Python for Kids

Connect with me: Youtube | LinkedIn | WhatsApp Channel | Web | Facebook | Twitter

- Download PDF
- To access the updated handouts, please click on the following link: index.html

Python Turtle is a fantastic tool for kids to learn coding concepts in a fun and visual way. By using simple commands, kids can control a virtual turtle to draw shapes, designs, and even create animations! Here's a quick guide to get you started:

- Setting Up:
  - Download Python from [www.python.org] and install it on your computer.
  - Once installed, open a text editor like Notepad or use a Python IDE like Thonny which is beginner-friendly.
- Importing Turtle:
  - In your text editor, type the following line:

```
import turtle
```

- This line imports the Turtle library, giving you access to all the turtle commands.
- Controlling the Turtle:
  - Let's create a turtle! Type the following line:

```
t = turtle.Turtle()
```

- Now, you can use commands to move the turtle around the screen. Here are some basic commands:
  - t.forward(distance): Moves the turtle forward by the specified distance.
  - t.backward(distance): Moves the turtle backward by the specified distance.
  - t.right(angle): Turns the turtle right by the specified angle (in degrees).
  - t.left(angle): Turns the turtle left by the specified angle (in degrees).

- Drawing Shapes:
  - By combining these commands, you can draw shapes! For example, to draw a square, you can use the following code:

```
for i in range(4):
    t.forward(100)
    t.right(90)
```

- Experimenting:
  - Try these commands to explore what the turtle can do:

    - t.pensize(width): Change the thickness of the turtle's pen.
    - t.pencolor("color"): Change the color of the turtle's pen (e.g., "red", "blue", "green").
    - t.penup(): Lift the turtle's pen to stop drawing.
    - t.pendown(): Put the turtle's pen down to start drawing again.
-

```
import turtle

# Create a turtle named "t"
t = turtle.Turtle()

# Set the background color (optional)
t.screen.bgcolor("lightblue")

# Make the pen thicker (change the number to see the difference)
t.pensize(5)  # Try values like 2, 8, or even 15!

# Move the turtle forward to draw a line
t.forward(100)

# Let's draw another line with a different size
t.pensize(2)
t.left(90)  # Turn the turtle 90 degrees left
t.forward(50)

# Keep playing! Try different colors and pen sizes
t.pencolor("red")
t.pensize(10)
t.right(180)  # Turn 180 degrees (full circle)
t.forward(80)
```

Run this code and see how the thickness of the lines changes with different pensize values. You can even change the pen color using pencolor("color name") (like "red", "blue", or "green").

Imagine a turtle with a marker strapped to its belly. In Python's turtle graphics, this turtle can draw on a screen! The `pendown()` and `penup()` commands control the marker, just like a pen.

- `pendown()`: This puts the marker down, so the turtle will draw a line as it scoots around.
- `penup()`: This picks the marker up, so the turtle can move without leaving a trace.

Here's a fun example:

```
import turtle

# Create a turtle named "t"
t = turtle.Turtle()

# Set the turtle's speed (optional, slower is easier to see)
t.speed(1)
```

```
# Make the turtle draw a square
t.pendown()
t.forward(100)  # Move forward 100 pixels
t.right(90)     # Turn right 90 degrees
t.forward(100)
t.right(90)
t.forward(100)
t.right(90)
t.forward(100)

# Now, let's move the turtle to a new spot without drawing
t.penup()
t.forward(200)  # Move forward 200 pixels without drawing
t.right(90)
t.pendown()

# Draw a triangle with a different color (optional)
t.pencolor("red")
t.forward(150)
t.left(120)
t.forward(150)
t.left(120)
t.forward(150)

# Keep the window open until you close it
turtle.done()
```

Try running this code! You'll see the turtle draw a square, then move without drawing to a new spot, and finally draw a red triangle. Play around with the code:

- Change the numbers in `forward()` to draw different shapes.
- Try different colors with `pencolor("blue")`.
- Can you make the turtle draw a star?

