Generative Design Workshop

Fridays at UDK Raum 4/5

November, Friday 6th, 2015

Variable

A variable is a storage location that has
a name, a type and a value.

```
float noseX = -10.5;
 type    name        value
```

The type defines the kind of values you can store
in the variable. "float" stands for numbers with decimal point.
The name is “noseX”. Descriptive names are good.
The value is `-10.5`.

Other valid types are “int” (for integers like 3 and 7),
"color" for colors like #FF0000, "String" for text like "hello".

Variable

Variable names must begin with a letter or underscore.
The name can contain letters, numbers and underscores.

Short variable names are easy to type, but their purpose is easy to forget: what do a, b, c stand for?

When the name contains several words, some use the underscore:

```
sunset_weight
```

and others use camel case:

```
sunsetWeight
```

Variable

Processing automatically declares
some variables for us:

`width, height, frameCount,`
`mousePressed, mouseX, mouseY`

Some of them are automatically updated
on every animation frame:

`frameCount, mousePressed, mouseX, mouseY`

Why use variables

They represent properties or
the current state of things,
which is needed in dynamic systems.

Variables let us reuse values.

They let us modify values over time.

Variables let us name values, making the program
easier to understand and modify.

They allow complex operations to be split
into more readable steps.

Variable

Declare a variable:

```
    float x;
```

Initialize a variable:

```
    x = 10.1;
```

Declare and initialize a variable:

```
    float x = 10.1;
```

Use a variable:

```
    println(x);
    line(x, x, 20, 20);
```

Exercise with variables

Write a program that includes several global variables that allow adjusting your design.

```
float minX = 200;
float maxX = 400;
color bgColor = #000000;
color fgColor = #FFFFFF;
float x;

void setup() {
  size(400, 400);
  background(bgColor);
  stroke(fgColor);
}
void draw() {
  x = random(minX, maxX);
  // vertical line
  line(x, 0, x, height);
}
```

Conditions with if/else

The <u>if</u> statement lets you run code conditionally:

```
if(mouseX > 100) {
  ellipse(mouseX, mouseY, 20, 20);
}
```

Use <u>else</u> to provide two alternatives:

```
if(mouseX < 200) {
  ellipse(mouseX, mouseY, 20, 20);
} else {
  line(mouseX, mouseY, width, height);
}
```

Conditions and probability

if() combined with random() enables code that runs with a certain probability %

```
if(random(100) < 20) {
  fill(#FF0000); // red = 20% probability
} else {
  fill(#FFFFFF); // white = 80% probability
}
```

Example. A grid.

```
float x = 0;
float y = 0;
float sz;
void setup() {
  size(600, 600);
}
void draw() {
  sz = random(10, 50);
  ellipse(x, y, sz, sz);
  // increase x
  x = x + 50;
  // if x is too large, make it 0 and move down
  if(x > width) {
    x = 0;
    y = y + 50;
  }
}
```

Creating a grid with two variables and an if statement

Function

<u>function</u>: a sequence of program instructions that perform a specific task

also known as: a procedure, a subroutine, a method, a subprogram

Function

We can use functions provided by Processing
or declare our own.

Functions have a name.
Names of some Processing functions:

```
ellipse, line, stroke, noStroke, random...
```

Function

Calling = running = executing functions:

Calling a function that takes no arguments:

```
bakeCake();
drawElephant();
noStroke();
```

Calling a function that does take arguments:

```
stroke(255);
```

Function

Functions that return no value:

```
noStroke();
line(11, 22, 33, 44);
// and most drawing functions
```

Functions that return a value:

```
println(random(10, 20));
println(sin(3));
// and other math functions
```

Function

Functions that take no arguments:

```
noStroke();
noFill();
```

Functions that do take arguments:

```
stroke(255);
line(11, 22, 33, 44);
```

Creating and calling functions

```
// declaring a function that takes no arguments
void drawRandomShapes() {
  fill(#FF0000);
  ellipse(random(width), random(height), 50, 50);
  fill(#00FF00);
  rect(random(width), random(height), 80, 30);
}

// calling your function
drawRandomShapes();
```

