



# COMPSCI 105 S1 2017 Principles of Computer Science

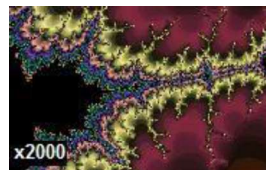
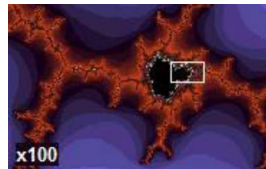
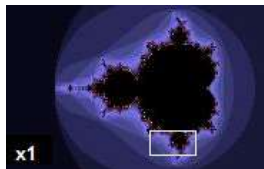
21-Recursion(2)



## 21.1 Introduction

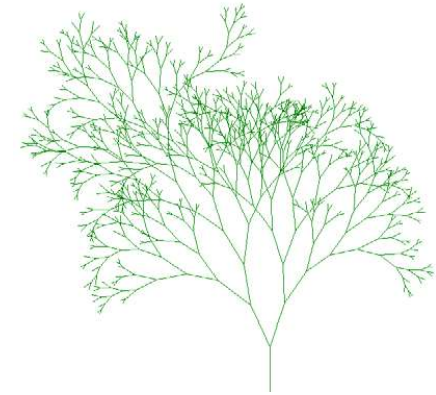
### Self-similarity

- ▶ A fractal is a rough or fragmented geometric shape that can be split into parts, each of which is (at least approximately) a reduced-size copy of the whole
- ▶ This a property is called self-similarity



## Agenda & Readings

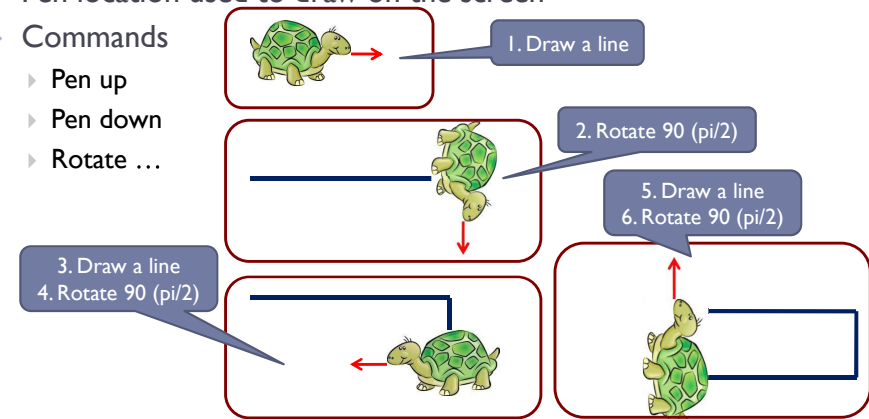
- ▶ **Agenda**
  - ▶ Introduction
  - ▶ Using the Python Turtle
    - ▶ Recursive Drawing
    - ▶ Drawing a Spiral
    - ▶ Drawing a KochUp
    - ▶ Drawing a C-curve
    - ▶ Call Tree
- ▶ **Reference:**
  - ▶ **Textbook:**
    - ▶ Problem Solving with Algorithms and Data Structures
      - Chapter 4 – Recursion



## 21.2 Python Turtle

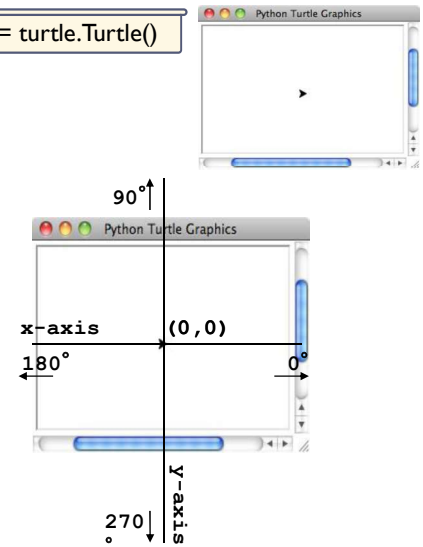
### Turtle Class

- ▶ Can be drawn using a “Turtle”
- ▶ Named after Logo programming language
- ▶ Pen location used to draw on the screen
- ▶ **Commands**
  - ▶ Pen up
  - ▶ Pen down
  - ▶ Rotate ...