Remember, `pendown()` is for drawing, and `penup()` is for moving without a trace. With these commands, you can control your turtle to create all sorts of cool pictures!

With these basic commands, kids can start creating their own drawings and animations using Python Turtle. There are many online resources and tutorials that provide more advanced projects as well!

Sure, here's some more you can explore with Python Turtle:

- Loops and Functions: Let's try drawing a spiral! You can use a loop to repeat the same drawing commands multiple times, slightly changing the angle each time to create a spiral effect. Here's an example:

```
import turtle
t = turtle.Turtle()

for i in range(30):
```

```
    t.forward(i * 5)
    t.right(24)
```

- Challenge: Try modifying this code to change the direction of the spiral (clockwise or counter-clockwise) or the spacing between the loops.
- Variables: Introduce variables to customize your drawings. For instance, you can store the side length of a square in a variable and use it to draw squares of different sizes.

Let's draw some fun shapes with our turtle friend! Here are a few examples in Python using turtle graphics:

1. Square:

This is a basic shape to get started.

```
import turtle

# Create a turtle named "t"
t = turtle.Turtle()

# Set the turtle's speed (optional, slower is easier to see)
t.speed(1)

# Make the turtle draw a square with a black pen
t.pendown()
t.forward(100)  # Move forward 100 pixels to draw a side
t.right(90)     # Turn right 90 degrees to change direction
t.forward(100)
t.right(90)
t.forward(100)
t.right(90)
t.forward(100)

# Keep the window open until you close it
turtle.done()
```

2. Triangle:

Change the angles slightly to draw a triangle!

```
import turtle

# Create a turtle named "t"
t = turtle.Turtle()

# Set the turtle's speed (optional)
t.speed(1)

# Make the turtle draw a triangle with a blue pen
t.pendown()
```

```
t.pencolor("blue")
t.forward(150)  # Move forward for the first side
t.left(120)     # Turn left 120 degrees for triangle angles
t.forward(150)
t.left(120)
t.forward(150)

# Keep the window open
turtle.done()
```

3. Circle:

The turtle doesn't draw a perfect circle by itself, but we can get close!

```
import turtle

# Create a turtle named "t"
t = turtle.Turtle()

# Set the turtle's speed (optional)
t.speed(1)

# Make the turtle draw a circle (many small lines)
t.pendown()
for i in range(360):  # Loop 360 times for a full circle
    t.forward(1)
    t.right(1)  # Small turn for each line

# Keep the window open
turtle.done()
```

4. Star:

Combine lines and turns to create a cool star!

```
import turtle

# Create a turtle named "t"
t = turtle.Turtle()

# Set the turtle's speed (optional)
t.speed(1)

# Make the turtle draw a yellow star
t.pendown()
t.pencolor("yellow")
for i in range(5):  # Draw 5 lines for the star points
    t.forward(100)
    t.right(144)  # Angle to create a pointed star
```

```
# Keep the window open
turtle.done()
```

Remember, you can change the colors with `pencolor("color")`, adjust the distances with `forward(distance)`, and play with the turning angles (`right(angle)`) to create all sorts of shapes! Explore and have fun!

```
import turtle
t = turtle.Turtle()

size = 100

# Draw a square with side length 'size'

for i in range(4):
    t.forward(size)
    t.right(90)
```

- Challenge: Modify this code to draw a rectangle using separate variables for width and height.

I'll provide the next lecture content on enhancing Python Turtle creations: Adding Color and Customization Python Turtle offers a vibrant range of colors and customization options to bring your drawings to life!

