EECS 1710
Programming for Digital Media

Lecture 9 :: Loops & Repetition

YORK UNIVERSITÉ UNIVERSITY

Midterm Reminder

- Again, Midterm on Wednesday Oct 5, 2022 (12.30pm-1:15pm)
- Please arrive early for a 12.30pm start
  - If you have accommodations but did not schedule test with the accommodation centre you will have to apply to sit at an alternative date
    - we cannot accommodate all of the variances in the normal lecture room/timeslot
    - come see me/contact me after today's lecture if you are in this category
- Test is 45 mins (not hard, but there is time pressure)
- 3 questions (each multi part)
    - Language basics + data types
    - Expressions
    - Methods + branching (simple IF/ELSE only; no SWITCH/CASE)
- Aim for ~10-12 mins per question
    - Gives you a bit more time to check over, or spend on questions that require more thinking
      - E.g. last part of q3 (complete a simple method that uses conditionals)
      - review conditional methods and simple if statements used in lecture examples +
      - try a couple of the coding bat questions (link in last lecture) to practice

Iteration/Repetition

- So far, we know that the draw() method automatically gets called over and over - so any code/statements we put inside this method, get repeated until we quit/stop our application

- Say we would like to execute the same block of code a fixed number of times (and control this ourselves)?
  - Copy & Paste (issues)
  - More efficient – execute a loop !!

Basic Loops

Iteration

- *Say we would like to execute the same code a fixed number of times (and control this ourselves) ?*
  - *Copy & Paste (issues)*
  - *More efficient – execute a loop !!*

  - Idea:
    - loop enters and repeats according to some condition
    - If the condition fails, it exits

- Two primary forms we will consider (initially)
  - FOR and WHILE

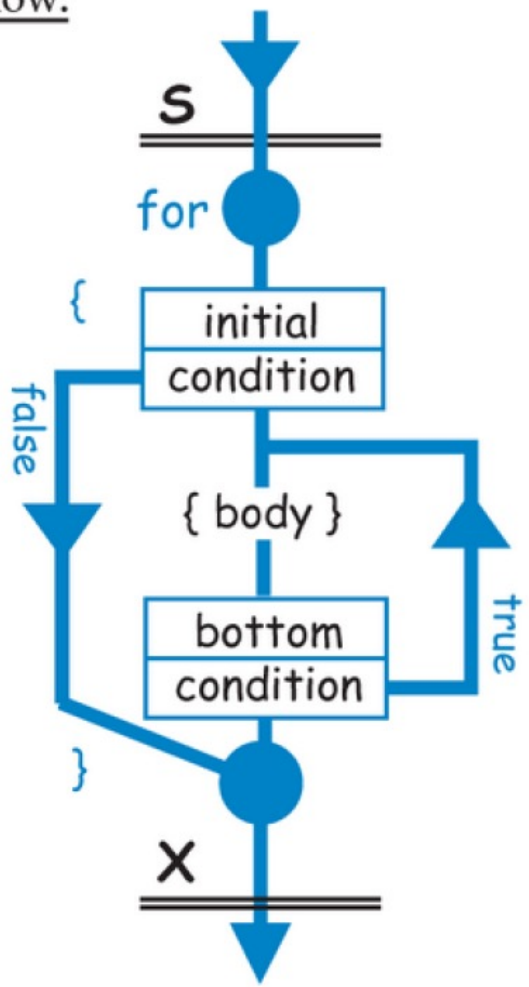
Looping control flow

- Fixed iteration:
  - FOR
    - N repetitions
- Indefinite iteration:
  - WHILE
    - 0 or more repetitions
  - DO WHILE
    - 1 or more repetitions

draw() is actually called from within an indefinite looping mechanism - however this is hidden from us in processing

The FOR statement

Flow:



Syntax:

```
Statement-S
for (initial; condition; bottom)
{
    body;
}
Statement-X
```

Algorithm:

1. Start the `for` scope
2. Execute `initial`
3. If `condition` is `false` go to 9
4. Start the body scope {
5. Execute the body
6. End the body scope }
7. Execute `bottom`
8. If `condition` is `true` go to 4
9. End the `for` scope

Example 1

```
final int MAX = 10;
final float SQUARE_ROOT = 0.5;

for (int i=0; i<MAX; i=i+1) {

    double sqrt = pow(i, SQUARE_ROOT);
    print(i);
    print("\t");
    println(sqrt);

}
```

What does this code fragment do?

for (initial; condition; bottom)

```
for (int i = 0; i < MAX; i = i + 1)
{
    ...

}
```

Can the initial statement be omitted? Try it and see!

```
int i;
for ( ; i < MAX; i = i + 1)
{
    ...

}
```

for (initial; **condition**; bottom)

- What about the condition?
  - Can it be omitted?
  - Can it be set to the literal true?
  - What if it were false at the beginning?
  - Is it monitored throughout the body?

Try it and see!

for (initial; condition; **bottom**)

- What about the bottom statement?
  - Can it be any statement?
  - Will the loop be infinite if it is omitted?

