03
변수와 연산

03.01
변수와 데이터 타입

변수의 필요성

지금까지 여러 도형을 그렸습니다. 도형을 정의하는데 위치나 크기값이 필요하다는 것을 알았습니다. 어떤 때는 숫자를 바로 입력하고 어떤 때는 예를 들어 mouseX나 mouseY를 사용할 때 처럼 숫자가 아닌 어떤 이름을 사용할 때가 있었습니다. mouseX와 같은 이런 이름을 변수(variable)이라고 부르며 앞으로 가능하면 자주 사용하려고 합니다.

변수는 어떤 정보를 담을 수 있는 그릇과 같습니다. 변수(variable)이라는 말이 암시하듯 그릇 속의 내용물은 달라질 수 있습니다. 어떤 정보를 담고 있든 변수의 이름을 사용하면 변수가 담고 있는 정보를 이용할 수 있습니다.

만약 변수를 사용하지 않고 모든 입력을 숫자로 한다면 어떨까요? 규모가 작은 프로그램은 큰 상관이 없지만 규모가 크거나 값이 빈번하게 바뀌는 경우 매번 숫자를 고쳐서 입력하는 것이 여간 곤혹스러운 일이 아닐 것입니다. 한 번 사용하는 프로그램이 아니라 여러번 사용할 프로그램을 만들고 싶다면 변수의 사용은 자연스러운 일입니다.

데이터 타입의 필요성

아직까지 프로그래밍 언어는 사람보다는 컴퓨터에 친절합니다. 프로그래밍 언어를 이해하려면 컴퓨터가 어떻게 동작하는지 기초적인 지식이 필요합니다. 컴퓨터를 정말 단순하게 보면 긴 테이프를 읽고 테이프에 적힌 정보를 마찬가지로 테이프에 적혀있는 방법대로 처리하는 기계와 같습니다. 조금 더 빠르고(아닙니다. 아주 빠릅니다.) 조금 더 작을 뿐 동작하는 기본 아이디어는 동일합니다.

이런 기계는 2진법을 사용해 정보를 기록합니다. 0과 1을 이용해 정보를 기록하는 것은 전등의 스위치를 끄거나(0) 켜는 것(1)에 대응됩니다. 왜 2진법을 배우는지 궁금하면 컴퓨터가 어떻게 숫자를 이해할 수 있을지 생각해보세요.

그런데 말입니다 1 0 0 0 0 1 0과 같이 2진수로 적힌 데이터는 여러가지 방법으로 해석될 수 있습니다. 하나씩 끊어 읽는다면 true false false false false true false처럼 읽을 수 있습니다. 스위치가 켜지면

true, 꺼지면 false인 것이지요.

다른 방식으로 읽을 수 있습니다. 1 0 0 0 0 1 0은 정수 66(1 x 2^6 + 1 x 2^1 = 66)을 말할지도 모릅니다. 혹은 문자 'B'를 의미할 수도 있습니다. ASCII 코드 테이블의 66번째 문자가 'B'입니다.

결론을 말씀드리면 변수를 사용할 때는 사용자가 변수의 타입을 정해야 합니다. 타입이란 말하자면 어떤 정보를 담고있는지 알려주는 표지(marker)입니다. 만약 타입이 없다면 담겨있는 정보를 어떻게 읽어야 할지 알 수 없는 경우가 생깁니다. 혹은 다른 타입으로 저장되어 있는 정보를 잘못된 타입으로 읽는 실수를 범할 수 있습니다.

프로그래밍 언어에 따라 이런 타입을 엄격하게 지키는 경우가 있고 느슨하게 프로그래밍 언어가 '알아서' 짐작하는 경우도 있습니다. 프로세싱은 엄격하게 타입을 지키려는 프로그래밍 언어입니다.

변수 사용하기

변수를 사용할 때는 두가지 단계를 거칩니다.

변수의 선언: 변수의 타입과 이름을 정합니다. 변수의 타입을 int 즉 정수로 정했습니다. 저장공간에 타입에 맞는 크기의 메모리가 확보됩니다.

```
int myVariable;
```

변수의 초기화: 실제 값을 저장합니다.

