LDOC

De-stressing with Recursive Art

Congratulations, you made it to LDOC!

This semester, we've covered a LOT:

- Objects
- Data types
- Expressions
- Functions
- Memory diagrams
- Boolean expressions
- Conditionals
- Scope
- User input
- While loops
- For loops
- Importing modules
- Lists
- Unit tests
- Dictionaries
- Object-oriented prog.
- Classes and methods
- Recursive structures
- Recursive functions
- Importing and reading files

This is no small feat!

Announcements

Final Exam Preparation:

- Review practice quizzes, your past quizzes
- I will post *supplemental* final exam practice to the site today
- Review session recordings
- Visit Office Hours (11am-5pm) and Tutoring (5-7pm)
    - Today is the last day of office hours!
- Final exam review session tomorrow at 4pm! (More details soon)

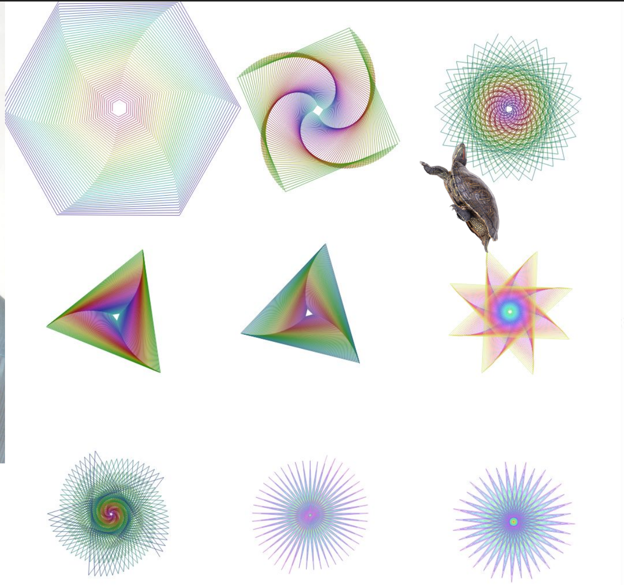
Interested in serving as an Undergrad TA for COMP110 in Spring 2026? Please apply by Dec 10!

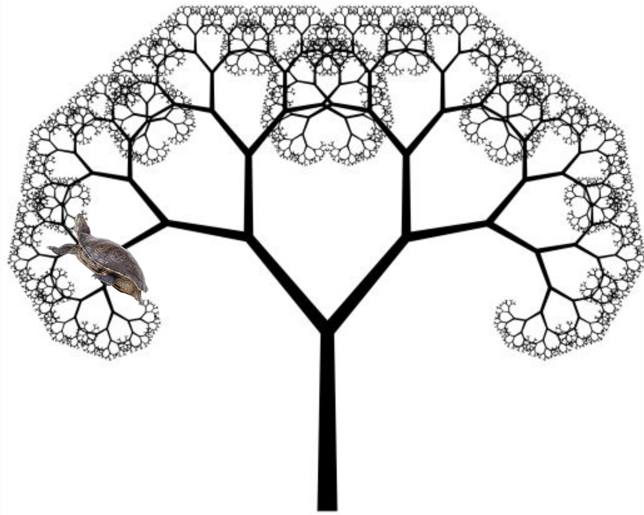
Time to de-stress: recursive art with `turtle` graphics!

Python library that lets us draw shapes on a virtual canvas

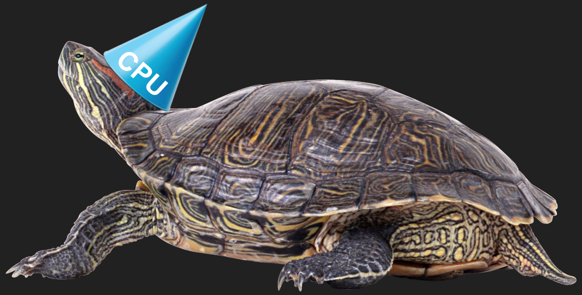
Imagine dipping a virtual turtle's tail in (non-toxic) paint and directing the turtle around a virtual canvas!

lots of recursion!





