# CS 100: Practice on Python Drawing

Chris Kauffman

Week 4

# Mini-Exam 1 Back: Results are very good

## Summary Stats

| Stat | Raw | % |
|---|---|---|
| Count | 43 | - |
| Max | 39.00 | 97.50 |
| Average | 35.26 | 88.15 |
| Median | 36.00 | 90.00 |
| Stddev | 3.24 | 8.10 |

## Percentage Frequencies

| Range | Count |
|---|---|
| 90 - 100 | 24 |
| 80 - 89 | 15 |
| 70 - 79 | 2 |
| 60 - 69 | 2 |
| 50 - 59 | 0 |

# Logistics

## Homework 3

- Due next Thursday
- Can work with partner
- Submit both Word Doc/PDF AND Python code

## Reading

How to Think Like a Computer Scientist Ch 3-7

## Mini-Exam

Will return and discuss on Thursday

## Goals Today

- Python basics
- Drawing Exercises

# Quick Review

- Where can you find example code we work on in class?
- What will appear at the top of python files which use the `turtle` to draw?
- Describe 3 primitive movement operations the turtle knows?
- How does one change the color of the turtle?
- How does one get the turtle to fill in shapes with color?
- How does one stop and start the turtle from drawing while it moves?

# Staying Organized

- HW and python files
- Single `Desktop/cs100/hw3` directory
  - `Homework 3.doc` (written HW)
  - `hw3.py` which contains code for the HW
- When HW 4 rolls around, make `Desktop/cs100/hw4`
  - `Homework 4.doc` (written HW)
  - `hw4.py` for code
- When working in class, create a file for the days work
  - `classwork_9_16.py` (spaces screw things up)

# Exercise: Draw a plain house

## Basic commands

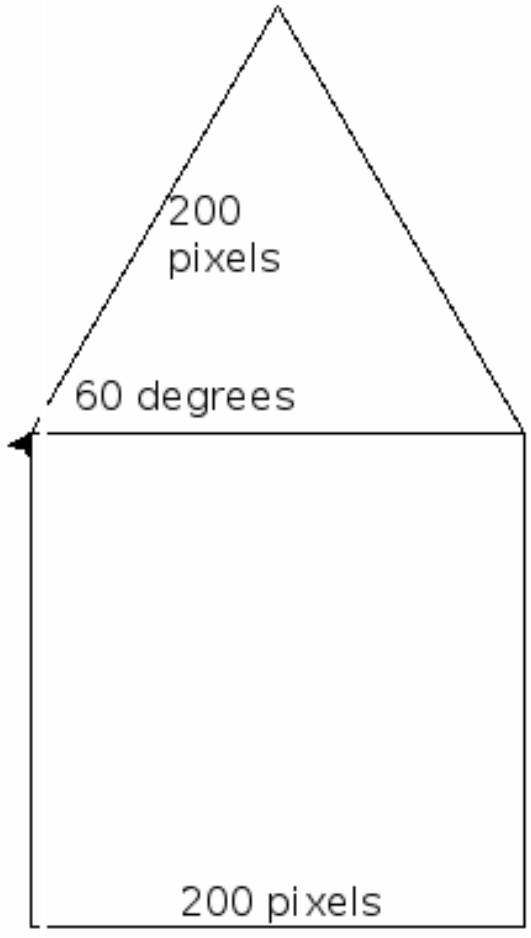
```
forward(length)
right(angle)
left(angle)
```

## Repetition

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

backward(200)
```

Spaces to indent loops



# Spaces in Python

Spaces between things doen't matter too much

```
x = 1                                   # Assign x to be 1
x=2                                     # Assign x to be 2
x    =   3                              # Assign x to be 3

for i in range(4):                      # Repeat 4 times
    print(i)

for   i  in range(  4):                 # Repeat 4 times
    print(i)
```

# Spaces in Python

Spaces in front of things matter a lot

```
x = 1                                   # Assign x to be 1
  y=2                                   # Error!

if (x > 2):                             # Indent things that should
    print("x > 2")                      # be done if x > 2
    print(x)
else:                                   # Indent things to do
    print("x <= 2")                     # when x <= 2
    if(x == 2):                         # Check if x is 2
        print("x is 2")                 # Print if it is
print("All done")                       # ALWAYS do this

for i in range(4):
    print(i)                            # Do this 4 times
print("hi")                             # Do this once
```

# Color Names as Strings