- Pen Color:
  - Use t.pencolor("color_name") to set the pen color. Explore various color names like "red", "blue", "yellow", or experiment with hex codes for more specific shades (e.g., t.pencolor("#FF0000") for red).
- Fill Color:
  - Create filled shapes using t.fillcolor("color_name") before drawing. Combine it with t.begin_fill() and t.end_fill() to enclose the area you want to fill.

t.fillcolor("yellow") t.begin_fill()

Draw rectangle here

t.end_fill()

Here's an example of using `fillcolor` to draw a colored rectangle in Python turtle:

```
import turtle

# Create a turtle named "t"
t = turtle.Turtle()

# Set the turtle's speed (optional, slower is easier to see)
t.speed(1)
```

```
# Set the fill color (example: blue)
t.fillcolor("blue")

# Tell the turtle to start filling upcoming shapes
t.begin_fill()

# Draw the rectangle (using forward and right)
t.pendown()
t.forward(100)  # Length of the rectangle
t.right(90)     # Turn 90 degrees
t.forward(50)   # Width of the rectangle
t.right(90)
t.forward(100)
t.right(90)
t.forward(50)

# Tell the turtle to stop filling
t.end_fill()

# Keep the window open until you close it
turtle.done()
```

Explanation:

1. We import the `turtle` library.
2. We create a turtle object named `t`.
3. We set the turtle's speed (optional).
4. We set the fill color using `t.fillcolor("blue")`. You can replace "blue" with any color name or hex code.
5. We use `t.begin_fill()` to tell the turtle to start filling the upcoming closed shape.
6. We draw the rectangle using `pendown()`, `forward()`, and `right()`.
    - `forward(100)` moves the turtle forward 100 pixels, creating the length of the rectangle.
    - `right(90)` turns the turtle 90 degrees four times to complete the rectangle.
    - Adjust these values to change the size of your rectangle.
7. We use `t.end_fill()` to tell the turtle to stop filling.
8. Finally, `turtle.done()` keeps the window open until you close it.

This code will draw a blue rectangle on the screen. You can experiment with different colors and rectangle sizes!

- Pen Size:
    - Control the thickness of the line using t.pensize(width). Higher values create thicker lines.
- Background Color:
    - Change the background color of the drawing window using t.bgcolor("color_name").

Imagine you're painting a picture. Before you start drawing all the fun shapes and characters, you choose a background color, right? In Python's turtle graphics, the `bgcolor()` function lets you set the background color for your drawings!

Think of it like painting the canvas before your turtle friend starts creating its masterpiece. Here's how it works:

```
import turtle

# Create a turtle named "t"
t = turtle.Turtle()

# Set the background color (example: light blue)
turtle.bgcolor("lightblue")

# Now the turtle can draw on a light blue background!

# (You can add your drawing code here)

# Keep the window open until you close it
turtle.done()
```

In this example:

1. We import the `turtle` library.
2. We create a turtle object named `t`.
3. We use `turtle.bgcolor("lightblue")` to set the background color to light blue. You can try other color names or hex codes.
4. We add a comment `# (You can add your drawing code here)` to show where you would place your drawing commands.
5. Finally, `turtle.done()` keeps the window open.

Experimenting with bgcolor:

- Try different colors like "green", "yellow", or even "purple"!
- Can you draw a sunny day with a yellow background and a blue square for the sky?
- Maybe a nighttime scene with a black background and white stars drawn by the turtle?

Remember, `bgcolor` sets the color behind everything the turtle draws. Play around and see what cool backgrounds you can create for your turtle pictures!

Drawing More Complex Shapes

Circles and Arcs:

- Use t.circle(radius) to draw a full circle. For arcs, use t.arc(radius, angle, extent), where radius is the circle's radius, angle is the starting angle, and extent is the arc's central angle in degrees.

Using the circle() function:

The turtle library actually has a built-in function for circles! This is a bit more advanced, but it's a shortcut to draw a perfect circle.

```
import turtle

# Create a turtle named "t"
t = turtle.Turtle()

# Set the turtle's speed (optional)
t.speed(1)

# Make the turtle draw a circle with a radius of 50
t.pendown()
t.circle(50)  # Circle with radius 50

# Keep the window open
turtle.done()
```

Explanation:

1. We import the `turtle` library.
2. We create a turtle object named `t`.
3. We set the turtle's speed (optional).
4. We use `t.circle(50)`. This function draws a circle with a radius of 50 pixels. The bigger the number, the bigger the circle.

