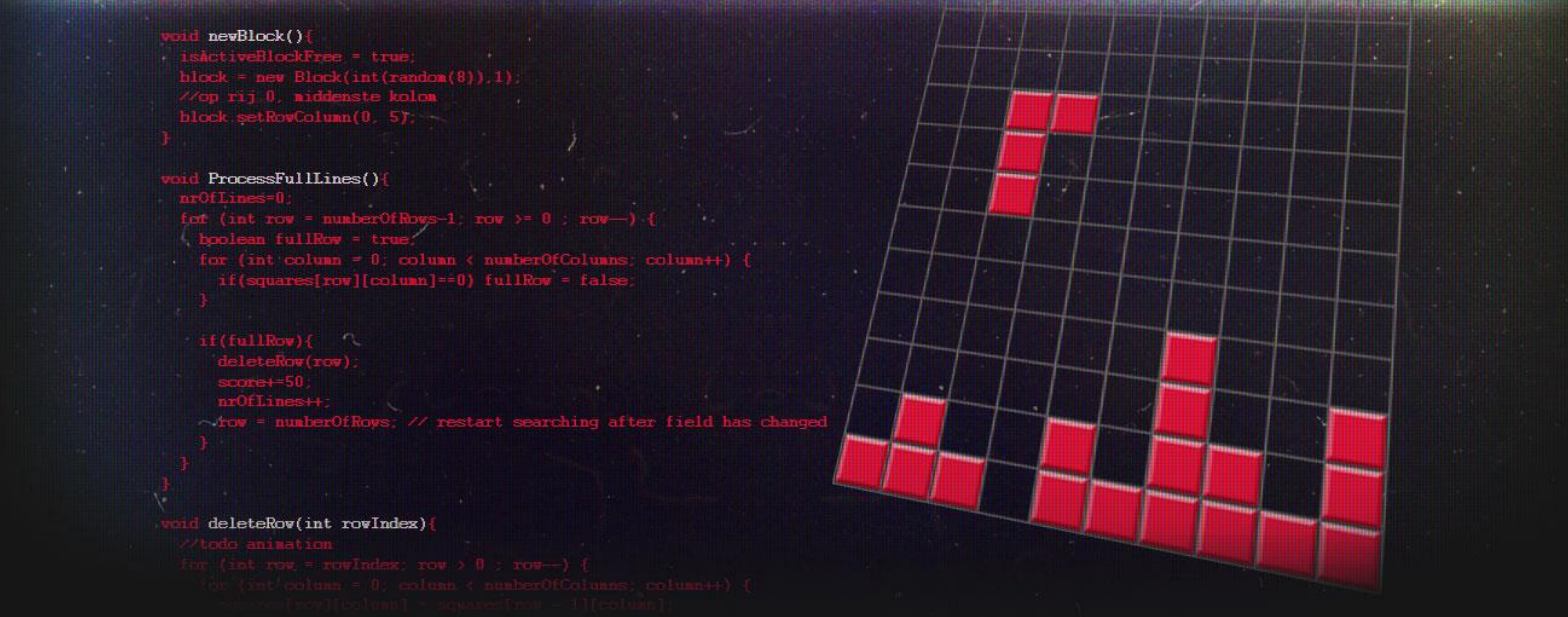


```
void newBlock(){
  isActiveBlockFree = true;
  block = new Block(int(random(8)),1);
  //op rij 0, middenste kolom
  block.setRowColumn(0, 5);
}

void ProcessFullLines(){
  nrOfLines=0;
  for (int row = numberOfRows-1; row >= 0 ; row--) {
    boolean fullRow = true;
    for (int column = 0; column < numberOfColumns; column++) {
      if(squares[row][column]==0) fullRow = false;
    }

    if(fullRow){
      deleteRow(row);
      score+=50;
      nrOfLines++;
      row = numberOfRows; // restart searching after field has changed
    }
  }
}

void deleteRow(int rowIndex){
  //todo animation
  for (int row = rowIndex; row > 0 ; row--) {
    for (int column = 0; column < numberOfColumns; column++) {
      squares[row][column] = squares[row - 1][column];
```