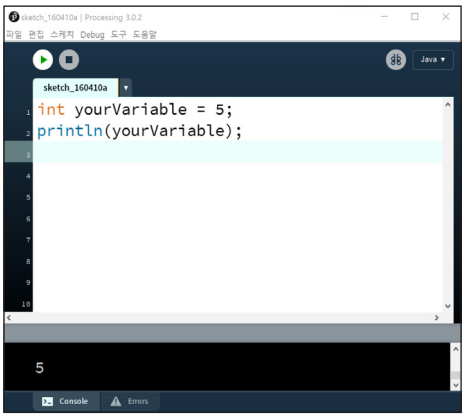
```
myVariable = 5;
```

두가지 단계를 한꺼번에 할 수 있습니다. 변수를 선언하면서 값을 저장합니다. 값을 콘솔화면에 확인해 보겠습니다. println() 함수를 사용합니다.

```
int yourVariable = 5;
println(yourVariable);
```

저장한 값은 필요에 따라 바꿀 수 있습니다.

```
int yourVariable = 5;
println(yourVariable);
```



println

```
yourVariable = 10;
println(yourVariable);
```

지금부터는 프로세싱에서 자주 볼 수 있는 몇가지 변수의 타입에 대해 배워보겠습니다.

논리값

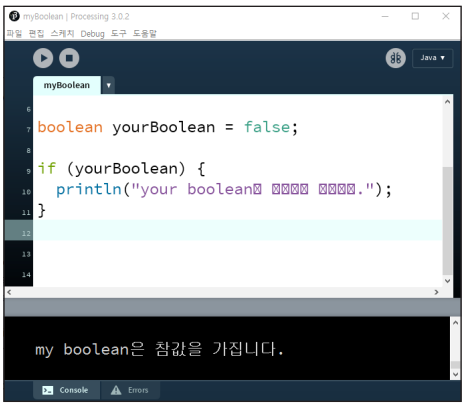
George Bool이라는 논리학자의 이름을 딴 데이터 타입입니다. boolean 타입은 true와 false값 두 개 중 하나의 값을 가집니다.

```
boolean myBoolean = true;
```

boolean 타입은 조건문에 자주 사용됩니다. if 다음의 () 사이에 true나 false값을 가지는 구문이 위치합니다. 만약 구문이 true라면 if() {...} 사이의 코드를 실행하고 false값을 가지면 {...} 을 건너뛰며 실행하지 않습니다.

```
if (myBoolean) {
    println("my boolean은 참값을 가집니다.");
}

boolean yourBoolean = false;

if (yourBoolean) {
    println("your boolean은 거짓값을 가집니다.");
}
```

위의 코드를 실행하면 yourBoolean 관련 문자열은 출력되지 않습니다.



myBoolean

정수

-2 -1 0 1 2와 같이 똑 떨어지는 숫자를 정수하고 합니다. 영어 integer의 처음 세 자를 따서 int라고 부릅니다.

```
int myInteger = 30;

if (myInteger < 40) {
    println("myInteger는 40보다 작습니다.");
}
```

```
myInteger = 50;
if (myInteger < 40) {
    println("myInteger는 40보다 작습니
다.");
}
```

부동소수점

3.14와 같은 수를 표현하기에 정수는 부족한 부분이 있습니다. 실수를 표현하기 위해 프로세싱은 부동소수점을 준비했습니다.

한자로 된 용어가 잘못된 뜻을 유도하는 경우 중의 하나입니다. 부동이라는 말은 '움직이지 않는다'의 不動이 아니라 '떠서 움직인다'의 浮動입니다. 영어로 하면 뜻이 분명해지는데 부동소수점 수를 영어로는 floating point number라고 칭합니다.

floating point의 float을 타입 이름으로 사용합니다.

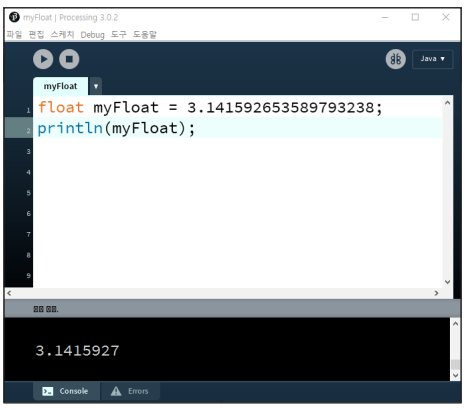
부동소수점 수는 수를 정수와 다르게 표현합니다. 수를 부호와 가수(mantissa) 그리고 지수(exponent)로 분리해서 다룹니다. 예를 들어 11.625는 + 1.1625 x 10^1 와 같은 형식으로 이해됩니다. 116.25는 어떻게 표현될까요? + 1.1625 x 10^2로 가수부위는 동일하고 지수만 다르게 표현됩니다. 지수의 크기에 따라 가수의 소수점이 왼쪽으로 오른쪽으로 둥둥 떠다니는 걸 이해하셨으면 왜 floating point number라고 하는지 쉽게 이해할 수 있습니다.

부동소수점은 아주 큰 수와 작은 수를 표현할 수 있지만 정확도는 떨어집니다. 가수 다시 말해 소수점으로 표현되는 숫자를 표현하는데 무한히 많은 자리를 사용할 수 없기 때문에 한계를 넘어서는 경우에는 표현할 수 있는 최대한을 사용하고 나머지는 반올림해버립니다.

