EECS 1710
Programming for Digital Media

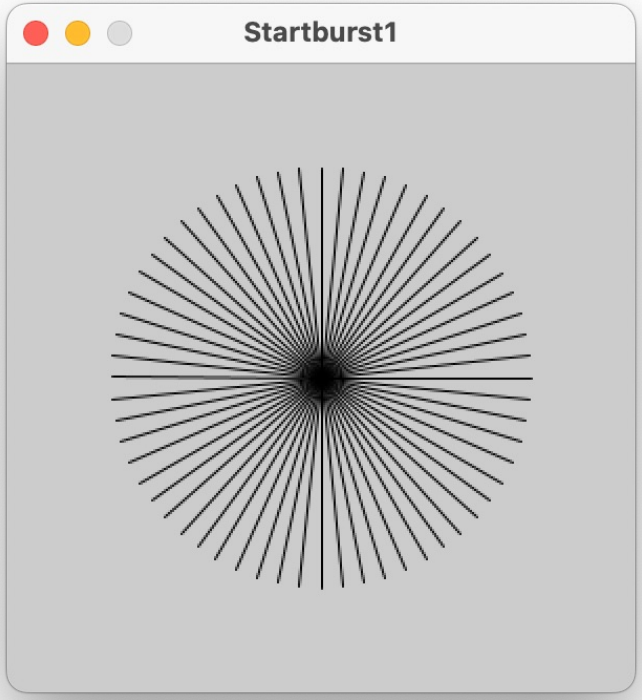
Lecture 10 :: Loops & Repetition [2] / P5

YORK UNIVERSITÉ UNIVERSITY

Loop examples:

- Starburst
  - iterating over a loop counter
  - restricting when we do something in a loop
    - restricting using if/else
    - restricting using %
  - iterating/stepping over the range of a variable
  - randomizing properties
  - smoothly varying a property
- Nested loops

How would you draw this type of pattern using a loop?

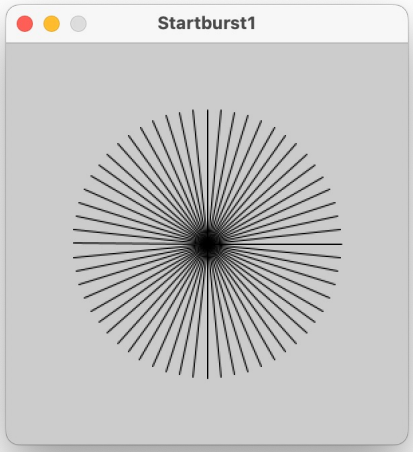


calc angle step based on # rays

iterate over number of rays
- use a counter i
- start at i=0
- calc next angle (inc by angle step)
- draw from centre to point on circle
  (using trig, r*cos(angle), r*sin(angle)

Iterating over a loop counter

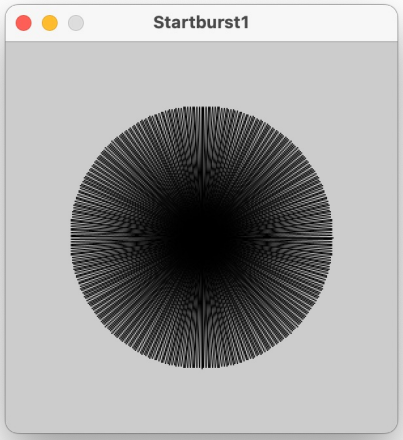
- Previously we iterated over a loop counter i (can think of i as the “index” of a counter



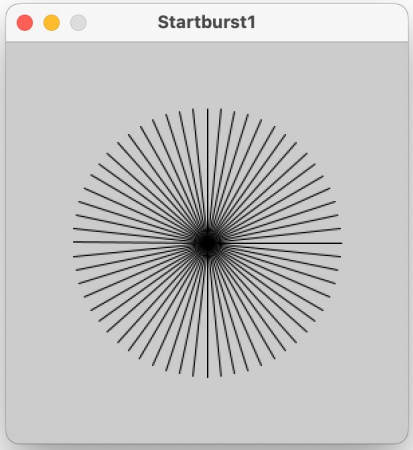

```
thetaStep = 2*PI / DEFAULT_RAYS
          = 360 / DEFAULT_RAYS

theta   =   thetaStep * i
```