Getting started with `turtle` graphics



Steps to get us started:

1. Create a new folder in your workspace called **‘art’**
2. Inside that folder, create a new file called **\`turtle.py\`**
3. In your browser, navigate to:

   go.unc.edu/turtle

4. Select all the code on that page (ctrl+A or command+A) and copy it (ctrl+C or command+C)
5. Paste the code into your \`turtle.py\` file (ctrl+V or command+V). Then, save it!
6. Also in your **‘art’** folder, create new files called **\`tree.py\`** and **\`flower.py\`**

Code-along: Type the following into **flower.py:**

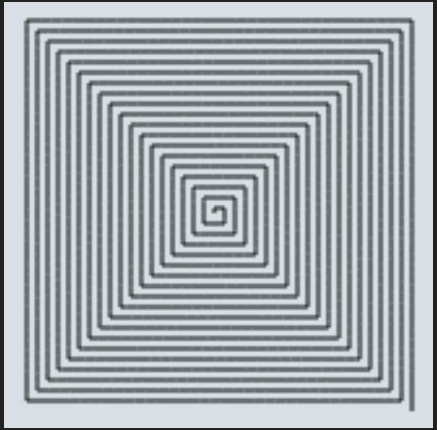
```python
"""Turtle art!"""

from .turtle import Turtle
from math import pi

__template__ = "https://comp110-25f.github.io/static/turtle/"

def main() -> Turtle:
    t: Turtle = Turtle()
    t.setSpeed(0.25)

    t.forward(150)
    t.left(pi / 2.0)

    t.forward(140)
    t.left(pi / 2.0)

    return t
```

Before you run the code, what shape do you think the Turtle will draw?

Your turn! Write a loop to draw a shape

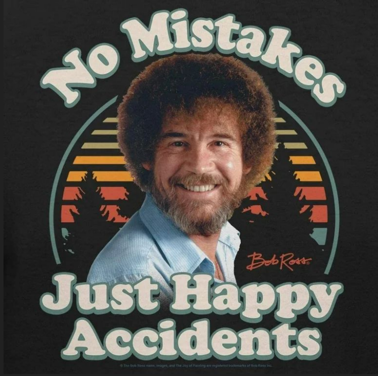
- Write a while loop (don’t forget a counter variable!) that, inside of the loop:
  - Turns the **Turtle t** left by **pi/2.0**
  - Moves the **Turtle t** forward by 150, 148, 146, and so on, until 0 (not moving forward at all)
  - Update your variable so that it moves toward the loop’s terminating condition
- Once you’re finished, try running it. What shape do you see?

- A spiral!!
- Try increasing the speed to 10 or 100 once you have it working. Additionally, try playing with the angle the turtle is turning to develop different spirals.



Next: Drawing happy, little trees!

```
"""Some happy, little trees!"""

from .turtle import Turtle
from math import pi
from random import random

__template__ = "https://comp110-25f.github.io/static/turtle/"


DEGREE: float = -pi / 180.0  # Constant


def main() -> None: ...


def click(x: float, y: float) -> Turtle:
    """Moves turtle to wherever we click on the canvas + draws line!"""
    t: Turtle = Turtle()
    t.moveTo(x, y)
    t.turnTo(90 * DEGREE)
    t.forward(100)
    return t
```



TODO: Complete the **`branch`** function to make the trees a little happier (add branches)!