```
from turtle import *
color(x,x)                          # what is x?
color(blue,blue)                    # what is blue?
color("blue","blue")                # I know the "word" blue!
```

Bare names like

```
blue red
```

are treated as variables, often undefined
Things in quotes like

```
"blue" "red" "Several colors at once"
```

are string literals: "wordy" data

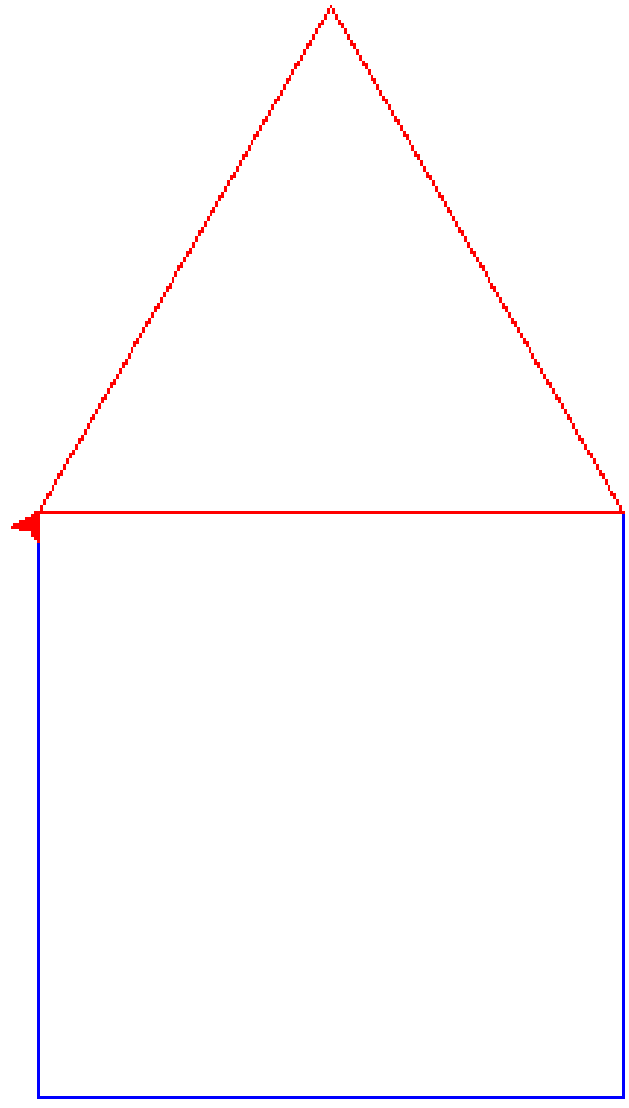
# Exercise: Colored House

Add `color("something")` commands

```
from turtle import *

for i in range(4):
  forward(200)
  right(90)

