CIS 122

Turtles all the way down
Turtle Graphics

- Open IDLE in "No Subprocess mode"
  - Command Prompt / Terminal
  - `<IDLE location> -n`

- Import turtle module
  - `import turtle`

- Start drawing
  - `turtle.forward(dist)`
  - `turtle.backward(dist)`
  - `turtle.left(angle)`
  - `turtle.right(angle)`
  - `turtle.reset()`
Yesterday, we tried to draw this image

Here's one way to do it:

turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(100)
turtle.left(90)
turtle.forward(70)
...

● We don't need that much code

● Let's write a square function instead
  ○ Then we can call it when needed
All Squared Away

● We don't need that much code

● Let's write a square function instead
  ○ Then we can call it when needed

```python
def square(length):
    turtle.forward(length)
    turtle.left(90)
    turtle.forward(length)
    turtle.left(90)
    turtle.forward(length)
    turtle.left(90)
```
Now we can rewrite our drawing code

```
square(100)
turtle.forward(20)
square(60)
```

Much cleaner

○ But our square code feels overly complicated
Our square function does the same stuff repeatedly
  ○ Go Forward
  ○ Turn Left

Let's write write square recursively
  ○ But what is there to recurse on?
  ○ What gets smaller as we draw our square?
All Squared Away

- Recurse on the number of sides left to draw
  - square(length, sidesLeft)

- Base Case

- Recursive Step
All Squared Away

- Recurse on the number of sides left to draw
  - square(length, sidesLeft)

- Base Case
  - No sides left to draw

- Recursive Step
All Squared Away

● Recurse on the number of sides left to draw
  ○ square(length, sidesLeft)

● Base Case
  ○ No sides left to draw

● Recursive Step
  ○ To draw a square with x sides left
  ○ Draw one side
  ○ Then draw a square with x-1 sides left
def square(length, sidesLeft):
    if sidesLeft == 0:
        return
    else:
        turtle.forward(length)
        turtle.left(90)
        square(length, sidesLeft - 1)
def square(length, sidesLeft):
    if sidesLeft == 0:
        return
    else:
        turtle.forward(length)
        turtle.left(90)
        square(length, sidesLeft - 1)

● This function takes two arguments
  ○ What if we want a square function with only one?

● Outsiders shouldn't care how our function is implemented
  ○ Want to call square(50), not square(50, 4)
def square(length, sidesLeft = 4):
    if sidesLeft == 0:
        return
    else:
        turtle.forward(length)
        turtle.left(90)
        square(length, sidesLeft - 1)

● Default arguments
  ○ If you don't specify a value, default to the given one

● Now, we can call square(50)
  ○ and sidesLeft will default to 4
More Cool Turtle Functions

- `turtle.width(size)`
  - Sets the width of your lines in pixels
  - Minimum 1 pixel
  - No maximum
  - What happens if you set width to...
    - 50?
    - 100?
    - 1000?
More Cool Turtle Functions

- `turtle.setpos(pos)`
  - Moves turtle to given coordinate position
  - Only takes one argument
  - But we need two coordinates...

- How can we store two coordinates in only one variable?
  - Use a tuple
Tuple Aside

- Tuples are another type of values
  - Store multiple values together
    - (1, 2, 3)
    - (1, "b", True)
  - We'll see them more in the future
More Cool Turtle Functions

- `turtle.setpos( pos )`
  - Moves turtle to given coordinate position
  - Only takes one argument
  - But we need two coordinates...

- How can we store two coordinates in only one variable?
  - Use a tuple
  - `turtle.setpos( (25, 50) )`

- NOT the same as calling `setpos` with two arguments
  - `turtle.setpos(25, 50)`
  - This code will not run
A Turtle of a Different Color

- `turtle.color(color)`
  - Sets the color of your turtle
  - And the lines it draws

- Color can be a string
  - `turtle.color("red")`
  - `turtle.color("blue")`

- But what if you want finer color control?
  - Only so many color names...
A Turtle of a Different Color

- Display colors are made by combining primary colors
  - Red
  - Green
  - Blue

- We can describe a color with these components
  - (Red Intensity, Green Intensity, Blue Intensity)
  - More tuples...

- A few common colors
  - Red = (1, 0, 0)
  - Yellow = (1, 1, 0)
  - White = (1, 1, 1)
A Turtle of a Different Color

- Color intensities range from 0 to 1
  - (0.0, 0.0, 0.0) - Black
  - (0.3, 0.3, 0.3) - Dark Gray
  - (0.6, 0.6, 0.6) - Light Gray
  - (1.0, 1.0, 1.0) - White
Let's draw a line that blends from one color into another

```python
def blend(greenValue, redValue):
    if redValue >= 1:
        return
    else:
        myColor = (redValue, greenValue, 0)
        turtle.color(myColor)
        turtle.forward(15)
        blend(greenValue + 0.5, redValue - 0.5)

>>> blend(1, 0)
```