cis3.5 fall2009 lecture II.1

topics:

- internet overview
- creating graphics on the internet with “Processing”

resources:

- Processing web site: `http://www.processing.org/`
- getting started tutorial: `http://www.processing.org/learning/gettingstarted/`
- drawing tutorial: `http://www.processing.org/learning/drawing/`
- reference: `http://www.processing.org/reference/index.html`

what is a network?

- when computers talk to each other, this is called a **network**
- the network can have different kinds of computers and peripherals attached to it
- networks in which computers are physically connected to each other in the close geographical proximity are called **local area networks** (LANs)
- other networks are called **wide area networks** (WANs)
- the **internet** is a wide area network
- the internet is an *open system* = “a system whose architecture is not a secret”
- *protocol* = set of rules for how computers communicate with each other; for example:
  - TCP: transmission control protocol (computer ↔ computer)
  - IP: internet protocol (computer ↔ computer)
  - HTTP: hypertext transfer protocol (computer ↔ browser)
  - FTP: file transfer protocol (computer ↔ computer)
  - SMTP: simple mail transfer protocol (computer ↔ mail client)

what is the internet?

- history
  - ARPAnet (circa 1971): used “NCP”
  - TCP (1974): hardware independent, open
  - internet was standardized in September 1981
- the internet is NOT the world-wide web (WWW)
  - the idea of the world-wide web was conceived by Tim Berners-Lee
  - developed and discussed at CERN in Switzerland from about 1989
  - made public in 1994
  - the WWW uses the internet, but is not the internet itself—it is a way of organizing and viewing data that is accessible through the internet

some internet facilities

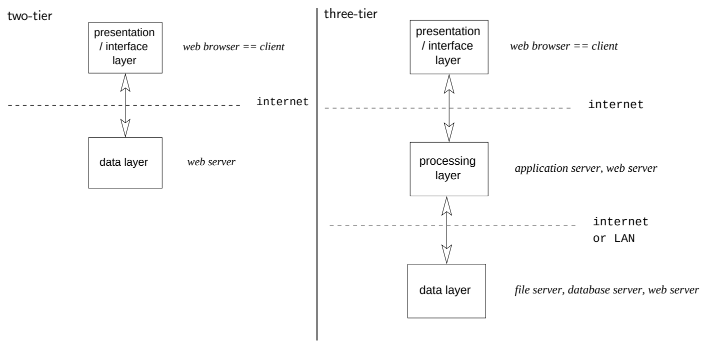
- the world wide web
  - HTML = hypertext markup language
  - *hyperlink*
  - *browser*
  - *web page*, *web site*, *web server*
- ftp (file transfer protocol)
  - *download*
  - *upload*
- email
- newsgroups
  - *posting*
  - *thread*
- mailing lists

clients and servers

- *server*:
  - computer on a network which carries out some **service** for another computer
- *client*:
  - the other computer for whom the server is carrying out the service
- types of servers:
  - *file server*
    - provides files for clients
  - *database server*
    - specialized file server that provides databases (structured files) for clients
  - *web server*
    - specialized file server that provides files that make up the components of a web site,
    - for example: HTML documents, CSS files, images, video clips, etc.

- *groupware server*
  - manages scheduling for individuals and groups of co-workers/collaborators
  - provides reports (e.g., billing) for collaborators
  - supports mailing lists for collaborators
  - e.g., Lotus Notes
- *mail server*
  - sends mail
  - receives mail
  - stores mail
- *application server*
  - provides access to particular applications
  - e.g., game server

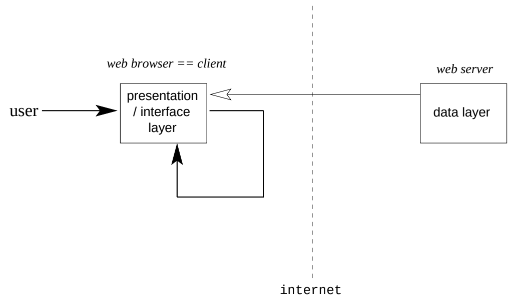
client-server architectures



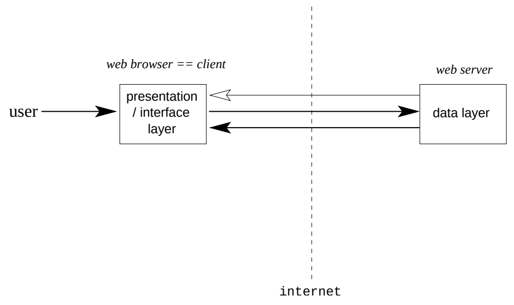
- isolates data storage technology
- places more burden on server (instead of client)
- distributes tasks amongst server(s)
- follows object-oriented and modular programming paradigms

interactive web programming

- user initiates some action
- which causes the web page to change in some way
- changes can happen locally, on the "client"



- changes can happen on the server and be reflected on the client



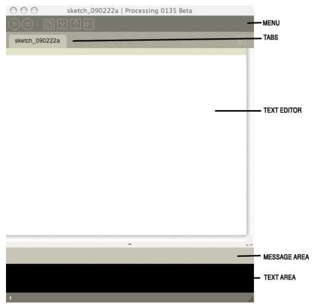
interactive web programming languages

- Javascript
  - scripting language based on Java
  - write programs using a text editor (like with HTML)
  - run programs in a browser
- Processing
  - language originally written for artists
  - programs in Processing are called *sketches*
  - text-based programming language
  - write and run using an *integrated development environment (IDE)* that is part of Processing
  - programs can also be saved as *applets* and run inside a browser