PROGRAMMING FOR ARTISTS II

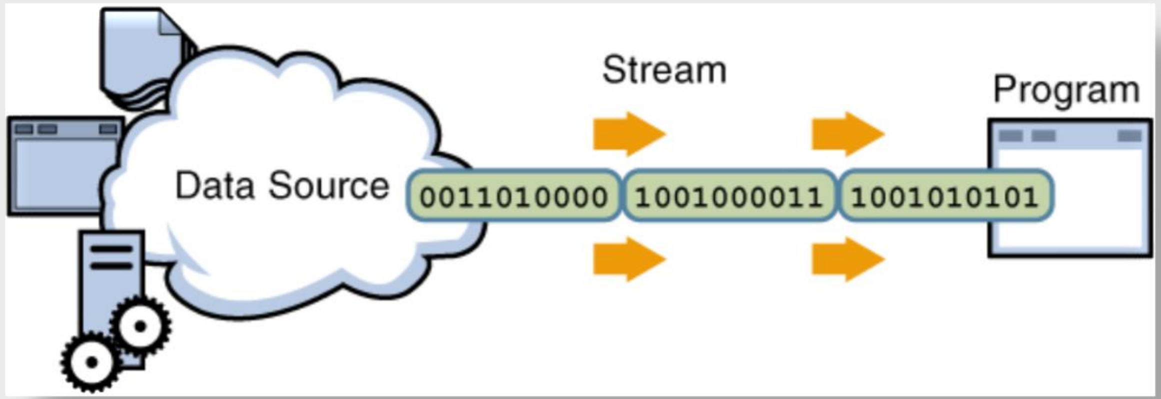
FILE INPUT / OUTPUT

INTERACTION WITH FILES

- Variables are destroyed when application stops
- Store settings for program, list of high scores, ...

To create a text file:
- Use text editor
- Save the file with the ".txt" extension

Each line
from this text file
will be an element
in an array
of strings



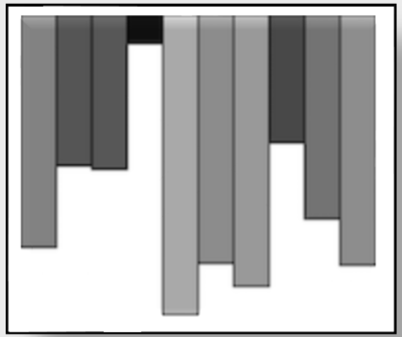
READ FROM A FILE: LOADSTRINGS

- read the content of the file or hyperlink into a String array
- each line in the file is an individual element
- file must be placed in the data folder or loaded from an absolute path
    - **not available**? **null** will be returned!

numbers.txt

```
131,85,87,16,169,140,153,72,115,141
```

The result of visualizing these numbers is shown below:



```
int[] bars;

void setup() {
  size(200, 200);

  // Load text file as a String
  String[] stuff = loadStrings(“data/numbers.txt");

  // Convert string into an array using ',' as a delimiter
  bars = int(split(stuff[0], ','));
}

void draw() {
  background(255);
  stroke(0);

   /*Use array of integers to set the color and height of
     each rectangle.*/

  for (int i=0; i < bars.length; i++) {
    rect(i*20, 0, 20, bars[i]);
  }
  noLoop();
}
```

programming for artists II

READ FROM A FILE: EXAMPLE

The example below shows how to read from a text file, using an array of pvectors:

coords.txt

```
880.4895 373.5595
797.7594 551.85364
582.87476 590.6258
```

Output coords

```
[0] "880.4895 373.5595"
[1] "797.7594 551.85364"
[2] "582.87476 590.6258"
```

Output pairs

```
[0] [ 880.4895, 373.5595, 0.0 ]
[1] [ 797.7594, 551.85364, 0.0 ]
[2] [ 582.87476, 590.6258, 0.0 ]
```

```
String[] coords = loadStrings("coords.txt");
int numLines = coords.length;

//initialize an array of PVectors
PVector[] pairs = new PVector[numLines];

for (int i=0; i < coords.length; i++) {
  /*
  splitTokens(): If no delim characters are specified,
  any whitespace character is used to split.
  Whitespace characters include tab (\t), line feed (\n),
  carriage return (\r), form feed (\f), and space.
  */
  float[] pair = float(trim(splitTokens(coords[i])));
  pairs[i] = new PVector(pair[0], pair[1]);
}

printArray(coords);

printArray(pairs);
```

WRITE TO A FILE

The function saveStrings() **writes** an **array of Strings** to a file, **one line per String**.