YORK UNIVERSITÉ UNIVERSITY

```
// Starburst

final int DEFAULT_RADIUS = 100;
final int DEFAULT_RAYS = 360;

int centreX;
int centreY;

void setup() {
  size(600, 600);
  centreX = width/2;
  centreY = height/2;
  starburst();
}

void starburst() {
  // a default startburst

  float theta = 0;
  float thetaStep = TWO_PI / DEFAULT_RAYS ;  // in radians

  float radius = DEFAULT_RADIUS;

  // loop for pattern
  for (int i=0; i<DEFAULT_RAYS; i++) {
    theta = thetaStep*i;
    line( centreX, centreY, centreX+radius*cos(theta),
                            centreY+radius*sin(theta));
  }
}
```



360 rays



60 rays

YORK UNIVERSITÉ UNIVERSITY

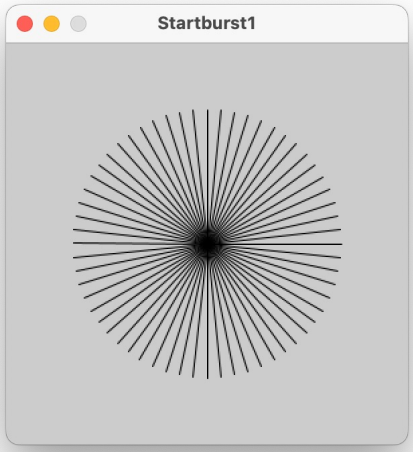
Notes & Modifications?

- Note: if we don't need interaction …
  - we don't **really** need `void draw() { }`
  - can launch and control flow from a single `void setup() {}` method.. any repetitions needed we can control ourself!

- We can restrict rays drawn
  - by wrapping our line() method call in an if statement
  - by using %

Iterating over a loop counter

- Previously we iterated over a loop counter i (can think of i as the “index” of a counter



in which order are we drawing these rays?

```
// RESTRICTING VIA IF STATEMENTS

if (i<10) {
  line( centreX, centreY,
       centreX+radius*cos(theta),
       centreY+radius*sin(theta));
}


if (i==10) {
  stroke(255,0,0);
  line( centreX, centreY,
       centreX+radius*cos(theta),
       centreY+radius*sin(theta));
  stroke(0,0,0);
}


if ((i>10) && (i<50)) {
      stroke(0,0,255);
      line( centreX, centreY,
        centreX+radius*cos(theta),
        centreY+radius*sin(theta));
      stroke(0,0,0);
}
```

Starburst over limited angle

colour to indicate order in which rays are drawn

range of rays drawn

YORK UNIVERSITÉ UNIVERSITY

Controlling the number of drawn rays?

- Control thetaStep = control # rays

  OR

- Iterate over large range, and draw at regular intervals
  - e.g. could iterate with a step of 1deg (TWO_PI/360 radians)
  - this would mean 360 steps at TWO_PI/360 radians per step

  - How else could we draw a single ray every 10 degrees (if we have already chosen a 1 degree step)?

Recall: modulo operator %

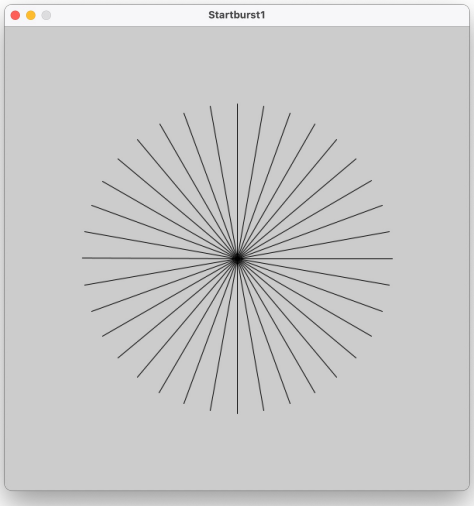
- X%Y → will equal zero every time Y divides evenly into X

