INTRODUCTION

The course

- Lecture Tuesdays 10:45 to 12:30
- Tutorial Thursday 8:45 to 12:30 or Fridays 12:45 to 17:30
- Blackboard
- Textbook: Learning Processing, by Daniel Shiffman
INTRODUCTION

The people

- Lecturers
  - Ansgar Fehnker (Coordinator)
  - Angelika Mader
  - Marcus Gerhold

- Student Assistants
  - Margot Rutgers
  - Oliver Horst
  - Carmen Burghardt
  - Thijs Dortmann
  - Marrit Schellekens
  - Wybe Westra
  - Dennis Vinke
  - Lukas Bos
INTRODUCTION

The students
NASA High Speed Flight Station "Computer Room", Dryden Flight Research Center, 1949
Fig. 6. A portion of the worksheet for the Mathieu Functions tables.

Figure courtesy of the National Archives and Records Administration

Computing

The organization of the Math Tables Project.

Diagram courtesy of the WPA (795) Records, National Archives and Records Administration

COMPUTERS

Glen Beck (background) and Betty Snyder (foreground) with ENIAC, 1947 to 1955
UNIVERSAL TURING MACHINE

Turing's insight:

"It is possible to invent a single machine which can be used to compute any computable sequence.

If this machine $U$ is supplied with a tape on the beginning of which is written the S.D ["standard description"] of some computing machine $M$, then $U$ will compute the same sequence as $M$.

Turing 1936
STORED PROGRAM COMPUTER

EDVAC, 1949
Go to http://www.digibarn.com/collections/posters/tongues/ComputerLanguagesChart-med.png for a full-size copy of this chart.
PROGRAMMING
PROCESSING LANGUAGE
PROCESSING LANGUAGE

Aim and purpose

- For electronic arts, new media, and visual design
- A dialect of Java
- Designed by Casey Reas and Benjamin Fry
- Popularized by Processing Foundation (Reas, Fry and Shiffman)
- Textbook and additional material Learning Processing (Shiffman)
SETUP AND DRAW

Setup

- This method gets executed once
- Use it to setup the canvas
- Use it to create shared data structure
- Use it for everything you want to do at the beginning

Draw

- This method will be repeated as long as the program runs
- Use it to deal with interaction and animation
**SETUP AND DRAW**

**Setup**
- This method gets executed once
- Use it to setup the canvas
- Use it to create shared data structure
- Use it for everything you want to do at the beginning

**Draw**
- This method will be repeated as long as the program runs
- Use it to deal with interaction and animation

**Other standard methods**
- mousePressed, called when mouse is pressed
- KeyPressed, called when a key is pressed
PREDEFINED

- mouseX, mouseY
- pmouseX, pmouseY
- width, height
- keyCode
- keyPressed
- mousePressed
- mouseButton

BASIC TYPES

- int
- float
- double (rarely used in processing)
- bool
- char
VARIABLE DECLARATIONS

A variable is used to store information. It can contain one piece of information at a time.

Creating a variable is called **declaring a variable**.

When declaring variables, the programmer specifies the **type** of information to be stored.

The compiler will set aside that space in memory and give it the name we choose.

*Initialization* is putting a value into a variable when the variable is created.

Initialization is not required. But usually a good idea.
int numSegments;
int numSegments = 5;
DATATYPES

There are different **types** of data.

Every variable contains only data of one **type**.

Common **basic types**

- **int**  
  integers ..., -1, 0, 1, 2, ...

- **float**  
  floating point numbers 1.0, 3.14, -2.1

- **char**  
  characters a, b, c, ..., A, B, C, ..., !, @, #, $, ...

- **boolean**  
  true and false
LITERALS

- Literals are specific values used in the program.
- Like 1, 3.4, true, or ‘z’.
- They have a type, too.
- There is for example a difference between
  - the integer 2
  - the floating point number 2.0
  - the character ‘2’.
There is an important difference between a variable definition and an assignment statement:

```java
int numSegments = 6; // Variable definition
...
numSegments = 8; // Assignment statement
```

The first statement is a *declaration* of `numSegments`.

The second statement is an *assignment statement*. An existing variable’s contents are replaced.
THE ASSIGNMENT STATEMENT

The = in an assignment does not mean the left hand side is equal to the right hand side as it does in math.

= is an instruction to do something:

*copy* the value of the expression on the right

*into* the variable on the left.

Consider what it would mean, mathematically, to state:

```
numSegments = numSegments + 2;
```
For type float use +, -, *, / with the usual meaning.

For type int use +, -, * with the usual meaning.

Division / will be integer division.
- The result will be integer if numerator and divisor are integer.

Another useful operator is modulo %
- Computes the remainder of division
- Useful for integer values that are bound and repeat.