By default, this file is saved to the sketch's folder (always place the files in the data folder of your sketch!)

```
String[] dwarfs = new String[7];
dwarfs[0] = “Grumpy";
dwarfs[1] = “Happy";
…

/* Writes the array string to a file,
   each element on a separate line   */

saveStrings("data/sevendwarfs.txt", dwarfs);
```

WRITE TO A FILE USING PRINTWRITER

If you want to write multiple lines without first placing them into an array, use the **PrintWriter** class.

PrintWriter allows **characters** to print to a **text-output stream**.

A new PrintWriter object is created with the **createWriter()** function.
For the file to be made correctly, it should be **flushed** and must be **closed** with its **flush()** and **close()** methods.

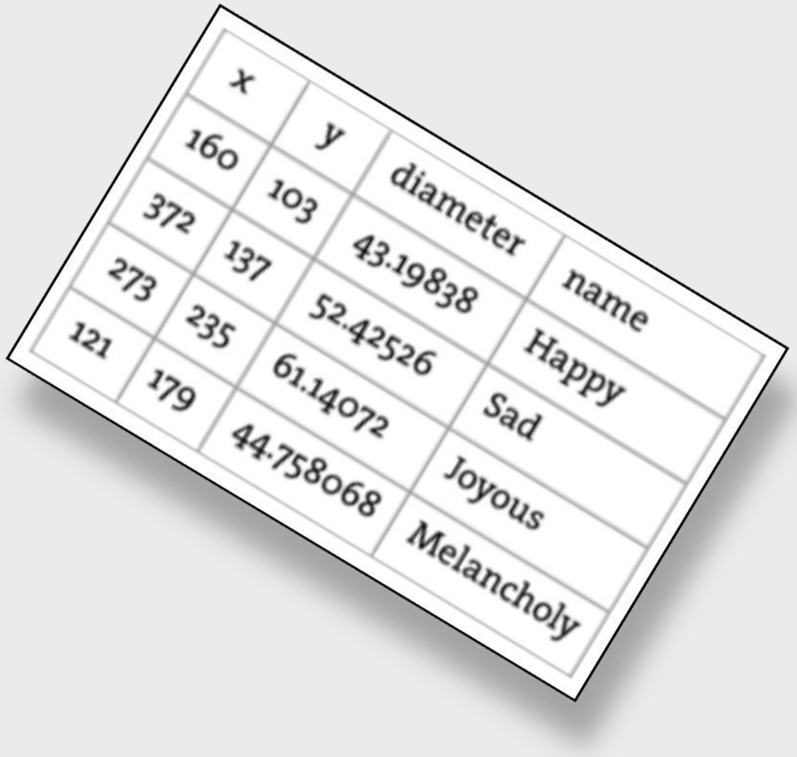
```
PrintWriter output;

void setup() {
  // Create a new file in the sketch’s data directory
  output = createWriter(“data/positions.txt");
}

void draw() {
  point(mouseX, mouseY);
  output.println(mouseX);  // Write the coordinate to the file
}

void keyPressed() {
  output.flush();  // Writes the remaining data to the file
  output.close();  // Finishes the file
  exit();  // Stops the program
}
```

READING TABULAR DATA

A table consists of data arranged as a set of rows and columns, also called "tabular data." If you've ever used a spreadsheet, this is tabular data. Processing's **loadTable()** function takes comma-separated (csv) or tab-separated (tsv) values and automatically places the contents into a Table object storing the data in columns and rows.

data.csv

```
x,y,diameter,name
160,103,43.19838,Happy
372,137,52.42526,Sad
273,235,61.14072,Joyous
121,179,44.758068,Melancholy
```



x	y	diameter	name
160	103	43.19838	Happy
372	137	52.42526	Sad
273	235	61.14072	Joyous
121	179	44.758068	Melancholy