left(60)
for i in range(3):
  forward(200)
  right(120)
```



# Filling Areas with Color

## New Commands

`begin_fill()` and `end_fill()` can create shapes filled with color.

- Call `begin_fill()` to start coloring
- Looks like nothing happens
- When `end_fill()` is called, will fill in an area

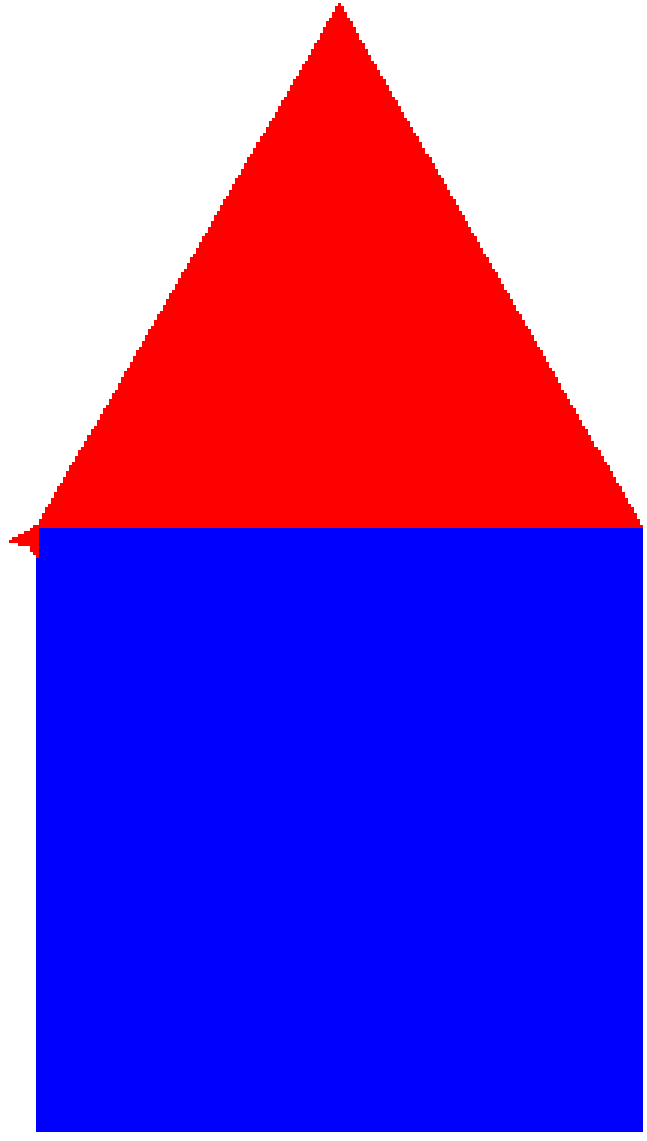
## Try the Following Code

```
color("green")
begin_fill()
for i in range(5):
  forward(100)
  right(72)
end_fill()
```

Can do this directly in interactive loop or in a file

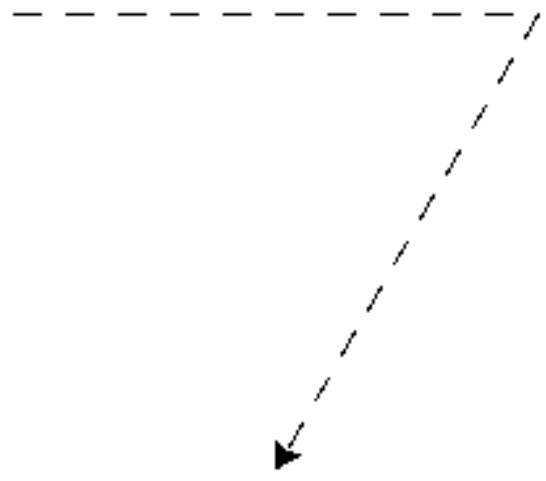
# Exercise: The Pretty House

Add `begin_fill()` and `end_fill()` to your code to produce the pretty house at the right



# Pen goes up, Pen goes down

- `penup()` stops drawing lines, allows turtle to move without drawing
- `pendown()` starts drawing lines again
- Useful for dashes and for `face.py`



```
for i in range(10):
  forward(10)
  penup()
  forward(10)
  pendown()

right(120)
for i in range(10):
  forward(10)
  penup()
  forward(10)
  pendown()
```

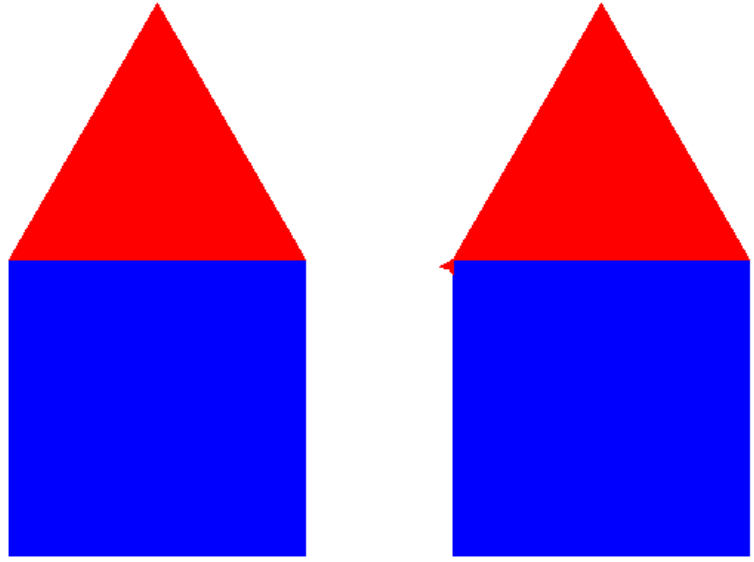
# Exercise: Two Houses

## Single House