Challenge:

- Try changing the number inside `t.circle( )` to see how it affects the circle size.
- Can you draw multiple circles of different sizes?
- Add some color with `t.pencolor("color")` before drawing the circle!

Custom Shapes:

- Create custom shapes using t.penup(), t.goto(x, y), and t.pendown() to move the turtle without drawing and define the shape's outline.

Absolutely! By combining `t.penup()`, `t.goto(x, y)`, and `t.pendown()`, you can create various custom shapes in Python's turtle graphics. Here's how:

1. **Plan your shape:** Think about the basic lines and turns that make up your desired shape. Break it down into smaller segments.
2. **Use `t.penup()` for movement:** When the turtle doesn't need to draw, use `t.penup()` to lift the pen and move it to the starting point of the next line segment.
3. **Use `t.goto(x, y)` for positioning:** This function precisely positions the turtle at a specific coordinate (x, y) on the screen.
4. **Use `t.pendown()` for drawing:** When you're ready to draw a line segment, use `t.pendown()` to put the pen down and start drawing.

Here's an example to draw a house:

```
import turtle

# Create a turtle named "t"
t = turtle.Turtle()

# Set the turtle's speed (optional)
t.speed(1)

# House base (square)
t.pendown()
t.forward(100)
t.right(90)
t.forward(50)
t.right(90)
t.forward(100)
t.right(90)
t.forward(50)
t.penup()  # Lift the pen to move to the roof

# Move the turtle to the roof starting point
t.goto(50, 50)  # Move to (x, y) coordinates
t.pendown()

# Roof (triangle)
t.left(60)  # Adjust angle for roof slope
t.forward(70)
t.right(120)
t.forward(140)
t.right(120)
t.forward(70)

# Door (rectangle)
t.penup()
t.goto(20, 0)  # Move to door position
t.pendown()
t.forward(30)
t.right(90)
t.forward(20)
t.right(90)
t.forward(30)
t.right(90)
t.forward(20)

# Keep the window open
turtle.done()
```

Challenge:

- Try drawing a car or a simple flower using these techniques.
- Experiment with different colors for the pen using `t.pencolor("color")`.
- Can you add details like windows to the house or wheels to the car?

Exploring Creative Techniques

- Patterns and Repetition:
  - Leverage loops (like for loops) to repeat drawing commands, creating intricate patterns and designs.
- Animations:
  - Combine multiple shapes, movements, and color changes within loops to produce basic animations. Example: Colorful Spiral

Absolutely! By combining `t.penup()`, `t.goto(x, y)`, and `t.pendown()`, you can create various custom shapes in Python's turtle graphics. Here's how:

1. **Plan your shape:** Think about the basic lines and turns that make up your desired shape. Break it down into smaller segments.
2. **Use `t.penup()` for movement:** When the turtle doesn't need to draw, use `t.penup()` to lift the pen and move it to the starting point of the next line segment.
3. **Use `t.goto(x, y)` for positioning:** This function precisely positions the turtle at a specific coordinate (x, y) on the screen.
4. **Use `t.pendown()` for drawing:** When you're ready to draw a line segment, use `t.pendown()` to put the pen down and start drawing.