Notice how the first line of text is not the data itself, but rather a header row. This row includes labels that describe the data included in each subsequent row. Processing can automatically interpret and store the headers for you, if you pass in the option "header" when loading the table:

```
Table table = loadTable("data.csv", "header");
```

READING TABULAR DATA

In the above image you can see that the data is organized in terms of rows and columns. One way to access the data, therefore, would be to request a value by its numeric row and column location (with zero being the first row or first column)

```
int val1 = table.getInt(2, 1);      // val now has the value 235
float val2 = table.getFloat(3, 2);  // val2 now has the value 44.758068
String s = table.getString(0, 3);   // s now has the value “Happy”
```

While the numeric index is sometimes useful, it’s generally going to be more convenient to access each piece of data by the column name, by getting a specific row from the Table.

```
TableRow row = table.getRow(2); // Gets the third row (index 2)
```

Once you have the TableRow object, you can ask for data from some or all of the columns.

```
int x = row.getInt("x");              // x has the value 273
int y = row.getInt("y");              // y has the value 235
float d = row.getFloat("diameter");   // d has the value 61.14072
String s = row.getString("name");     // s has the value “Joyous”
```

programming for artists II

WRITING TABULAR DATA

```
Table newTable = new Table();
```

To add a new row to a Table, simply call the method addRow() and set the values of each column.

```
TableRow myRow = newTable.addRow();

//Set the values of all columns in that row.
myRow.setFloat("x", mouseX);
myRow.setFloat("y", mouseY);
myRow.setFloat("diameter", random(40, 80));
myRow.setString("name", "new label");
```

To delete a row, call the method removeRow() and pass in the numeric index of the row you would like removed.

```
// If the table has more than 10 rows
if (newTable.getRowCount()>10) {

//Delete the first row (index 0).
 newTable.removeRow(0);
}
```

WRITING TABULAR DATA: EXAMPLE

```
Table newTable;

void setup() {

 newTable = new Table();

 newTable.addColumn("id");
 newTable.addColumn("species");
 newTable.addColumn("name");

 TableRow newRow = newTable.addRow();
 newRow.setInt("id", newTable.lastRowIndex());
 newRow.setString("species", "Panthera leo");
 newRow.setString("name", "Lion");

 saveTable(newTable, "data/new.csv");
}
```

