

Programming 02

`pushMatrix()` and `popMatrix()`

Interpolated Motion

`mousePressed`, `mousePressed()`

`mouseDragged()`, `mouseMoved()`, `mouseReleased()`

User-defined Functions

Functions Returning Values

`dist()`

Variable Scope

pushMatrix() and popMatrix()

Pushes the current transformation matrix onto the matrix stack. The push() function saves the current coordinate system to the stack and pop() restores the prior coordinate system. Think of it as a new piece of paper on the stack with its own matrix/coordinate system. In a loop you always need both push() and pop() to avoid a stack overflow.

```
void setup() {
  size(200,200);
  fill(80);
  noStroke();
  smooth();
  rectMode(CENTER);
}

void draw() {
  background(255);
  pushMatrix(); // adds new matrix/coordinate system
  translate(50, 50);
  rotate(radians(mouseX));
  rect(0, 0, 50, 50);
  popMatrix(); // goes back to the prior matrix/coordinate system
  translate(150, 50);
  rotate(radians(mouseY));
  rect(0, 0, 50,50);
}
```

Without pushMatrix() and popMatrix()

```
void setup() {  
  size(200,200);  
  fill(80);  
  noStroke();  
  smooth();  
  rectMode(CENTER);  
}  
  
void draw() {  
  background(255);  
  translate(50, 50);  
  rotate(radians(mouseX));  
  rect(0, 0, 50, 50);  
  translate(150, 50);  
  rotate(radians(mouseY));  
  rect(0, 0, 50,50);  
}
```

pmouseX, pmouseY

pmouseX and pmouseY contains the previous horizontal and previous vertical coordinate of the mouse. It is the position of the mouse in the frame previous to the current frame.

This is very useful to determine the velocity of a mouse movement or gesture. By subtracting the previous from the current mouse position the current mouse velocity can be determined.

```
void draw()  
{  
  background(204);  
  line(mouseX, 20, pmouseX, 80);  
}
```

Interpolated Motion

```
float ox; // the object's current horizontal position
float oy; // the object's current vertical position
float dx; // delta X, subtraction between mouseX and ox
float dy; // delta Y, subtraction between mouseY and oy
float divisor = 50.0; // between the mouse Position and the object's position

void setup() {
  size(200, 200);
  noStroke();
  rectMode(CENTER);
}

void draw() {
  background(100);
  dx = mouseX - ox;
  // do it only if the object's horizontal distance to the mouse is > 1 pixel
  if(abs(dx) > 1) {
    ox = ox + (dx/divisor);
  }
  dy = mouseY - oy;
  // do it only if the object's vertical distance to the mouse is > 1 pixel
  if(abs(dy) > 1) {
    oy = oy + (dy/divisor);
  }
  rect(ox, oy, 10, 10);
}
```

mousePressed

mousePressed is a system variable which is true if the button is pressed and false if the button is not pressed.

```
void setup() {
  size(200, 200);
  rectMode(CENTER);
  background(255);
  smooth();
  noStroke();
}

void draw() {
  if(mousePressed == true) {
    fill(random(255), 100);
  } else {
    fill(0);
  }
  rect(mouseX, mouseY, 30, 30);
}
```

mousePressed()

The mousePressed() system function is called every time the mouse button is pressed

```
int value = 0;

void draw() {
  fill(value);
  rect(25, 25, 50, 50);
}

void mousePressed()
{
  if(value == 0) {
    value = 255;
  } else {
    value = 0;
  }
}
```

Other System Mouse Functions

```
void mouseReleased() { ... }
```

called every time when the mouse is released

```
void mouseDragged() { ... }
```

called every time when the mouse is dragged (pressed and moved)

```
void mouseMoved() { ... }
```

called every time when the mouse moves and not pressed

User-defined Functions

```
void setup() {
  size(200, 200);
  background(51);
  noStroke();
  ellipseMode(CENTER);
}
void draw() {
  draw_target(68, 34, 100, 10);
  draw_target(152, 16, 50, 3);
  draw_target(100, 144, 40, 5);
}
void draw_target(int xloc, int yloc, int size, int num) {
  float grayvalues = 255/num;
  float steps = size/num;
  for(int i=0; i<num; i++) {
    fill(i*grayvalues);
    ellipse(xloc, yloc, size-i*steps, size-i*steps);
  }
}
```

Functions Returning Values

`void` functions are only executed, but do not return values, e.g.:

```
void setup() {... }  
void myFunction() {... }
```

You can also define functions which are returning values

```
myNumber(1,3); // calls myNumber with parameters  
int myNumber(int value1, int value2) {  
    return value1*value2;  
}
```

the keyword `return` indicates the value to be returned

```
float myFloat() {...} // returns float value  
String myString() {...} // returns String value  
boolean checkThis() {... } // returns true or false
```

dist()