아래의 코드를 실행하면 콘솔에는 몇자리까지 나올까요?

```
float myFloat = 3.141592653589793238;
println(myFloat);
```

콘솔에는 3.1415927까지 나옵니다. 나머지 자리는 반올림되었습니다. 혹시 정확하게 표현해야 할 숫자라면 float을 사용하는 것은 좋은 방법이 아닙니다.

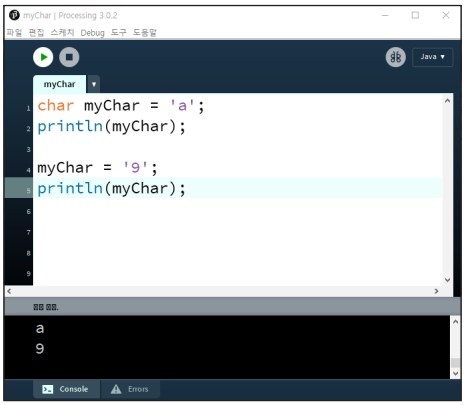


myFloat

문자

문자(character)는 ′a′, ′A′, ′9′, ′g′와 같은 영문자나 숫자 하나를 말합니다. 보통 홑따옴표를 이용해 문자라는 것을 알립니다. 영어 character의 처음 4자를 따서 char 타입이라고 지칭합니다.

```
char myChar = ′a′;
println(myChar);

myChar = ′9′;
println(myChar);
```

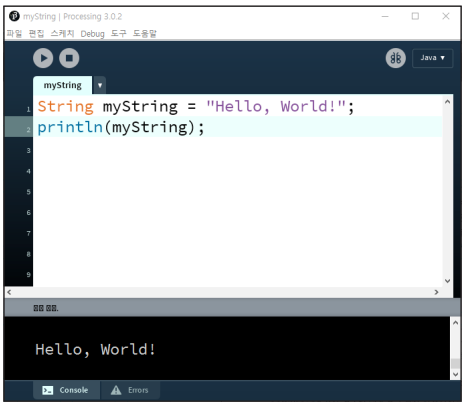


myChar

문자열

사실 하나의 문자만 다루는 일은 드뭅니다. 문자가 여러 개 있을 때 이를 문자열이라고 합니다. 프로세싱은 이런 데이터 타입을 위해 String이라는 데이터 타입을 준비했습니다. 보통 겹따옴표를 이용해 문자열을 표현합니다.

```
String myString = "Hello, World!";
println(myString);
```



myString

정확하게 말하면 String은 문자열을 다루는 클래스(class)입니다. String 클래스에는 문자열을 다루는 데 도움이 되는 여러 함수(클래스의 함수는 따로 메소드라고 부릅니다)가 있습니다. 예를 들어 length() 메소드는 문자열의 길이를 구합니다. 메소드를 사용할 때는 지금까지 사용했던 함수와 사용법이 약간 다릅니다. 아래의 예제를 참고하세요.

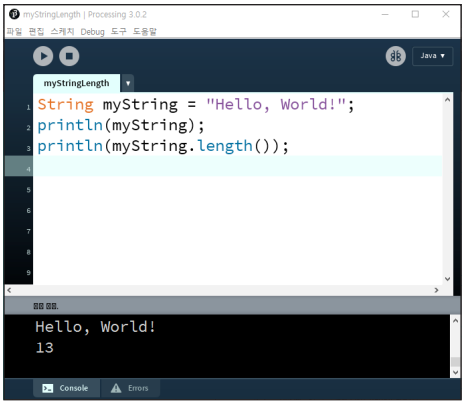
```
println(myString.length());
```

새로운 용어가 나와도 겁먹지 마세요. 일단 사용하고 나중에 이해해도 괜찮습니다. 모든 것을 한 번에 알 수는 없는 것이니 마음을 편하게 가지세요.

String 타입(클래스)를 조금 더 살펴볼까요? toLowerCase() 메서드는 문자열을 모두 소문자로 바꿉니다.

```
println(myString.toLowerCase());
```

이와 비슷하게 toUpperCase() 메서드는 문자열을 대문자로 바꿉니다.



myStringLength

```
println(myString.toUpperCase());
```

배열

프로그램이 복잡해지면 더 많은 데이터가 필요합니다. 예를 들어 원을 100개 그려야 할 때 모든 원의 위치를 일일이 변수로 정의해서 사용해야 한다면 차라리 종이에 원을 그리고 싶은 심정일 것입니다.

배열(array)을 사용하면 이런 불편함이 줄어듭니다. 배열은 **같은** 타입의 여러 변수를 하나의 묶음으로 다루게 합니다.

