Two Views of
Programming Languages

Mechanical vs. Linguistic

Franklyn Turbak
Computer Science Department

Science Center Faculty Seminar
Wellesley College
Thursday, April 8, 2010

Overview

- What is a programming language? Mechanical vs. linguistic views.
- Along the way: what do programming language designers & implementers think about? syntax, semantics, pragmatics.
- Several shameless plugs

Plug #1: Grand Challenges Summit

Boston Grand Challenge Summit

The Educational Imperatives of the Grand Challenges

Can we transform our educational system to better prepare students to tackle the big issues facing our planet? Join us for in-depth discussions touching on energy, health, security, learning, and more.

Part of a national summit series based on the **National Academy of Engineering's 14 "Grand Challenges,"** critical problems we must solve to ensure a sustainable future.

General series info: http://www.grandchallengesummit.org/

April 21, 2010

Wellesley College
Wellesley, MA

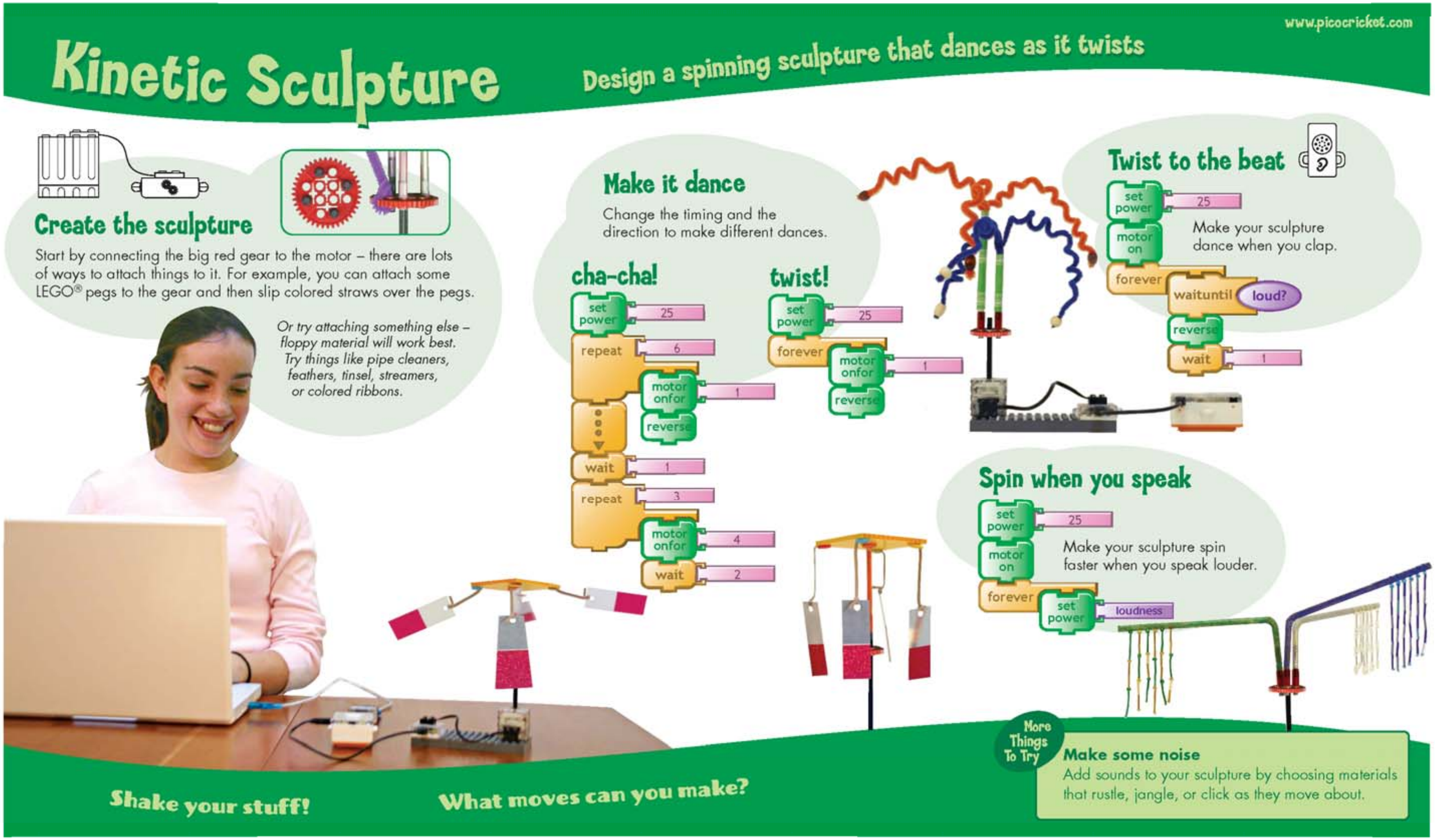
http://grandchallengesummit.olin.edu/

Programming Languages: Mechanical View

A computer is a machine. Our aim is to make the machine perform some specified actions. With some machines we might express our intentions by depressing keys, pushing buttons, rotating knobs, etc. For a computer, we construct a sequence of instructions (this is a ``program'') and present this sequence to the machine.

– Laurence Atkinson, Pascal Programming

PicoBlocks



www.picocricket.com

Kinetic Sculpture

Design a spinning sculpture that dances as it twists

Create the sculpture

Start by connecting the big red gear to the motor – there are lots of ways to attach things to it. For example, you can attach some LEGO® pegs to the gear and then slip colored straws over the pegs.

Or try attaching something else – floppy material will work best. Try things like pipe cleaners, feathers, tinsel, streamers, or colored ribbons.

Make it dance

Change the timing and the direction to make different dances.

cha-cha!

```
set power 25
repeat 6
    motor onfor 1
    reverse
wait 1
repeat 3
    motor onfor 4
    wait 2
```

twist!

```
set power 25
forever
    motor onfor 1
    reverse
```

Twist to the beat

Make your sculpture dance when you clap.

```
set power 25
motor on
forever
    waituntil loud?
    reverse
    wait 1
```

Spin when you speak

Make your sculpture spin faster when you speak louder.

```
set power 25
motor on
forever
    set power loudness
```

More Things To Try

Make some noise

Add sounds to your sculpture by choosing materials that rustle, jangle, or click as they move about.

Shake your stuff!

What moves can you make?

Syntax (Form) vs. Semantics (Meaning)

Furiously sleep ideas green colorless.

Colorless green ideas sleep furiously.

Little white rabbits sleep soundly.

Syntax Examples: Absolute Value Function

Logo: `to abs :n ifelse :n < 0 [output (0 - :n)] [output :n] end`

Javascript: `function abs (n) {if (n < 0) return -n; else return n;}`

Java: `public static int abs (int n) {if (n < 0) return -n; else return n;}`

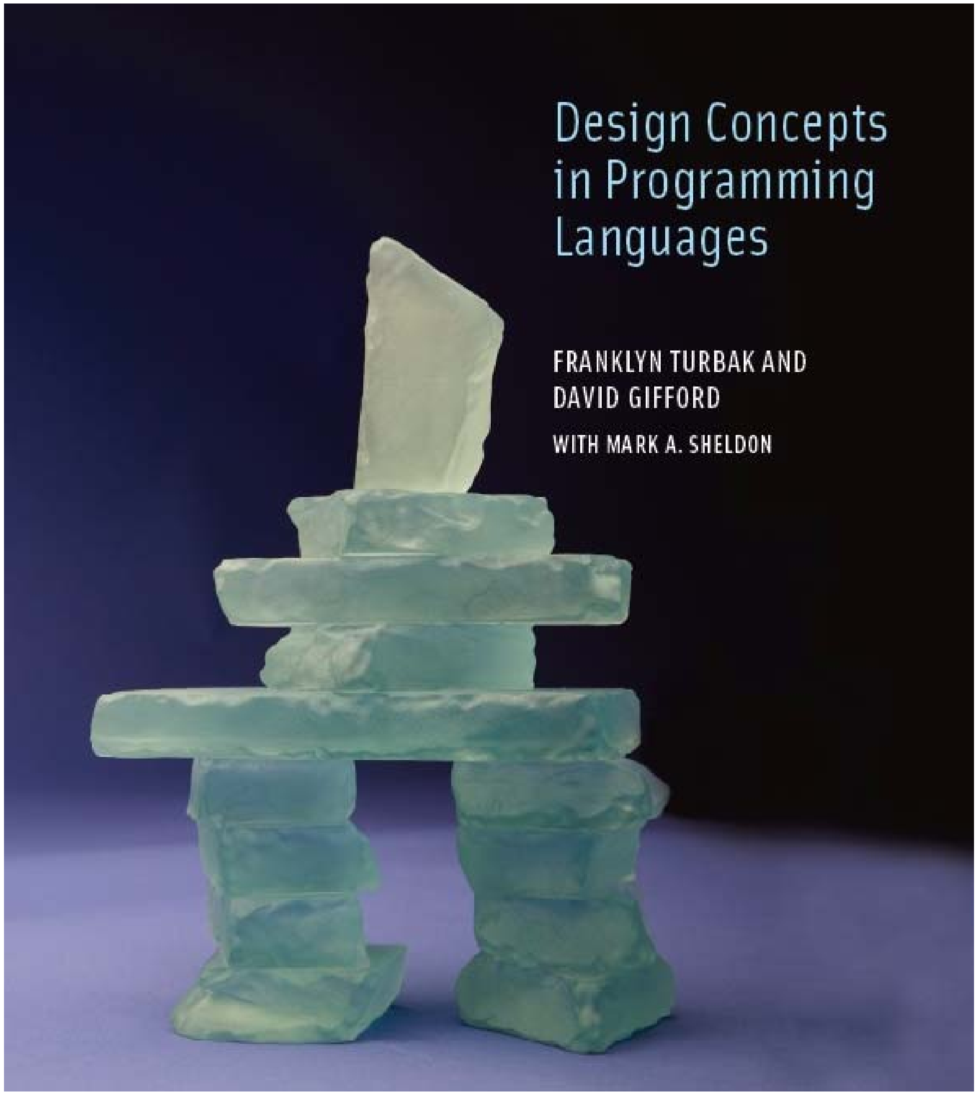
Python:

```
def abs(n):
  if n < 0:
    return -n
  else:
    return n
```

