Loops & Turtle Graphics

Loops are ideal for repeating the same thing over and over. This program draws the 5 points of a star:

```python
import turtle
for count in range(5):
    turtle.forward(100)
    turtle.right(144)
turtle.hideturtle()
```

To make the result clean, hide the turtle using `hideturtle`.

Loops & Turtle Graphics

Repetition allows you to create more complex shapes easily:

```python
import turtle
turtle.pencolor("green")
for i in range(41):
    turtle.forward(100)
    turtle.right(123)
turtle.hideturtle()
```

Experimentation is the best way to discover new designs.

Other Design Ideas

Flowers

```python
for count1 in range(10):
    for count2 in range(2):
        turtle.forward(100)
        turtle.right(60)
        turtle.forward(100)
        turtle.right(120)
turtle.right(36)
```

The key piece is a nested loop... a `for` loop inside another `for` loop.
Using Randomness

```python
import turtle
import random
for i in range(20):
    if (random.choice([True, False])):
        turtle.right(30)
    else:
        turtle.left(30)
    turtle.forward(40)
    turtle.stamp()
```

Using random values can make computer generated graphics more interesting and unique. Explore methods in the random module.

Turtle Graphic Functions

```python
def draw_two_circles(diam):
    turtle.hideturtle()
    turtle.circle(diam)
    turtle.penup()
    turtle.goto(diam * 2, 0)
    turtle.pendown()
    turtle.circle(50)
```

A function encapsulates a reusable shape or design.

Honeycomb Example

**Step 1: Create reusable functions that draw an individual shape**

```python
def draw_hexagon():
    for count in range(6):
        turtle.forward(50)
        turtle.left(60)
```

**Step 2: Create a function that repeatedly calls the first function**

```python
# Draws a honeycomb shape.
def draw_honeycomb():
    for draw_honeycomb in range(6):
        draw_hexagon()
        turtle.forward(50)
        turtle.right(60)
```

**Step 3: Set color and pen size, and call draw_hexagon.**

```python
# Main program
turtle.penup()
turtle.goto(-35, 50)
turtle.pendown()
turtle.pencolor('gold')
turtle.pensize(7)
draw_honeycomb()
turtle.hideturtle()
```

How to Approach Creating Designs

- Look for examples to inspire new ideas.
- Invent ideas of your own.
- Reuse ideas that you see with your own wrinkles.
- Think Functions! Break up code into smaller units.
- Think Loops! Use for or while loops to repeat things.
- Break up your idea into smaller pieces.
- Create a function for each piece.
Random Numbers

A random number generator is an object that produces a stream of pseudorandom numbers.

They are based on a seed value that factors into a set of calculations.

To the user, they certainly appear random.

The mechanism for generating random numbers in Python is the random module.

Random Numbers

The random module has several methods for generating random numbers.

The `randrange` method accepts an argument `N` and returns an integer in the range 0 to N-1.

```
sum = random.randrange(10)
```

The variable `num` now contains a single integer between 0 and 9, inclusive.

There’s an equal probability of getting any value in that range.

```
um = random.randrange(50) + 1
```

The call to `randrange` returns a value between 0 and 49, which is then shifted into the range 1 to 50.

Random Numbers

In general, a scale factor of X and a shift value of Y produces an integer in the range Y to X + Y – 1.

```
<table>
<thead>
<tr>
<th>Expression</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>random.randint(100)</td>
<td>0 to 99</td>
</tr>
<tr>
<td>random.randint(256)</td>
<td>0 to 255</td>
</tr>
<tr>
<td>random.randint(6) + 1</td>
<td>1 to 6</td>
</tr>
<tr>
<td>random.randint(30) + 100</td>
<td>100 to 119</td>
</tr>
<tr>
<td>random.randint(30) - 10</td>
<td>-10 to 20</td>
</tr>
<tr>
<td>random.randint(10) - 50</td>
<td>-50 to -41</td>
</tr>
</tbody>
</table>
```

Random Numbers

A summary of some Random methods:

<table>
<thead>
<tr>
<th>Method</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>randrange(step)</td>
<td>0 to step - 1</td>
</tr>
<tr>
<td>randrange(start, stop)</td>
<td>start to stop - 1</td>
</tr>
<tr>
<td>randrange(start, stop, step)</td>
<td>start to stop - 1, in increments of step</td>
</tr>
<tr>
<td>randint(a, b)</td>
<td>a to b, integer</td>
</tr>
<tr>
<td>random()</td>
<td>0.0 to 1.0</td>
</tr>
<tr>
<td>uniform(a, b)</td>
<td>a to b, float</td>
</tr>
<tr>
<td>choice([True, False])</td>
<td>random value from a list (e.g., True or False)</td>
</tr>
</tbody>
</table>
Random Numbers

The seed value (an integer) can be set for a random object using the `seed` method

```python
random.seed(546721)
```

The seed value determines exactly the stream of numbers that will be produced

If you don't call `seed`, the current time (in milliseconds) is automatically used instead