배열은 항상 '무엇의' 배열이라는 형식을 따릅니다. 예를 들어 char의 배열인지, String의 배열인지 아니면 다른 타입의 배열인지가 중요합니다.

배열을 이용하려면 두 단계를 밟아야 합니다.

배열의 선언(declaration)

배열의 이름을 선언하는 것이 첫 단계입니다.

예를 들어 int의 배열 integers을 사용하고 싶다면,

```
int[] integers;
```

을 실행합니다. 이제 integers는 int타입 배열의 이름으로 선언되었습니다.

배열의 생성

배열이 선언된 다음 배열을 생성해야 합니다. 배열의 선언은 단지 이름만 선언할 뿐 데이터를 저장할 공간은 만드는 것은 배열의 생성단계에서 담당합니다.

```
integers = new int[5];
```

new 명령어를 이용해 다섯개의 원소를 가지는 int 타입 배열을 생성하고 이를 integers로 지칭합니다.

배열의 선언과 생성 두단계가 아니라 한단계로 줄일 수 있습니다.

```
int[] integers = new int[5];
```

위의 방법으로 배열의 선언과 생성을 한번에 할 수도 있습니다.

배열의 초기화

배열의 선언과 생성을 마쳤지만 사실 아직 배열에는 정보가 들어있지 않습니다. 배열에 값을 지정해주는 작업을 따로 해주어야 합니다. 그러려면 배열의 각 원소에 접근해야 합니다. 배열의 인덱스를 이용해 배열의 각 원소에 접근할 수 있습니다.

```
int[] integers = new int[5];
integers[0] = 1;
integers[1] = 2;
integers[2] = 3;
integers[3] = 4;
integers[4] = 5;

println(integers[0]);
println(integers[4]);
```

프로세싱에서 배열의 인덱스는 0부터 시작합니다. 조금 어색하지만 많은 프르그래밍 언어가 따르는 관습입니다.

위에 제시한 방법은 번거롭습니다. 프로그래머들은 번거로움을 죄악시하기 때문에 거의 언제나 좀 덜 번거로운 방법을 고안합니다.

```
int[] integers;
integers = new int[]{1, 2, 3, 4, 5};

println(integers[0]);
println(integers[4]);

String[] texts = new String[]{"안녕하세요","
반갑습니다","즐거운 하루되세요"};

println(texts[0]);
println(texts[2]);
```

03.02
연산

사칙연산

데이터가 주어지면 원하는 결과를 얻기 위해 연산을 해야 합니다. 가장 먼저 사칙연산(더하기, 빼기, 곱하기, 나누기)을 해봅시다.

```
int a = 5;
int b = 10;

println(a + b);
println(a - b);
println(a * b);
println(a / b);
```

마지막 나누기 결과에 주목해주세요. 5 ÷ 10 = 0.5인데 결과가 0으로 나옵니다. 무슨 일이 일어난 것일까요?

0.5는 정수로 표현할 수 없습니다. 0.5는 실수 다시 말해 float 타입으로 표현할 수 있습니다. a와 b를 float타입으로 바꾸면 원하는 결과가 나옵니다. float() 함수는 변수의 타입을 float으로 바꿉니다.

```
println(float(a) / float(b));
```

타입의 중요성을 잘 이해하셨나요?

String 타입도 연산이 가능합니다. 문자열에서 정의된 + 연산은 어떤 결과를 내놓을까요?

```
String myName = "tool of thought";
String myString = "안녕하세요. 내 이름은 " +
myName + " 입니다.";
println(myString);
```

숫자와 문자열을 섞어서 연산할 수도 있습니다.

```
float height = 175.5;
String yourHeight = "당신은 " + height + "cm
입니다.";
println(yourHeight);
```

수학과 관련된 연산

프로세싱은 사칙연산 이외에 많은 연산을 지원합니다. 이런 연산은 함수 형태로 지원됩니다.

예를 들어 절대값을 구하려면 abs() 함수를 사용합니다.

```
println(abs(-4.5));
println(abs(3.2));
```

가장 큰 값을 구하려면 max() 함수를 사용합니다.

```
println(max(-4, -3.5));
```

단순히 두 값을 비교하는 것이 아니라 전체 배열에서 가장 큰 수를 찾을 수도 있습니다.