If X is changing from 0 to 360 …

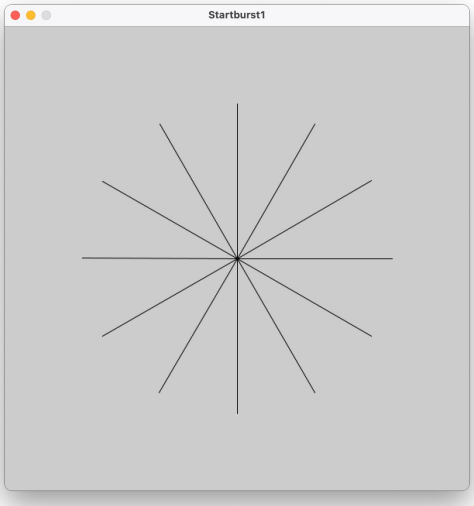
X

- 0 % 10 == 0
- 10 % 10 == 0
- 20 % 10 == 0
- 30 % 10 == 0
- …

```
// Starburst (with modulo)
final int DEFAULT_RADIUS = 200;
final int DEFAULT_RAYS = 360;

int centreX;
int centreY;

void setup() {
  size(600, 600);
  centreX = width/2;
  centreY = height/2;
  starburst();
}

void starburst() {
  float theta = 0;
  float thetaStep = TWO_PI / DEFAULT_RAYS ;
  float radius = DEFAULT_RADIUS;

  // loop for pattern
  for (int i=0; i<DEFAULT_RAYS; i++) {

    theta = thetaStep*i;
    if (i%10==0)
        line( centreX, centreY, centreX+radius*cos(theta),
                                centreY+radius*sin(theta));

  }
}
```



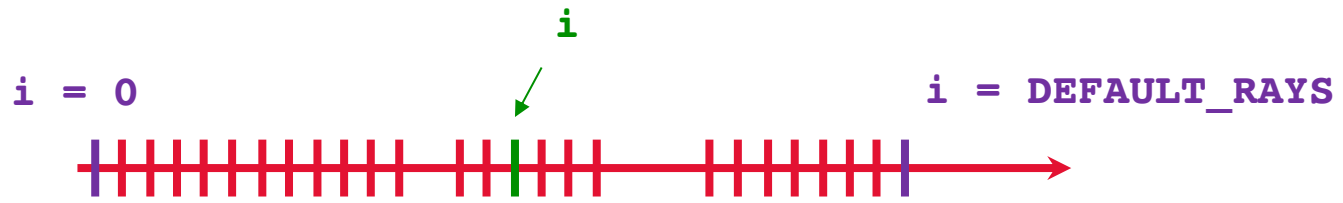
i%10==0



i%30==0

Iterating directly over the range of a variable

- Previously we iterated over a loop counter i (where i is the "index" of a counter)



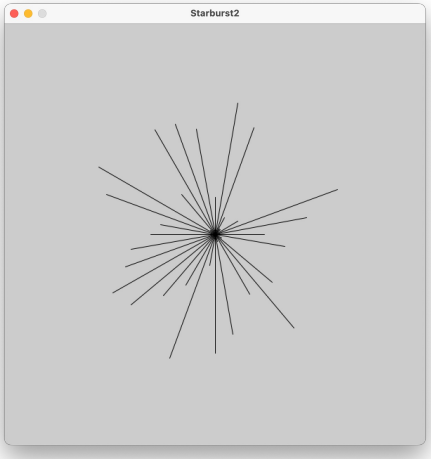
- Could also iterate directly over a variable we are using:

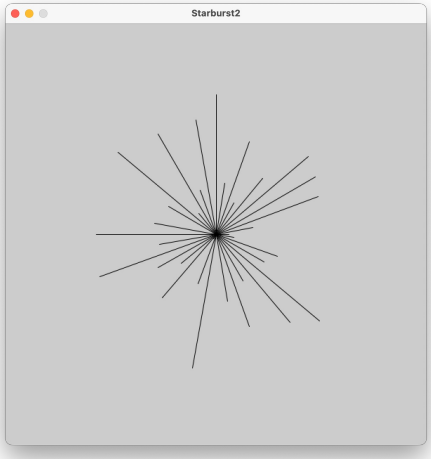
```
for (theta=0; theta < TWO_PI; theta += thetaStep) {
   line( centreX, centreY,
              centreX+radius*cos(theta),
              centreY+radius*sin(theta));
}
```

Modifying/varying other variables in the loop?

```
for (float theta=0; theta<360; theta++) {

    radius = DEFAULT_RADIUS * random(1);

    if ( theta%10==0 )
        line( centreX, centreY,
              centreX+radius*cos(radians(theta)),
              centreY+radius*sin(radians(theta)));

}
```