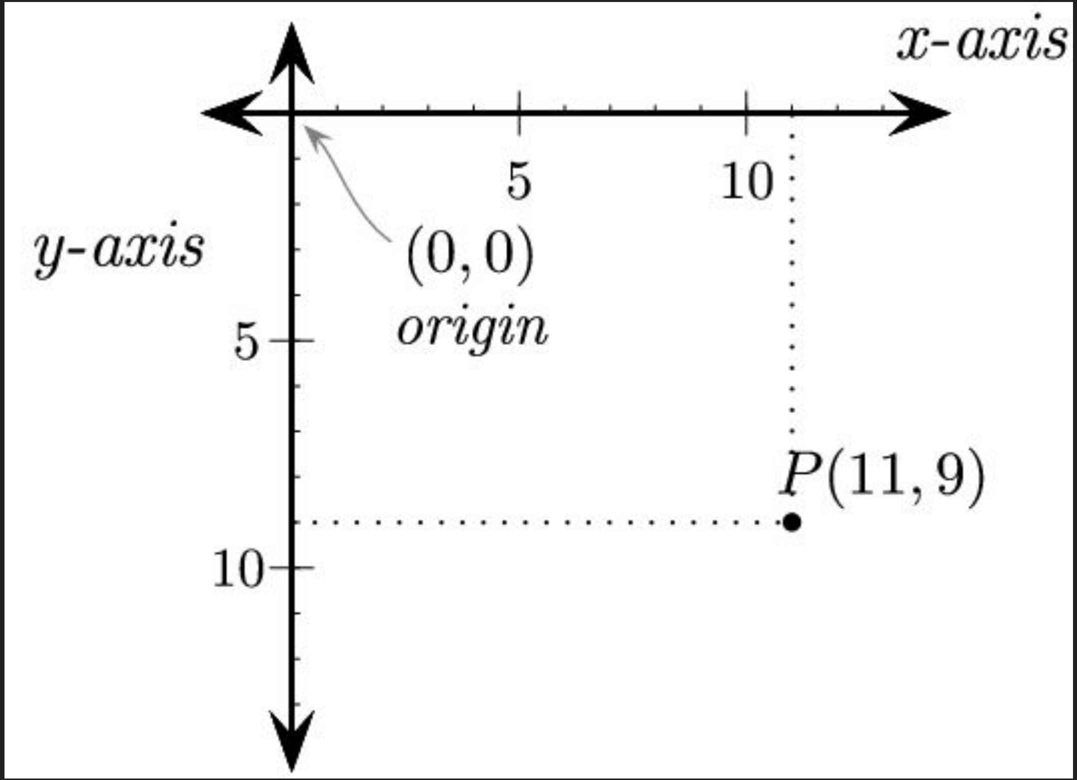
```
def branch(t: Turtle, length: float, angle: float) -> None:
    t.turnTo(angle)
    t.forward(length)
    # TODO: if length is greater than 10:
    # THEN call the branch function again (recursion!)
    #   The turtle argument of the call should be t, the same object
    #   The length argument of the call should be 75% of length's value
    #   The angle is angle + 30 * DEGREE
    t.turnTo(angle + pi)
    t.forward(length)
```

TODO: Call the **`branch`** function again to add branches on *both sides*

```
def branch(t: Turtle, length: float, angle: float) -> None:
    t.turnTo(angle)
    t.forward(length)
    if length > 10.0:
        branch(t, 0.75 * length, angle + 35 * DEGREE)
        # TODO: Make the same branch function call, but
        # Rather than adding 35 degrees, subtract 35 degrees!
        # ... (do this here)
    t.turnTo(angle + pi)
    t.forward(length)
```

Question: Which line of code could we change to alter the sizes of the trees?

Note: the origin of a **computer graphics coordinate system** starts in the *top left corner*, so a coordinate high up on the canvas has a *low* y value!