- ▶ **Steps:**
  - ▶ Import the turtle module which defines the Turtle and the Screen types `import turtle`
  - ▶ Create and open a window `my_win = turtle.Screen()`
    - ▶ The window contains a canvas, which is the area inside the window on which the turtle draws
  - ▶ Create a turtle object which can move forward, backwards, turn left, turn right, the turtle can have its tail up/down
  - ▶ If the tail is down, the turtle draws as it moves `tess = turtle.Turtle()`
    - ▶ The width and colour of the turtle tail can be changed
  - ▶ When the user clicks somewhere in the window, the turtle window closes and execution of the Python program stops `turtle.exitonclick()`

- ▶ **Instantiate a Turtle object:** `tess = turtle.Turtle()`
- ▶ **The turtle appears as an icon**
  - ▶ Initial position: (0, 0)
  - ▶ Initial direction: East (0°)
  - ▶ Colour: black
  - ▶ Line width: 1
  - ▶ pixel Pen: down (ready to draw)



- ▶ `forward(distance)` – move the turtle forward
- ▶ `backward(distance)` – move the turtle backwards
- ▶ `right(angle)` – turn the turtle clockwise
- ▶ `left(angle)` – turn the turtle anti-clockwise
- ▶ `up()` – puts the turtle tail/pen up, i.e., no drawing
- ▶ `down()` – puts the turtle tail/pen down, i.e., drawing

- ▶ `pencolor(colour_name)` – changes the colour of the turtle's tail
- ▶ `heading()` – returns the direction in which the turtle is pointing
- ▶ `setheading(angle)` – set the direction in which the turtle is pointing
- ▶ `position()` – returns the position of the turtle
- ▶ `goto(x,y)` – moves the turtle to position x, y
- ▶ `speed(number)` – set the speed of the turtle movement

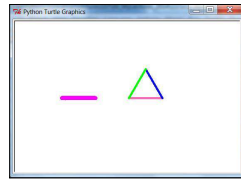


## Drawing Examples

### ▶ Examples:

```
my_win = turtle.Screen()
tess = turtle.Turtle()
tess.pencolor("hotpink")
tess.pensize(5)
tess.forward(80)
tess.pencolor("blue")
tess.left(120)
tess.forward(80)
tess.pencolor("green")
tess.left(120)
tess.forward(80)
```

```
tess.pensize(10)
tess.pencolor("magenta")
tess.left(120)
tess.right(180)
tess.up()
tess.forward(80)
tess.down()
tess.forward(80)
my_win.exitonclick()
```



## Recursive Drawing

- ▶ In the previous section, we looked at some problems that were easy to solve using recursion
- ▶ In this section, we will look at a couple of examples of using recursion to draw some interesting pictures
  - ▶ Drawing a spiral recursively
  - ▶ Drawing a Koch Up shape
  - ▶ Drawing a C Curve

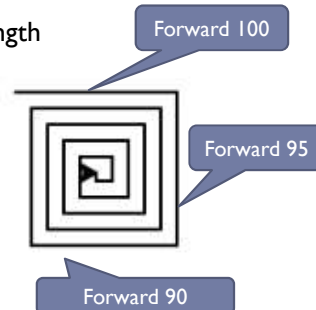


## Drawing a Spiral

### ▶ Define the `draw_spiral` function

- ▶ The base case is when the length of the line is zero or less
- ▶ The recursive step: (length of the line > 0; `len` > 0)
  - ▶ Instruct the turtle to go forward by `len` units, and
  - ▶ Turn right 90 degrees
  - ▶ Call `draw_spiral` again with a reduced length

```
draw_spiral(my_turtle, 100)
draw_spiral(my_turtle, 95)
draw_spiral(my_turtle, 90)
...
draw_spiral(my_turtle, 0)
```