Try it and see!

Example 2

Write a fragment of code that outputs the exponents of all powers of 2 that are smaller than one million. The program needs to keep computing successive powers of 2, i.e. 2^0, 2^1, 2^2, … as long as each is less than a million.

Desired output:

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

```
final int MILLION = 1000000;

for (int expo = 0;  pow(2, expo)<MILLION; expo++ ) {
  print(expo + " ");
}
```

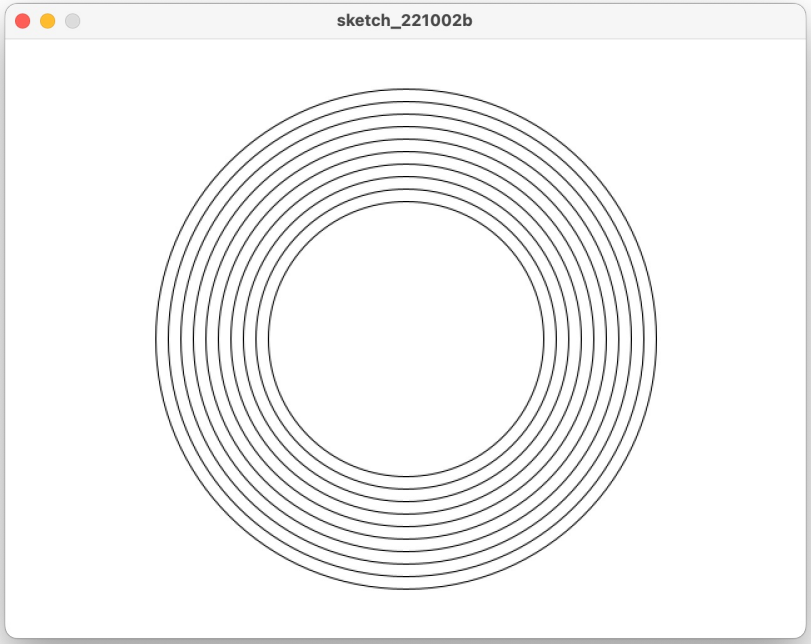
Example 3

- Modify the previous example to only output the exponent of the greatest power of 2 that is smaller than one million.

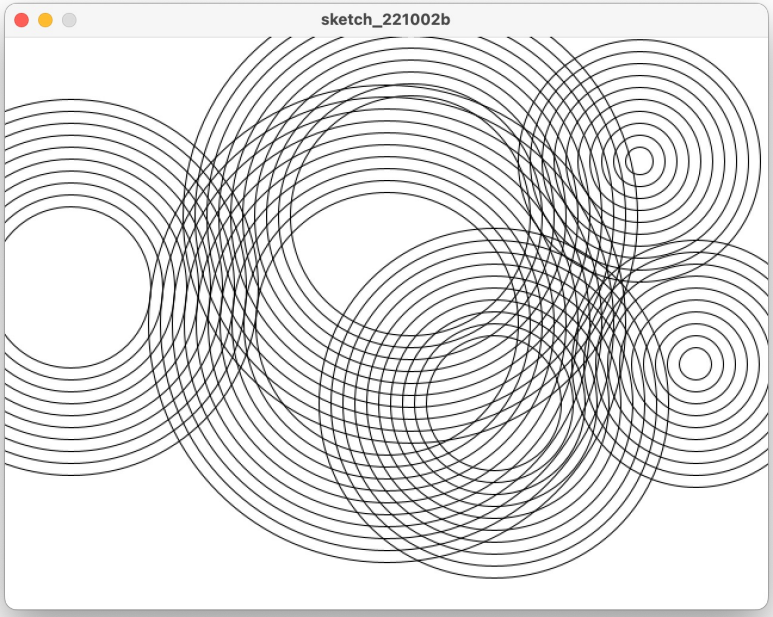
```
final int MILLION = 1000000;

int expo = 0;

for (  ;  pow(2, expo)<MILLION; expo++ ) {

}

// expo was incremented before the condition fails
// so the last value of expo was 1 before this
print(expo-1 + " ");
```

Example (drawing concentric rings)

```
int startX;
int startY;

void setup() {
  size(640, 480);
  startX = width/2;
  startY = height/2;

  final int NUM_ITER = 50;
  final int INIT_RADIUS = 200;

  ellipseMode(RADIUS);
  background(255, 255, 255);
  int radius = INIT_RADIUS;

  for (int i=0; i<NUM_ITER; i++) {
    circle(startX, startY, radius);
    radius = radius - 10;
  }
}

void draw() {
}
```