Scheme: `(define abs (lambda (n) (if (< n 0) (- n) n)))`

PostScript: `/abs {dup 0 lt {0 swap sub} if} def`

Plug #2: Design Concepts in Programming Languages

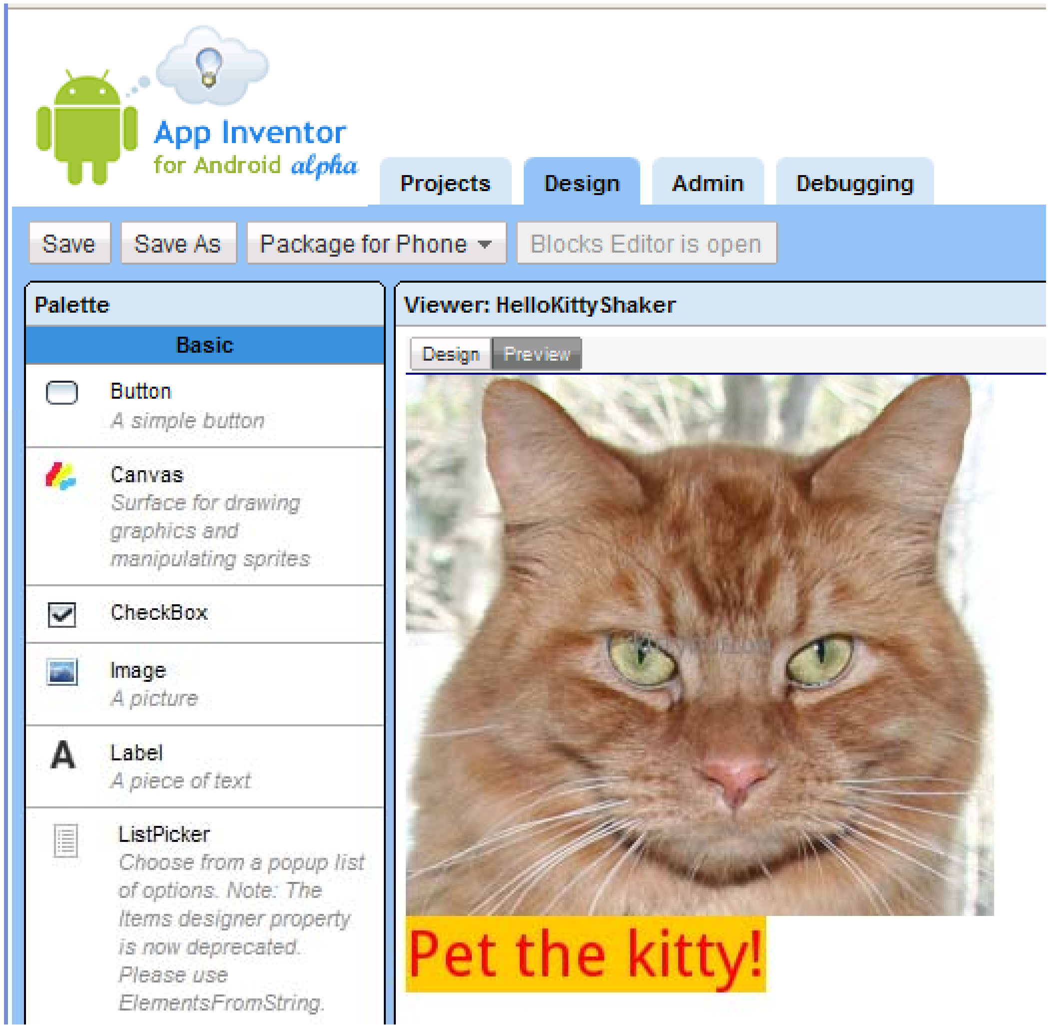


Design Concepts
in Programming
Languages

FRANKLYN TURBAK AND
DAVID GIFFORD

WITH MARK A. SHELDON

App Inventor For Android: Designer Window



Button
A simple button

Canvas
Surface for drawing graphics and manipulating sprites

CheckBox

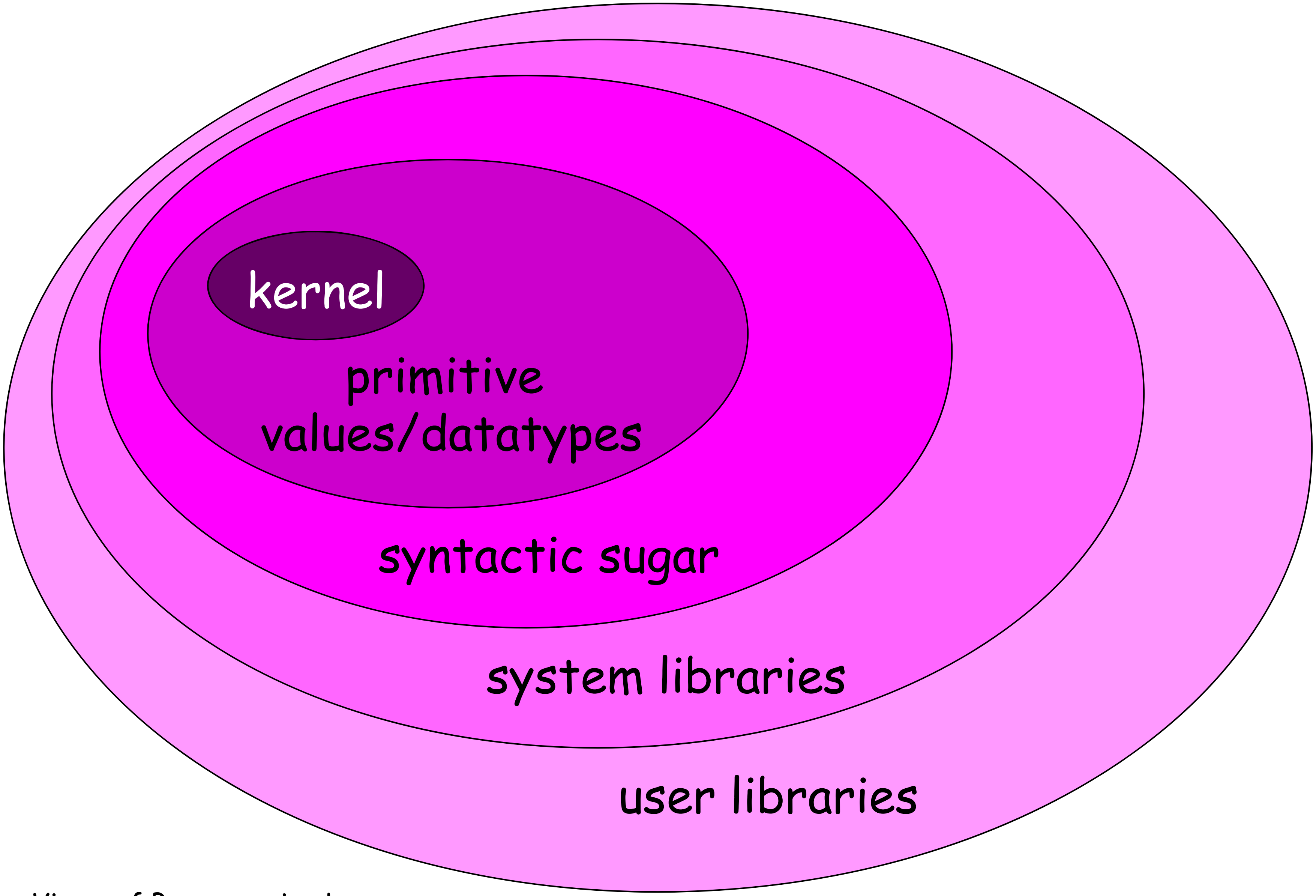
Image
A picture

Label
A piece of text

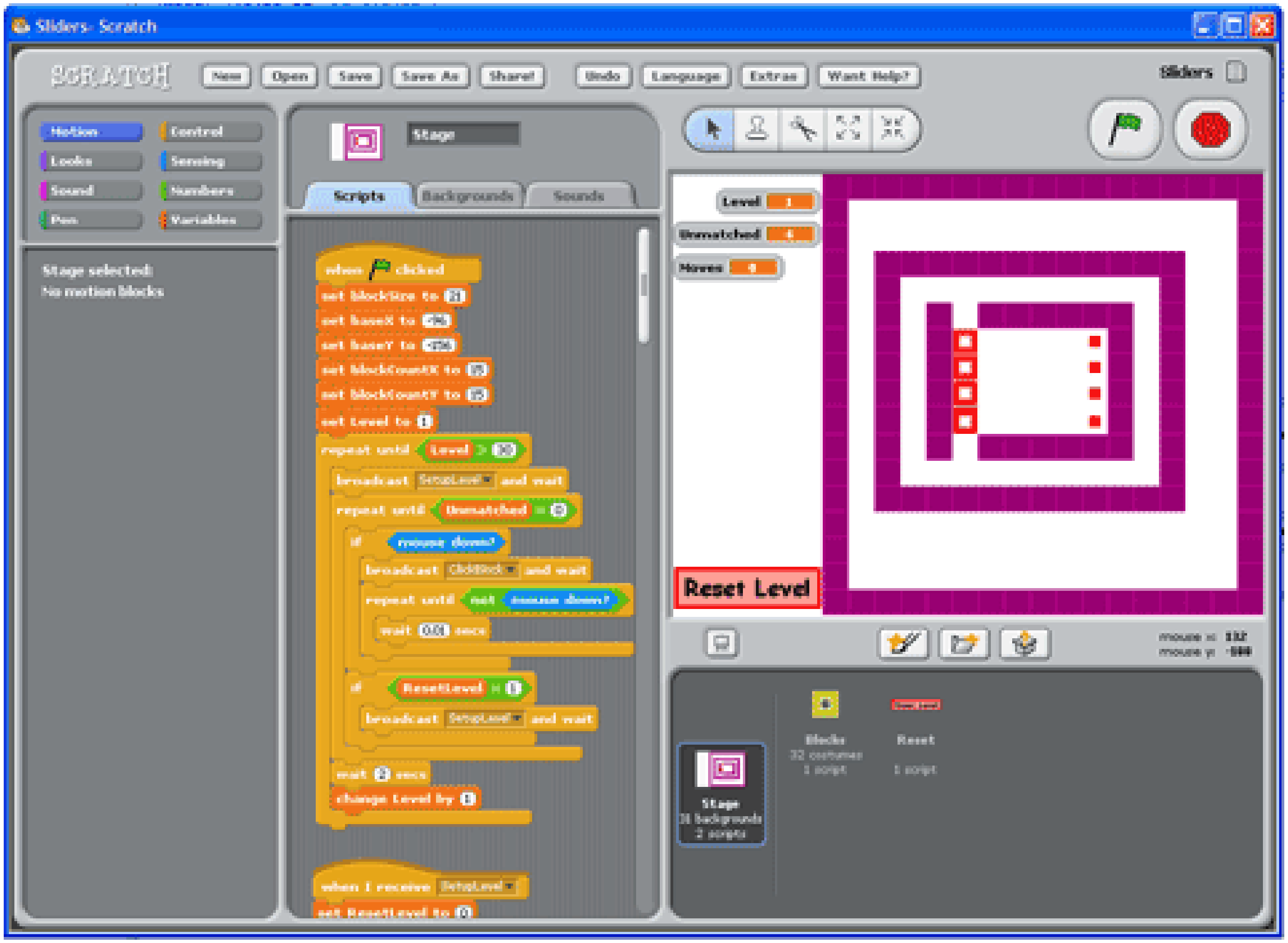
ListPicker
Choose from a popup list of options. Note: The Items designer property is now deprecated. Please use ElementsFromString.

App Inventor For Android: Blocks Window

Programming Language Layers



Plug #3: Scratch



http://scratch.mit.edu/

Example: Line Follower

Line Following Code: Abstract Version

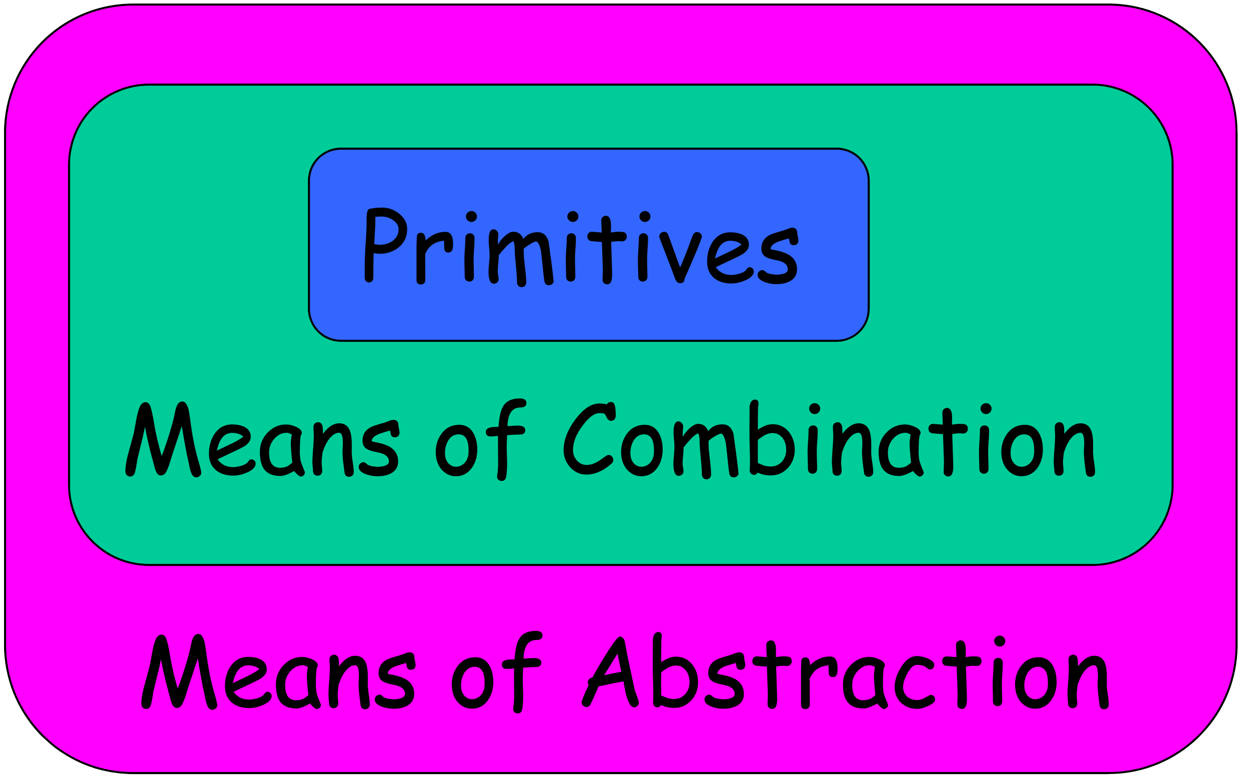
```
to follow-line
  go-forward
  loop [if sees-black? left-sensor [pivot-left]
          if sees-black? right-sensor [pivot-right]]
end

to go-forward
  left-wheel on thisway
  right-wheel on thisway
end

to pivot-left
  left-wheel off
  right-wheel on thisway
end

to pivot-right
  right-wheel off
  left-wheel on thisway
end
```

```
to left-wheel
  a,
end

to right-wheel
  b,
end

to sees-black? :sensor-value
  output :sensor-value > 100
end

to left-sensor
  output sensor 0
end

to right-sensor
  output sensor 1
end
```

