```
//Scala Code
def compute(xc: Double, yc: Double, threshold: Int): Int = {
  var i = 0
  var x = 0.0
  var y = 0.0
  while (x * x + y * y < 2 && i < threshold) {
    val xt = x * x - y * y + xc
    val yt = 2 * x * y + yc

    x = xt
    y = yt

    i += 1
  }
  i
}
```

```
#Maple version
compute := proc(xc::float, yc::float, threshold::integer)
     local i,x,y,xt,yt;

     x := 0.0;
     y := 0.0;
     for i from 0 by 1 while ((x*x + y*y < 2) and i<threshold) d
          xt := x*x - y*y + xc;
          yt := 2*x*y + yc;
          x := xt;
          y := yt;
     end do;
     return i;
end proc
```

```
> compute(.5, .5, 10)
```

```
//Scala code
for (idx <- 0 until (wdt * hgt)) {
  val x = idx % wdt
```

```
  val y = idx / wdt
  val xc = xlo + (xhi - xlo) * x / wdt
  val yc = ylo + (yhi - ylo) * y / hgt

  val iters = compute(xc, yc, threshold)
  val color = if (iters < threshold) 0xff000000 else 0xffffffff
  pixels(idx) = color
}
```

```
#Maple Version
mandelbrot := proc( xlo::numeric, xhi::numeric, ylo::numeric, yh
     threshold::integer)
     option hfloat; #Use hardware floating point rather than soft
     #xlo, xhhi, ylo, yhi should be in the range -1 to 1.
     local x,y,xc, yc, iters, color, pixels, pixelsrgb, idx, segm
          ONETHIRD, TWOTHIRDS, MINCOLORVALUE, THREE;
     pixels := Array(0..wdt,0..hgt,datatype = float);
     pixelsrgb := Array(0..wdt, 0..hgt, 1..3, datatype=float);
     ONETHIRD := convert(1/3, float); #floating constant
     TWOTHIRDS := 2*ONETHIRD; #another floating constant
     MINCOLORVALUE := .1; #another constant defining minimum colo
     THREE := 3.0;
     WHITE := 0.0;
     BLACK := 1.0;
     for idx from 0 to wdt*hgt do
          x := idx mod wdt;
          y := iquo(idx,wdt);
          xc := convert(xlo + (xhi - xlo) * (x /wdt),float);
          yc := convert(ylo + (yhi-ylo)*y/hgt, float);
          iters := compute(xc, yc, threshold);
          #color a pixel black if it converges within the thresh
          if (iters<threshold) then color := BLACK else color :=
          pixels[x,y] := color;
     #    if (iters >=threshold/2) then print(xc,yc,iters, color
          #Color a pixel black if it doesn't convert, white if it
          #faster than the threshold limit.
          segment := convert(iters/threshold,float );

          pixelsrgb[x,y,1] := ifelse (segment <= ONETHIRD, max(M
          pixelsrgb[x,y,2] := ifelse(segment > ONETHIRD and segm
          pixelsrgb[x,y,3] := ifelse (segment >=TWOTHIRDS, max(M
          od;
     return pixels, pixelsrgb;
end;


>
```

```
mandelRun := proc(SIZE::integer, THRESHOLD::integer,
     XLO::float, XHI::float, YLO::float, YHI::float)
     local startTime, pixels, pixelsrgb, endTime, runTime;
     with(ImageTools);
     startTime := time();
     pixels, pixelsrgb := mandelbrot(XLO, XHI, YLO, YHI, SIZE, S
     endTime := time();
     runTime := endTime-startTime;
     return runTime, pixels, pixelsrgb;
end;
```

```
#Whole300 Benchmark
SIZE := 300;
THRESHOLD := 300;
XLO := -1.5;
XHI := 1.5;
YLO := -1.5;
YHI := 1.5;

runTime, pixels, pixelsrgb := mandelRun(SIZE, THRESHOLD, XLO, XH

imageM1 := Create(SIZE, SIZE, 1, pixels);
imageM2 := Create(SIZE, SIZE, 3, pixelsrgb);

BWfilename := sprintf("BW-%d-%d- %3.2f, %3.2f, %3.2f, %3.2f.JPEG
Colorfilename := sprintf("C-%d-%d- %3.2f, %3.2f, %3.2f, %3.2f.JP
sprintf("filename = %s, time=%5.2f",BWfilename, runTime);
Write(BWfilename,imageM1);
Write(Colorfilename, imageM2);
View([imageM1, imageM2], title = sprintf("Mandelbrot set from (%
XLO, YLO, XHI, YHI, THRESHOLD, SIZE, runTime));
#result := Embed([imageM1, imageM2], title = sprintf("Mandelbrot
#XLO, YLO, XHI, YHI, THRESHOLD, SIZE, runTime));
#print(result);
```

>

Seahorse400 Benchmark

>

Turtle300 Benchmark

>

Turtle400 Benchmark

>