Creating Algorithms to Draw Polygons

Do this on pencilcode.net

We will develop algorithms to draw polygons using *turtle graphics*. Turtle graphics were developed by Seymour Papert to create algorithmic art by moving a geometric "turtle."

Five Basic Turtle Commands

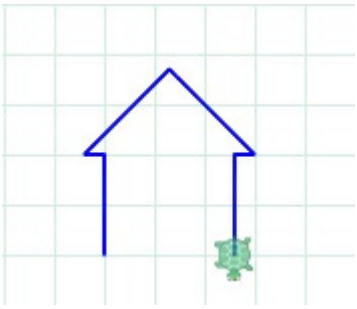
With turtle graphics, you have the following **Primitive functions**:

`pen red`	Selects a line color for drawing (`pen null` to switch to no-color later).
`fd 100`	Moves the turtle forward by some pixels.
`rt 90`	Pivots the turtle right by some degrees.
`lt 90`	Pivots the turtle left by some degrees.
`bk 100`	Slides the turtle back by some pixels.

There are other turtle primitives, but these five are already enough to do a lot. Here is a house:

A Little Blue House

```
pen blue
fd 50
rt 90
bk 10
lt 45
fd 60
lt 90
bk 60
lt 45
fd 10
lt 90
fd 50
```



Activity #1

Polygons: Square, Triangle, and Pentagon.

Create programs to make a square, an equilateral triangle, and a regular pentagon. You can save them using the names "square", "triangle" and "pentagon".

Since a square has four corners and four sides, it should take no more than 4+4=8 commands to draw a square after choosing a pen color. Similarly, a triangle should take no more than six steps after picking a pen, and a pentagon should take no more than ten steps after the pen.

```
# Purple Square

pen purple

fd 50

rt 90
```

```
# Orange Triangle

pen orange

fd 100
```

```
# Green Pentagon
```

Even though our triangle has a smaller angle, the turtle must pivot a larger number of degrees to make that angle. Why is that?

Exercise:

Write programs for the three regular polygons above, using only the primitives `pen`, `fd` and `rt`. Include final turns to return the turtle to its starting direction in each program.

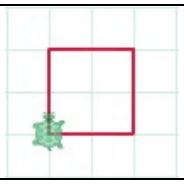
Activity #2

Simplifying an Algorithm with Loops

An algorithm can be written with **Primitives**, **Control flow**, **Memory**, and **Arithmetic**. So far we have been able to make fine drawings using only **Primitives**: why use anything else?

Here is a simple loop using `for` to demonstrate **Control flow** in CoffeeScript:

```
pen crimson
for [1..4]     # This line controls the indented block below.
  fd 50        # This indented line is repeated 4 times.
  rt 90        # This indented line is also repeated 4 times.
```



The result is a square again, but with a shorter program.

The `for` keyword repeats a block of code once for each number in a range we provide.

In CoffeeScript, indenting is important! The program will not run correctly if you do not line up the indenting neatly.

Drawing Faster and Filling Shapes

Here are two more turtle primitives for solving common problems:

`speed 10`	`speed 10` sets the turtle speed to 10 moves per second. `speed Infinity` works.
`pen path` `fill purple`	`pen path` makes the turtle trace out an invisible path. Later, `fill purple` fills the invisible path with color.

Exercises:

1. Which written description of a square is easier to understand: A, or B? _____
   A. "A shape with a side and a corner and an equal side and an equal corner and another equal side and another equal corner and another equal side and another equal corner."
   B. "A shape with four equal sides and corners."
2. Use the `for [1..n]` technique to simplify your triangle and pentagon programs. How many lines of code are in each program now? _____
3. Write a program to draw a regular polygon with 30 corners. Use `speed 10`.
   How many lines is your program? _____
4. Write a program called "stopsign" that quickly draws a solid red octagon. Use `speed Infinity` and `fill red`.

Shorter algorithms are usually clearer.

Activity #3

Generalizing an Algorithm Using Variables

Variables can let us generalize an algorithm. Consider a polygon algorithm that defines a variable `n` to be the number of corners:

```
n = 6          # n is the number of corners.
k = plum       # k is the color
a = 60         # a is the size of an exterior angle.

speed Infinity
pen path
for [1..n]     # Loop n times:
  fd 50        # Move by 50 pixels.
  rt a         # Turn by a degrees.
fill k
```

Wherever the program needs to know the number of corners, it uses `n` instead of the number. The code above defines three variables `n`, `a`, and `k`, and uses them instead of numbers and colors.

Using Arithmetic To Compute Side-Length

To draw a 10-sided polygon, we would just alter `n` to be 10 and `a` to be 36 degrees.

When we add more sides, the polygon gets too big to be useful. Instead of always using `fd 50`, it might be more convenient to specify the target perimeter, and automatically adjust the lengths based on the number of corners. Some simple arithmetic does the trick:

```
n = 10
k = skyblue
a = 36
p = 300        # p is the desired perimeter.

speed Infinity
pen path
for [1..n]
  fd p / n     # This will calculated to be 30.
  rt a
fill k
```

Exercise: Writing Elegant Algorithms for Polygons

Create a program similar to the examples on this page, but eliminate the variable `a` and compute the needed angle automatically.

Activity #4

Introduction to Functions

Our goal is to create a new function polygon so that we have a way to draw *any* regular polygon with *n* sides of *l* length, and color *k*. For example, polygon 12, 100, blue draws a blue regular 12-sided polygon (dodecagon) where each side length is 100.