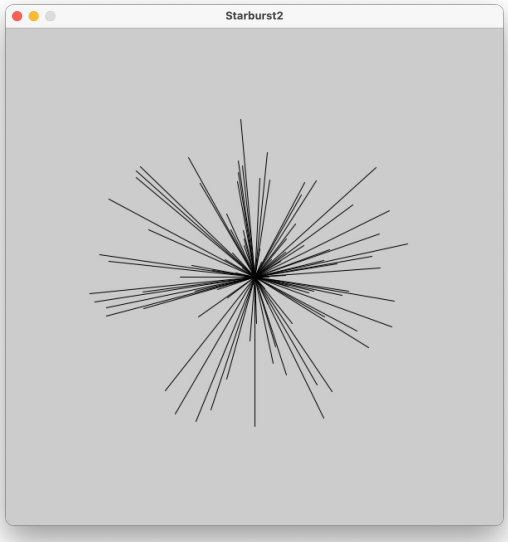
randomize radius





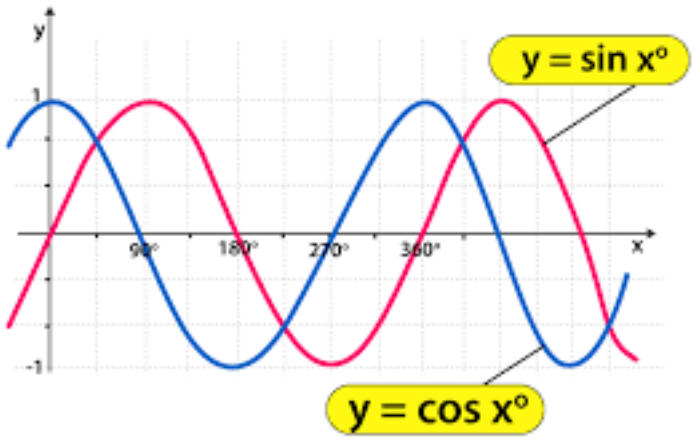
degrees convenient to iterate over (so use radians() to convert degrees to radians)

```
for (float theta=0; theta<360; theta++) {

    radius = DEFAULT_RADIUS * random(1);

    if ( theta%(int)random(10) ==0 )
      line( centreX, centreY,
          centreX+radius*cos(radians(theta)),
          centreY+radius*sin(radians(theta)));
}
```

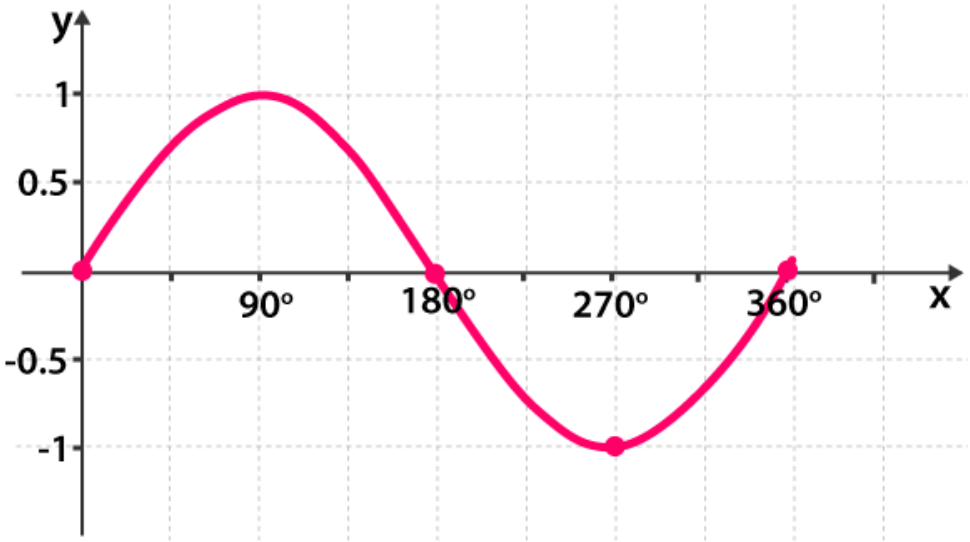


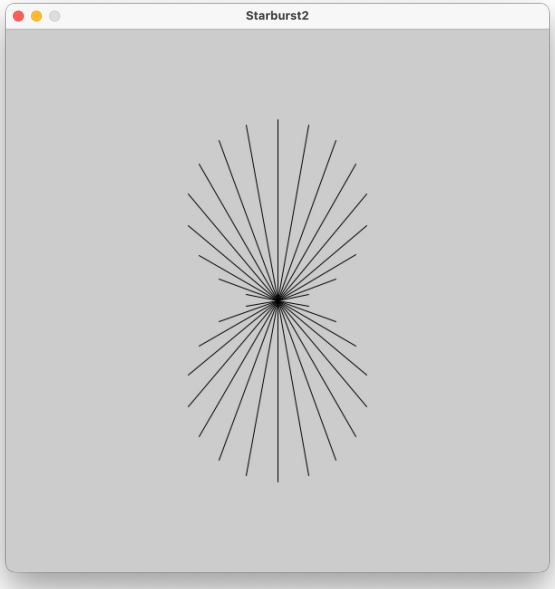
Smoothly varying a variable?

