle

**- Examples**

>>> import turtle

>>> origin = turtle.position()

>>> origin.right(25)

>>> origin

25.00

1. left(angle)

Moves cursor by the angle to the left

**- Attributes**

Angle: (float) Moves cursor by the given angle

**- Examples**

>>> import turtle

>>> origin = turtle.position()

>>> origin.left(25)

>>> origin

-25.00

1. setposition(x,y)

Sets coordinates to the inputted position

**- Attributes**

Coordinates: (float) Sets the position to these coordinates

**- Examples**

>>> import turtle

>>> point = turtle.setposition(10, 10)

>>> point

(10,10)

1. home()

Returns the coordinates to home

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> point = turtle.home()

>>> point

(0,0)

1. setheading()

Gets the input of last line

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> point = turtle.setposition(10, 10)

>>> point.setheading()

(10,10)

1. circle(radius, extent, steps)

Creates circle

**- Attributes**

Radius: (float) radius value

Extent: (float) angle

Steps: (integer) number of steps taken to draw

**- Examples**

>>> import turtle

>>> circle = turtle.circle(10, 15, 20)

>>> circle

(10, 15, 20)

1. undo()

Undoes previous line

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> circle = turtle.circle(10, 15, 20)

>>> circle.undo()

>>> circle

None

1. speed(‘speed’)

Moves cursor at inputted speed

**- Attributes**

Speed = (list/integer) Speed at which cursor moves

**- Examples**

>>> import turtle

>>> circle = turtle.circle(10, 15, 20)

>>> circle.speed(‘fast’)

1. setx(coordinates)

Sets x coordinate to given input

**- Attributes**

Coordinates: (float) Coordinate input for x

**- Examples**

>>> import turtle

>>> x\_cord = turtle.setx(10)

>>> x\_cord

(10, 0)

1. sety(coordinates)

Sets y coordinate to given input

**- Attributes**

Coordinates: (float) Coordinate input for y

**- Examples**

>>> import turtle

>>> y\_cord = turtle.sety(10)

>>> y\_cord

(0,10)

Turtle Canvas

1. stamp()

Stamps the canvas

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> canvas = turtle.position()

>>> canvas.stamp()

1. clearstamp()

Clears stamp off canvas

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> canvas = turtle.position()

>>> canvas.stamp()

>>> canvas.clearstamp()

1. position()

Finds the position of the cursor

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> canvas = turtle.position()

>>> canvas.stamp()

>>> canvas.clearstamp()

>>> canvas.position()

>>> canvas

(0, 0)

1. xcor()

Find x coordinate of cursor

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> canvas = turtle.position()

>>> canvas.xcor()

0.00

1. ycor()

Find y coordinate of cursor

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> canvas = turtle.position()

>>> canvas.ycor()

>>> 0.00

1. heading()

Return the current heading (heading depends on the turtle mode)

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> canvas = turtle.position()

>>> canvas.heading()

1. color(Color)

Return pencolor and fillcolor

**- Attributes**

Color: (String) Color input

**- Examples**

>>> import turtle

>>> canvas = turtle.position()

>>> canvas.color(‘blue’)

1. filling()

Determines whether the shape is colored

**- Attributes**

Null = (Boolean) color

**- Examples**

>>> import turtle

>>> canvas = turtle.position()

>>> canvas.filling()

1. reset()

Reset the canvas

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> canvas = turtle.position()

>>> canvas.reset()

1. clear()

Clears the graphic window completely

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> canvas = turtle.position()

>>> canvas.clear()

1. Showturtle()

Shows the cursor

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> canvas = turtle.position()

>>> canvas.showturtle()

Turtle Appearance

1. Shape(shape)

Changes the cursor to inputted shape

**- Attributes**

Shape = (string) Shape for cursor

**- Examples**

>>> import turtle

>>> cursor = turtle.shape(‘triangle’)

1. Resizemode(x, y)

Resizes the shape drawn by cursor

**- Attributes**

Sizes = (float, float)

**- Examples**

>>> import turtle

>>> cursor = turtle.shape(‘triangle’)

>>> cursor.rmode(1, 2, 4)

1. shapesize(magnification)

Changes the size of the shape

**- Attributes**

Magnification = (integer) increases the size of the shape by the magnification

**- Examples**

>>> import turtle

>>> shape = turtle.circle(5, 10, 20)

>>> shape.shapesize(3)

1. turtlesize(magnification)

Changes the cursor sides

**- Attributes**

Magnification = (integer) increases the size of the cursor

**- Examples**

>>> import turtle

>>> cursor = turtle.shape(‘triangle’)

>>> cursor.turtlesize(5)

Turtle Inputs

1. onclicks

Inputs are mouse clicks

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> user\_in = turtle.onclicks()

1. onrelease

Inputs are on key release

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> user\_in = turtle.release()

1. ondrag

Inputs are on key press

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> user\_in = turtle.ondrag()

Special Turtle Methods

1. begin\_poly(function)

Begin drawing polygon

**- Attributes**

Input = (function) function that needs to be started

**- Examples**

>>> import turtle

>>> circle = circle(5, 10, 20)

>>> poly = circle.begin\_poly(‘circle’)

1. end\_poly

Ends polygon drawing

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> circle = circle(5, 10, 20)

>>> poly = circle.begin\_poly(‘circle’)

>>> poly.end\_poly()

1. get\_poly

Determine if polygon was drawn

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> circle = circle(5, 10, 20)

>>> poly = circle.begin\_poly(‘circle’)

>>> poly.end\_poly()

>>> poly.get\_poly()

True

1. clone

Draws duplicate of the polygon

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> circle = circle(5, 10, 20)

>>> poly = circle.begin\_poly(‘circle’)

>>> poly.end\_poly()

>>> poly.get\_poly()

True

>>> poly.clone()

1. getturtle

Finds the position of the cursor in the drawing

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> circle = circle(5, 10, 20)

>>> poly = circle.begin\_poly(‘circle’)

>>> poly.end\_poly()

>>> poly.get\_poly()

True

>>> poly.clone()

>>> poly.getturtle()

1. getscreen

Finds the screen upon which drawing was made

**- Attributes**

Null

**- Examples**

>>> import turtle

>>> circle = circle(5, 10, 20)

>>> poly = circle.begin\_poly(‘circle’)

>>> poly.end\_poly()

>>> poly.get\_poly()

True

>>> poly.clone()

>>> poly.getturtle()

>>> poly.getscreen()

Conclusion:

Python is a powerful language that comes equipped with many powerful libraries, but what makes Turtle stand out is its unique casualness. It is not as intensive as other libraries and is very simply to use with extremely good results at the end of a program. With customizable cursors and colors, Turtle is definitely a great way to learn about the graphics aspect of programming.