The function `dist()` calculates the distance between two points

```
dist(x1, y1, x2, y2);  
dist(x1, y1, z1, x2, y2, z2);  
  
void setup() {  
    size(200, 200);  
    fill(0);  
}  
void draw() {  
    background(255);  
    point(100, 100);  
    rect(0, 0, dist(100, 100, mouseX, mouseY), 5);  
}
```

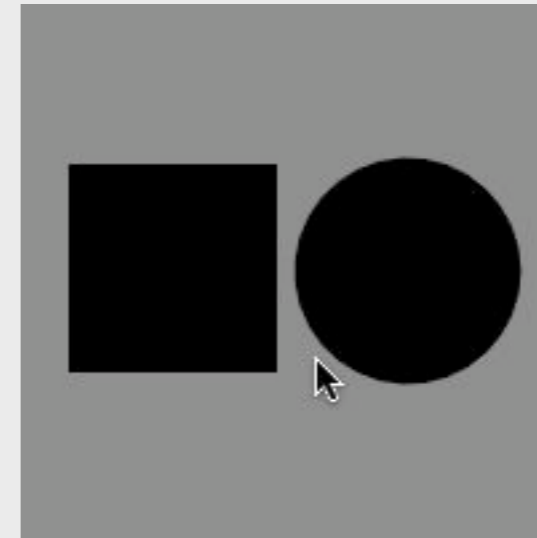
Determining mouseOver

```
void setup() {
  size(200, 200);
  noStroke();
  smooth();
}

void draw() {
  background(126);
  if(overRect(18, 60, 78, 78) == true) {
    fill(255);
  } else {
    fill(0);
  }
  rect(18, 60, 78, 78);
  if(overCircle(145, 100, 42.5) == true) {
    fill(255);
  } else {
    fill(0);
  }
  ellipse(145, 100, 85, 85);
}

boolean overRect(float x, float y, float w, float h) {
  if(mouseX > x && mouseX < x+w && mouseY > y && mouseY < y+h) {
    return true;
  } else {
    return false;
  }
}

boolean overCircle(float x, float y, float radius) {
  if(dist(mouseX, mouseY, x, y) < radius) {
    return true;
  } else {
    return false;
  }
}
```



Using User-defined Functions

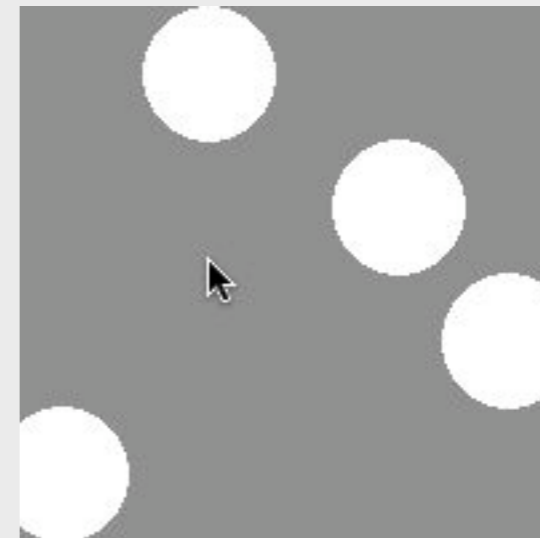
```
void setup() {
  size(200, 200);
  noStroke();
}

void draw() {
  float mx = mouseX;
  // Normalized version of mouseX (ranging in value from 0 to 1)
  float nmx = mx/width;
  background(126);
  ellipse(mx, 25, 50, 50);
  ellipse(chevron(nmx)*200.0, 75, 50, 50);
  ellipse(bump(nmx)*200.0, 125, 50, 50);
  ellipse(inverse_bump(nmx)*200.0, 175, 50, 50);
}

float chevron(float x) {
  if(x < .5) {
    x = x * 2;
  } else {
    x = 1.0 - ((x-0.5) * 2);
  }
  return x;
}

float bump(float x) {
  x = (x - 0.5) * 2.0; // Scale from -1 to 1
  x = 1 - x*x;
  return x;
}

float inverse_bump(float x) {
  x = (x - 0.5) * 2.0; // Scale from -1 to 1
  x = 1 - x*x;
  return 1 - x;
}
```



Variable Scope

Variables may either have a global or local "scope". For example, variables declared within either the `setup()` or `loop()` functions may be only used in these functions.

Global variables, variables declared outside of `setup()` and `loop()`, may be used anywhere within the program. If a local variable is declared with the same name as a global variable, the program will use the local variable to make its calculations within the current scope.

```
int c = 100; // global variable

void setup(){
  int s = 255; // local variable
  background(c); // use global variable
  stroke(s);
}

void draw(){
  background(c); //use global variable
  line(i, 0, i, height);
  int i = 0; // local variable
  stroke(c);
  if (i > width) {
    i = 0;
  }

  i++;
}
```