Line Following Code w/o Abstractions

```
to follow-line
  a, on thisway b, on thisway
  loop [if (sensor 0) > 100
              [a, off b, on thisway]
          if (sensor 1) > 100
             [b, off a, on thisway]]
end
```

Programming Language Essentials



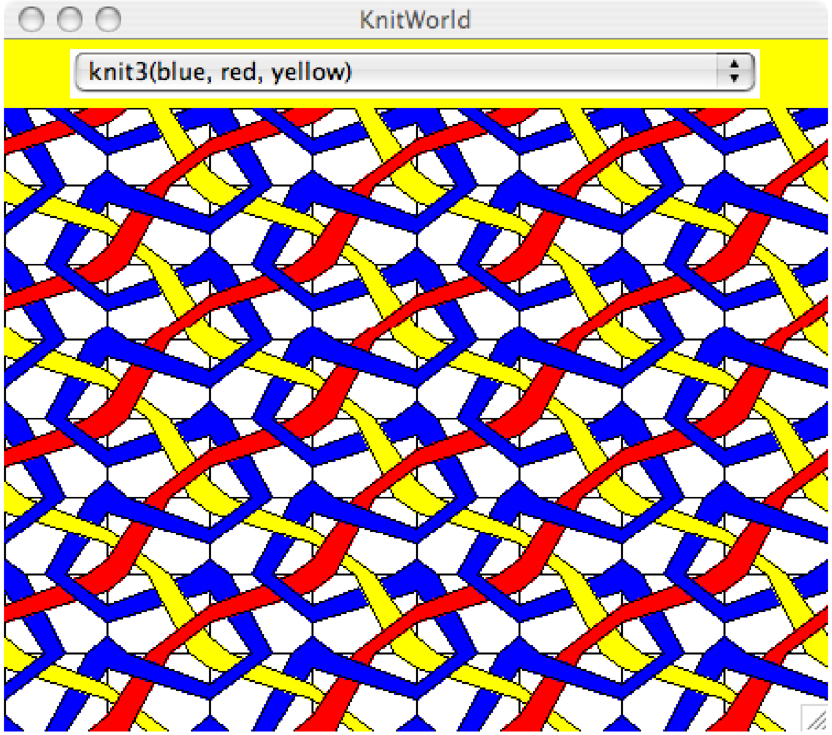
Primitives

Means of Combination

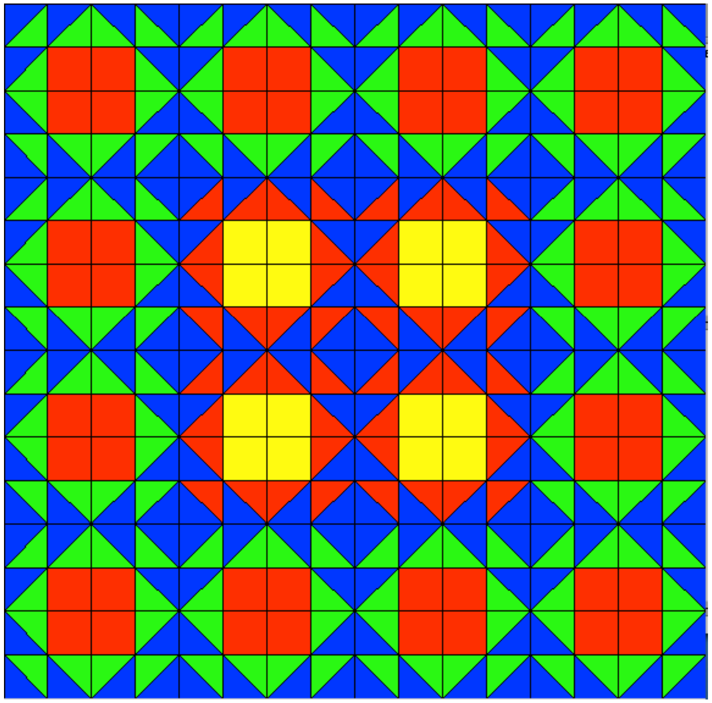
Means of Abstraction

Plug #4: SICP

PictureWorld

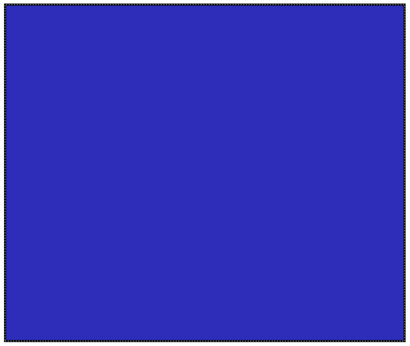


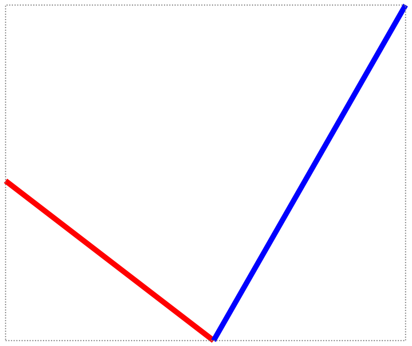
knit3(blue, red, yellow)



PictureWorld: Some Primitive Pictures

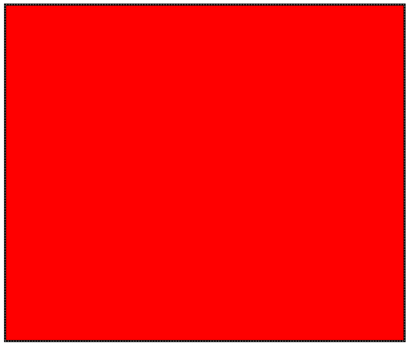
bp
(blue patch)

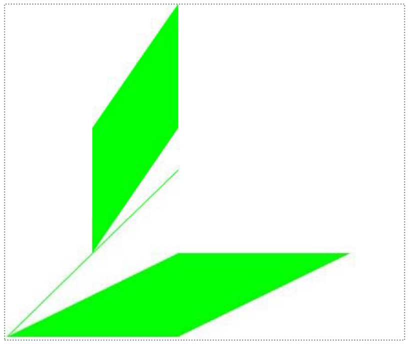




mark

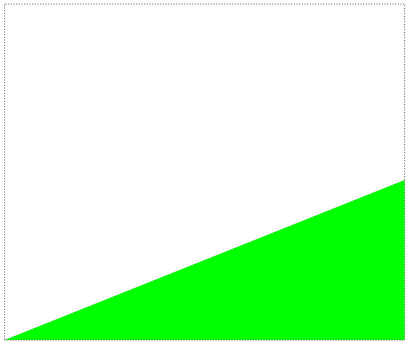
rp
(red patch)

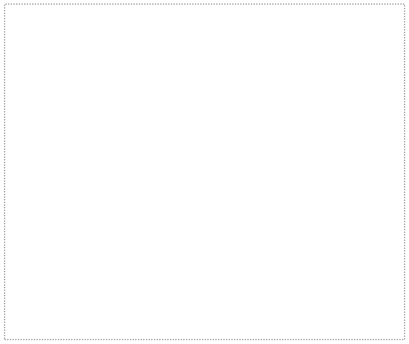




leaves

gw
(green
wedge)

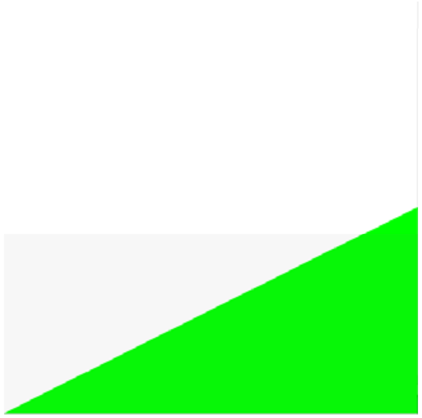




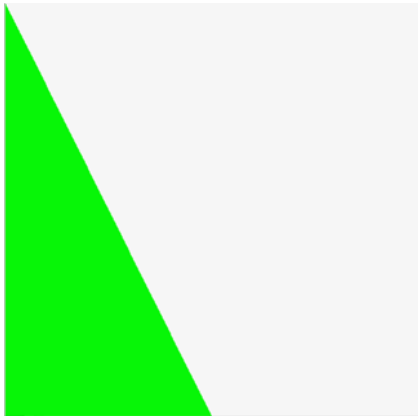
empty

Rotating Pictures

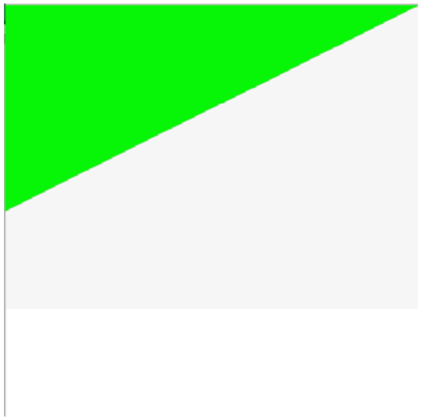
```
public Picture clockwise90(Picture p);  // Returns p rotated  90° clockwise
public Picture clockwise180(Picture p); // Returns p rotated 180° clockwise
public Picture clockwise270(Picture p); // Returns p rotated 270° clockwise
```



gw



clockwise90(gw)

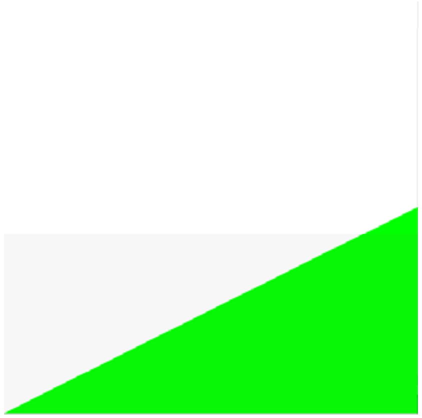


clockwise180(gw)

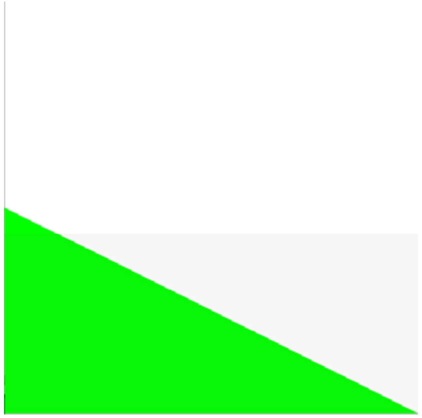
clockwise270(gw)

Flipping Pictures

```
public Picture flipHorizontally(Picture p); // Returns p flipped across vert axis
public Picture flipVertically(Picture p);   // Returns p flipped across horiz axis
public Picture flipDiagonally(Picture p);   // Returns p flipped acros diag axis
```