Functions can take inputs (called arguments). This code defines a polygon function with three inputs, *n*, *l*, and *k*. Can you complete the function using your algorithm to correctly draw the polygon based on the input arguments?

```
polygon = (n, l, k) -> # define a function polygon with three inputs, code to follow:
  pen k                # use the chosen color.

                       # insert your algorithm here (remember to indent)!!

speed 10               # draw fast!
polygon 12, 100, blue  # use our newly-defined function!
```

Exercises:

1. Custom Star: Instead of drawing a regular polygon, can you draw a custom star using input arguments? The function `star` below has one argument *n* that is used to represent how many corners (points) are drawn in the star. Can you draw a five-pointed star or a nine-pointed star? Can you make it work for all numbers?

```
star = (n) ->
```

2. Super Star: Add more arguments by declaring `star = (n, m, s) ->`. *m* is the multiplier used when calculating the angle, and *s* is the length of a side. Can you draw an 8-pointed star?

Reference — pencilcode.net

Movement

`fd 50`	forward 50 pixels
`bk 10`	backward 10 pixels
`rt 90`	turn right 90 degrees
`lt 120`	turn left 120 degrees
`rt 90, 50`	do a 90° arc of radius 50
`home()`	go to the page center
`slide x, y`	slide right x and forward y
`moveto x, y`	go to x, y relative to home
`turnto 45`	set direction to 45 (NE)
`turnto obj`	point toward obj
`speed 30`	do 30 moves per second

Appearance

`ht()`	hide the turtle
`st()`	show the turtle
`scale 8`	do everything 8x bigger
`wear yellow`	wear a yellow shell
`fadeOut()`	fade and hide the turtle
`remove()`	totally remove the turtle

Output

`see obj`	debug the value of obj
`write 'hello'`	writes a line of HTML
`p = write 'hm'`	saves an HTML element
`p.html 'done'`	changes old text
`append 'ok'`	text without a new line
`button 'go', -> fd 10`	adds a button with an action
`read (n) -> write n*n`	adds a text input with an action
`t = table 3,5`	adds a 3x5 <table>
`t.cell(0, 0). text 'aloha'`	selects the first cell of the table and sets its text
`ct()`	clear text

Drawing

`pen blue`	draw in blue
`pen red, 9`	9 pixel wide red pen
`pen null`	use no color
`pen off`	pause use of the pen
`pen on`	use the pen again
`label 'X'`	draw the letter X
`dot green`	draw a green dot
`dot gold, 30`	30 pixel gold circle
`pen path`	trace an invisible path
`fill cyan`	fill traced path in cyan
`cg()`	clear graphics

Properties

`turtle`	name of the main turtle
`getxy()`	[x, y] position relative to home
`direction()`	direction of turtle
`hidden()`	if the turtle is hidden
`touches(obj)`	if the turtle touches obj
`inside(window)`	if enclosed in the window

Objects

`t = new Turtle`	make a new turtle
`p = new Piano 88`	make a new 88-key piano
`g = $('img')`	select all <img> as a set

Other Functions

`tick 5, -> fd 10`	go 5 times per second
`click -> fd 10`	go when clicked
`random [3,5,7]`	return 3, 5, or 7
`random 100`	random [0..99]
`play 'ceg'`	play musical notes
`$(window)`	the visible window
`$('p').eq(0)`	the first <p> element
`$('#zed')`	the element with id="zed"

Colors

white	gainsboro	silver	darkgray	gray	dimgray	black
whitesmoke	lightgray	lightcoral	rosybrown	indianred	red	maroon
snow	mistyrose	salmon	orangered	chocolate	brown	darkred
seashell	peachpuff	tomato	darkorange	peru	firebrick	olive
linen	bisque	darksalmon	orange	goldenrod	sienna	darkolivegreen
oldlace	antiquewhite	coral	gold	limegreen	saddlebrown	darkgreen
floralwhite	navajowhite	lightsalmon	darkkhaki	lime	darkgoldenrod	green
cornsilk	blanchedalmond	sandybrown	yellow	mediumseagreen	olivedrab	forestgreen
ivory	papayawhip	burlywood	yellowgreen	springgreen	seagreen	darkslategray
beige	moccasin	tan	chartreuse	mediumspringgreen	lightseagreen	teal
lightyellow	wheat	khaki	lawngreen	aqua	darkturquoise	darkcyan
lightgoldenrodyellow	lemonchiffon	greenyellow	darkseagreen	cyan	deepskyblue	midnightblue
honeydew	palegoldenrod	lightgreen	mediumaquamarine	cadetblue	steelblue	navy
mintcream	palegreen	skyblue	turquoise	dodgerblue	blue	darkblue
azure	aquamarine	lightskyblue	mediumturquoise	lightslategray	blueviolet	mediumblue
lightcyan	paleturquoise	lightsteelblue	cornflowerblue	slategray	darkorchid	darkslateblue
aliceblue	powderblue	thistle	mediumslateblue	royalblue	fuchsia	indigo
ghostwhite	lightblue	plum	mediumpurple	slateblue	magenta	darkviolet
lavender	pink	violet	orchid	mediumorchid	mediumvioletred	purple
lavenderblush	lightpink	hotpink	palevioletred	deeppink	crimson	darkmagenta