For other type +, -, *, / may or may not be defined. Result may be surprising. Check what they mean before using them.

11/4 is 2. Not 2.75.

E.g. 60 minutes
ARITHMETIC

\[ x = (3 + x) \times (7 + y) + z; \]
\[ y = \text{abs}(x - z) \mod 50; \]
\[ z = (x - y) / 2; \]

- some operations behave differently for different types, e.g., integer division
- not all operations defined for all types
- left of = is variables that get a new value
- right of = the expression to compute the new value
- computed top to bottom
VARIABLE NAMES

When you define a variable, you should pick a name that explains its purpose.

- For example, it is better to use a descriptive name, such as `numSegments`, than a terse name, such as `ns`.

- Use common Java/Processing camelCase style for variables.

- You do this for yourself, and your colleagues.
Comments are explanations for human readers of your code (other programmers).

The compiler ignores comments completely.

```c
float canVolume = 0.355; // Liters in a 12-ounce can
```
Comments can be written in two styles:

- Single line:
  ```
  float canVolume = 0.355; // Liters in a 12-ounce can
  The compiler ignores everything after // to the end of line
  ```

- Multiline for longer comments:

  ```
  /*
  This program computes the volume (in liters)
  of a six-pack of soda cans.
  */
  ```
In Processing you can declare CONSTANTS.

```
static final int ITEM_SIZE = 20;
```

Common in Java. Rarely done in Processing.

A good idea, also in Processing

- It tells that this value should not/cannot change
- It gives an explanatory NAME to a value.
- Makes code less obscure.
- Use ALL_CAPS for the name.
Every pair of curly brackets {} defines a Local Scope. Scopes can be nested. Each variable that is declared exists from the point of declaration until the end of the scope. You cannot use a variable outside of its scope. Declarations outside of any {} are in Global Scope. Variables in global scope can be used everywhere.
float topX;
float topY;

void setup(){
    ...
    topX = width/2;
    topY = height/2;
    ...
}

void draw(){
    float bottomX;
    float bottomY;
    ...
    bottomX = topX + width/10;
    bottomY = topY + height/10;
    ...
    topX = (topY + 1) % 200;
    topY = (topY + 1) % 100;
}
It is considered good practice to minimize the scope

- Few global variables.
- Except for data that has to be shared.
- Global constants are ok.
- Most variables should be local.

Global variables make understanding and debugging difficult.
Learning Processing

- http://learningprocessing.com/exercises/chp03/exercise-03-07-absolute-value
- http://learningprocessing.com/exercises/chp03/exercise-03-07-absolute-value
- http://learningprocessing.com/examples/chp03/example-03-06-interactive-zoog
- http://learningprocessing.com/examples/chp04/example-04-08-zoogvars
Programs must be written for people to read, and only incidentally for machines to execute.

- H. Abelson and G. Sussman
Great software, likewise, requires a fanatical devotion to beauty.

- Paul Graham
STYLE

A good program

→ Uses layout to convey the structure.
→ Different components are easily recognized.
→ The code adheres to a given coding guideline.
→ All output is spelled correctly, with a proper layout

A poor program

→ The layout is coincidental,
→ It is difficult to identify the structure.
→ Names are cryptic.
→ No discernible effort.
→ Spelling mistakes, poorly formatting of output.
STYLE

Make yourself familiar with professional style guides
Using a good IDE will help to program in style.
They will automatically “fix” many problems in your code.
TOPICS COVERED

Getting Started

Setup and draw

Variables
  - mouseX, mouseY
  - other system variables
  - variable declaration
  - types

Arithmetic

Predefined methods
  - abs, max, delay …

Topics
  - Look for examples
  - Look at manual

Programming as communication
  - Style
  - Naming
  - Comments
  - Constants
Lectures: Tuesday 11:45 to 12:39
Tutorials: Thursdays 8:45 to 12:30 or Friday 12:30 to 17:30
Assessment:
- Week 4 tutorial project assessed in week 5 (1/10)
- Multiple choice test (3/10)
- Final project (6/10)
The tutorial and lab sessions are there for practicing programming. They are an opportunity to get help and advice or to help and advise. Don’t use the lab time for other purposes (like facebook):

- Wastes valuable time for multitasking
- It is distracting to you, and others
- Creates a poor learning environment

If it turns out that you are using lab time and space for non-related issues, you may be asked to leave.
This is (not) a difficult course…

Students who do well do these things

- Attend lectures regularly
- Attend tutorials regularly
- Including doing prep work before tutorial
- Keep up with the textbook
- Ask questions
- Ask for help

Programming is a practical subject. Besides book knowledge you need to learn by doing.
GOOD LUCK