2015
정보문화기술입문

Week 10

이차원 배열, 알고리즘

이 준 환 / 오 종 환
서울대학교 _ 연합전공 정보문화학

Assignment 5 리뷰

- Short presentation
  - 자기 작품을 화면에 띄우면 10-60초간 짧은 설명
  - 자기 이름 소개
  - 작품 설명
  - 다른 사람 작품에 질문이 있을 경우 바로 하기

지난 시간 리뷰

- Quiz 2 리뷰
- 배열
  - 변수의 배열
  - 클래스의 배열
  - 배열의 선언, 생성, 초기화
  - 배열의 확장
  - Lab: Snake
- Timer

오늘 다룰 내용

- 이차원 배열
- Algorithm의 작성

이차원 배열

이차원 배열

- 지금까지 복수의 정보를 순서대로 저장하기 위해서 일차원 데이터의 집합인 배열에 대해서 다루었다.
- 하지만 이미지나, 보드게임과 같은 데이터를 표현하기 위해서는 이차원 배열이 필요.
- 이차원 배열 → 배열들의 배열
  - ```
    int[] myArray = { 0, 1, 2, 3};
    int[][] myArray = { { 0, 1, 2, 3},
                        { 4, 5, 6, 7},
                        { 8, 9, 10, 11},
                        { 12, 13, 14, 15} };
    int[][] myArray = new int[rows][cols];
    ```
    ```

이차원 배열

- `int[][] myArray = new int[rows][cols];`

[0][0]	[0][1]	[0][2]	[0][3]	[0][4]	[0][5]	[0][6]	[0][7]	[0][8]	[0][9]
[1][0]	[1][1]	[1][2]	[1][3]	[1][4]	[1][5]	[1][6]	[1][7]	[1][8]	[1][9]
[2][0]	[2][1]	[2][2]	[2][3]	[2][4]	[2][5]	[2][6]	[2][7]	[2][8]	[2][9]
[3][0]	[3][1]	[3][2]	[3][3]	[3][4]	[3][5]	[3][6]	[3][7]	[3][8]	[3][9]
[4][0]	[4][1]	[4][2]	[4][3]	[4][4]	[4][5]	[4][6]	[4][7]	[4][8]	[4][9]

이차원 배열

```
for(int i = 0; i < myArray.length; i++) {
  for(int j = 0; j < myArray[0].length; j++) {
    print(myArray[i][j] + " ");
  }
  println();
}