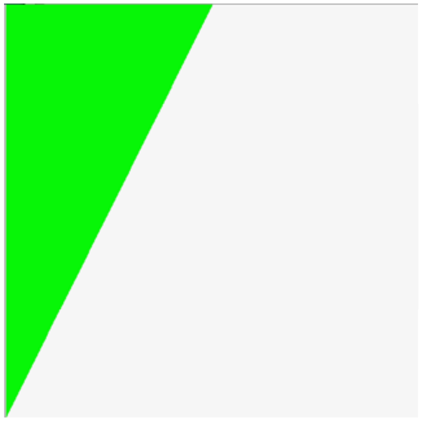


gw



flipHorizontally(gw)

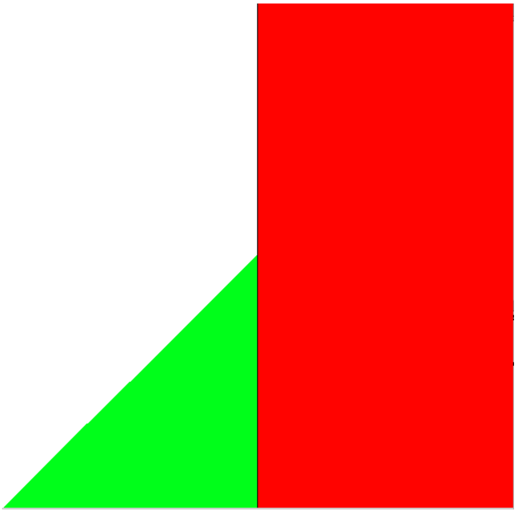
flipVertically(gw)



flipDiagonally(gw)

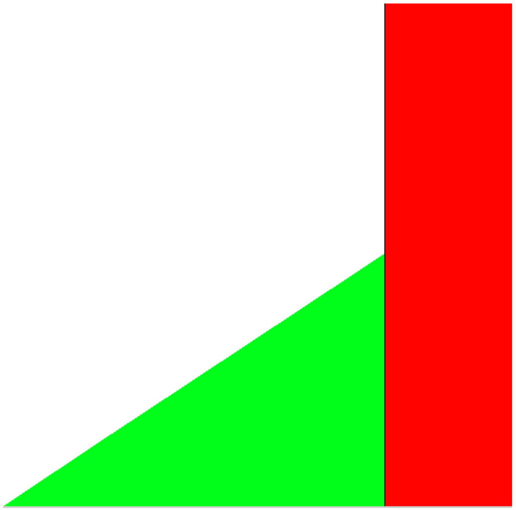
Putting one picture beside another

```
// Returns picture resulting from putting p1 beside p2
public Picture beside(Picture p1, Picture p2);

// Returns picture resulting from putting p1 beside p2,
// where p1 uses the specified fraction of the rectangle.
public Picture beside(Picture p1, Picture p2, double fraction);
```



beside(gw,rp)

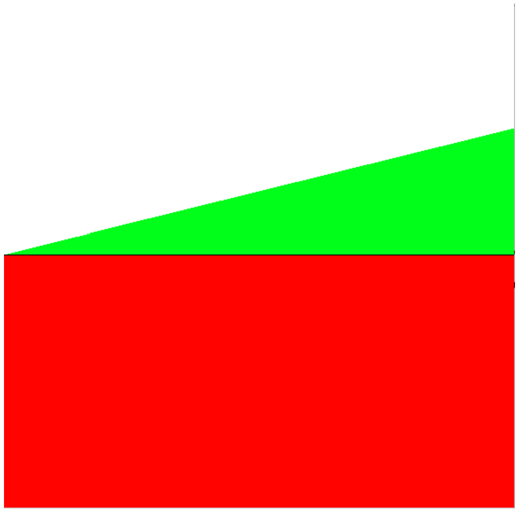
beside(gw,rp,0.25)



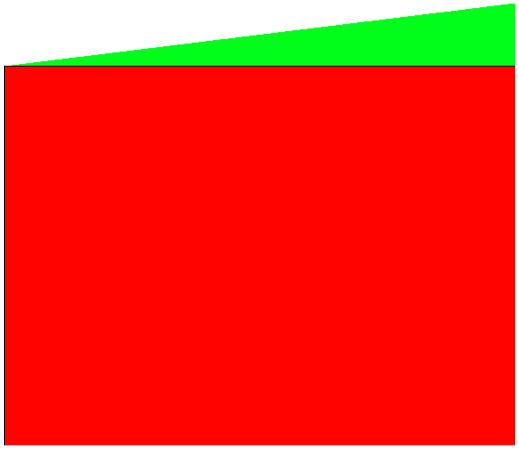
beside(gw,rp,0.75)

Putting one picture above another

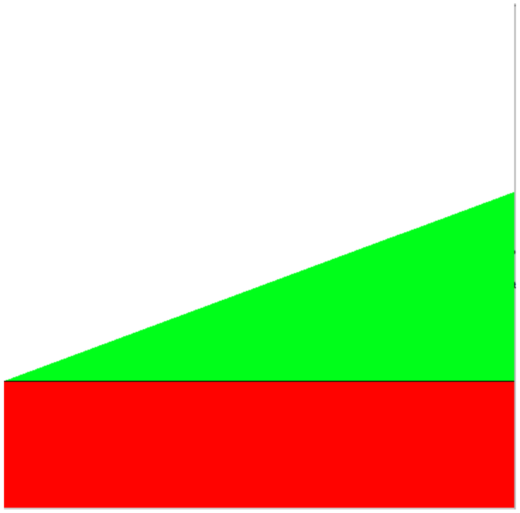
```
// Returns picture resulting from putting p1 above p2
public Picture above(Picture p1, Picture p2);

// Returns picture resulting from putting p1 above p2,
// where p1 uses the specified fraction of rectangle.
public Picture above(Picture p1, Picture p2, double fraction);
```



above(gw,rp)



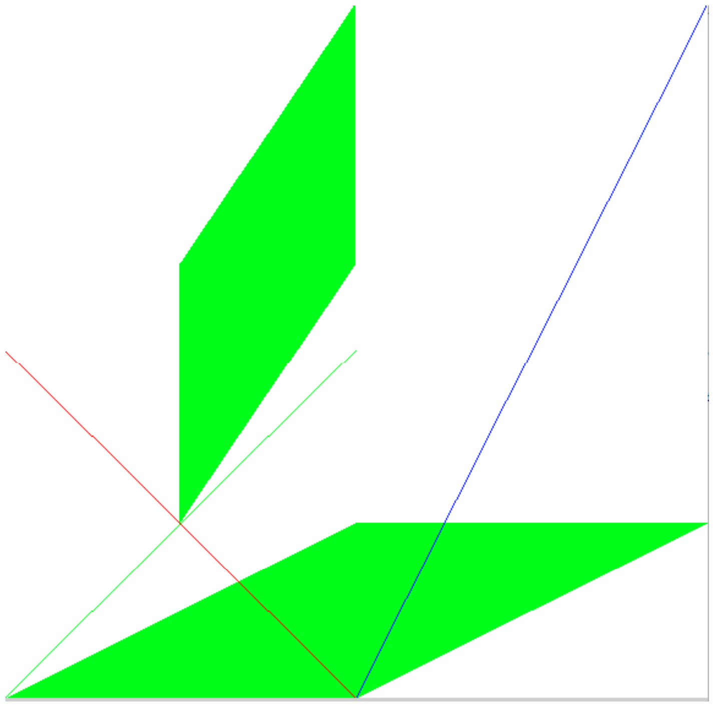
above(gw,rp,0.25)



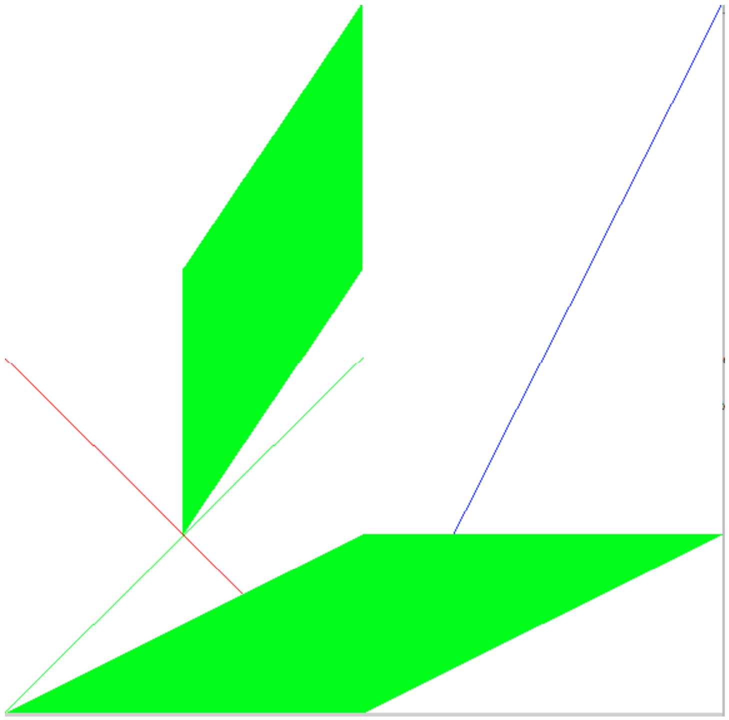
above(gw,rp,0.75)

Putting one picture over another

```
// Returns picture resulting from overlaying p1 on top of p2
public Picture overlay(Picture p1, Picture p2);
```



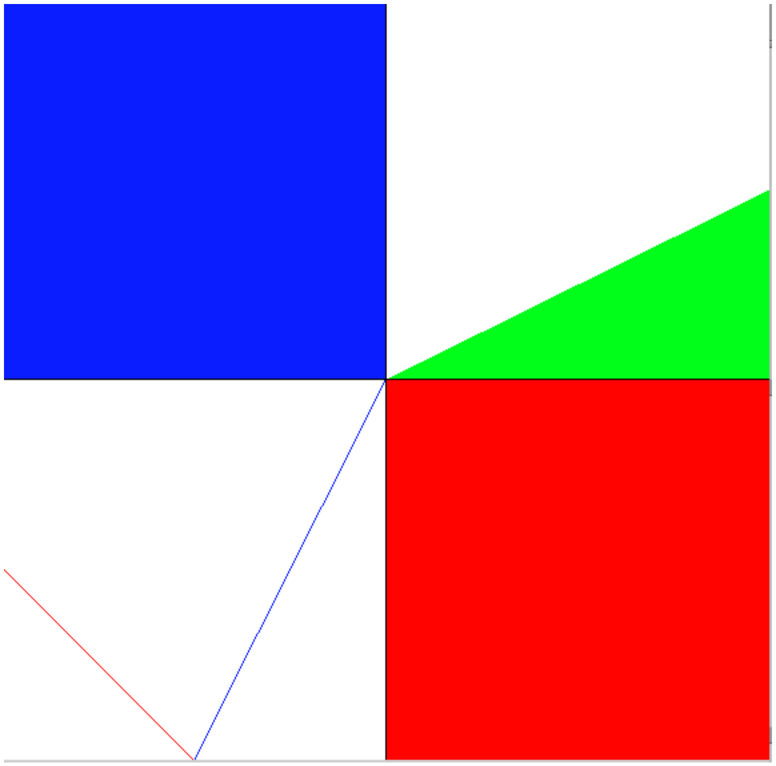
overlay(mark,leaves)



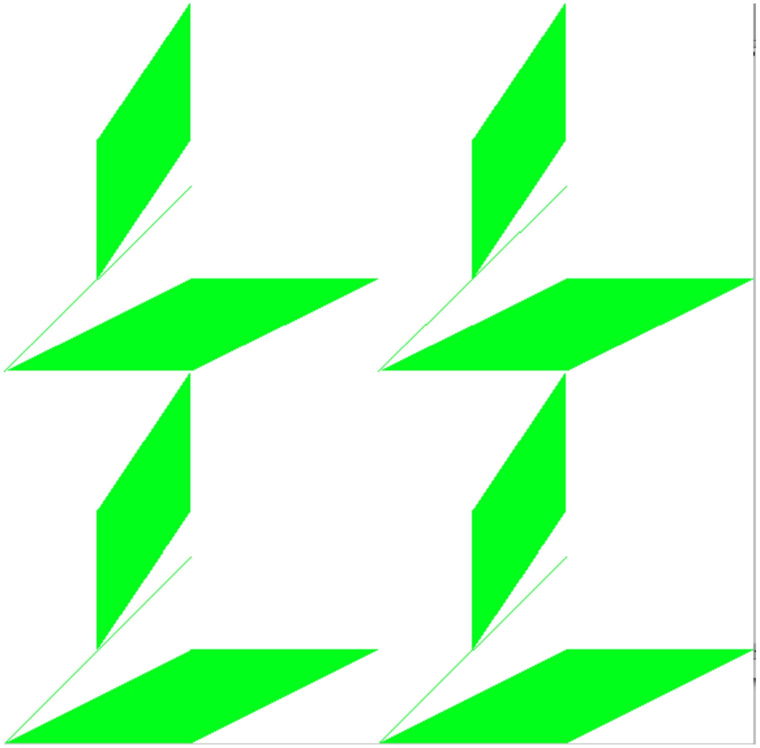
overlay(leaves,mark)