Creating and calling functions

```
// declaring a function that takes no arguments
void drawRandomShapes() {
  fill(#FF0000);
  ellipse(random(width), random(height), 50, 50);
  fill(#00FF00);
  rect(random(width), random(height), 80, 30);
}
// declaring a function that takes arguments
void drawCircles(float x, float y) {
  fill(100);
  ellipse(x, y, 50, 50);
  fill(200);
  ellipse(x, y, 20, 20);
}
// calling functions
drawRandomShapes();
drawEllipses(80, 80);
```

Repetition: for loop examples

The for loop is used to run code multiple times.

Example: count from 1 to 5:

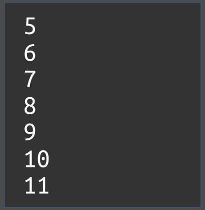
```
for(int i=1; i<6; i=i+1) {
   println(i);
}
```

```
1
2
3
4
5
```

Repetition: for loop examples

Count from 5 to 11:

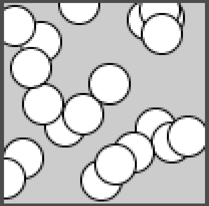
```
for(int i=5; i<12; i=i+1) {
    println(i);
}
```



```
5
6
7
8
9
10
11
```

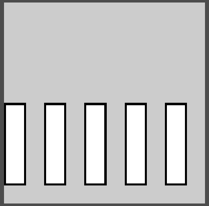
Draw 20 random circles:

```
for(int i=0; i<20; i=i+1) {
    ellipse(random(width), random(height), 20, 20);
}
```



Draw a rectangle every 20 pixels:

```
for(int x=0; x<width; x=x+20) {
    rect(x, 50, 10, 40);
}
```



Repetition

The structure of the for loop:

```
for(init; test; update) {
    //statements you want to repeat
}
```

1. The init statement is run.
2. If the test is false, go to step 6,
   otherwise continue.
3. Run the statements within the block.
4. Run the update statement
5. Jump to step 2.
6. Exit the loop.

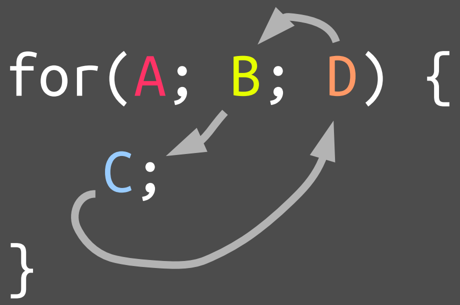
```
for(int i=0; i<3; i=i+1) {
    println(i);
}

                 // pseudocode!

int i=0          // i = 0
i<3?             // 0<3? yes:cont.
println(i)       // 0
i=i+1            // i = 1
i<3?             // 1<3? yes:cont.
println(i)       // 1
i=i+1            // i = 2
i<3?             // 2<3? yes:cont.
println(i)       // 2
i=i+1            // i = 3
i<3?             // 3<3? no:exit
```

Repetition

For loop flow



```
for(A; B; D) {
    C;
}
```

A BCD BCD … BCD B

```
for(int i=0; i<3; i=i+1) {
    println(i);
}
```

```
                // pseudocode!

int i=0         // i = 0
i<3?            // 0<3? yes:cont.
println(i)      // 0
i=i+1           // i = 1
i<3?            // 1<3? yes:cont.
println(i)      // 1
i=i+1           // i = 2
i<3?            // 2<3? yes:cont.
println(i)      // 2
i=i+1           // i = 3
i<3?            // 3<3? no:exit
```

Generative Design Workshop

October / November 2015 at UDK - Berlin

Abe Pazos
@fun_pro | fun@funprogramming.org
http://hamoid.com | http://funprogramming.org