- Can use any smoothly changing function to modify a variable..
- E.g. if you want to make a variable oscillate, can use sin() or cos()



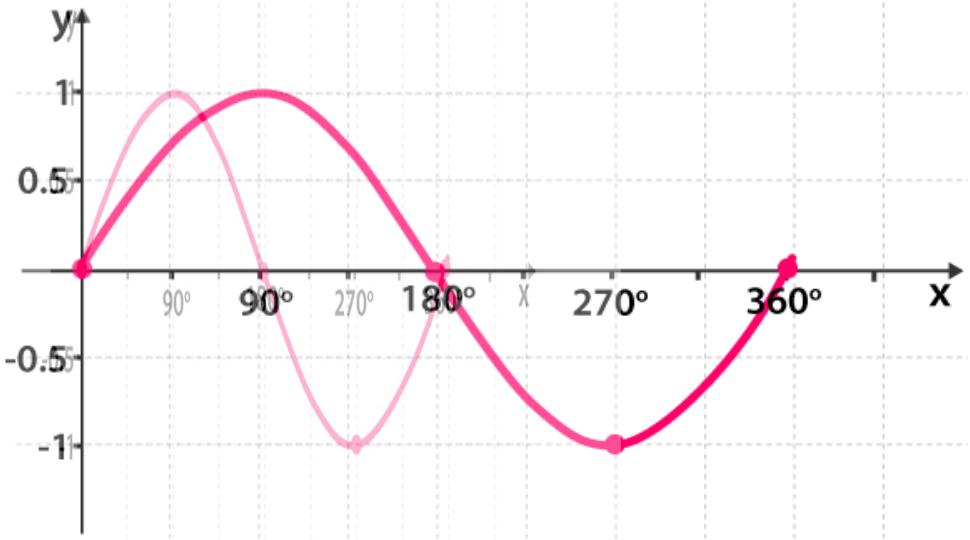
Like random(-1,1) they also vary between -1 and 1 (but in a regular way)

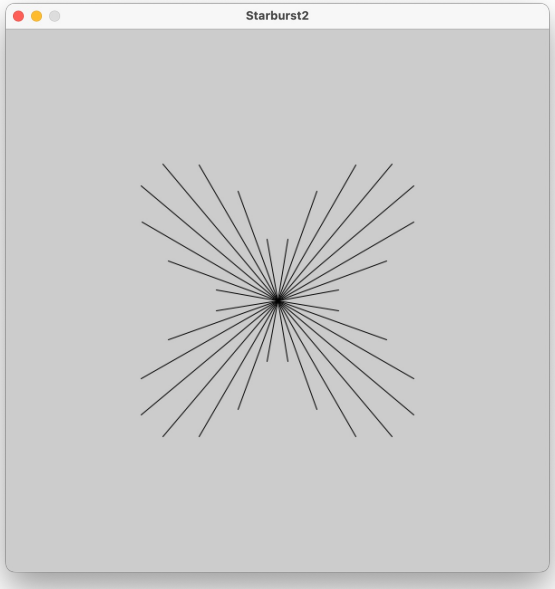





```
for (float theta=0; theta<360; theta++) {

    radius = DEFAULT_RADIUS * abs(sin(radians(theta)));

    if ( theta% 10 ==0 )
         line( centreX, centreY,
          centreX+radius*cos(radians(theta)),
          centreY+radius*sin(radians(theta)));
}
```