Combining Four Pictures

```
public Picture fourPics (Picture p1, Picture p2, Picture p3, Picture p4) {
   return above(beside(p1,p2), beside(p3, p4)); }

public Picture fourSame (Picture p)  {   return fourPics(p, p, p, p);  }
```



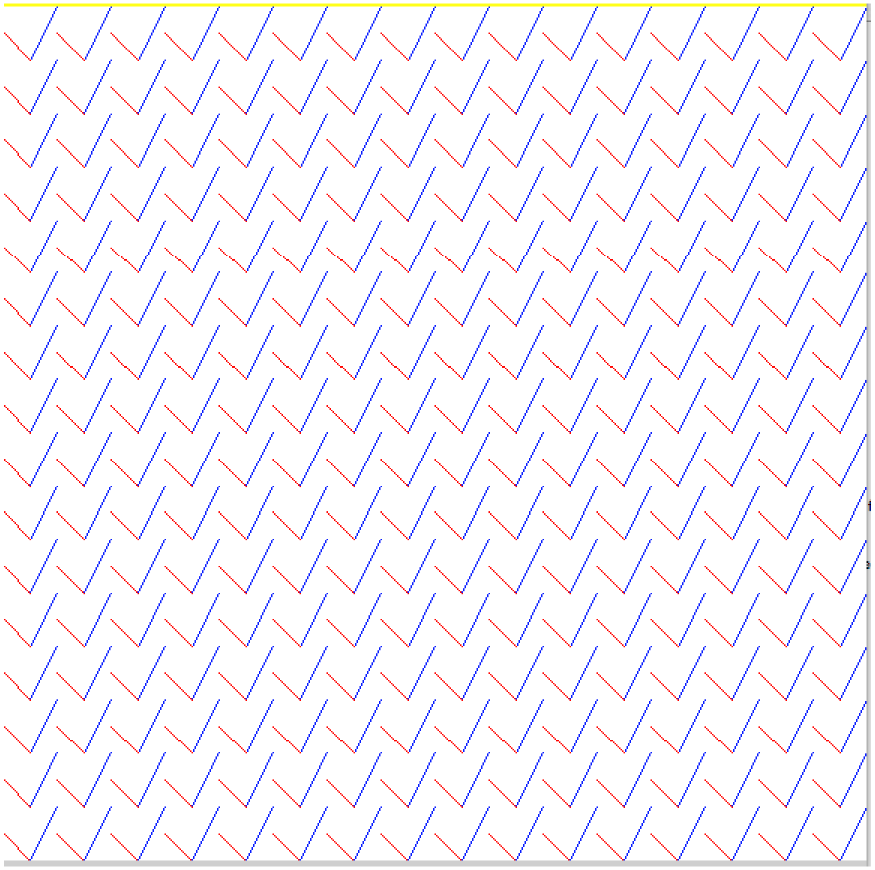
fourPics(bp,gw,mark,rp)



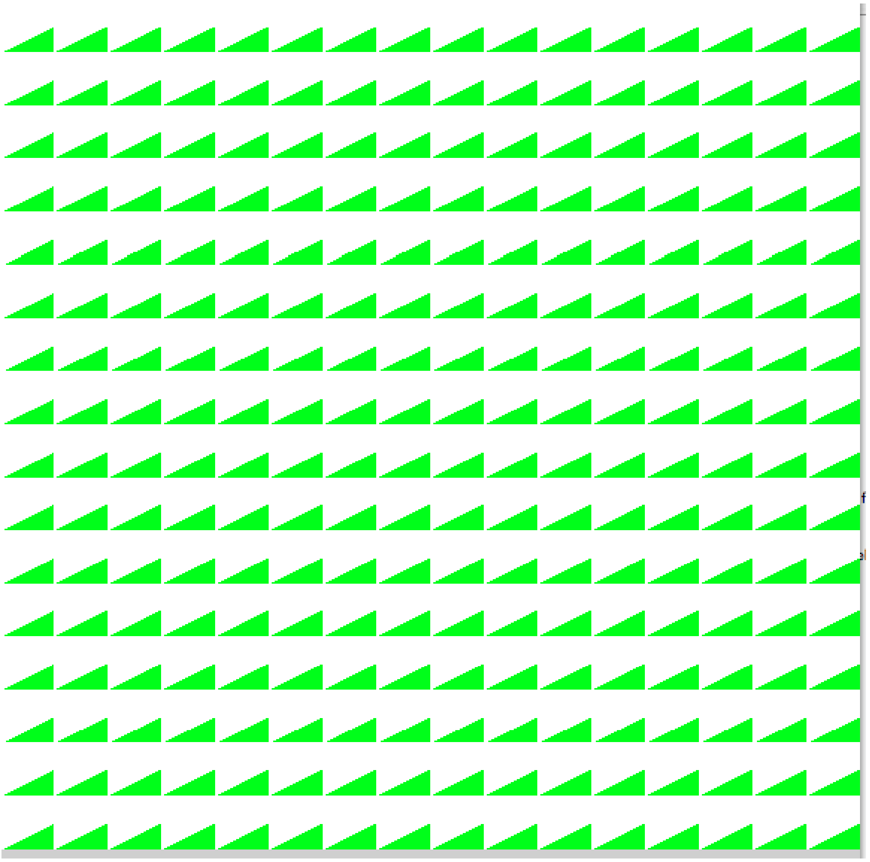
fourSame(leaves)

Repeated Tiling

```
public Picture tiling (Picture p)  {
    return fourSame(fourSame(fourSame(fourSame(p)))); }
```



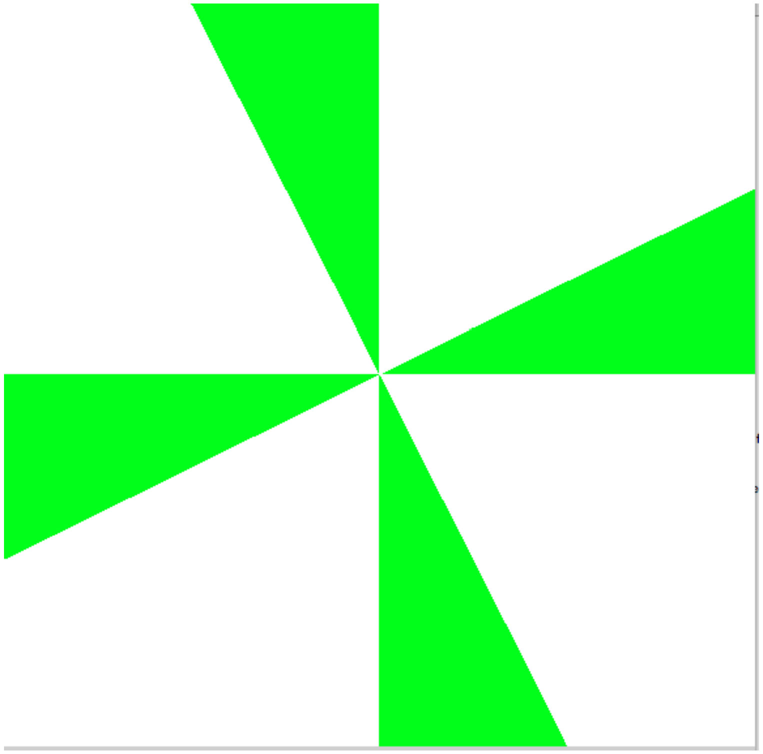
tiling(mark)



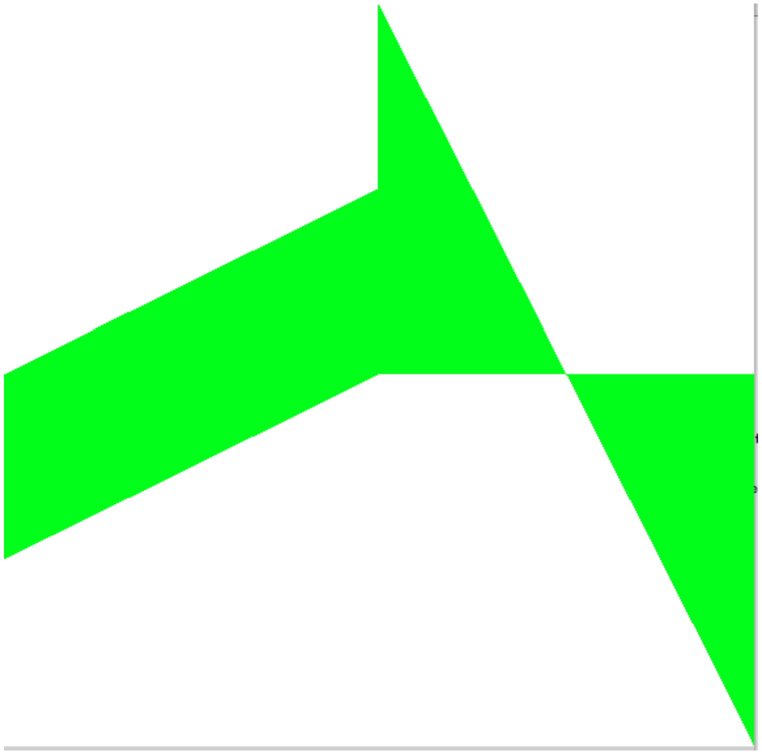
tiling(gw)

Rotation Combinators

```
public Picture rotations (Picture p)  {
    return fourPics(clockwise270(p), p, clockwise180(p), clockwise90(p)); }

public Picture rotations2 (Picture p)  {
   return fourPics(p, clockwise90(p), clockwise180(p), clockwise270(p)); }
```



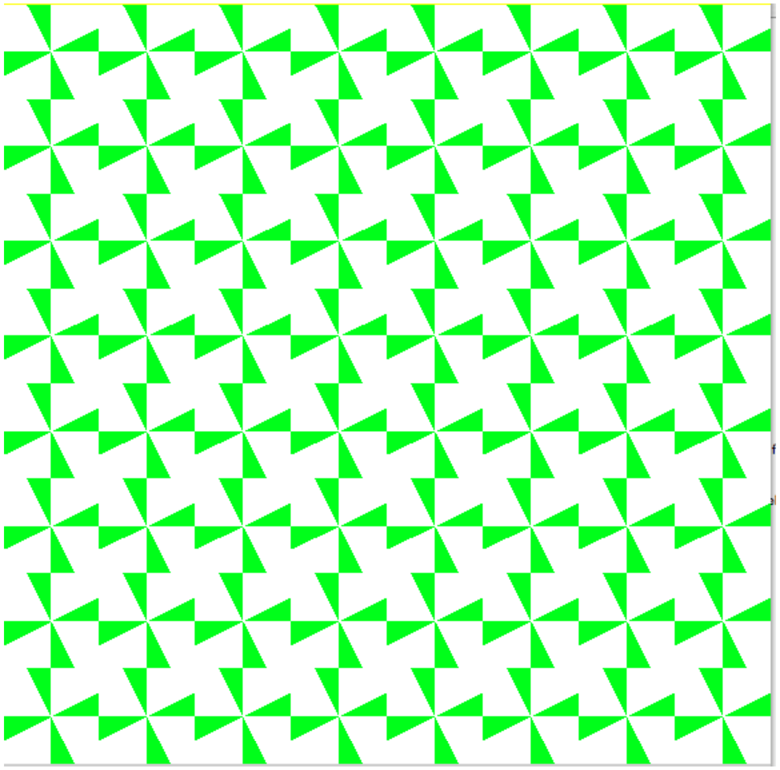
rotations(gw)