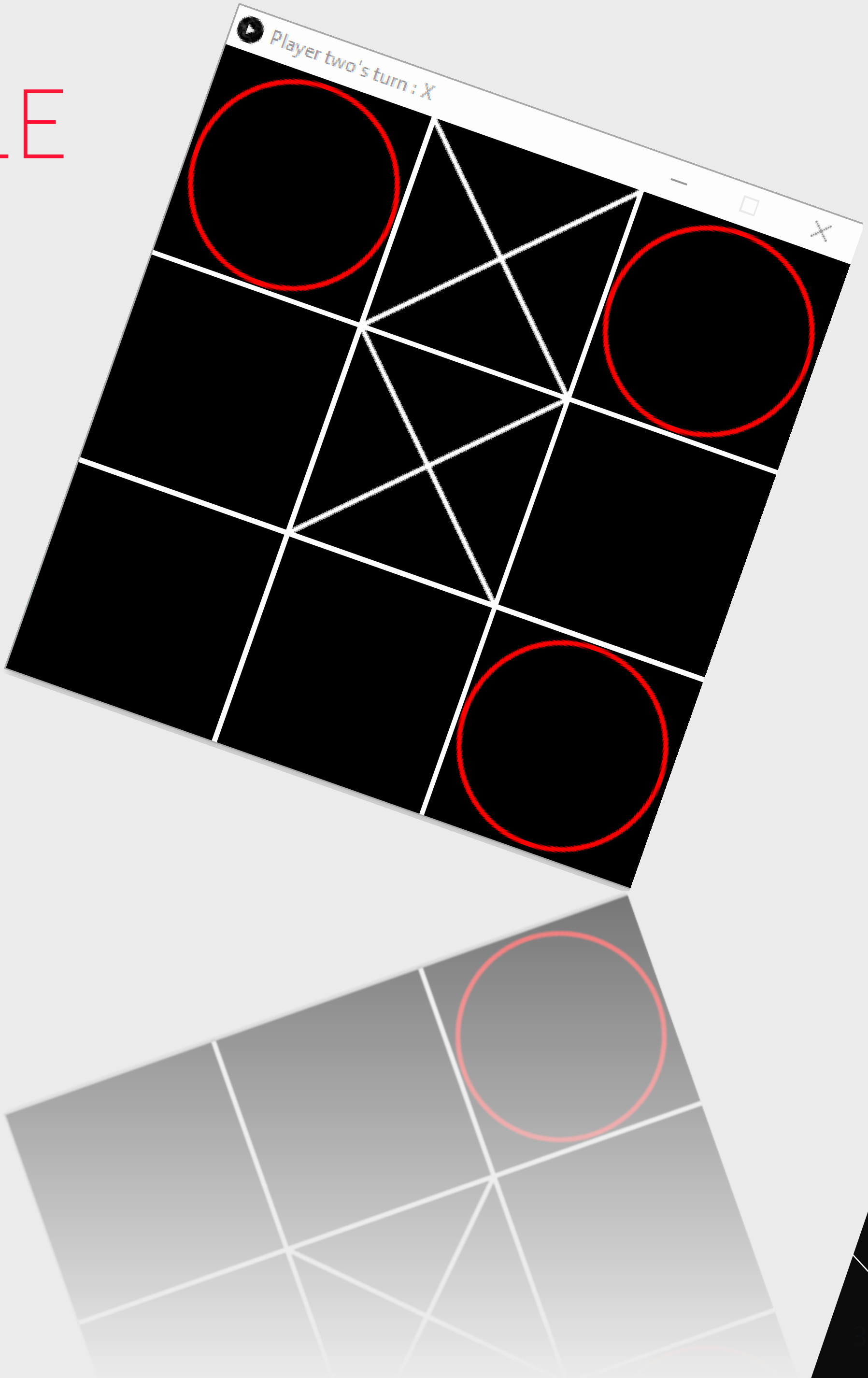
More info: https://processing.org/reference/Table.html

WRITING TABULAR DATA: EXAMPLE

For the **Tic Tac Toe game**, save the score for each player after each game:

```
if(winner > 0){
  setTitle("PLAYER " + winner + " WINS THE GAME!") ;
  gameOver=true;
  //save score for player
  if(winner==1) totalScorePlayer1++;
  else if(winner==2) totalScorePlayer2++;
```

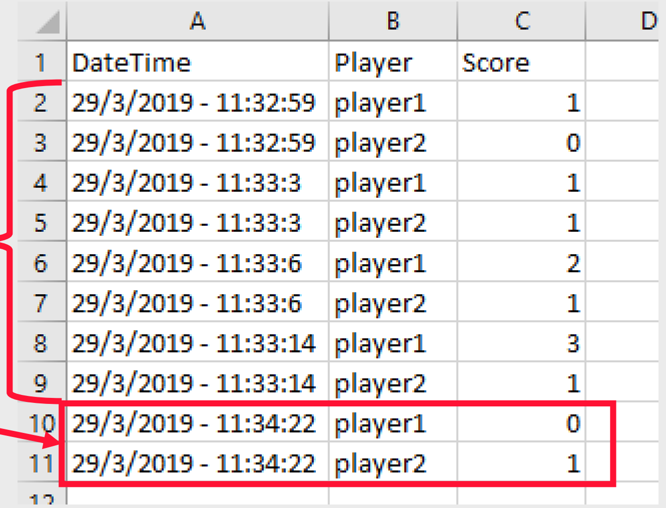
```
if(gameOver){
  imageMode(CENTER);
  image(imgGameover, width/2, height/2);
  saveScoresToFile();
  noLoop();
}
```