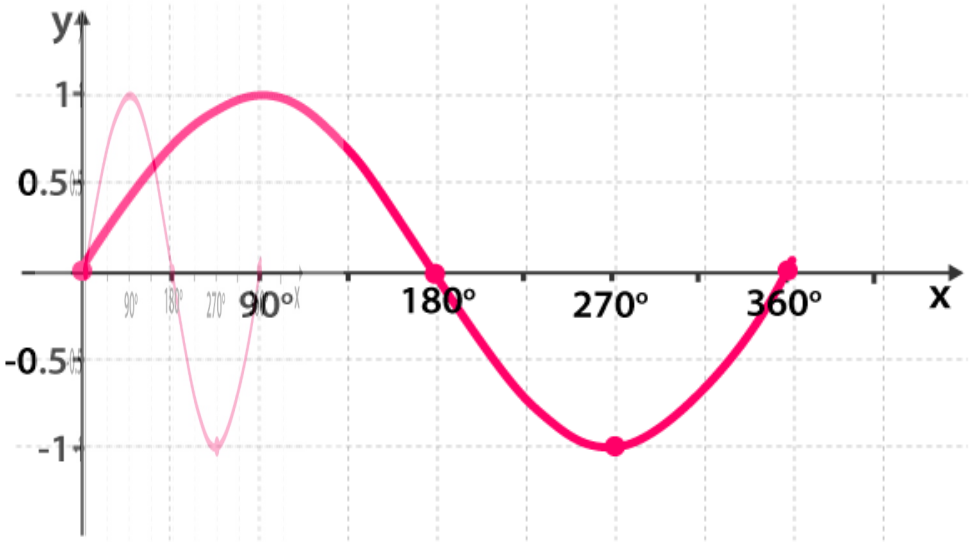
YORK UNIVERSITÉ UNIVERSITY

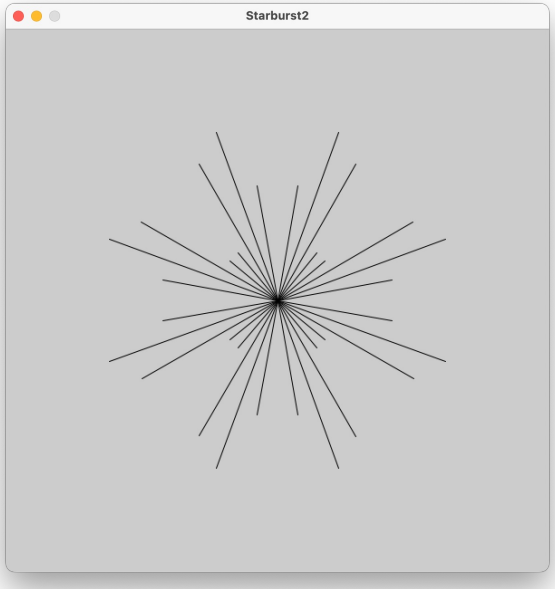





```
for (float theta=0; theta<360; theta++) {

    radius = DEFAULT_RADIUS * abs(sin(radians(2*theta)));

    if ( theta% 10 ==0 )
         line( centreX, centreY,
          centreX+radius*cos(radians(theta)),
          centreY+radius*sin(radians(theta)));
}
```

YORK UNIVERSITÉ UNIVERSITY



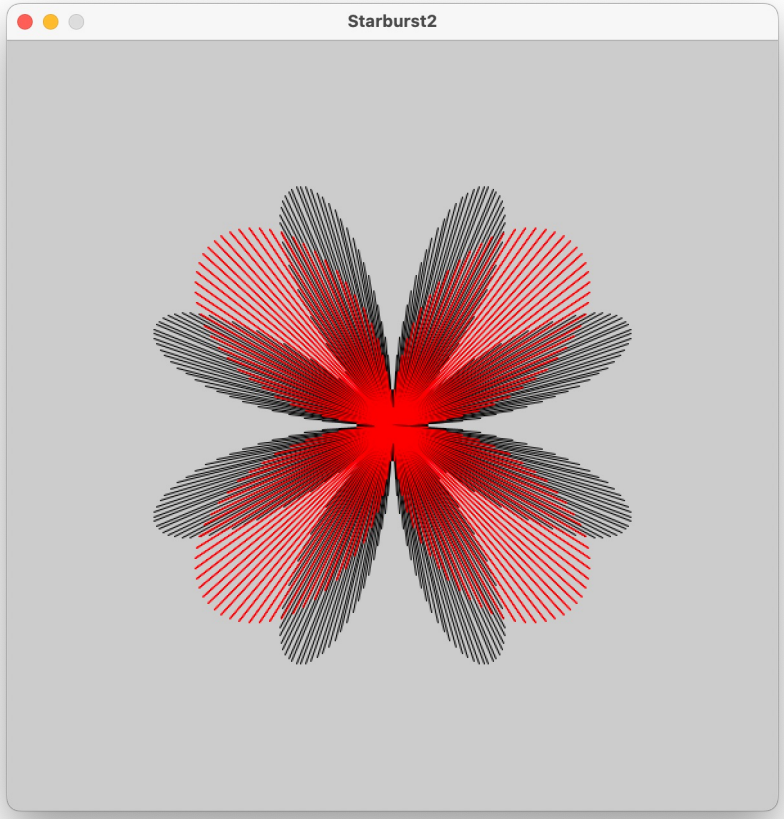



```
for (float theta=0; theta<360; theta++) {

    radius = DEFAULT_RADIUS * abs(sin(radians(4*theta)));

    if ( theta% 10 ==0 )
         line( centreX, centreY,
          centreX+radius*cos(radians(theta)),
          centreY+radius*sin(radians(theta)));
}
```