# myArray.length → row length

# myArray[0].length → column length

# http://stackoverflow.com/questions/4000169/getting-the-array-length-of-a-2d-array-in-java
```

Lab: 이차원 배열

- 다음과 같은 그림을 이차원 배열을 이용해 그려보자.
  - `size(400, 400);`
  - `int pixelSize = 8;`
  - `int cols = width/pixelSize;`
  - `int rows = height/pixelSize;`
  - 각각의 셀은 `random()`함수로 만들어진 숫자 (0~255)를 가진다.

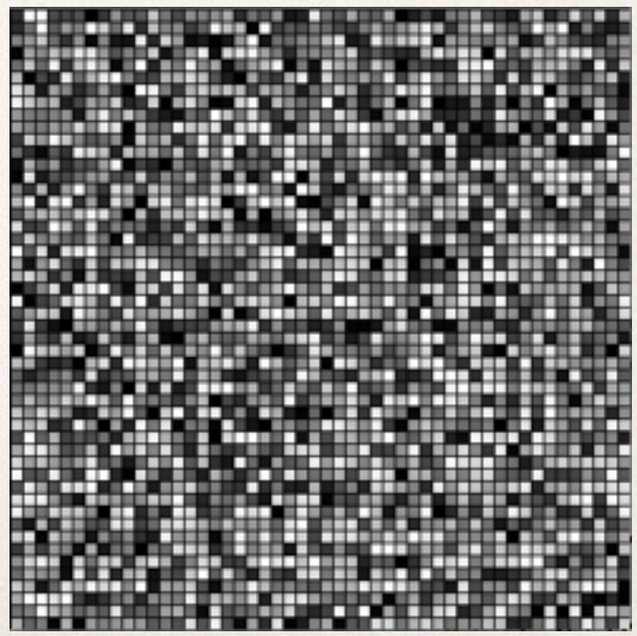
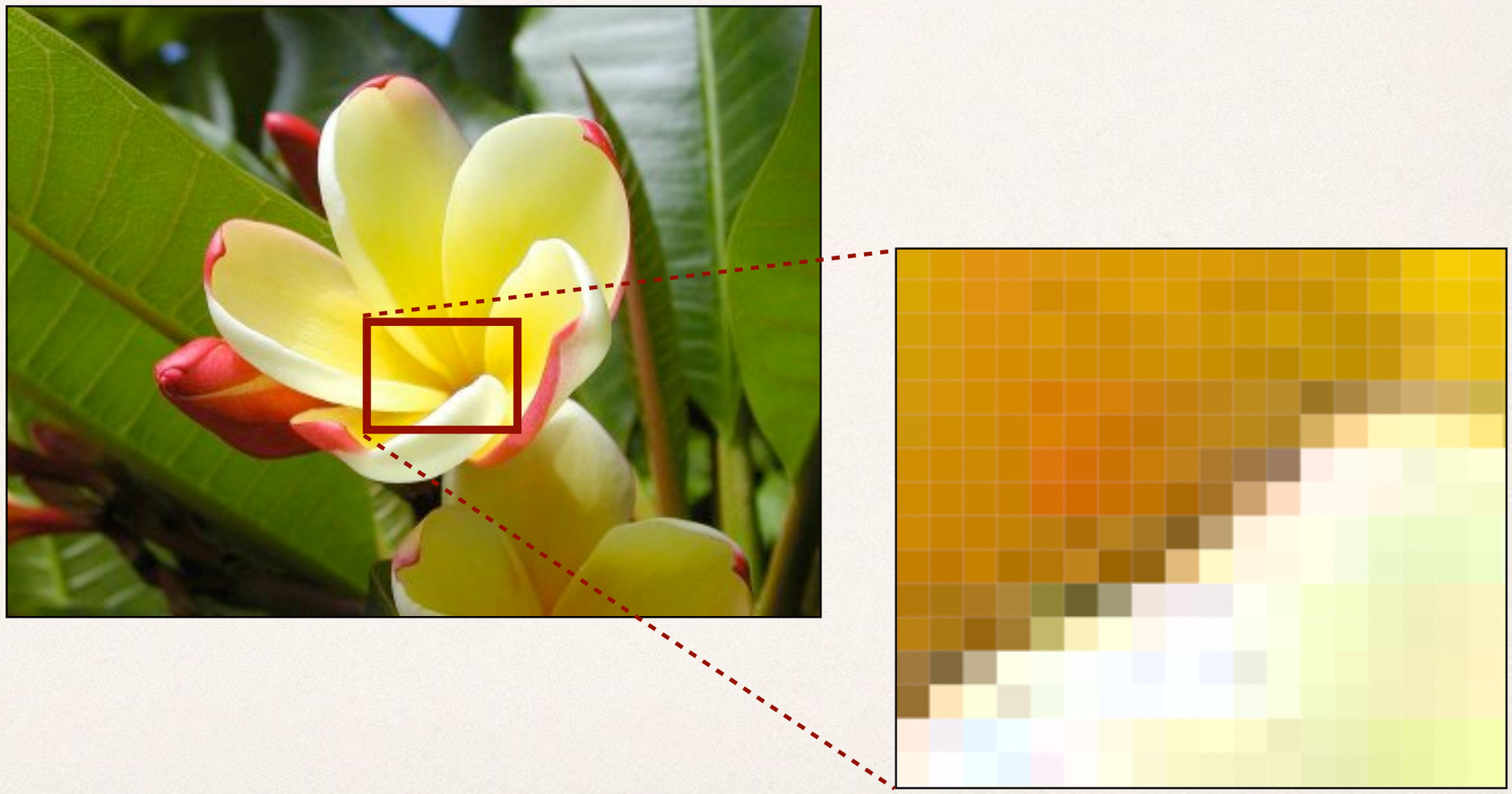
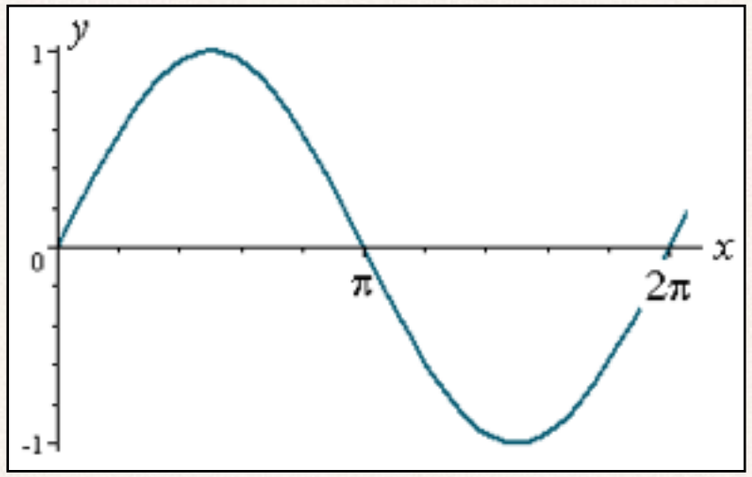


Image Array

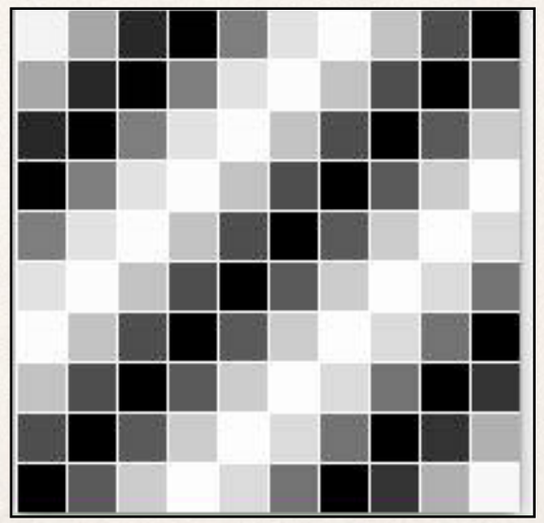


Lab: 객체의 이차원 배열

- Cell 클래스를 이용하여 다음과 같은 프로그램을 작성.
  - `fill(127+127*sin(angle));`
  - sin(angle)은 -1에서 1까지의 숫자를 만들어 줌.



- Cell을 이용하여 2차원 배열을 생성하고, angle 값을 바꾸어주면서 사각형을 그려보자.



이미지를 픽셀 단위로 다루기

- 이미지는 픽셀의 배열로 나타낼 수 있다.

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24

- 이차원에서 일차원 index로 변환: **(row, col)**

→ **row * width + col**

Lab: 이미지 돋보기 만들기

- PImage를 픽셀 단위로 다루는 순서와 방법

1 - 이미지의 color를 배열로 불러오기	img.loadPixels();
2 - 배열에 저장된 픽셀의 color 활용하기	img.pixels[**y*img.width + x**]