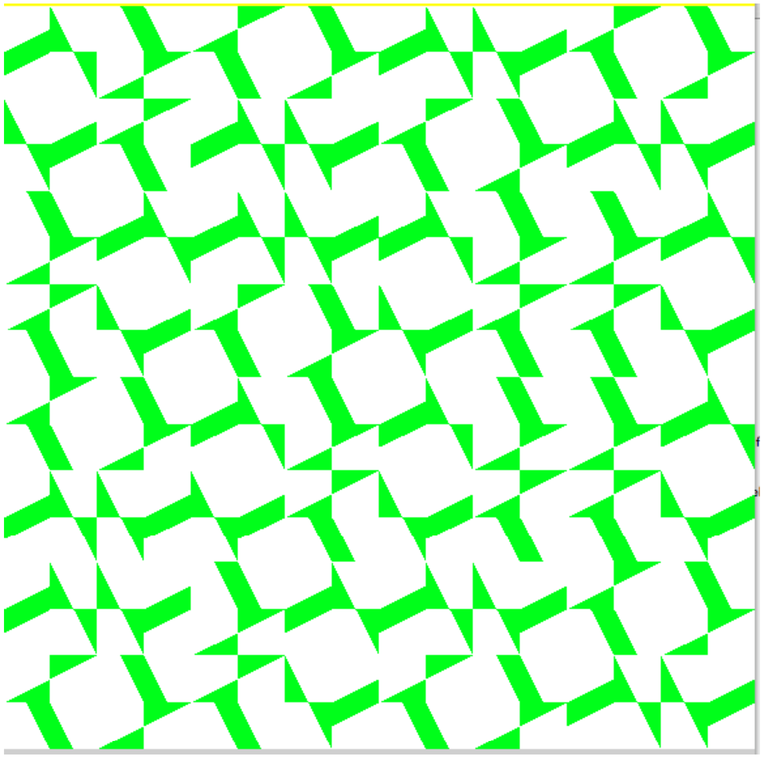
rotations2(gw)

A Simple Recipe for Complexity

```
public Picture wallpaper (Picture p) {
    return rotations(rotations(rotations(rotations(p))); }

public Picture design (Picture p)  {
    return rotations2(rotations2(rotations2(rotations2(p))); }
```



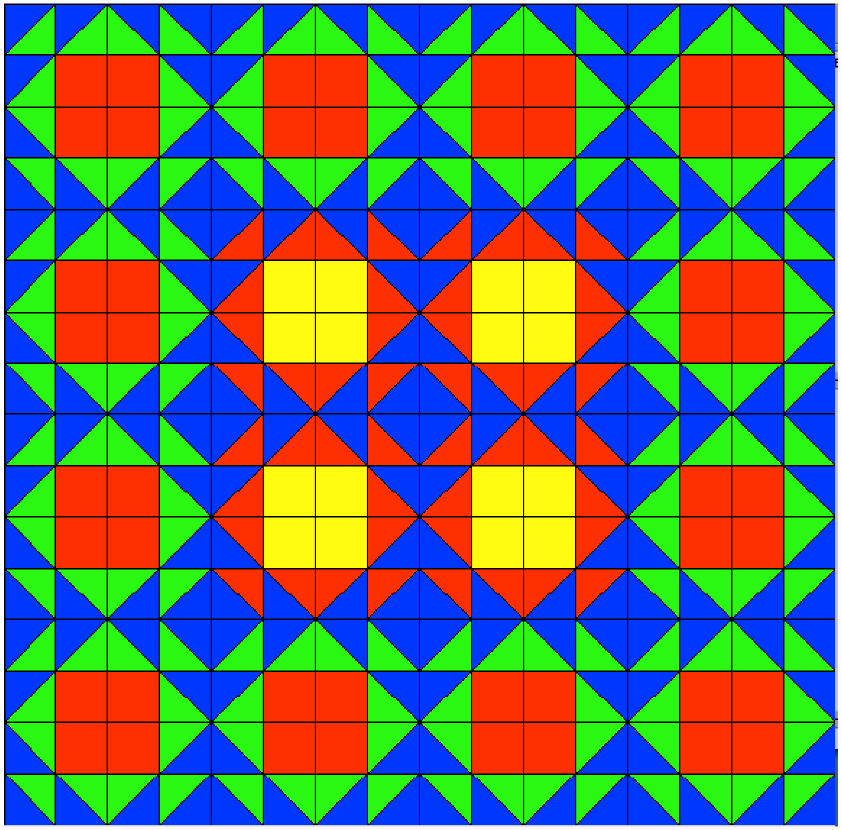
wallpaper(gw)



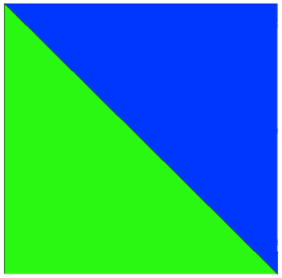
design(gw)

A Quilt Problem

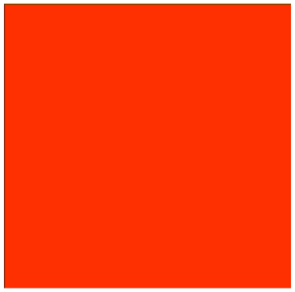
How do we build this complex quilt …



… from simple primitive parts?



triangles(Color.green,
Color.blue)



patch(Color.red)

Divide, conquer & glue

Divide

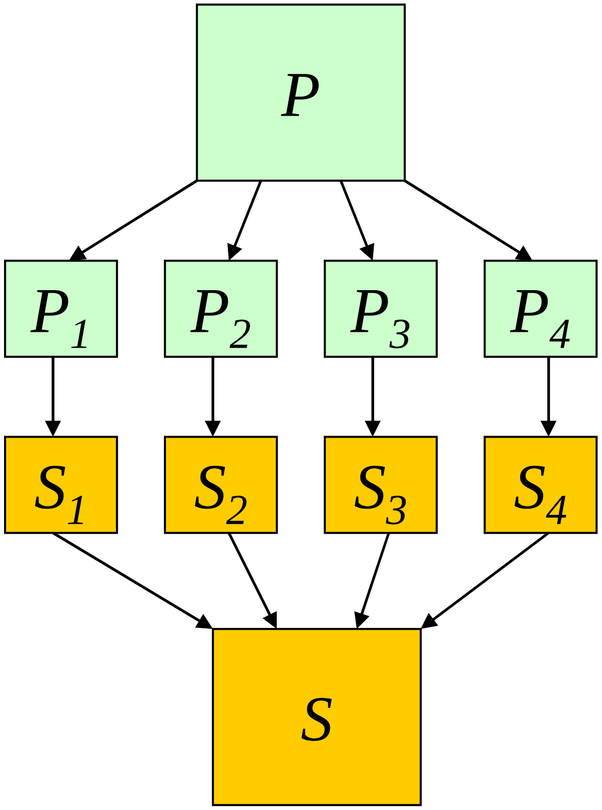
problem P into subproblems.

Conquer

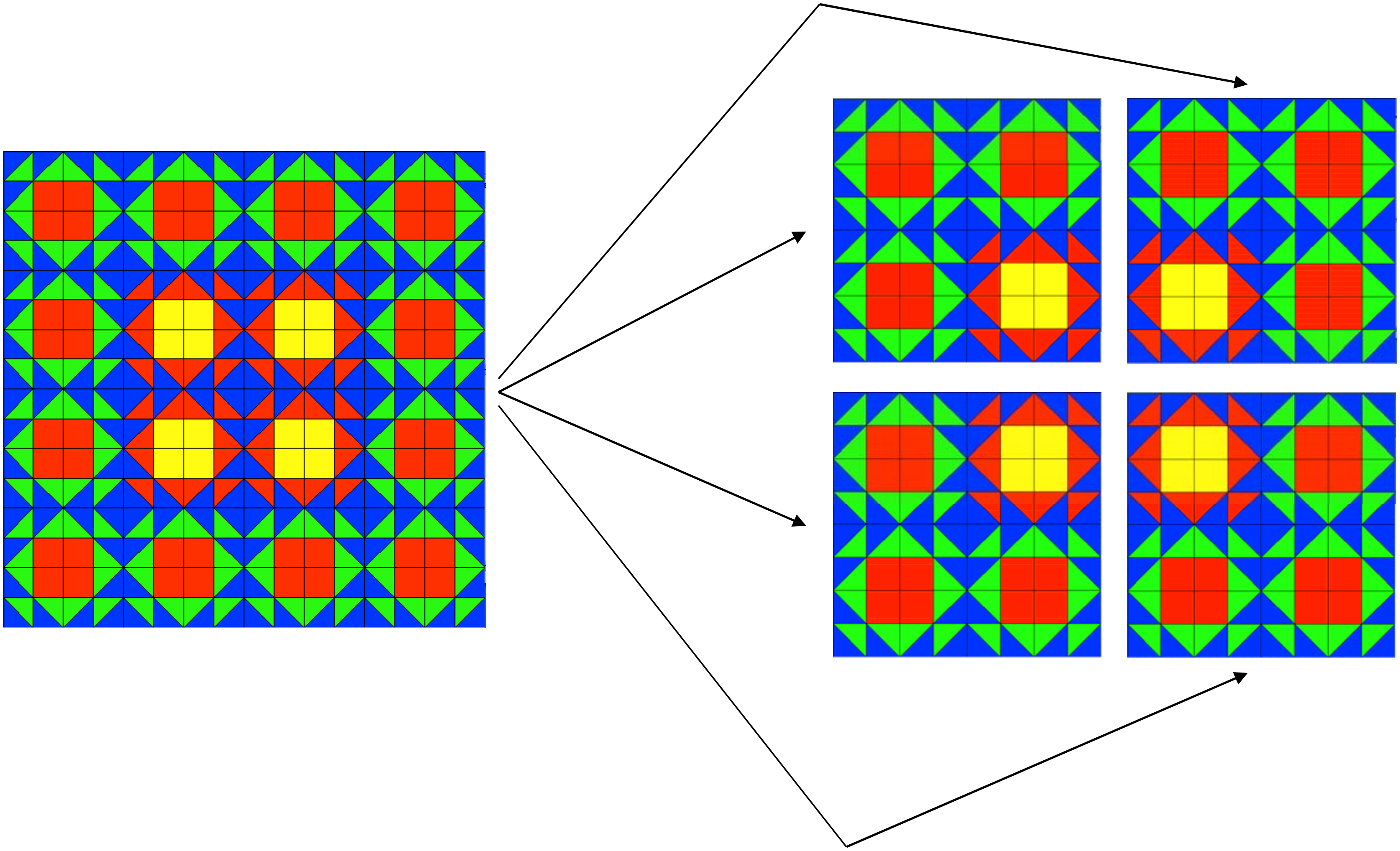
each of the subproblems, &

Glue (combine)

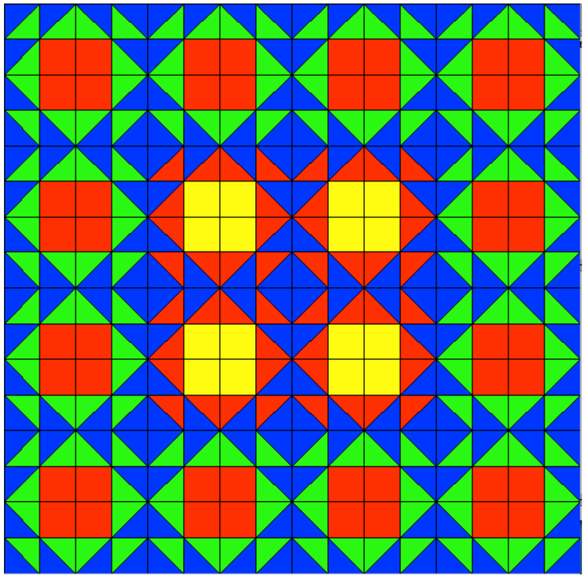
the solutions to the subproblems into a solution S for P.