YORK UNIVERSITÉ UNIVERSITY

Experiment!

```
for (float theta=0; theta<360; theta++) {

    radius = DEFAULT_RADIUS * abs(sin(radians(4*theta)));

    line( centreX, centreY,
      centreX+radius*cos(radians(theta)),
      centreY+radius*sin(radians(theta)));

    stroke(255, 0, 0);
    line( centreX, centreY,
      centreX+radius*cos(radians(2*theta)),
      centreY+radius*sin(radians(2*theta)));
    stroke(0, 0, 0);
}
```



Ok, anything that varies can also then be called from draw() to animate

```
void draw() {
    background(255,255,255);
    starburst();
}
```

Nested loops

- Just as we can do with if/else (within if/else), we can put for loops inside other for loops

```
void setup() {

 for (int i=0; i<4; i++) {
   for (int j=0; j<5; j++) {

     print("(i=" + i + ",j=" + j + ") ");

   }
   println();
 }

}
```

Scope of i is both loops

Scope of j is inner loop

j varies faster than i

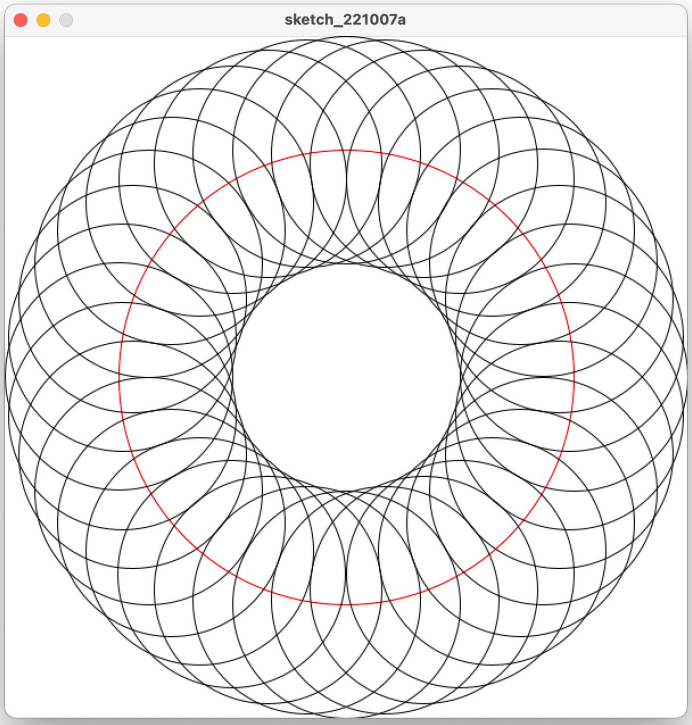
```
(i=0,j=0) (i=0,j=1) (i=0,j=2) (i=0,j=3) (i=0,j=4)
(i=1,j=0) (i=1,j=1) (i=1,j=2) (i=1,j=3) (i=1,j=4)
(i=2,j=0) (i=2,j=1) (i=2,j=2) (i=2,j=3) (i=2,j=4)
(i=3,j=0) (i=3,j=1) (i=3,j=2) (i=3,j=3) (i=3,j=4)
```

A more interesting example

- Outer loop:
  - Lets continue with iteration over 360 degrees (points on a circle for example
    - theta = 0 to 360
    - Don’t draw a line, lets draw a smaller circle centred on the main circle

- Inner loop:
    - Iterate over the inner circle, and draw smaller circles!