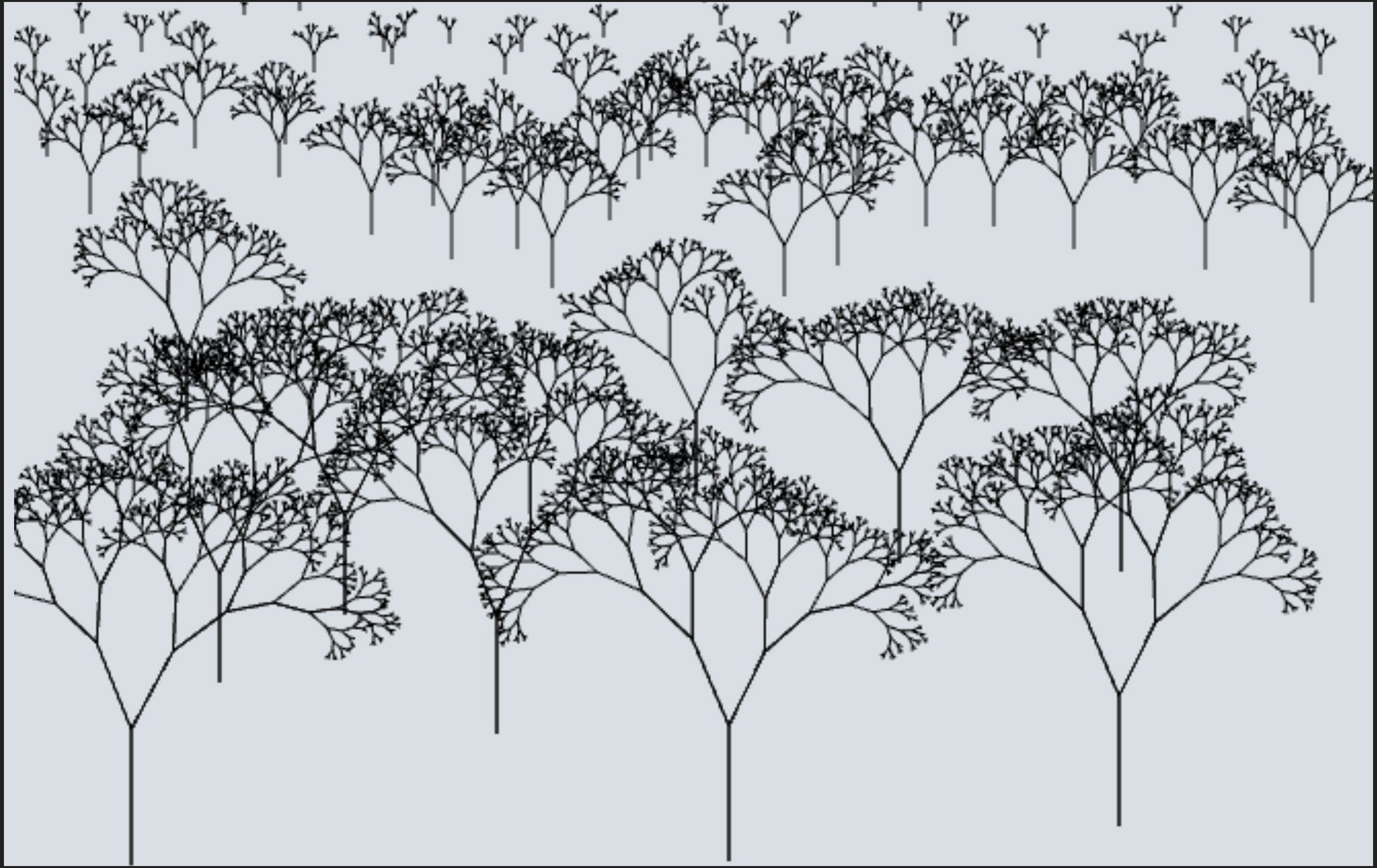
Not all trees in nature look the same… What if we randomize the length of the branches?

```
def branch(t: Turtle, length: float, angle: float) -> None:
    t.turnTo(angle)
    t.forward(length)
    if length > 3.0:
        branch(t, random_length(length), angle + 35 * DEGREE)
        branch(t, random_length(length), angle - 35 * DEGREE)
    t.turnTo(angle + pi)
    t.forward(length)


def random_length(length: float) -> float:
    random_value: float = random()
    factor: float = 0.20 * random_value
    return length * (0.6 + factor)
```

Now, what if we randomize the angles of the branches?

```python
def random_angle() -> float:
    # TODO: Return a value between 20 and 30 * DEGREE at random
```

Seeing the forest for the trees



Thank you for a great semester!

❤️, your COMP110 Team

P.S. Please complete your Course Evaluation;

it helps us improve the course!