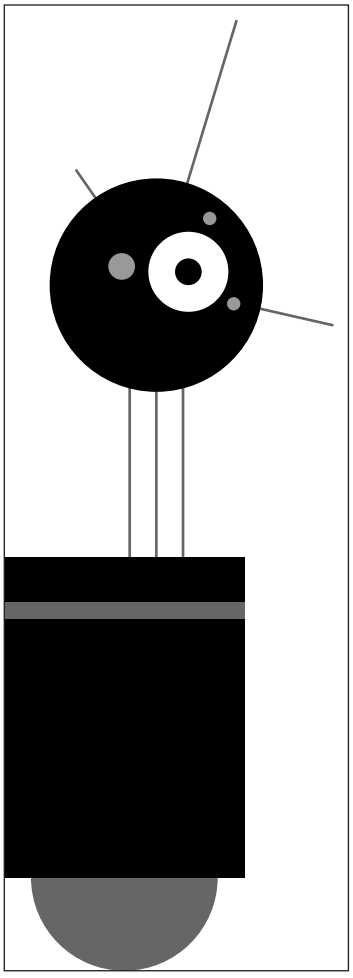
```
float[] myFloats = new float[]{1.2, 2.3,
4.9, 5.3, 0.7};
println(max(myFloats));
```

이밖에도 min(), exp(), dist() 함수 등이 있습니다. 모든 함수를 다 알 수는 없습니다. 다 알 필요도 없습니다. 어떤 연산이 필요하면 프로세싱 레퍼런스(https://processing.org/reference/)를 찾아보세요. 아마 높은 확률로 여러분이 필요한 함수가 이미 존재할 것입니다.

03.03
변수를 이용해 P5 그리기

지난 시간에 그렸던 P5를 다시 그려봅시다. 이번에는 변수를 이용해 그립니다.

```
  void setup() {
   size(640, 480);
   smooth();
   strokeWeight(2);
   background(204);
  }

  void draw() {
   int x = 219;
   int y = 257;

   int neckLength = 95;
   int headWidth = 80;
   int headX = x + 57;
   int headY = y - neckLength - 7;
   int bodyWidth = 90;
   int bodyHeight = 120;
   int bodyBandHeight = 6;
   int bodySphereWidth = 70;

   //neck
stroke(102);
```



```
  line(x + 47, y, x + 47, y - neckLength);
  line(x + 57, y, x + 57, y - neckLength);
  line(x + 67, y, x + 67, y - neckLength);

  //antenna
  line(headX, headY, headX - 30, headY -
43);
  line(headX, headY, headX + 30, headY -
99);
  line(headX, headY, headX + 66, headY +
15);

  //body
  noStroke();
  fill(102);
  ellipse(x + bodyWidth / 2, y +
bodyHeight, bodySphereWidth,
bodySphereWidth);
  fill(0);
  rect(x, y, bodyWidth, bodyHeight);
  fill(102);
  rect(x, y + 17, bodyWidth,
bodyBandHeight);

  //head
  fill(0);
  ellipse(headX, headY, headWidth,
headWidth);
  fill(255);
  ellipse(headX + 12, headY - 5,
headWidth * 3 / 8, headWidth * 3 / 8);
  fill(0);
  ellipse(headX + 12, headY - 5,
headWidth / 8, headWidth / 8);
  fill(153);
  ellipse(headX - 13, headY - 7,
headWidth / 8, headWidth / 8);
  ellipse(headX + 20, headY - 25,
headWidth / 16, headWidth / 16);
  ellipse(headX + 29, headY + 7,
headWidth / 16, headWidth / 16);

  println("(" + headX + ", " + headY + ")");
}
```

이제 변수값을 바꿔봅시다. 목을 늘이고 몸통을 줄여보세요. 변수를 사용하면 이런 점이 편리합니다.

03.04
시스템 변수(system variable)

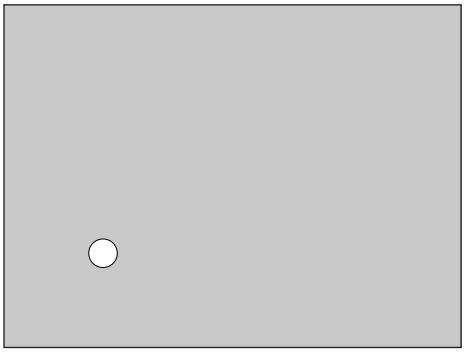
변수이야기가 나온 김에 몇가지 더 살펴봅시다. 프로세싱은 시스템 변수를 가지고 있습니다. 프로세싱이 미리 사용하는 변수로 프로그래밍을 하면서 필요에 따라 언제든 사용할 수 있습니다. 모든 변수를 살펴보기는 어렵고 앞으로 만나게 될 몇가지만 먼저 살펴봅시다.

마우스

mouseX와 mouseY는 이미 만난 적이 있습니다. 마우스의 위치를 픽셀 단위로 담고있는 변수입니다.

```
void setup() {
    size(480, 360);
}

void draw() {
    background(255);
    ellipse(mouseX, mouseY, 30, 30);
}
```

mouseX와 mouseY를 이용해 도형이 마우스를 따라다니게 하는 코드입니다.



mouse

키보드

key

프로세싱은 프로그램이 실행되는 동안 키보드에 입력되는 가장 마지막 문자를 저장합니다. 프로세싱이 저장하고 있는 문자값에 접근하려면 key 변수를 이용합니다.

```
void draw() {
    println(key);
}
```

keyPressed

지금 키보드가 눌러져 있는지 알고 싶은 경우도 있습니다. 이런 경우 시스템 변수 key는 큰 관련이 없습니다. key는 마지막에 입력된 문자값일 뿐 지금 키보드에 입력이 있는지를 알 수 있는 변수는 아닙니다.