Here's an example to draw a house:

```
import turtle

# Create a turtle named "t"
t = turtle.Turtle()

# Set the turtle's speed (optional)
t.speed(1)

# House base (square)
t.pendown()
t.forward(100)
t.right(90)
t.forward(50)
t.right(90)
t.forward(100)
t.right(90)
t.forward(50)
t.penup()  # Lift the pen to move to the roof

# Move the turtle to the roof starting point
t.goto(50, 50)  # Move to (x, y) coordinates
t.pendown()

# Roof (triangle)
t.left(60)  # Adjust angle for roof slope
t.forward(70)
t.right(120)
t.forward(140)
```

```
t.right(120)
t.forward(70)

# Door (rectangle)
t.penup()
t.goto(20, 0)  # Move to door position
t.pendown()
t.forward(30)
t.right(90)
t.forward(20)
t.right(90)
t.forward(30)
t.right(90)
t.forward(20)

# Keep the window open
turtle.done()
```

Challenge:

- Try drawing a car or a simple flower using these techniques.
- Experiment with different colors for the pen using `t.pencolor("color")`.
- Can you add details like windows to the house or wheels to the car?

```
import turtle
t = turtle.Turtle()
speed = 10

for i in range(100):
    t.pencolor("lightblue")
    t.circle(i * speed)
    t.right(90)
    t.pencolor("orange")
    t.circle(i * speed - 20)
    t.left(90)
```

With these enhancements, you're well on your way to crafting even more expressive and visually engaging Python Turtle creations!

In the next lecture on Python Turtle, we'll explore exciting ways to make your programs more interactive and visually interesting:

- Working with Images:
    - Use t.screen.addshape("image_name.gif") to load an image file (GIF format). Then, t.shape("image_name") sets the turtle's shape to the loaded image. While Python Turtle doesn't directly add images like GIFs, it can add custom shapes defined elsewhere. Here's an explanation with a fun alternative for kids:

Using `t.screen.addshape()` (for older kids):

This method is for slightly more advanced users. It involves creating a separate image file (like a GIF) and then linking it to the turtle program. Here's a basic example:

1. **Create a GIF image:** You'll need an external image editing tool to create a GIF image. This could be a simple drawing or animation.
2. **Save the GIF:** Save the image with a recognizable name, for example, "myshape.gif". Make sure you remember where you save it!
3. **Link the GIF in your Python code:**

```
import turtle

# Create a turtle named "t"
t = turtle.Turtle()

# Set the screen object
screen = t.screen

# Add the image shape (replace "myshape.gif" with your filename)
screen.addshape("myshape.gif")

# Now you can use t.shape("myshape.gif") to draw the image!
```

Explanation:

1. We import the `turtle` library.
2. We create a turtle object named "t" and get the screen object.
3. We use `screen.addshape("myshape.gif")` to link the image file named "myshape.gif" to the turtle program.
4. Once the shape is added, you can use `t.shape("myshape.gif")` to change the turtle's shape to the image. Then you can draw with that shape!

Important note: This approach might be a bit complex for younger kids.

Alternative for younger kids:

- Use Python Turtle's built-in drawing functions and `t.fillcolor()` to create colorful and interesting shapes!
- Explore examples that combine lines, circles, squares, and triangles to make cool pictures.
- Focus on the fun of using code to draw and experiment with different colors and shapes.

Remember, Python Turtle is a great introduction to coding concepts, and building creative images with basic shapes is a fun way to learn!

- Interactive Drawing:
    - Leverage Python's onclick() function to respond to mouse clicks. Here's an example:

```
def click_handler(x, y):
    t.goto(x, y)
```

```
    t.dot(10, "red")  # Draw a red dot at the click position
t.screen.onclick(click_handler)
```

This code creates a function (click_handler) that draws a red dot wherever you click on the screen.

- Event Listeners:
  - Capture keypresses using t.screen.onkeypress(). Here's an example:

```
   def change_color(key):
    if key == "r":
        t.pencolor("red")
    elif key == "g":
        t.pencolor("green")

t.screen.onkeypress(change_color, "r")  # Listen for 'r' keypress
t.screen.onkeypress(change_color, "g")  # Listen for 'g' keypress
```

This code changes the turtle's pen color to red when you press 'r' and green when you press 'g'. By combining these techniques, you can create dynamic and engaging Python Turtle programs that respond to user interaction.