```
# Draw the body of the house
color("blue")
begin_fill()
for i in range(4):
  forward(200)
  right(90)
end_fill()

# Draw the roof of the house
color("red")
begin_fill()
right(300)
for i in range(3):
  forward(200)
  right(120)
end_fill()
```

Now `penup()`, change angle, move, `pendown()` and do it again



# Variables

- A name like `size` associated with a value
- Can change the value associated with the name with assignment

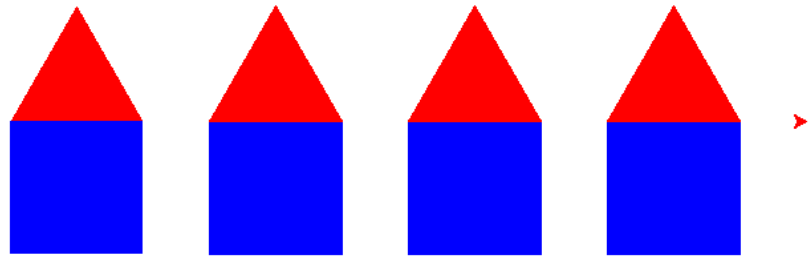
```
# size is 100
size = 100
# change size to 200
size = 200
# value of i is 3
i = 3
# change size to 300
size = i * 100
```

```
# Little square
size = 100
for i in range(4):
  forward(size)
  right(90)

# Big square
size = 200
for i in range(4):
  forward(size)
  right(90)
```

# Exercise: The Suburbs

- Smaller houses - size 100 sides
- Use a variable `size = 100`
- Change `forward(200)` to `forward(size)`
- Use a `for` loop to repeatedly draw houses and move turtle

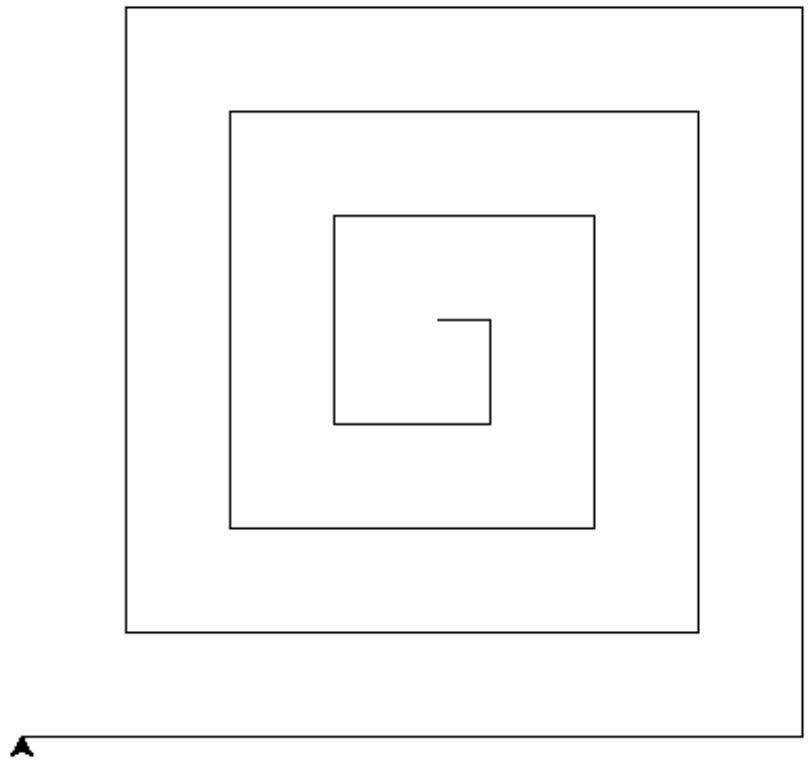


# Loop Variables Change Each iteration

The `range(N)` statement produces a sequence of numbers from `0` to `N`; good for loops