시스템 변수 keyPressed를 사용하면 이 문제를 풀 수 있습니다. 예를 들어 사각형을 그리다가 키보드를 누르고 있을 때만 원을 그리고 싶을 때는 아래와 같이 keyPressed를 활용합니다.

```
void setup() {
    size(480, 360);
}

void draw() {
    background(255);
    if (keyPressed) {
          ellipse(240, 180, 40, 40);
    }
    else {
          rect(220, 160, 40, 40);
    }
}
```

if 조건문에 else 조건을 추가한 것을 눈여겨 보세요.

keyCode

키보드의 UP, DOWN, LEFT, RIGHT, ALT, CONTROL, SHIFT 키를 입력받을 때는 주의가 필요합니

다. key 변수와 keyCode를 함께 사용해야 합니다.

아래의 코드는 UP과 DOWN키를 이용해 도형의 색을 밝게 혹은 어둡게 조정하는 예제입니다. 먼저 key == CODED 조건을 확인하고 keyCode 변수를 사용하는 것에 주의하세요.

```
color fillValue = color(125);

void setup() {
    size(480, 360);
}

void draw() {
    fill(fillValue);
    ellipse(240, 180, 40, 40);
    if (key == CODED) {
        if (keyCode == UP) {
            fillValue = 255;
        }
        else if (keyCode == DOWN) {
            fillValue = 0;
        }
    }
    else {
        fillValue = 125;
    }
}
```

프레임

width & height

언급하고 넘어가야 할 화면과 관련된 시스템 변수가 몇 가지 있습니다. 가장 중요한 시스템 변수는 width와 height입니다. 화면의 크기는 size() 함수에 의해서 결정됩니다. 화면 크기는 빈번하게 사용되는데 이를 위해 width는 화면의 너비를, height는 화면의 높이를 알려주는 시스템 변수로 사용됩니다.

예를 들어 화면 한가운데 원을 그릴 때 지금까지는 화면 크기를 고려해 적당한 수를 넣어야 했습니다. 하지만 width와 height를 사용하면 아래 코드처럼 간단하게 처리할 수 있습니다.

```
ellipse(width / 2, height / 2, 10, 10);
```

frameRate

draw() 함수는 1/60초마다 화면을 다시 그립니다. 그런데 복잡한 그림을 그릴 때 컴터의 성능이 허락하지 않는다면 초당 60번 미만의 프레임을 그릴 수 밖에 없습니다. 실제 1초당 몇 프레임이 그려지는지 알 수 있는 시스템 변수가 frameRate입니다.

```
println(frameRate);
```

비슷하게 프로그램이 시작된 후 지금 화면이 몇 번째 프레임인지 알고 싶다면 frameCount 변수를 사용합니다.

```
println(frameCount);
```

03.05
스스로 움직이는 도형

사각형이 화면을 부유하는 원을 프로그램을 실행해보세요. 오늘 배운 변수를 활용해 연산을 하고 조건문을 활용해 도형이 화면 안에 항상 머물게 합니다. 단계별로 코드를 실행해서 각 단계에 추가된 기능을 확인해보세요.

정지되어 있는 원

도형을 정의하려면 우선 도형의 위치를 결정해야 합니다. 도형의 위치를

```
float posX;
float posY;
```

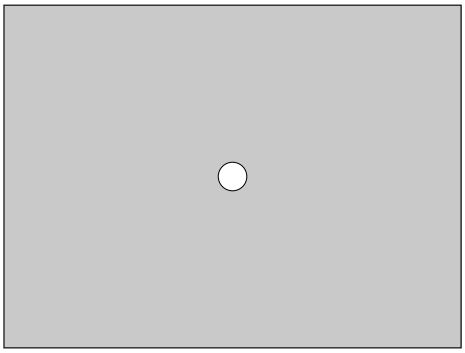
두 개의 변수로 정의합니다.