3 - 배열에 저장된 값을 이미지에 반영하기	img.updatePixels();

- mouseX, mouseY에 해당하는 위치에 원을 표시

이미지 프로세싱: 색상을 다루는 속성

- 루프(loop)를 활용하여 각 픽셀을 한 번씩 돌아가며 처리

- color 속성의 변수로 부터 밝기, 채도, R, G, B 등의 값을 가져올 수 있다.

  - `alpha(), blue(), brightness(), color(), green() ,hue() ,red(), saturation(), ...`

Lab: 점묘화 만들기

- 작은 이미지를 불러와서, n배 큰 창에 표시해보자.
- 이중루프(for문)를 사용하여 모든 픽셀을 돌며, 밝기 등의 속성을 불러와본다.
- 밝기 등의 속성에 따라 점의 크기를 달리해보자.
  - `ellipse(x, y, size, size)` 를 이용하여 점을 찍어 준다.
- mouseX, mouseY에 해당하는 위치에 원을 표시한다.

Lab: Tic-Tac-Toe 게임을 만들자

- Cell 객체를 이용하여 tic-tac-toe 게임을 만들어 보자.
- 각각의 cell은 state 를 가진다.
  - 0: nothing
  - 1: draw 0
    `ellipse(x, y, w, h)`를 이용하여 0을 그림.
  - 2: draw X
    `line(x, y, w, h)` 두개를 이용하여 X를 그림.
- 초기에는 이들 cell state를 0으로 지정
- 한번 클릭할 때 마다 state 를 0, 1, 2 로 변경.

Lab: Tic-Tac-Toe 게임의 확장 (선택)

- 현재 Tic-Tac-Toe 게임은 혼자서 두 플레이어 역할을 해야 해서 재미가 없다. (3개의 state가 클릭 횟수에 따라 보여짐)
- 마우스를 클릭했을 때 O 와 X 가 차례로 바뀌며 보여질 수 있도록 바뀌게 Lab 3 프로그램을 바꿔보자.
  - Player 1(O)와 Player 2(X)의 turn taking
- O 혹은 X, 둘중 하나의 플레이가 자동으로 이루어지게 프로그램을 고쳐보자.
  - 프로그램이 시작되고 사용자가 마우스를 클릭하면 O혹은 X가 표시된다 (random 선택)