```
# prints 0, 1, 2, 3
for i in range(4):
    print(i)

# Square spiral
size = 0
for i in range(15):
  size = (i+1) * 25
  forward(size)
  right(90)
```



# Exercise: Suburbs part 2

- Change `size` each loop iteration
- Remember that loop variables start at 0

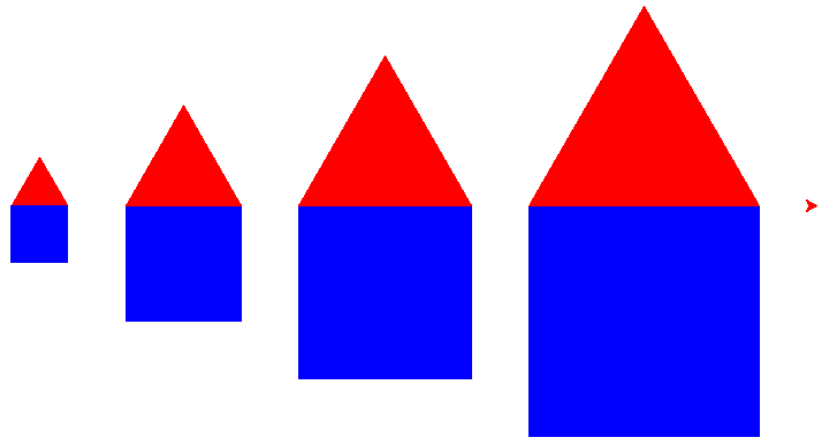
## Template for Suburbs

```
size = 50
for i in range(4):
  # draw a house size big

  # penup() and move turtle

  # pendown()

  # make size 50 pixels bigger
```



# Functions in Python

## Functions are Recipes

Define how to do something, an *algorithm*, but don't do it yet

## Syntax

- The `def` keyword for *define*
- Parentheses `( )` for parameters
- The colon `:`
- Indentation of commands belonging to the function

```
# Draw a square size 100
# No parameters
def draw_square_100():
    for i in range(4):
        forward(100)
        right(90)
# End of square_100() function

# Draw a square with given
# size which is a parameter
def draw_square(size):
    for i in range(4):
        forward(size)
        right(90)
# End of square(size) function
```

# Writing a Recipe versus Cooking

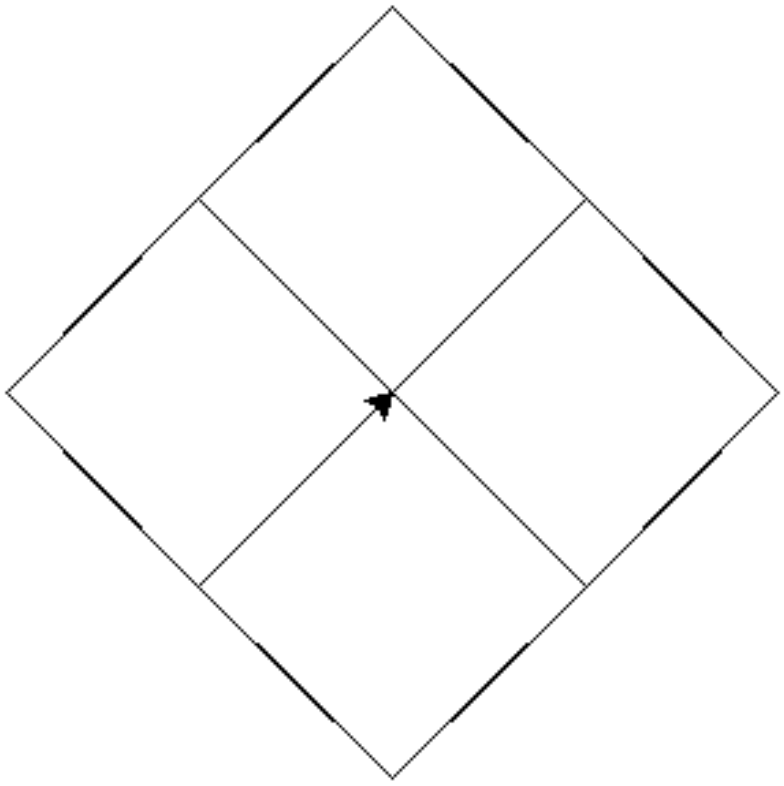
```
# How to draw a square with given
# size which is a parameter
def draw_square(size):
    for i in range(4):
        forward(size)
        right(90)
# End of square(size) function