Here are some examples of using `screen.onkeypress` in Python Turtle:

1. Drawing a Square:

This example demonstrates how to draw a square when the spacebar is pressed.

```
import turtle

# Create a turtle named "t"
t = turtle.Turtle()

# Set the turtle's speed (optional)
t.speed(1)

def draw_square():
  # Draw the square using forward and right
  t.pendown()
  t.forward(100)
  t.right(90)
  t.forward(100)
  t.right(90)
  t.forward(100)
  t.right(90)
  t.forward(100)
  t.penup()  # Lift the pen after drawing

# Listen for the spacebar key
screen = t.screen
```

```
screen.onkeypress(draw_square, "space")

# Listen for key presses
screen.listen()

# Keep the window open until you close it
turtle.done()
```

2. Moving the Turtle:

This example shows how to move the turtle up, down, left, and right using the arrow keys.

```
import turtle

# Create a turtle named "t"
t = turtle.Turtle()

# Set the turtle's speed (optional)
t.speed(1)

def move_up():
  t.setheading(90)  # Set heading to face north (up)
  t.forward(50)  # Move forward 50 pixels

def move_down():
  t.setheading(270)  # Set heading to face south (down)
  t.forward(50)

def move_left():
  t.setheading(180)  # Set heading to face west (left)
  t.forward(50)

def move_right():
  t.setheading(0)  # Set heading to face east (right)
  t.forward(50)

# Listen for key presses
screen = t.screen
screen.onkeypress(move_up, "Up")
screen.onkeypress(move_down, "Down")
screen.onkeypress(move_left, "Left")
screen.onkeypress(move_right, "Right")

screen.listen()

# Keep the window open until you close it
turtle.done()
```

3. Changing Pen Color:

This example allows you to change the turtle's pen color by pressing specific keys.

```
import turtle

# Create a turtle named "t"
t = turtle.Turtle()

# Set the turtle's speed (optional)
t.speed(1)

def change_color_red():
  t.pencolor("red")

def change_color_blue():
  t.pencolor("blue")

def change_color_green():
  t.pencolor("green")

# Listen for key presses
screen = t.screen
screen.onkeypress(change_color_red, "r")
screen.onkeypress(change_color_blue, "b")
screen.onkeypress(change_color_green, "g")

screen.listen()

# Keep the window open until you close it
turtle.done()
```

Remember:

- You need to call `screen.listen()` after defining the `onkeypress` events to start listening for key presses.
- Make sure to use quotation marks for the key you want to listen for (e.g., "space").
-

Projects

advanced concepts to elevate your Python Turtle mastery!

- Project Ideas:
  - Drawing Challenge: Pick an object (e.g., flower, car) and challenge yourself to draw it using Python Turtle commands.
  - Animated Games: Create simple games like Pong or Turtle Crossing using turtle movement, loops, and conditional statements.
  - Fractal Art: Explore fractals like the Koch snowflake or Sierpinski triangle using recursive functions to generate complex, self-similar patterns.
- Advanced Concepts:
  - Functions: Organize your code into reusable functions for better readability and maintainability.

- Modules: Create custom modules to store commonly used functions and import them into your projects.
- Classes: Object-oriented programming concepts like classes can help manage complex turtle objects and interactions.

Example: Fractal Snowflake

```
import turtle

def snowflake_arm(t, length, levels):
    if levels == 0:
        t.forward(length)
        return
    snowflake_arm(t, length / 3, levels - 1)
    t.left(60)
    snowflake_arm(t, length / 3, levels - 1)
    t.right(120)
    snowflake_arm(t, length / 3, levels - 1)
    t.left(60)
    snowflake_arm(t, length / 3, levels - 1)

t = turtle.Turtle()
t.speed(0)
snowflake_arm(t, 200, 3)
```

This code demonstrates a recursive function to draw a beautiful snowflake fractal. As you progress, Python Turtle offers a springboard to explore more intricate graphics programming concepts.