WRITING TABULAR DATA: EXAMPLE

```
void saveScoresToFile() {
  //create a new table
  Table newScores;
    newScores = new Table();
    newScores.addColumn("DateTime");
    newScores.addColumn("Player");
    newScores.addColumn("Score");

 //first load the existing table and add to new table
  Table oldScores = loadTable("data/scores.csv", "header");
  if (oldScores == null) {}
  else {
    TableRow oldRow;
    for (TableRow row : oldScores.rows()) {
      String dateTime = row.getString("DateTime");
      String player = row.getString("Player");
      int score = row.getInt("Score");

      oldRow = newScores.addRow();
      oldRow.setString("DateTime", dateTime);
      oldRow.setString("Player", player);
      oldRow.setInt("Score", score);
    }
  }
  //continue..
```

```
  //continue..
  TableRow newRow;
  String dt = day()+"/"+month()+"/"+year()+" - "+hour()+":"+minute()+":"+second();
  //add score of player 1
  newRow = newScores.addRow();
  newRow.setString("DateTime", dt);
  newRow.setString("Player", "player1");
  newRow.setInt("Score", totalScorePlayer1);
  //add score of player 2
  newRow = newScores.addRow();
  newRow.setString("DateTime", dt);
  newRow.setString("Player", "player2");
  newRow.setInt("Score", totalScorePlayer2);

  saveTable(newScores, "data/scores.csv");
//end of function
}
```



	A	B	C	D
1	DateTime	Player	Score	
2	29/3/2019 - 11:32:59	player1	1	
3	29/3/2019 - 11:32:59	player2	0	
4	29/3/2019 - 11:33:3	player1	1	
5	29/3/2019 - 11:33:3	player2	1	
6	29/3/2019 - 11:33:6	player1	2	
7	29/3/2019 - 11:33:6	player2	1	
8	29/3/2019 - 11:33:14	player1	3	
9	29/3/2019 - 11:33:14	player2	1	
10	29/3/2019 - 11:34:22	player1	0	
11	29/3/2019 - 11:34:22	player2	1	

Game played 4 times (with reset key)