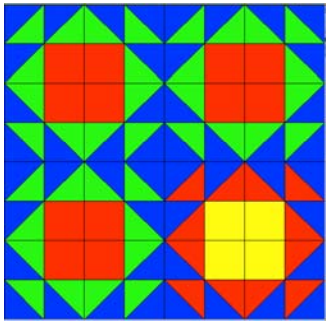


Divide the Quilt in Subproblems

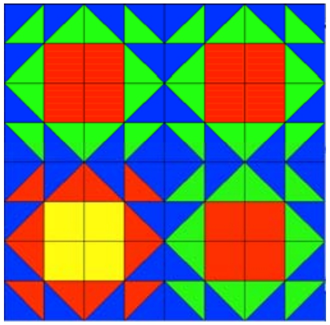


Conquer the Subproblems using wishful thinking

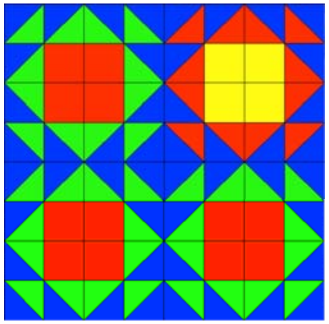




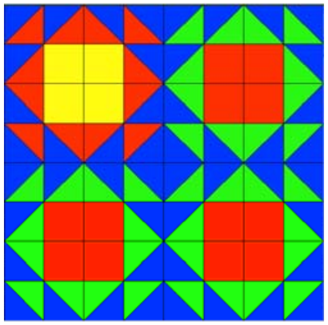
clockwise270(quadrant())



quadrant()

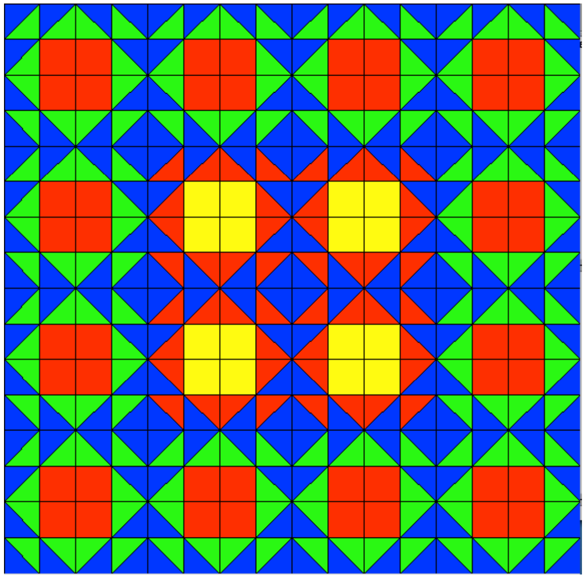


clockwise180(quadrant())



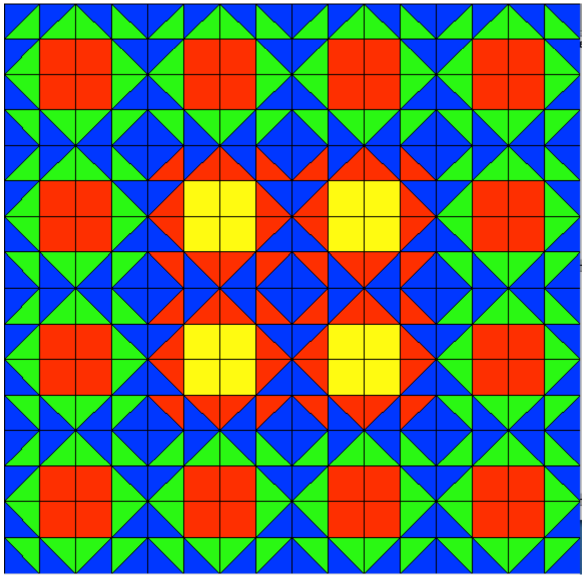
clockwise90(quadrant())

Glue the Solutions to Solve the Problem



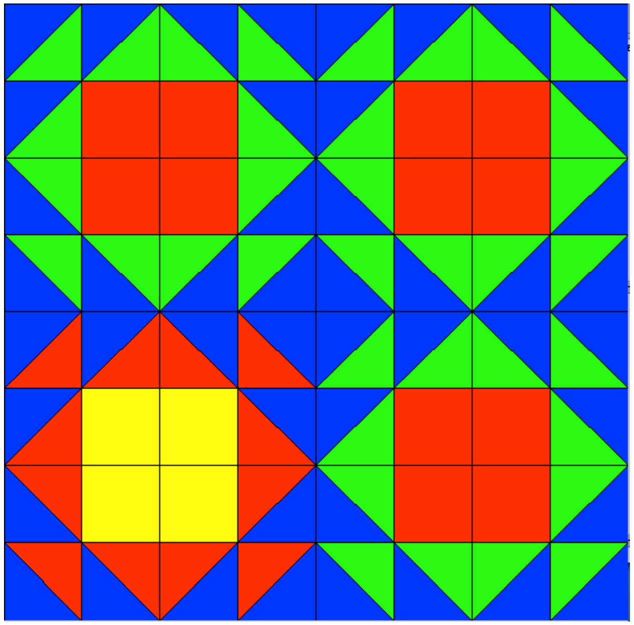
```
public Picture quilt () {
      return fourPics(clockwise270(quadrant()),
                      quadrant(),
                      clockwise180(quadrant()),
                      clockwise90(quadrant()));
}
```

Abstracting Over the Glue



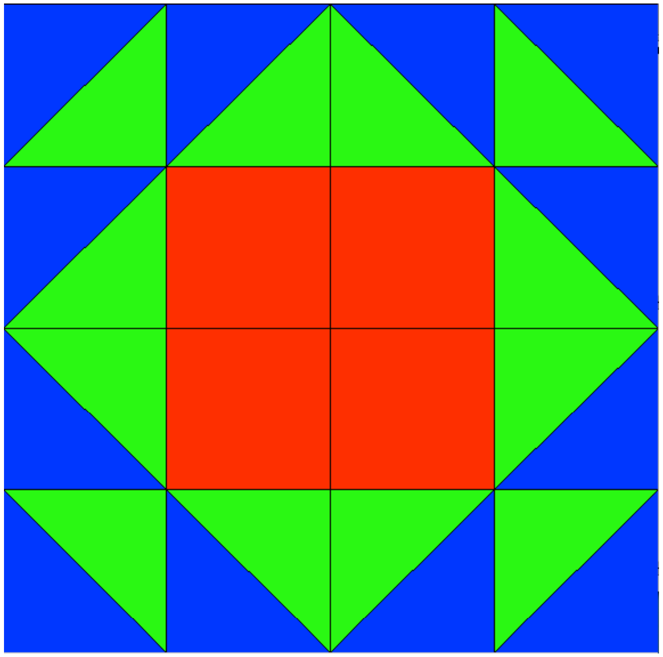
```
public Picture quilt()  {
    return rotations (quadrant());
}

public Picture rotations (Picture p)  {
    return fourPics(clockwise270(p), p,
                    clockwise180(p), clockwise90(p));

}
```

Now Figure out quadrant()



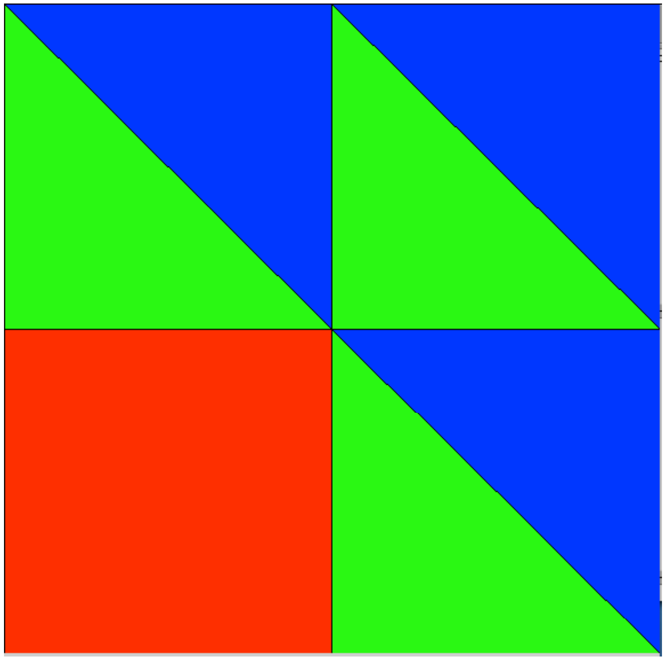
quadrant()

Continue the Descent …



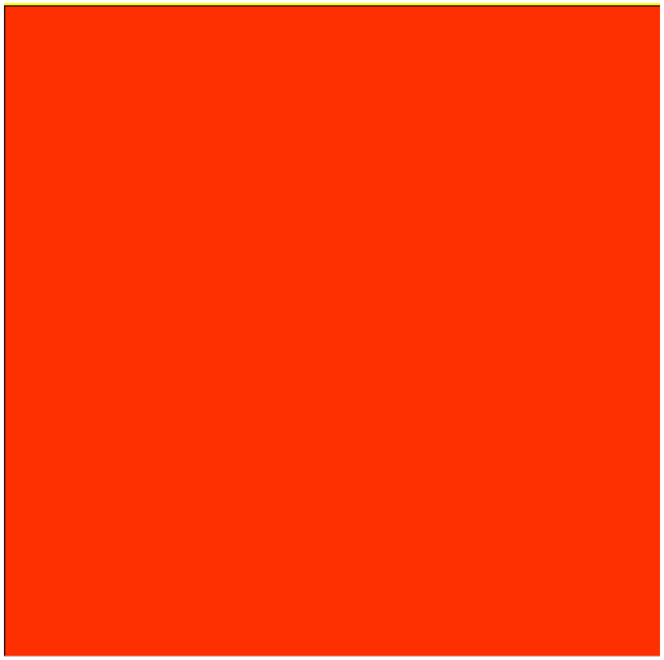
star(Color.red, Color.green, Color.blue)

And Descend Some More ...

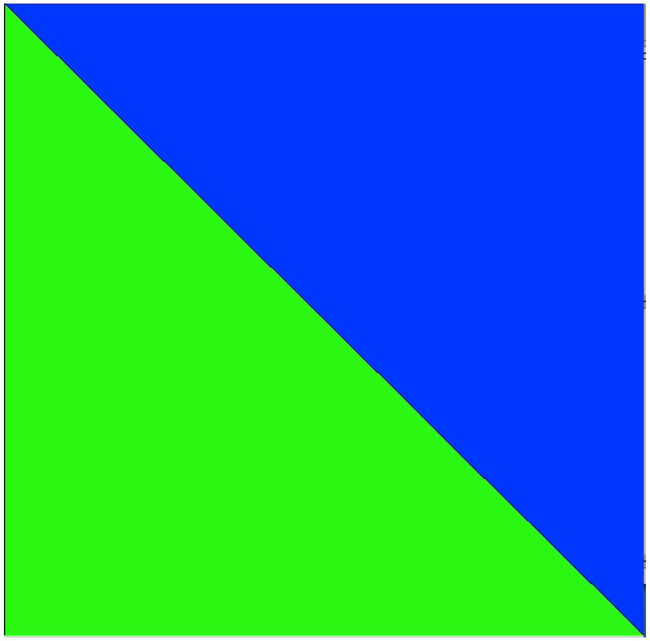


starQuadrant(Color.red, Color.green, Color.blue)

Until we Reach Primitives

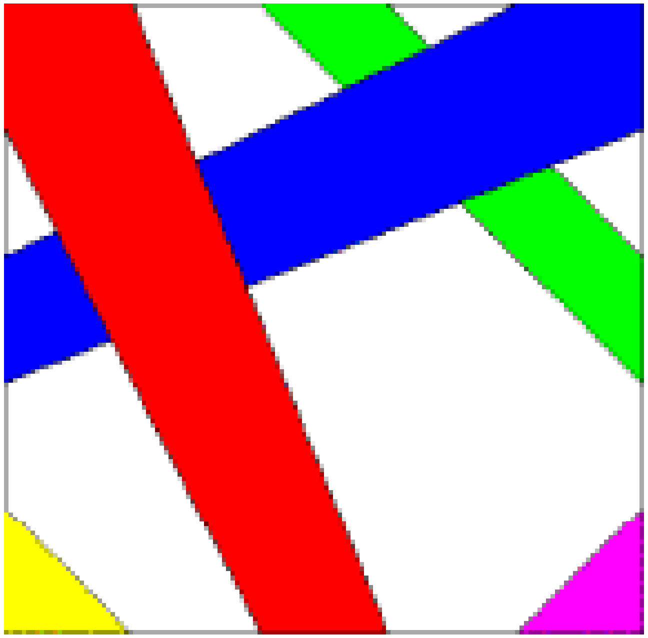


patch(Color.red)