## Drawing a Spiral

### ▶ The `draw_spiral` function

#### ▶ Steps:

- ▶ Define the `draw_spiral` function
- ▶ Create a turtle
- ▶ Call the recursive function

```
def draw_spiral(my_turtle, line_len):
    if line_len > 0:
        my_turtle.forward(line_len)
        my_turtle.right(90)
        draw_spiral(my_turtle, line_len-5)
```

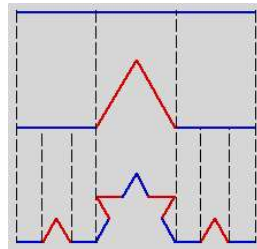
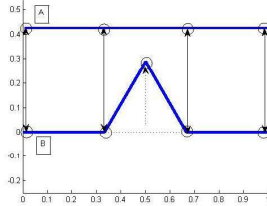
```
import turtle
...
my_win = turtle.Screen()
my_turtle = turtle.Turtle()
draw_spiral(my_turtle, 100)
turtle.exitonclick()
```



## Drawing a KochUp

- ▶ Idea: recursively applying a simple rule to each of the triangles sides

- ▶ Examples:



- ▶ The pattern:

- ▶ Cut the side ( $line\_len$ ) into 3 equal parts ( $line\_len/3$ )
- ▶ Replace the center part with 2 sides of length  $line\_len/3$ , such that it forms a spike
- ▶ Repeat the process for each of the 4 sides, until the length of each side is smaller than a given value.

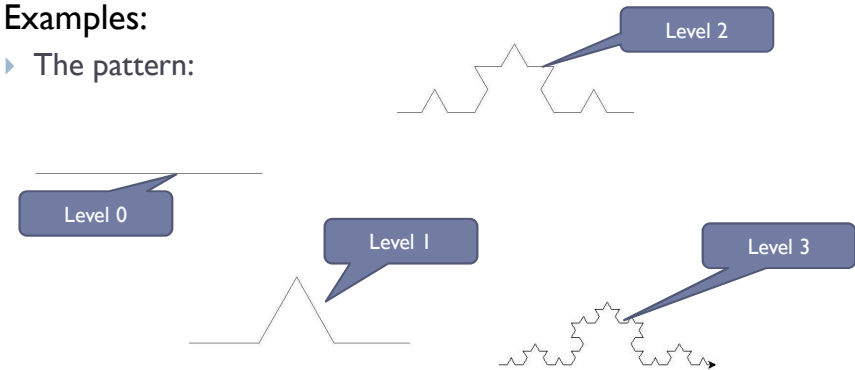


## Drawing a KochUp

- ▶ Idea: recursively applying a simple rule to each of the triangles sides

- ▶ Examples:

- ▶ The pattern:



## Drawing a KochUp

- ▶ Define the **draw\_kochup** function

- ▶ The base case is when the level is zero or less:
  - ▶ Instruct the turtle to go forward by **line\_len** units
- ▶ The recursive step: ( $level > 0$ )
  - ▶ Call **draw\_kochup** again with a reduced length and a reduced level
  - ▶ Turn left 60 degrees (anti-clockwise)
  - ▶ Call **draw\_kochup** again with a reduced length and a reduced level
  - ▶ Turn right 120 degrees
  - ▶ Call **draw\_kochup** again with a reduced length and a reduced level
  - ▶ Turn left 60 degrees (anti-clockwise)
  - ▶ Call **draw\_kochup** again with a reduced length and a reduced level



## Drawing a KochUp

- ▶ The **draw\_kochup** function

```
def draw_kockup(my_turtle, level, line_len):
    if level > 0:
        draw_kockup(my_turtle, level-1, line_len/3)
        my_turtle.left(60)
        draw_kockup(my_turtle, level-1, line_len/3)
        my_turtle.right(120)
        draw_kockup(my_turtle, level-1, line_len/3)
        my_turtle.left(60)
        draw_kockup(my_turtle, level-1, line_len/3)
    else:
        my_turtle.forward(line_len)
```



# Drawing a C Curve

- ▶ A C-curve is a fractal pattern
  - ▶ A level 0 C-curve is a vertical line segment
  - ▶ A level 1 C-curve is obtained by bisecting a level 0 C-curve and joining the sections at right angles
  - ▶ ...
  - ▶ A level N C-curve is obtained by joining two level N - 1 C-curves at right angles