After exiting the game, played once

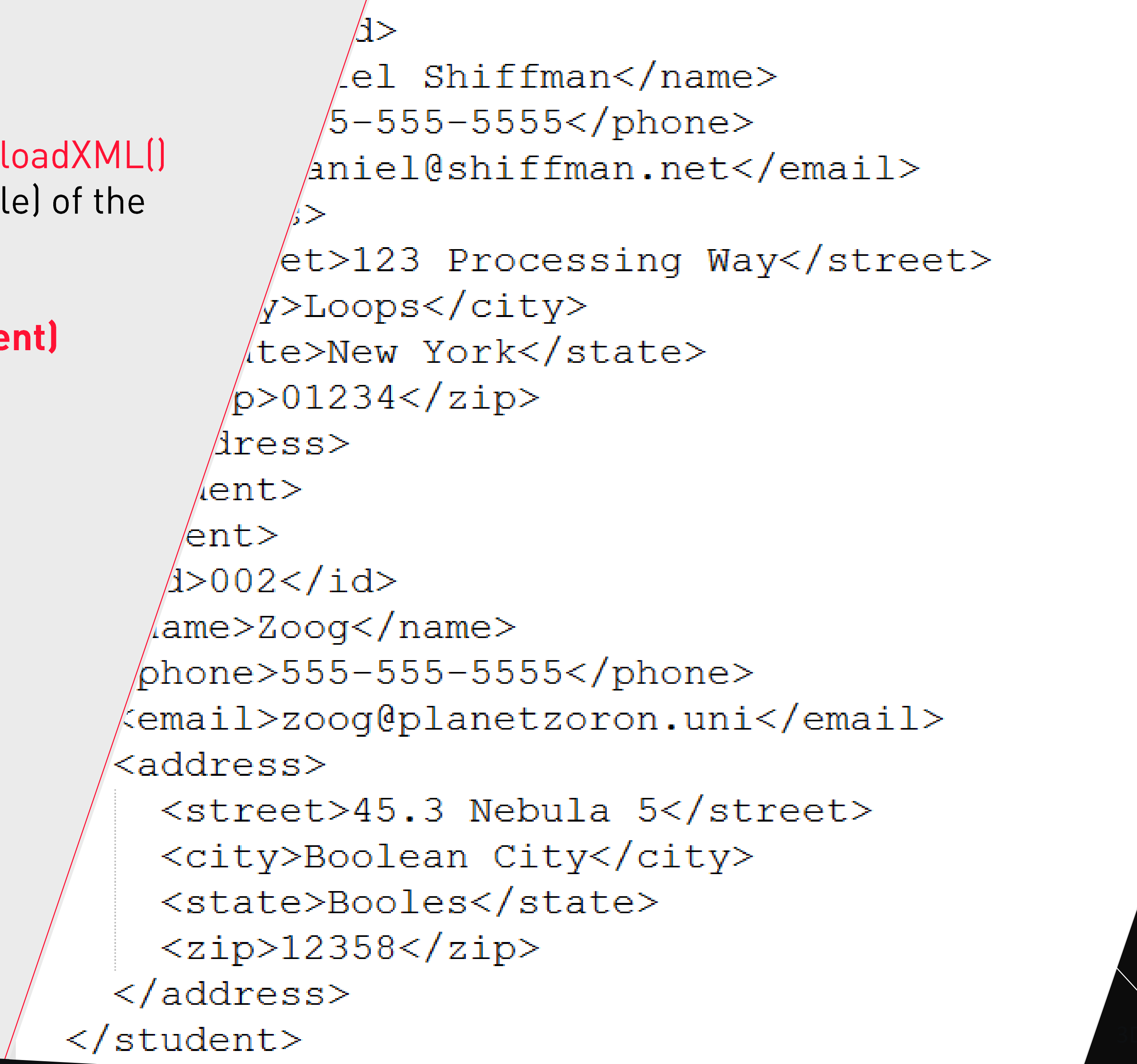
LOAD XML DATA

instead of loadStrings() or loadTable(), call the loadXML() method, passing in the address (URL or local file) of the XML document.

XML object → **XML tree (after load: root element)**

- Access children of an element → **getChild()**

- Access content:
    - **getContent()**
    - **getIntContent()**
    - **getFloatContent()**



```
d>
el Shiffman</name>
5-555-5555</phone>
aniel@shiffman.net</email>
>
et>123 Processing Way</street>
y>Loops</city>
te>New York</state>
p>01234</zip>
dress>
ent>
ent>
d>002</id>
ame>Zoog</name>
phone>555-555-5555</phone>
email>zoog@planetzoron.uni</email>
<address>
  <street>45.3 Nebula 5</street>
  <city>Boolean City</city>
  <state>Booles</state>
  <zip>12358</zip>
</address>
</student>
```

LOAD XML DATA: EXAMPLE

```
// An Array of Bubble objects
Bubble[] bubbles;

// An XML object
XML xml;

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

void draw() {
  background(255);
  // Display all bubbles
  for (int i=0; i < bubbles.length; i++) {
    bubbles[i].display();
    bubbles[i].rollover(mouseX, mouseY);
  }
}
```

```
void loadData() {
  // Load XML file
  xml = loadXML(“data.xml");
  // Get all the child nodes named "bubble"
  XML[] children = xml.getChildren("bubble");

  bubbles = new Bubble[children.length];
  /* The size of the Bubble array is determined by the total
     XML elements named “bubble.”  */

  for (int i = 0; i < bubbles.length; i++) {

    XML positionElement = children[i].getChild("position");
    /* The position element has two attributes: “x” and “y”. Attributes can be
       accessed as an integer or float via getInt() and getFloat().   */
    float x = positionElement.getInt("x");
    float y = positionElement.getInt("y");

    // The diameter is the content of the child named "diameter"
    XML diameterElement = children[i].getChild("diameter");
    float diameter = diameterElement.getFloatContent();

    // The label is the content of the child named "label"
    XML labelElement = children[i].getChild("label");
    String label = labelElement.getContent();

    // Make a Bubble object out of the data read
    bubbles[i] = new Bubble(x, y, diameter, label);
  }
}
```