그리고 초기값을 입력합니다. 화면의 중간에 도형이 위치하도록 합시다.

```
posX = width / 2;
posY = height / 2;
```

이제 크기 30픽셀의 원을 그립니다. 전체 코드를 모두 적어보겠습니다.

```
float posX;
float posY;

void setup() {
    size(480, 360);
    smooth();
    strokeWeight(5);

    posX = width / 2;
    posY = height / 2;
}

void draw() {
    ellipse(posX, posY, 30, 30);
}
```

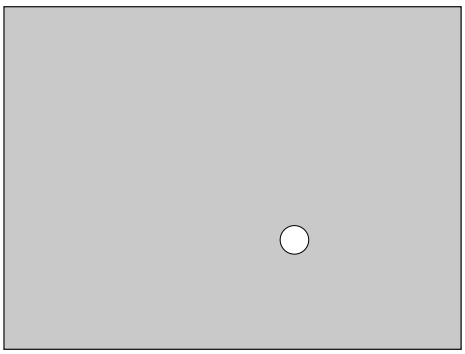


staticCircle

등속도 운동을 하는 원

정지해 있는 원은 재미가 없습니다. 같은 속도로 이동하는 원을 그려봅시다. 속도라고 해서 어렵게 생각할 필요는 없습니다. 현재 원은 (posX, posY)에 위치합니다. 다음 프레임에서 원의 위치를 (posX + 1, posY + 1)이라고 하면 한 프레임 사이에 오른쪽으로 1픽셀 아래쪽으로 1픽셀 이동했다는 뜻이고 변화량 (1, 1)이 이 원의 속도입니다. 한번 그려볼까요? 변수를 이용해 코드를 정리해 보았습니다.

```
float posX;
float posY;
float velocityX;
float velocityY;
float r;

void setup() {
    size(480, 360);
    smooth();
    strokeWeight(5);

    posX = width / 2;
    posY = height / 2;

    velocityX = 1;
    velocityY = 1;

    r = 30;
}
```



constantVelocity

```
void draw() {
    background(201);
    posX = posX + velocityX;
    posY = posY + velocityY;
    ellipse(posX, posY, 30, 30);
}
```

원을 화면 안으로 제한하기

위의 코드를 실행하면 원이 움직이다 곧 화면을 벗어납니다. 이렇게 되면 원을 그린 의미가 없으니 화면을 벗어나지 않도록 수정합시다.

아이디어는 간단합니다. 도형이 화면의 오른쪽을 벗어나면 도형이 다시 왼쪽으로 나오게 합시다. 만약 화면의 아래쪽으로 사라지면 다시 화면의 위쪽으로 나타나게 합니다. 이 아이디어를 실제로 실행하려면 어떻게 해야 할까요?

(posX, posY)를 조정해야 합니다. 아래와 같이 조건문을 더합니다.

```
if (posX > width) {
    posX = 0;
}

if (posY > height) {
    posY = 0;
}
```

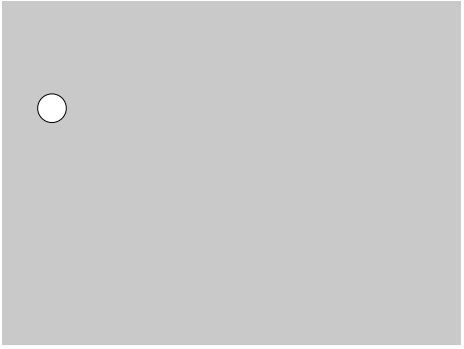
지금은 도형이 오른쪽 아래로 움직이기 때문에 오른쪽과 아래쪽 화면에서 사라지는 것만 고려했습니다. 만약 도형이 왼쪽이나 위로 움직일 경우에는 다음의 조건문을 추가해야 합니다.

```
if (posX < 0) {
    posX = width;
}

if (posY < 0) {
    posY = height;
}
```

조건문을 추가한 다음 다시 실행해볼까요?

```
float posX;
float posY;
float velocityX;
```

```
float velocityY;

void setup() {
 size(480, 360);
 smooth();
 strokeWeight(5);

 posX = width / 2;
 posY = height / 2;

 velocityX = 1;
 velocityY = 1;
}

void draw() {
 background(201);
 posX = posX + velocityX;
 posY = posY + velocityY;

 if (posX > width) {
 posX = 0;
 }

 if (posY > height) {
 posY = 0;
 }

 if (posX < 0) {
 posX = width;
 }

 if (posY < 0) {
 posY = height;
 }

 ellipse(posX, posY, 30, 30);
}
```



constantVelocityOverlapping

점점 빨라지는 원

천천히 움직이는 원을 보고 있으니 평온한 생각이 들기는 하지만 너무 지루합니다. 늘 똑같은 속도로 움직이는 원을 좀 더 활기차게 바꾸어 봅시다. 이제 가속도를 추가해 봅시다.