  - 사용자가 클릭을 하면 비어있는 공간에 컴퓨터가 random하게 표시를 한다. (사용자가 O면 컴퓨터는 X, 혹은 그 반대로)
  - 누군가 이기면 게임이 끝났다고 알려주자.

Algorithm의 작성 과정

프로그램의 작성 과정

- 프로세싱의 다양한 문법 학습
- 프로세싱을 이용한 여러 예제 탐색
  - 주로 단일한 기능을 구현
- 프로그래밍 → 여러 기능이 복합적으로 작동
  - 각각의 기능들이 어떻게 유기적으로 결합하는지를 고민해야
  - 프로그래밍을 시작하기 전에 pseudo code를 작성
    → 객체지향의 개념을 적극 활용

- **알고리즘의 작성**

알고리즘

- 문제를 해결하기 위한 절차나 공식
- 컴퓨터라는 지능을 가지고 있지 못한 기계에게 일의 순서를 알려주는 것
  - 모든 컴퓨터는 내부에 CPU를 가지고 있는데, 이 CPU는 다양한 명령어 집합(instruction sets)을 가지고 있다
  - 프로그래밍은 각각의 명령어 집합에 순차적으로 명령(instruction)을 내리는 과정
- 즉, 컴퓨터가 어떠한 일을 어떻게 처리해야 할지를 알려주는 방법

Ciao Espresso~

Ciao Espresso~

- Short Instruction
  - 그라인더에 간 커피를 portafilter에 넣고 꾹꾹 잘 누른 다음 커피머신에 돌려서 끼우고 버튼만 누르면 됨.

Ciao Espresso~

- portafilter에 뜨거운 물을 내려부어 따뜻하게 한다
- 커피를 portafilter에 담는다
- portafilter를 커피머신에 돌려 끼운다
- 에스프레소 잔을 올려놓고 버튼을 누른다

Ciao Espresso~

- 물이 없다면
  - 물탱크에 물을 담는다
- 물이 있다면
  - portafilter에 뜨거운 물을 내려부어 따뜻하게 한다
- 커피가 없다면
  - 커피를 그라인드로 간다
- 커피가 있다면
  - 커피를 portafilter에 담는다
- portafilter를 커피머신에 돌려 끼운다

Ciao Espresso~

- 한잔만 만들 경우
  - 에스프레소 잔을 올려놓고 버튼을 누른다