Processing: getting started

- the Processing desktop icon looks like this:

- the Processing window looks like this:



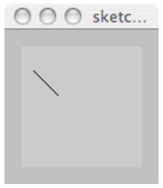
- interface buttons:

▷	**run**	compiles the code, opens a display window and runs the program.
□	**stop**	terminates a running program.
🗋	**new**	creates a new "sketch" in the current window.
⇧	**open**	provides a menu with options to open files from your "sketchbook", an example or another a sketch on your computer.
⇩	**save**	saves the current sketch with its current name and location.
⇨	**export**	exports the current sketch as a **Java** "applet".

first program

- first program, which draws a line:

```
line( 10, 20, 30, 40 );
```

- looks like this: 

- add to it, which draws a point:

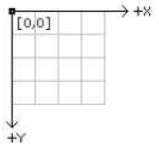
```
point( 50, 50 );
```

- add to it, which changes the background color:

```
background( #00ff00 );
```

coordinate system

- all graphics are drawn using the following coordinate system:

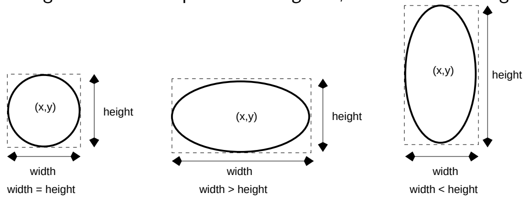


- think of it like a piece of graph paper
- a *point* fills in one cell on the graph paper
- a *line* fills in multiple cells, from one end-point of the line to the other

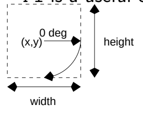
drawing things

- `point( x, y )`
  – draws one point (looks like a dot...)
- `line( x1, y1, x2, y2 )`
  – connects two points
- `triangle( x1, y1, x2, y2, x3, y3 )`
  – connects three points
- `quad( x1, y1, x2, y2, x3, y3, x4, y4 )`
  – connects four points
- `rect( x, y, width, height )`
  – origin + extent; square if width=height

- `ellipse( x, y, width, height )`
  – origin: center of ellipse's *bounding box*; circle if width=height



- `arc( x, y, width, height, start, stop )`
  – origin: center of arc's bounding box (see ellipse)
  – `start` and `stop`: can be whole numbers (`int`) or real numbers (`float`); expressed in degrees or radians, depending on current angle mode; 0 is due east; measured clockwise
  – `PI` is a useful constant



- attributes
  - `strokeWeight()`
    - line thickness
  - `strokeJoin()`
    - square (MITER, default), blunt (BEVEL), rounded (ROUND)
  - `strokeCap()`
    - SQUARE, PROJECT, ROUND (default)

programming basics

- each line contains a *statement*
- statements end with a semi-colon (;)
- *comments* are contained within `/**` and `*/`
- *functions*
  - provide a way to *modularize* code
  - makes it easier to read and re-use
  - also allows you to specify content for functionality built in to Processing
  - for example:

```
void draw() {
   line( 10, 20, 30, 40 );
}
```

  - `void` keyword that indicates a function which returns nothing
  - `draw()` = the name of the function
  - curly brackets ({ and }) delineate the beginning and end of the function
  - with Processing, your sketch has to use no functions or all functions

sample program

```
void setup() {
  background( #ffffff );
}

void keyPressed() {
  background( #0000ff );
}

void draw() {
  line( 10, 20, 30, 40 );
  point( 50, 50 );
}
```

keyboard interaction

- `keyPressed()`
  - handles behavior when user presses a key down
- `keyReleased()`
  - handles behavior when user releases a key
- `keyTyped()`
  - handles behavior when user types a key (press and release)
- `key`
  - indicates which key was pressed/released/typed
  - equals `CODED` when special key is pressed/released/typed, like an arrow key, shift, control, alt, etc.
- `keyCode`
  - indicates special key: UP, DOWN, LEFT, RIGHT, ALT, CONTROL, SHIFT

making decisions

- one decision—IF something is true:

```
if ( test ) {
  statements
}
```

- two decisions—IF something is true...or ELSE:

```
if ( test ) {
  statements
}
else {
  statements
}
```

- joint decisions—IF something is true OR something else is true:

```
if (( test1 ) || ( test2 )) {
  statements
}
```

modified sample program

```
void setup() {
  background( #ffffff );
}

void keyPressed() {
  if ( key == 'R' ) {
    background( #ff0000 );
  }
}

void draw() {
  line( 10, 20, 30, 40 );
  point( 50, 50 );
}
```

modified `keyPressed()` function

```
void keyPressed() {
  if ( key == 'R' ) {
    background( #ff0000 );
  }
  else if ( key == 'G' ) {
    background( #00ff00 );
  }
  else if ( key == 'B' ) {
    background( #0000ff );
  }
}
```

multiple conditions

- two vertical bars (||) mean "OR"

```
void keyPressed() {
  if (( key == 'R' ) || ( key == 'r' )) {
    background( #ff0000 );
  }
}
```

- you can also just list two conditions

```
void keyPressed() {
  if ( key == 'R' ) {
    background( #ff0000 );
  }
  else if ( key == 'r' ) {
    background( #ff0000 );
  }
}
```