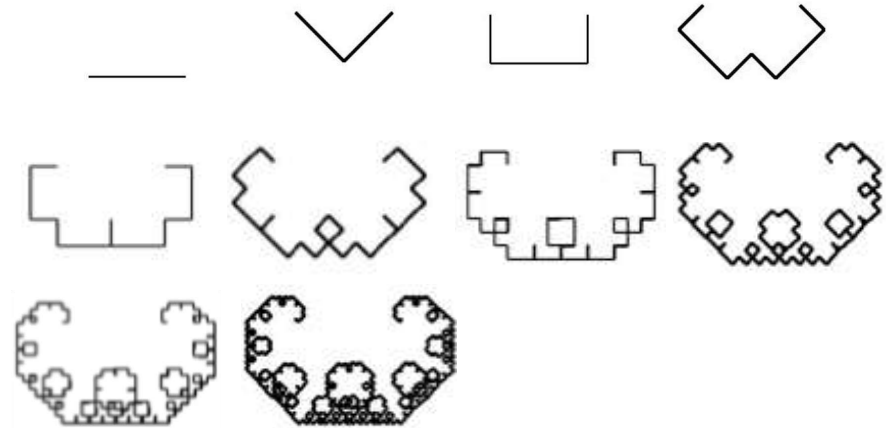
# Drawing a C Curve

- ▶ Define the **draw\_c\_curve** function
  - ▶ The base case is when the level is zero or less:
    - ▶ Instruct the turtle to go forward by line\_len units
  - ▶ The recursive step: (level > 0)
    - ▶ Turn right 45 degrees
    - ▶ Call **draw\_c\_curve** again with a reduced length and a reduced level
    - ▶ Turn left 90 degrees
    - ▶ Call **draw\_c\_curve** again with a reduced length and a reduced level
    - ▶ Turn right 45 degrees



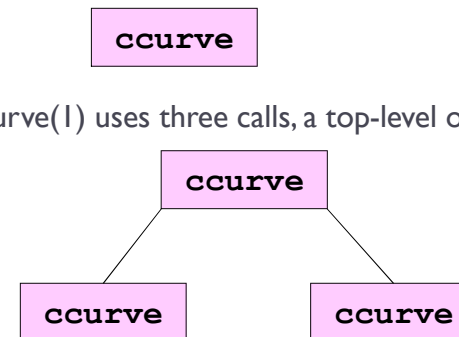
# Drawing a C Curve

- ▶ Examples:



# Drawing a C Curve

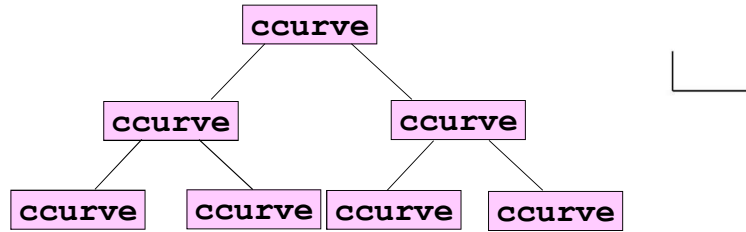
- ▶ A call tree diagram shows the number of calls of a function for a given argument value
  - ▶ `ccurve(0)` uses one call, the top-level one





## Drawing a C Curve

- ▶ A call tree diagram shows the number of calls of a function for a given argument value
  - ▶ `ccurve(2)` uses 7 calls, a top-level one and 6 recursive calls



- ▶ `ccurve(n)` uses  $2n+1$  calls, a top-level one and  $2n$  recursive calls



## Drawing a C Curve

- ▶ The `ccurve` function

```

def ccurve(my_turtle, level, line_len):
    if level > 0:
        my_turtle.right(45)
        ccurve(my_turtle, level-1, line_len/2)
        my_turtle.left(90)
        ccurve(my_turtle, level-1, line_len/2)
        my_turtle.right(45)
    else:
        my_turtle.forward(line_len)
  
```



## Summary

- ▶ The application of recursion is practiced by using Python Turtles