# Not indented so not part of function
# Like the "When Run" block in code.org
draw_square(100)  # draw square size 100
penup()
forward(200)      # move
pendown()
draw_square(200)  # draw square size 200
```

- Functions define how to do something new
- Won't do it until function is called or executed
- Code to left defines function `draw_square(size)`
- Calls that function twice
- Makes two different rectangles

# Exercise: Fancy Diamond

- Write a python function `fancy_diamond()` which draws a fancy diamond
- Tilt is 45 degrees
- Sides are 100 pixels long
- May want to use `draw_square(size)` as a fancy diamond is comprised of 4 squares



```
def draw_square(size):
    for i in range(4):
        forward(size)
        right(90)
```

# Function Gotchyas

## Define but forgot to call

Won't draw anything

```
def draw_square(size):
  for i in range(4):
    forward(size)
    right(90)
```

Will draw something

```
def draw_square(size):
  for i in range(4):
    forward(size)
    right(90)

draw_square(200)
```

## Define before Use

Error

```
draw_square(200)
def draw_square(size):
  for i in range(4):
    forward(size)
    right(90)
```

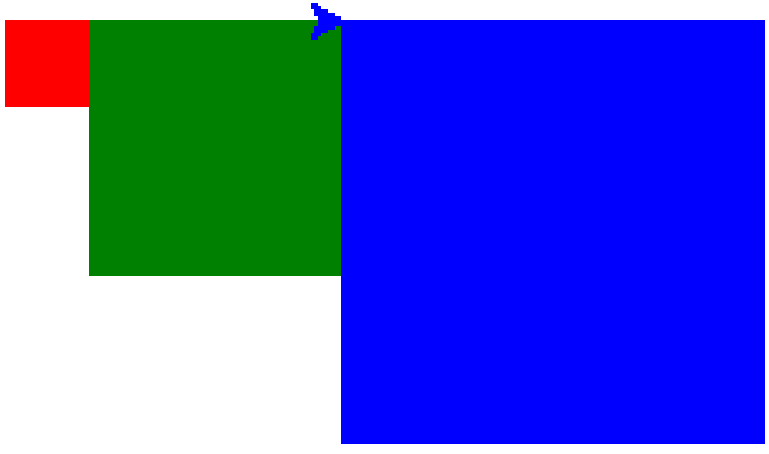
Okay

```
def draw_square(size):
  for i in range(4):
    forward(size)
    right(90)

draw_square(200)
```

# Multiple Arguments

- Functions can take multiple arguments such as size and color
- Each parameter is in between parenthesis separated by commas



```
# Draw a square with given size
# and color
def draw_color_square(size,col):
  color(col)
  begin_fill()
  for i in range(4):
    forward(size)
    right(90)
  end_fill()

draw_color_square(25,"red")
forward(25)
draw_color_square(75,"green")
forward(75)
draw_color_square(125,"blue")
```

# Exercise: Draw House Function

Create the function `draw_house(size,bodycol,roofcol)`:

- Draws house of given size
- Color body `bodycol` and roof `roofcol`
- Bonus: Use `draw_color_square(size,col)`
- Bonus: Create `draw_color_triangle(size,col)` and use it to draw house

```
size = 100

# Draw the body of the house
color("blue")
begin_fill()
for i in range(4):
  forward(size)
  right(90)
end_fill()

# Draw the roof of the house
color("red")
begin_fill()
left(60)
for i in range(3):
  forward(size)
  right(120)
end_fill()
```

# Draw House Solution

## Straight Code

```
def draw_house(size,bodycol,roofcol):
  # Draw the body of the house
  color(bodycol)
  begin_fill()
  for i in range(4):
    forward(size)
    right(90)
  end_fill()