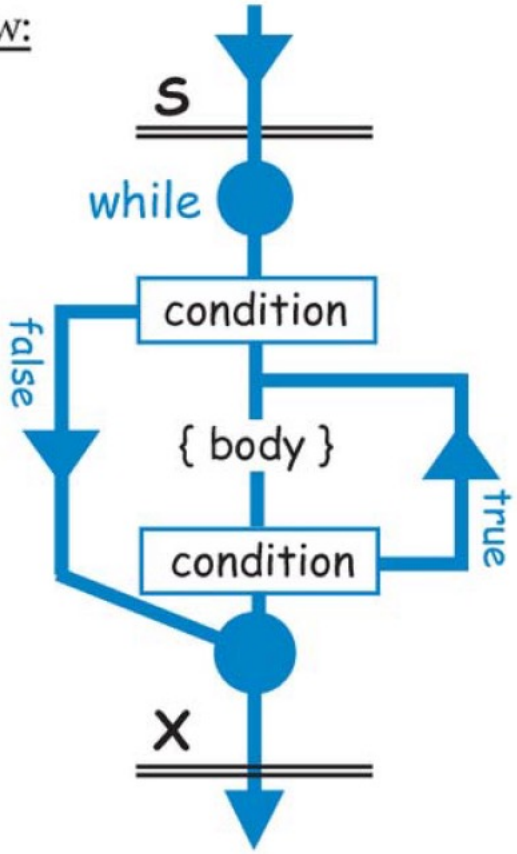
```
final int NUM_ITER = 10;
final int INIT_RADIUS = 200;
int startX;
int startY;

void setup() {
  size(640, 480);
  startX = width/2;
  startY = height/2;

  ellipseMode(RADIUS);
  background(255, 255, 255);
  noFill();
  drawRings(INIT_RADIUS);
}

void draw() {
}
```




```
void mousePressed() {
  startX = mouseX;
  startY = mouseY;
  drawRings(random(INIT_RADIUS/2,INIT_RADIUS));
}

void drawRings(float radius) {
  for (int i=0; i<NUM_ITER; i++) {
    circle(startX, startY, radius);
    radius = radius - 10;
  }
}
```

The WHILE statement

Flow:



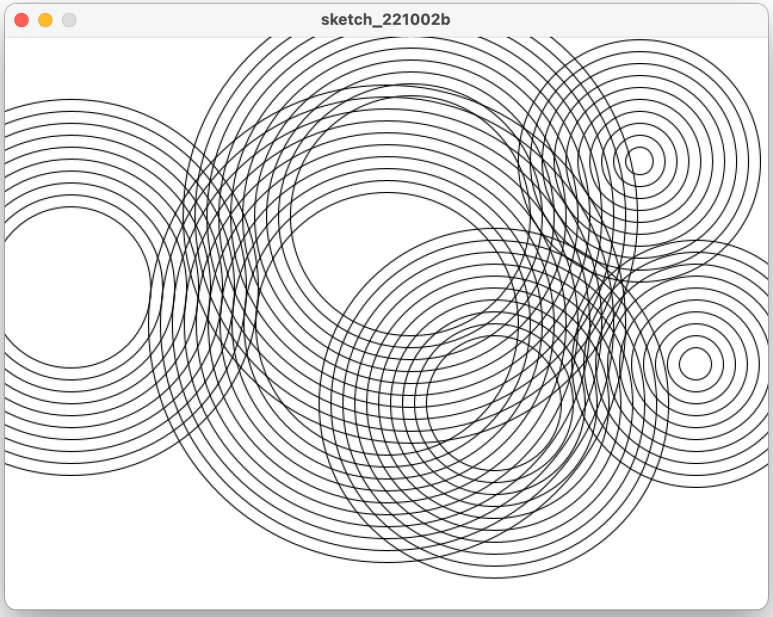
Syntax:

```
Statement-S
while(condition)
{
    body;
}
Statement-X
```

Algorithm:

1. If `condition` is `false` go to 6
2. Start the body scope {
3. Execute the body
4. End the body scope }
5. If `condition` is `true` go to 2
6. End the `while` scope

```
final int NUM_ITER = 10;
final int INIT_RADIUS = 200;
int startX;
int startY;

void setup() {
  size(640, 480);
  startX = width/2;
  startY = height/2;

  ellipseMode(RADIUS);
  background(255, 255, 255);
  noFill();
  drawRings(INIT_RADIUS);
}

void draw() {
}
```




```
void mousePressed() {
  startX = mouseX;
  startY = mouseY;
  drawRings(random(INIT_RADIUS/2,INIT_RADIUS));
}

void drawRings(float radius) {
  int i = 0;
  while (i<NUM_ITER) {
    circle(startX, startY, radius);
    radius = radius - 10;
    i++;
  }
}
```

While vs. For

- “for” loops tend to be used when we know we want to execute some code a fixed number of times
  - E.g. performing a calculation on a finite set of inputs (where we know the number of inputs to expect)
  - Traversing arrays (later)

- “while” loops tend to be used for situations where some code may/may not be executed, an arbitrary number of times
  - E.g. (friendly) validation of inputs – user asked until input entered successfully
  - Reading files (when we don’t know how big the file is)

More on Friday

- More examples
- Iteration & looping to generate some cool art patterns