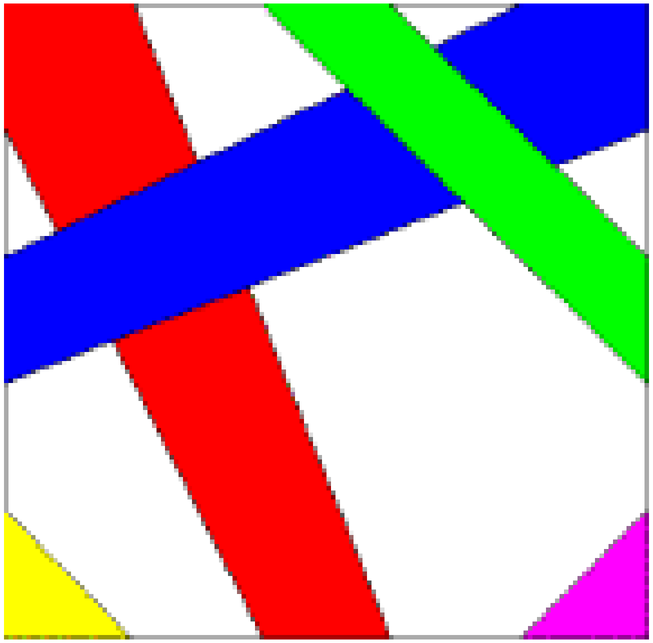


triangles(Color.green, Color.blue)

Knitting Primitives

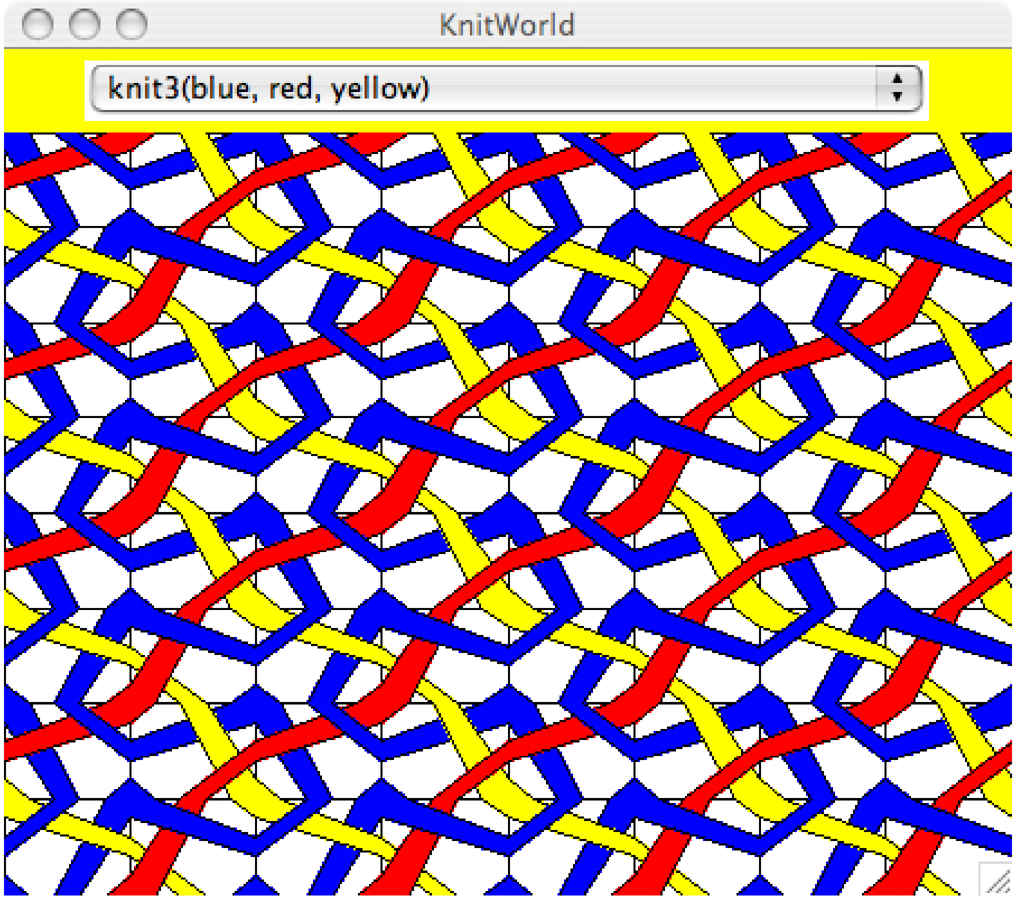


```
A( Color.red,
    Color.blue,
    Color.green,
    Color.yellow,
    Color.magenta);
```



```
B( Color.red,
    Color.blue,
    Color.green,
    Color.yellow,
    Color.magenta);
```

A Knitting Pattern



knit3(blue, red, yellow)

```
public Picture tileKnit
   (Picture p1, Picture p2,
    Picture p3, Picture p4) {
  return
    fourSame(
      fourSame(
        fourPics(p1, p2, p3, p4)));
}
```

```
public Picture knit3(Color c1, Color c2, Color c3) {
   return tileKnit(B(c1, c2, c1, c3, c1),
                        clockwise90(B(c1, c3, c2, c2, c1)),
                        flipHorizontally(B(c1, c3, c1, c2, c1)),
                        flipHorizontally(clockwise90(A(c1, c2, c3, c3, c1)))); }
```

"Religious" Views

The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offense. – *Edsger Dijkstra*

It is practically impossible to teach good programming to students that have had a prior exposure to BASIC: as potential programmers they are mentally mutilated beyond hope of regeneration. – *Edsger Dijstra*

You're introducing your students to programming in C? You might as well give them a frontal lobotomy! – *A colleague of mine*

A LISP programmer knows the value of everything, but the cost of nothing.
Alan Perlis

A language that doesn't affect the way you think about programming, is not worth knowing. . *Alan Perlis*

General Purpose PLs

JAVA

Python

FORTRAN

Perl

ML

Scheme

Haskell

Ruby

C/C++

CommonLisp

Domain Specific PLs

Excel

HTML

CSS

OpenGL

JavaScript

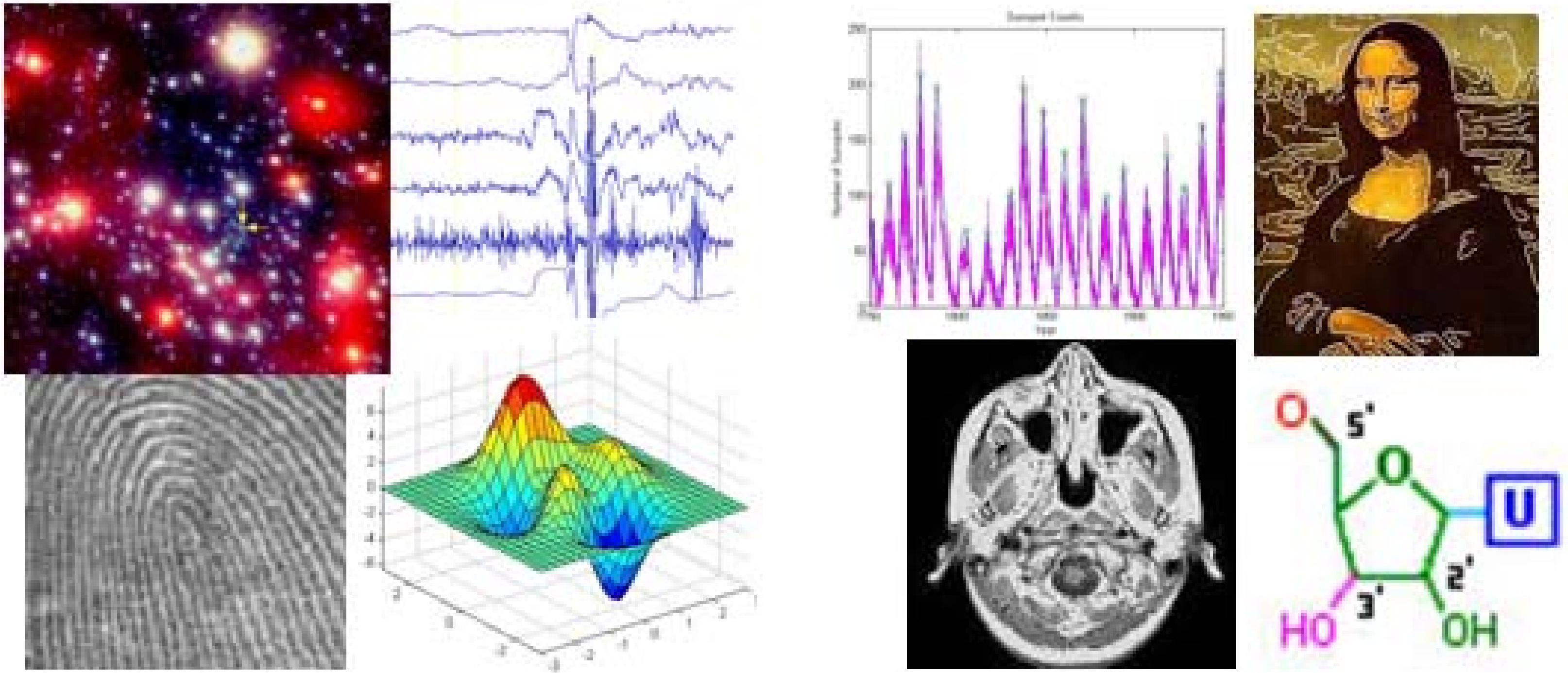
laTeX

Matlab

IDL

PostScript

Plug #5: CS112 Computation for the Sciences



http://cs.wellesley.edu/~cs112/

PL Implementation: Interpretation

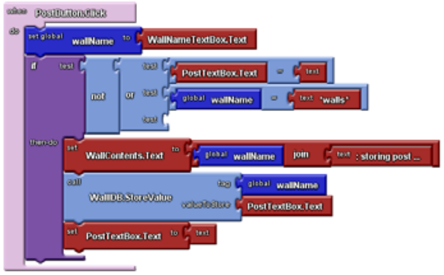
```
when PostButton.Click
do  set global wallName to WallNameTextBox.Text
    if test  not  or  test  PostTextBox.Text = text
                      test  global wallName = text 'walls'
                      test
    then do  set WallContents.Text to  global wallName  join  text : storing post ...
             call WallDB.StoreValue  tag  global wallName
                                     valueToStore  PostTextBox.Text
             set PostTextBox.Text to  text
```

Program in language L

Interpreter for language L on machine M

Machine M

PL Implementation: Translation



```
when PostButton.Click
do set global wallName to WallNameTextBox.Text
   if test not or test PostTextBox.Text = text
                  test global wallName = text 'walls'
                  test
   then-do set WallContents.Text to global wallName join text : storing post ...
           call WallDB.StoreValue tag global wallName
                                  valueToStore PostTextBox.Text
           set PostTextBox.Text to text
```

Program in
language A

A to B translator

Program in
language B

Interpreter
for language B
on machine M

Machine M

PL Implementation: Embedding

```
when PostButton.Click
do  set global wallName to WallNameTextBox.Text
    if test  not  or  test  PostTextBox.Text = text
                      test  global wallName = text 'walls'
                      test
    then do  set WallContents.Text to  global wallName  join  text : storing post ...
             call WallDB.StoreValue  tag  global wallName
                                     valueToStore  PostTextBox.Text
             set PostTextBox.Text to  text
```

Program in
language A
embedded in
language B

Interpreter
for language B
on machine M

Machine M

Future Work

Languages for making artifacts on
laser cutter & 3D printer

Generalizing tools for creating blocks languages.

Do you need a domain specific language?
Maybe I can help!