- 한잔 이상일 경우
  - 에스프레소 잔을 올려놓고 버튼을 누른다
  - 반복한다
- 아메리카노일 경우
  - 큰 컵에 뜨거운 물을 붓는다
  - 에스프레소 샷을 담는다
- 아이스 아메리카노일 경우
  - 큰 컵에 얼음을 넣는다
  - 찬 물을 붓는다
  - 에스프레소 샷을 담는다

절차를 만드는 과정

1. 아이디어 - 하나의 아이디어에서 시작
2. 부분 - 아이디어를 작은 부분으로 나눈다
    a. 알고리즘 pseudo code - 각 부분에서의 pseudo code 를 만들어 본다
    b. 알고리즘 코드 - pseudo code 를 실제 코드로 만든다
    c. 객체 - 알고리즘과 관련되게 데이터와 함수를 취하고, 클래스로 만든다
3. 종합 - 두번째 단계에서 반든 클래스를 모아 큰 알고리즘으로 통합

아이디어를 단계별로 구성

- Raindrops Game
  - 빗방울이 땅에 닿기 전에 빗방울을 잡는 게임
  - 하나의 빗방울도 땅에 (화면의 바닥에) 닿지 못하도록 마우스를 재빨리 움직이며 빗방울을 잡아야 한다
  - 가끔 새 빗방울이 화면에서 무작위적인 위치와 속도록 떨어진다

→ **떨어지는 빗방울 부분 (raindrops)**

: 빗방울을 가끔씩 떨어뜨리기 위해서는 시간을 조절해야 함

→ **빗방울을 잡는 부분 (catcher)**

: '빗방울을 잡았다'는 조건은?

아이디어를 단계별로 구성

- Raindrops Game
  - Part 1: 마우스에 의해 움직이는 원을 프로그래밍. 해당 원은 빗방울을 잡기 위한 catcher가 된다.
  - Part 2: 두개의 원(catcher와 raindrops)이 교차하는지 시험해 보기 위한 프로그램 작성 → 사용자가 빗방울을 잡는 순간 결정.
  - Part 3: n초마다 작동하는 시간함수를 실행하는 프로그램 작성.
  - Part 4: 화면의 위에서 바닥까지 떨어지는 원에 대한 프로그램 작성. (원: raindrop)

Part 1: Catcher의 작성

- Pseudo Code:
  - 배경을 지운다
  - 마우스 위치에 원을 그린다
- Real Code:

```
void setup() {
  size(400, 400);
  smooth();
}
void draw() {
  background(255);
  stroke(0);
  fill(175);
  ellipse(mouseX, mouseY, 64, 64);
}
```

Part 1: Catcher의 작성

- 전체 프로그램에서 Part 1의 해당 코드가 Catcher 객체로 작동하게 하기 위해서 다음과 같이 Catcher 클래스 작성.
  - `setup()`
    - Catcher 객체를 초기화
  - `draw()`
    - 배경을 지운다
    - Catcher의 위치를 마우스 위치로 삼는다 (`setLocation`)
    - Catcher를 그린다 (`display`)

Part 2: 교차(intersection) 알고리즘의 구현

- 교차(intersection) 알고리즘을 구현하기 위해 다음과 같은 클래스를 만들어 보자

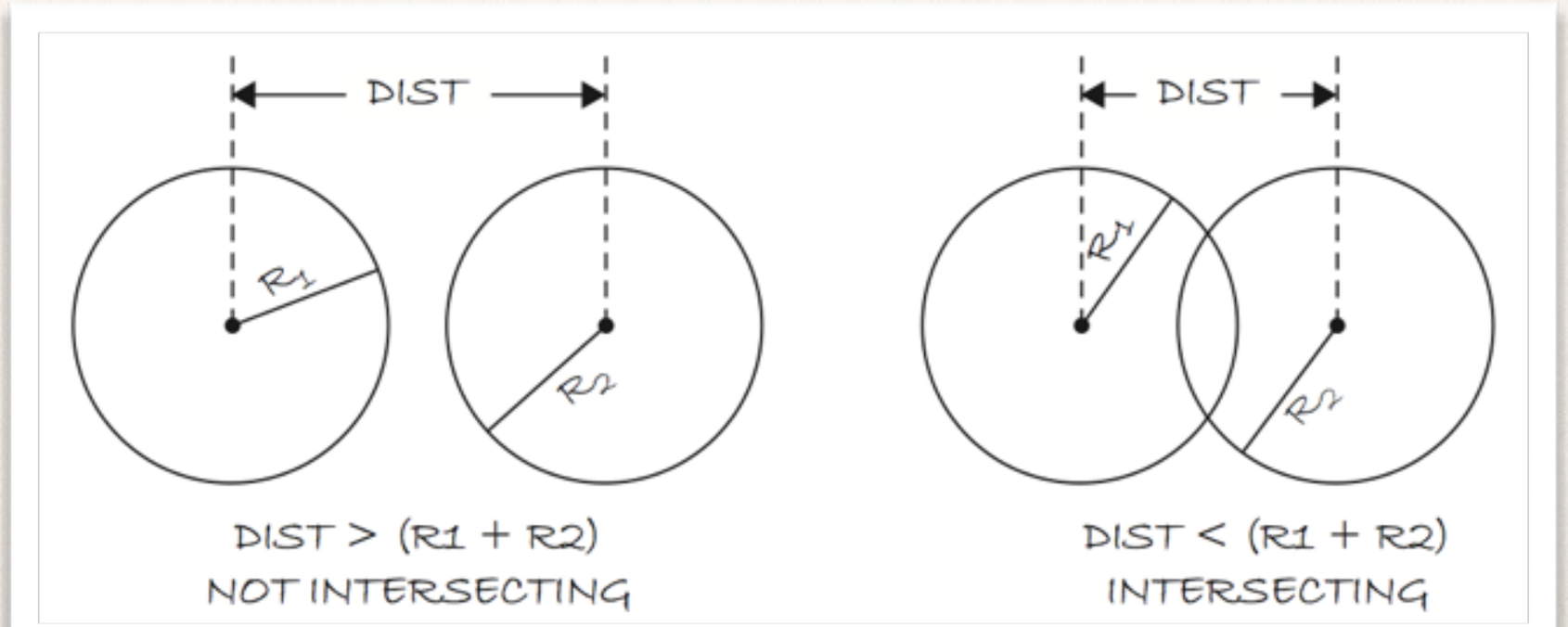
- Main Program
  - `setup()`
    - 두개의 공 객체를 만든다.
  - `draw()`
    - 공들을 움직인다. (`move`)
    - 만약 공1이 공2와 교차하면 각 공의 색이 하얗게 변하고, 그렇지 않다면 회색으로 그대로 있는다. (`intersect`)
    - 공들을 그린다. (`display`)

Part 2: 교차(intersection) 알고리즘의 구현

- Ball 클래스 참조
  - `data`
    - (공의) x와 y의 위치
    - (공의) 반지름
    - x, y 방향으로의 속도 (위아래로만 움직이지 않고 모든 방향으로 움직임)
  - `functions`
    - 생성자
      - 오브젝트 생성 시 주어진 패러미터에 근거해 반지름 설정.
      - 임의의 위치와 속도를 설정.
    - move
      - x를 x 방향의 가속도에 따라 증가시킨다.
      - y를 y 방향의 가속도에 따라 증가시킨다.
      - 공이 화면의 가장자리에 부딪히면 방향을 바꾼다.
    - display
      - 원을 x와 y 위치에 그린다.

Part 2: 교차(intersection) 알고리즘의 구현

- 교차(intersection) 알고리즘의 구현
  - processing은 `dist()`라는, 두 점의 거리를 계산할 수 있는 함수를 제공
  - `not intersecting: dist > (r1 + r2)`
  - `intersecting: dist < (r1 + r2)`



Part 2: 교차(intersection) 알고리즘의 구현

- 교차(intersection) 알고리즘의 구현
  - processing은 `dist()`라는, 두 점의 거리를 계산할 수 있는 함수를 제공
  - `not intersecting: dist > (r1 + r2)`
  - `intersecting: dist < (r1 + r2)`