  # Draw the roof of the house
  color(roofcol)
  begin_fill()
  left(60)
  for i in range(3):
    forward(size)
    right(120)
  end_fill()
```

## Using Other Functions

```
def draw_color_square(size,col):
  color(col)
  begin_fill()
  for i in range(4):
    forward(size)
    right(90)
  end_fill()

def draw_color_triangle(size,col):
  color(col)
  begin_fill()
  for i in range(3):
    forward(size)
    right(120)
  end_fill()

def draw_house(size,bodycol,roofcol):
  draw_color_square(size,bodycol)
  left(60)
  draw_color_triangle(size,roofcol)
```

# Suburbs Part 3

Use the `draw_house(size,bodycol,roofcol)`: function to simplify drawing the suburbs.

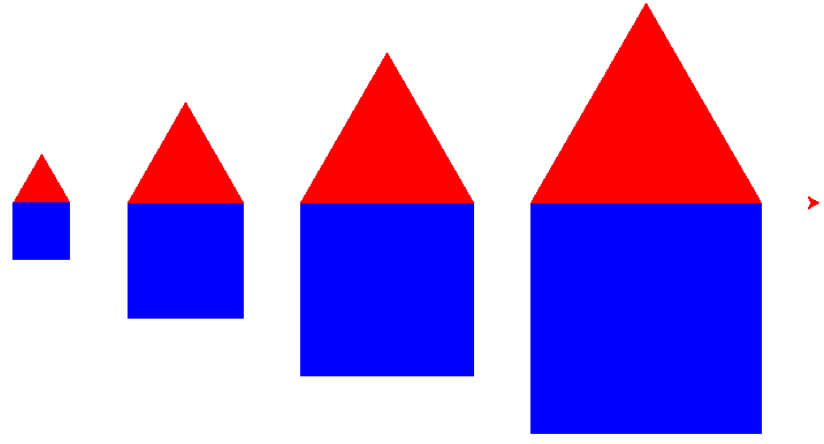
## Template for Suburbs

```
size = 50
for i in range(4):
  # draw a house size big

  # penup() and move turtle

  # pendown()

  # make size 50 pixels bigger
```



# Python Conditionals

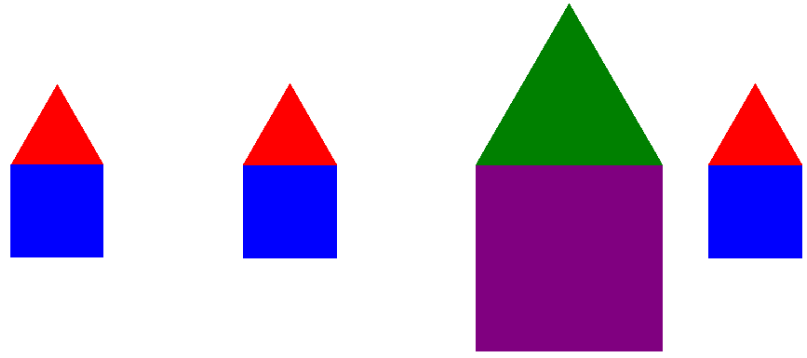
```
myVar = 7                                   # Assign a variable
if(myVar == 5):                             # Check something
    print("It's five");
else:
    print("It's not five");

for i in range(10):
    if i == 7:
        print("Lucky!")
    else:
        print("Boring")
```

- Using == allows one to check whether a variable is equal to a number
- An `if/else` statement allows conditional execution

# Exercise: Keeping up with The Kardashians

- Modify code below to produce the Kardashians neighborhood
- The Kardashians have a bigger house (200 pixels) with different coloring...
- Use an `if/else` statement



```
for house in range(4):
  draw_house(100,"blue","red")

  # Adjust position
  penup()
  right(60)
  forward(250)
  pendown()
```

# Alternating with Conditionals in Loops

```
# Print whether the numbers are odd or even
for i in range(10):
    if(i % 2 == 0):                 # % is remainder op
        print(str(i) + " is Even")
    else:
        print(str(i) + " is Odd")
```

- Useful when you want to alternate drawing different colors
- Nesting and combining things is what makes programming interesting

# The Alternating Neighborhood

Use remainder operator % and `if/else` to draw the alternating neighborhood which is crowded with Kardashians