위치 (posX, posY)가 매 프레임마다 (posX + velocityX, posY + velocityY)로 바뀌면 도형이 같은 속도로 운동하는 것이라고 했습니다. 같은 이치로 속도 (velocityX, velocityY)가 매 프레임마다 (velocityX + accelationX, velocityY +

accelerationY)로 바뀌면 가속도 운동이 됩니다.

```
velocityX = velocityX + accelerationX;
velocityY = velocityY + accelerationY;
```

속도가 너무 빨라지면 곤란하니 min()함수를 이용해 제한을 걸어둡니다.

```
velocityX = min(8, velocityX);
velocityY = min(8, velocityY);
```

우선 코드를 실행해볼까요?

```
float posX;
float posY;
float velocityX;
float velocityY;
float accelerationX;
float accelerationY;

void setup() {
 size(480, 360);
 smooth();
 strokeWeight(5);

 posX = width / 2;
 posY = height / 2;

 velocityX = 1;
 velocityY = 1;
 accelerationX = 1;
 accelerationY = 1;
}

void draw() {
 background(201);
 velocityX = velocityX + accelerationX;
 velocityY = velocityY + accelerationY;

 velocityX = min(8, velocityX);
 velocityY = min(8, velocityY);

 posX = posX + velocityX;
 posY = posY + velocityY;

  if (posX > width) {
 posX = 0;
 }

 if (posY > height) {
 posY = 0;
```

acceleration

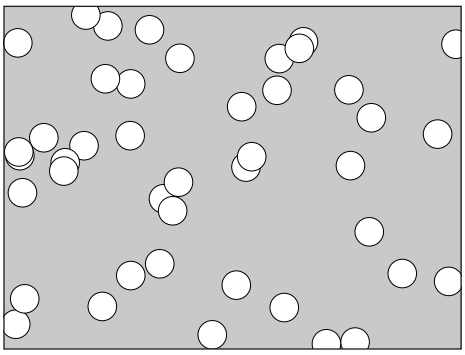
```
  }

  if (posX < 0) {
  posX = width;
  }

  if (posY < 0) {
  posY = height;
  }

  ellipse(posX, posY, 30, 30);
 }
```

이리저리 움직이는 원

한 방향으로 움직이는 원은 지루합니다. 그러지 말고 화면을 부유하는 원을 만들어 봅시다.

아이디어는 이렇습니다. 원의 위치 (posX, posY)가 임의의 값을 따르도록 합시다. random() 함수를 사용하면 임의의 값을 뽑을 수 있습니다. random(n)은 0에서 n사이의 값(n은 포함하지 않습니다)중 임의의 값을 선택하는 함수입니다.

```
float posX;
float posY;

void setup() {
 size(480, 360);
 smooth();
 strokeWeight(5);
}

void draw() {
 background(201);
 posX = random(width);
 posY = random(height);

 ellipse(posX, posY, 30, 30);
}
```



random.pdf

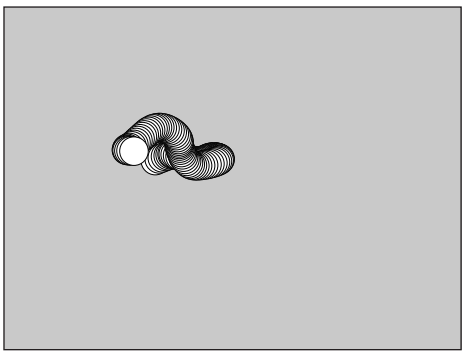
도형이 정신없이 움직입니다. random() 함수는 계속 다른 값을 뽑기 때문에 도형이 점프를 하는 것처럼 보입니다. 좀 더 부드럽게 움직일 수는 없을까요?

noise() 함수

noise() 함수를 사용합시다. noise() 함수는 부드럽게 임의의 값을 생성합니다. noise() 함수는 항상 0에

서 1사이의 값을 임의로 생성합니다. 비슷한 값에 대해서는 비슷한 값을 둘려주지만 차이가 커지면 커질수록 다른 값을 생성할 확률이 커집니다. 이 함수를 사용하면 부드럽게 움직이는 원을 관찰할 수 있습니다.

```
float posX;
float posY;
float noffX;
float noffY;

void setup() {
 size(480, 360);
 smooth();
 strokeWeight(5);
 noffX = random(1000);
 noffY = random(1000);
}

void draw() {
 background(201);
 posX = map(noise(noffX), 0, 1, 0,
width);
 posY = map(noise(noffY), 0, 1, 0,
height);

 ellipse(posX, posY, 30, 30);

 noffX = noffX + 0.01;
 noffY = noffY + 0.01;
}
```



noise

map() 함수를 이용해 noise() 함수를 이용해 얻은 0에서 1사이의 값을 화면의 좌표에 대응시키는 것에 주의하세요. 이런 테크닉은 앞으로도 자주 사용될 것입니다.

기본 형식: map(value, start1, stop1, start2, stop2);

(start1, stop1)구간 사이의 value값을 (start2, stop2) 구간의 값으로 다시 대응시킵니다.