```
boolean intersect(float x1, float y1, float
x2, float y2, float r1, float r2) {
  float distance = dist(x1,y1,x2,y2);
  if (distance < r1 + r2) {
    return true;
  } else {
    return false;
  }
}
```

Part 2: 교차(intersection) 알고리즘의 구현

- intersect 함수를 추가하고 draw 함수에 다음의 코드를 넣어보자.

```
boolean intersecting = intersect(ball1.x,
ball1.y, ball2.x, ball2.y, ball1.r, ball2.r);

if (intersecting) {
  println("The circles are intersecting!");
}
```

교차 함수를 Ball 클래스 내부로 이동

- intersect 함수는 Ball이 서로 겹치는지 여부를 판별하는 함수 → 논리적으로 Ball 클래스 내부에 있는 것이 좋다.
- 다음과 같이 사용할 수 있도록 Ball 클래스를 수정하고 메인프로그램도 수정
  - **`intersecting = ball1.intersect(ball2);`**

교차하는 순간에 공의 색 바꾸기

- 현재는 공이 교차하는 순간에 메시지를 보여준다.
- 메시지를 보여주는 대신에, 공의 색이 변화하게 하자.
- 전략:
  - 공의 속성(데이터)를 추가해 주자: Color 속성
  - `highlight()` 를 추가하여 공의 color 속성을 바꾸어 준다.
  - main program에서 intersecting 되는 순간에 메시지 출력 대신 `highlight()` 를 호출

Part 3: Timer 만들기

- 지난 시간에 만든 클래스

Part 4: Raindrops

- 지금까지 만든 것들...
  - Part 1: Catcher
  - Part 2: Intersection
  - Part 3: Timer
- 빗방울이 내리는 모습을 만들어야 함
  - Part 4: Raindrops
    - 하나의 빗방울이 떨어지는 것 부터 만든 후, 여러 빗방울이 떨어지는 모습으로 발전시켜 보자

Simple Raindrop

- 하늘에서 원(빗방울)이 떨어지는 모습을 만들어 보자.
  - 빗방울의 y값을 증가시킨다.
  - 빗방울을 그린다.

Simple Raindrop

```
float x, y;
void setup() {
  size(400, 400);
  x = random(width);
  y = 0;
}
void draw() {
  background(255);
  // draw raindrop
  fill(50, 100, 150);
  noStroke();
  ellipse(x, y, 16, 16);
  y++;
}
```

Raindrop 클래스

- 앞의 코드를 클래스로 만들고 main program을 변경하자.
- `data`
  - 빗방울 위치 (`x, y`)
  - 빗방울 속도 (`speed`)
  - 빗방울 칼라 (`c`)
  - 빗방울 크기 (`r`) - 반지름
- `functions`
  - 빗방울을 움직인다 (`move`)
  - 빗방울 그리기 (`display`)

Raindrop 클래스

- 앞의 코드를 클래스로 만들고 main program을 변경하자.

```
Raindrop drop;