  (self-repeating pattern)

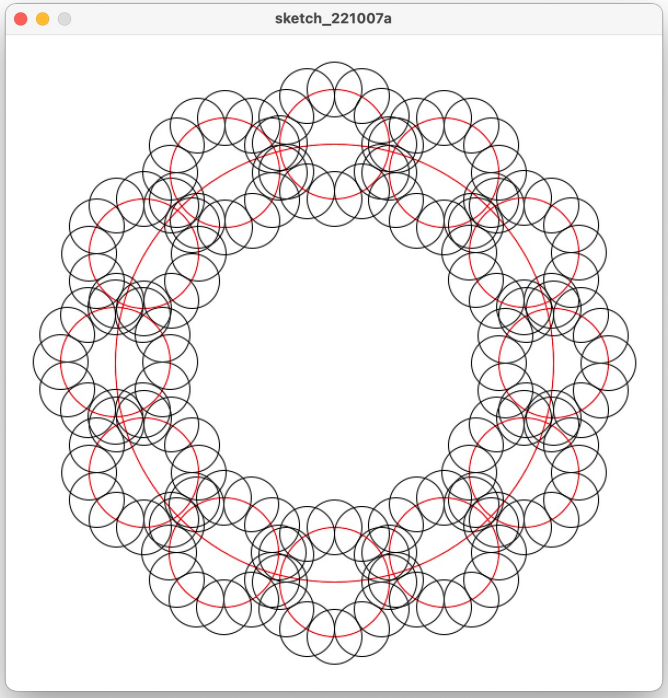



```
ellipseMode(RADIUS);
stroke(255,0,0);
circle(centreX,centreY,radius);
stroke(0,0,0);

for (float theta=0; theta<360; theta++) {

    if ( theta% 10 ==0 )
      circle( centreX+radius*cos(radians(theta)),
              centreY+radius*sin(radians(theta)),
              radius/2);


}
```

What if we want to vary the radius of these smaller circles?

YORK UNIVERSITÉ UNIVERSITY

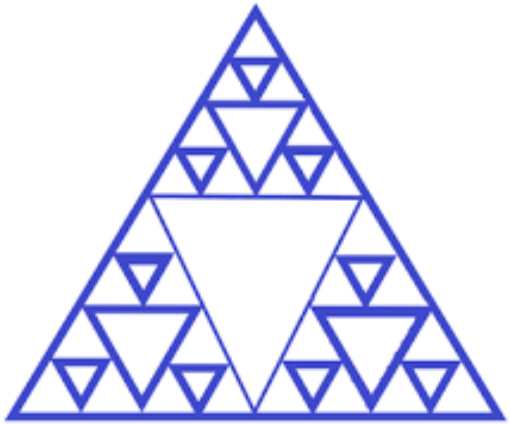
```
ellipseMode(RADIUS);
stroke(255, 0, 0);
circle(centreX, centreY, radius);
stroke(0, 0, 0);

for (float theta=0; theta<360; theta++) {

  if ( theta%30==0 ) {
    stroke(255, 0, 0);
    circle(centreX + radius*cos(radians(theta)),
           centreY + radius*sin(radians(theta)),
           radius/4 );
    stroke(0, 0, 0);
  }

  for (float theta2=0; theta2<360; theta2++) {

    if ( theta%30==0 && theta2%30==0 )
      circle( centreX + radius*cos(radians(theta))
                      + radius/4*cos(radians(theta2)),
              centreY + radius*sin(radians(theta))
                      + radius/4*sin(radians(theta2)),
              radius/8 );
    }
  }
}
```

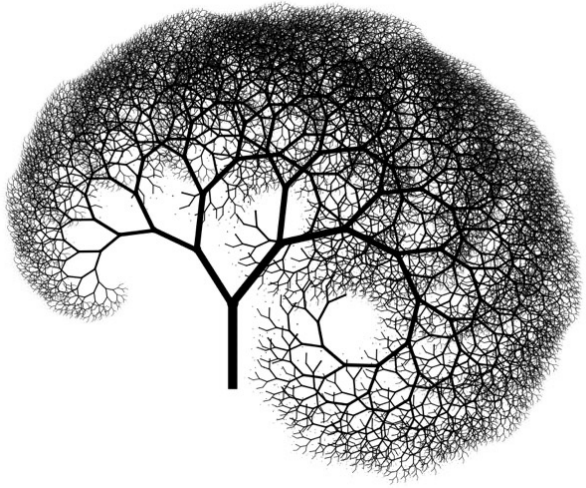


YORK UNIVERSITÉ UNIVERSITY

This is the basis of self-similarity!

- Fractals (repeating rule) – at each “level” of iteration repeat some pattern at a smaller scale
- Many patterns in nature are self-similar





YORK UNIVERSITÉ UNIVERSITY