void setup() {
  size(400, 400);
  ____________________________
}

void draw() {
  background(255);
  // move & draw raindrop
  ____________________________
  ____________________________
}
```

여러개의 빗방울 떨어뜨리기

- Raindrop 클래스를 이용하여 빗방울이 여러개 떨어지도록 바꾸어 보자.

- Array의 사용
- `Raindrop[] drops = new Raindrop[50];`

한 번에 빗방울 하나씩 떨어뜨리기

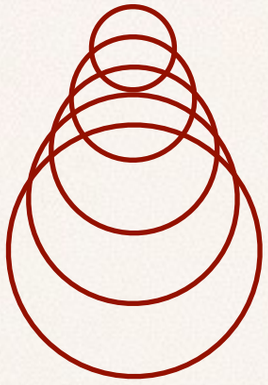
- 문제점:
  - 빗방울이 모두 한꺼번에 떨어져서 재미가 없다.
  - 한번에 빗방울이 하나씩 떨어지도록 바꾸어 보자.
- 아이디어
  - setup에서 모든 빗방울을 초기화 하지 않고, draw 함수가 반복적으로 실행될 때 한번에 하나씩 빗방울을 초기화 하자.
    → 현재까지 화면에 표시된 빗방울 수를 카운트 하는 변수가 필요 (`totalDrops`)
    → `totalDrops`를 draw가 호출될 때 마다 계속 증가
    → `totalDrops`가 전체 array 크기보다 크면 다시 0으로 초기화

한 번에 빗방울 하나씩 떨어뜨리기

- Pseudo Code
  - `setup()`
    - 1000개를 담을 수 있는 Raindrop 배열을 만든다.
    - 현재까지 화면에 표시된 모든 빗방울의 숫자를 담을 수 있는 변수를 만들고 0으로 초기화 한다 (`totalDrops`)
  - `draw()`
    - 하나의 빗방울을 초기하여 Raindrop 배열에 담는다.
    - `totalDrops` 를 증가시킨다. (배열에 담는 빗방울의 인덱스가 0, 1, 2, 3 등으로 증가)
    - `totalDrops` 가 전체 배열크기보다 크거나 같으면 0으로 초기화 한다. (빗방울을 다시 0부터 담기 시작)
    - 배열에 담긴 빗방울을 모두 그린다. (0 부터 `totalDrops` 까지)

한 번에 빗방울 하나씩 떨어뜨리기

- 빗방울 예쁘게 그리기
  - 원의 모습 → 빗방울의 모습으로
- 아이디어



for loop 를 이용하여 y 축으로 원의 중심을 이동하며 크기를 확대

Raindrop 클래스 `display()` 에 다음과 같이 추가

```
for (int i = 2; i < r; i++) {
  fill(c, 30 + i*30);
  ellipse(x, y + i*4, i*2, i*2);
}
```

Last Step: 통합하기

- 지금까지 최종 프로그램을 위한 여러 조각들이 완성.
- 마지막 단계는 이들 조각 프로그램을 합쳐서 하나의 통합된 완성품을 만드는 것!
- 먼저, 그동안 만든 클래스들을 모두 불러오자.
  - 개별 클래스는 지금까지 만들었던 조각 프로그램들에서 완벽하게 작동했기 때문에 수정할 필요는 없음.

Last Step: 통합하기

- Pseudo Code
- `setup()`
  - Catcher 객체를 만든다.
  - 빗방울 배열을 만든다.
  - `totalDrops`을 0으로 초기화.
  - 타이머 객체를 만든다.
  - 타이머를 시작한다.

Last Step: 통합하기

- Pseudo Code
- `draw()`
  - catcher 객체의 위치를 마우스 위치로 지정.
  - catcher를 그린다.
  - totalDrops 범위 내에 있는 모든 빗방울을 움직인다.
  - totalDrops 범위 내에 있는 모든 빗방울을 그린다.
  - catcher가 어떤 빗방울과 교차하면
    - 그 빗방울을 지운다.
  - 타이머가 끝났을 때
    - 빗방울을 하나 새로 생성한다
    - 타이머를 다시 시작한다.

Last Step: 통합하기

- Part 1 ~ Part 4 까지 성공적으로 통합이 이루어짐...
- Part 2에서는 `intersect()` 함수를 만들었음.

```
boolean intersecting = ball1.intersect(ball2);
if (intersecting) {
  // do something
}
```

Last Step: 통합하기

- 최종 프로그램에서는 catcher와 raindrop이 서로 `intersect`하여야 함.

```
if (catcher.intersect(drops[i])) {
  // erase drops[i]
}
```

- Catcher 클래스에 다음의 함수가 추가되어야 함.
  - `intersect()` : Ball 클래스로부터 카피
- Raindrop 클래스에 다음의 함수가 추가되어야 함.
  - `caught()` : 화면에서 사라지게 하기 위해 다음과 같이 설정.
    - `speed : 0`
    - `y : -1000`

Quiz 3

Quiz 3 공지

- 2015-11-16 수업 시작시
- 내용
  - 오늘 배운 내용까지
  - 코드의 오류 여부 판단, 코드의 빈 칸 채우기, 코드를 보고 출력되는 결과 쓰기 등
- Socrative 웹/앱 이용

Questions?