**Lecture 0- Scratch**

**Lecture 1- C**

* Compiler converts source code to machine code.
* Correctness, design, style
* Terminal:
  + Improves efficiency
  + **code filename.c** opens new file
  + **make hello** converts hello.c to hello(machine code)
  + **./hello** runs the code
* Syntax highlighting
* Code:

**#include <stdio.h> //standard input output**

**int main(void)**

**{**

**printf(“Hello, world\n”); // to display**

**return 0;**

**}**

**\n** is escape sequence that changes line

stdio.h is header file which contains blocks of code that allows you to use standard input output codes.

* Consider this block of code:

**#include <stdio.h>**

**int main(void)**

**{**

**printf(“You got 100%\n”);**

**}**

This will generate an error which can be solved using double %%.

* Conditional:

Code snippet:

if (x<y)

{

printf(“ x is less than y.\n”);

}

else if(x>y)

{

printf(“ x is greater than y.\n”);

}

else

{

printf(“ x is equal to y.\n”);

}

* Comparing user input:
* #include <stdio.h>
* int main(void)
* {
* int x,y;
* printf("What's x?: ");
* scanf("%d",&x);
* printf("What's y?: ");
* scanf("%d",&y);
* if(x>y)
* {
* printf("x is greater than y.");
* }
* else if(x<y)
* {
* printf("x is less than y");
* }
* else
* {
* printf("x is equal to y");
* }
* }
* Loop:

**for (i=0;i>3;i++)**

**{**

**//code to do something**

**}**

Repetition is discouraged.

**int i=0;**

**while(i<3)**

**{**

**//code to do something**

**i++; //This is same as i=i+1 or i += 1**

**}**

infinite repetition:

**while(true)**

**{**

**//block of code**

**}**

**Boolean values are inside <stdbool.h> now**

* Useful Command line instructions:

**mv** filename1 filename2 to rename 2 to 1

**ls** can list things

**cp** for copy

**ls** for list

**mkdir** for make directory

**rm** for remove

* Adding **const** in front of variable declaration makes it constant.
* **do**

**{**

**//block of code**

**} while (condition);**

* For loop is repetition controlled, while loop is entry controlled and do-while loop is exit controlled.
* //Single line comment
* It is good idea to use comment for pseudo code
* Function definition:

**return\_value function\_name(collection of datatype+formal parameters)**

**{**

**//block of code**

**return var;**

**}**

* It is better to write functions after main(){} function and use function declaration:

**return\_value function\_name ( collection of data type);**

before main() function.

* Integer (32bits), long(64bits) %d, %li
* %f for float (32 bits)
* Truncation error: Try dividing integer by integer and store in float (all variables)
* Typecasting:

float z = (float) x/y

* printf(“%.20f\n”,z); will display upto 20 digits but inaccurate.
* Double will provide more precision (%f is used still)
* Void is a type, but not a data type.
* bool data type is found in <stdbool>, also in cs50.h
* string data type is derived data type that is just a sequence of characters.
* There are also structures which is derived data type that can be made up of various types of data.
* declaration, assignment and initialization (declaration + assignment)
* Operators in C:

+, - ,\*, /, % (mod),

shorthand for x = x+5 is x+=5.

Increment x++ increases value of x by 1.

&& (and), || (or),! (not) are logical operators

every non-zero value is treated as true in operator.

<, >, ==, >=, <=, != are relational operators.

* Conditional statements:

if, if-else, if-else if-… else

swich(variable)

{

case value1:

//block of code1

break;

case value2:

//block of code 2

break;

……

default:

//default code doesn’t require break

}

Ternary operator: (x = condition?value if true: value if not true)

* Loops:

A variable only exists inside the most recent curly bracket

while(Boolean-expression)

{

//code to run till it is true

}

while(true)

{

//code with some form of break

}

do

{

}while(condition);

for(initialization, check, update)

{

}

**Week 2: Arrays**

* Generally, how a compiler works.
* How to debug your code using four methods.
* How to utilize arrays within your code.
* How arrays store data in back to back portions of memory.
* How strings are simply arrays of characters.
* How to interact with arrays in your code.
* How command-line arguments can be passed to your programs.
* The basic building-blocks of cryptography

Compilers: clang, gcc.

**clang –o hello hello.c**

for using cs50 library during compilation:

**clang –o hello hello.c –lcs50**

make is doing this all under the hood.

* Preprocessing:

includes files and definitions required for processing

* Compiling:

our code is changed to assembly language

* Assembling:

assembly to binary

* Linking:

header files, our code all these binaries combined

* Decompiling is a thing:

Exploits, debugging, copyrighting.

Compiling doesn’t retain your actual code but its action.

i.e. Reverse engineering is super duper hard.

* Debugging:

Harvard mark-II contained first physical bug.

How to debug: (logical bugs)

* + Using printf to check if intermediate steps are working properly
  + Debugger
    - You can add a breakpoint by clicking next to line number in vscode
    - you can use debug50 filename which will give you options to go forward or back …
  + Rubber duck debugging
    - Explain things to someone or something which might just click things for you.
* bool uses 1 byte, int 4bytes, long 8bytes, float 4bytes, double 8bytes. char 1byte, string can have arbitrary number of bytes.
* Array – collection of primitive data type

int arr[2] is collection of array with 3 integer value (at position 0,1,and 2)

arr[0] can be used to access first integer of that array

* Array are contiguously allocated.
* You can pass arrays as argument.

float average(int arr[]);

* string is array of characters.

It contains \0 value at end (just 0 character) to indicate end of string.

It is called NUL (not NULL) character.

* string.h contains useful string functions.
* ctype.h contains useful functions like isupper(), islower(), toupper()etc.
* Uppercase and lowercase letters are 32 values apart.

(A=65,Z=90 and a=97,z=122)

* for(int i=0; i < strlen(s); i++){//code here}

can be written in better way as:

for(int i=0,n=strlen(s); i < n ; i++){//code here}

as initialization part is executed only once which means strlen() is called only once. (Some compilers recognize this and optimize but its better to write better code)

* Consider this example:

#include <cs50.h>

#include <stdio.h>

int main(void)

{

string name = get\_string(“What’s your name? “);

printf(“hello, %s\n”, name);

}

Here main is getting void (i.e nothing) as command line argument

So, that means it can accept command line argument:

#include <cs50.h>

#include <stdio.h>

int main(int argc, string argv[])

{

printf(“hello, %s\n”, argv[1]);

}

If the file is named as hello, you can run this as make hello and ./hello Bishal and get the output hello, Bishal

argv[0] contains program name. Try using argv[0] in the above program

You can use argv[1] and argv[2] in the above program and ./hello Bishal Tiwari to get output hello, Bishal Tiwari

Consider this program now:

#include <cs50.h>

#include <stdio.h>

int main( int argc, string argv[])

{

if(argc == 2) //if the user passed two words as argument in Command line (including file name)

{

printf(“hello, %s\n”, argv[1]);

}

else

{

printf(“hello, world\n”);

}

}

* Exit status:

int in front of main is for exit status that main can return.

i.e. you can return 1 or 0 or anything etc.

You can see this exit status after ending program using **echo $?**

You can use different return statement for different exit condition so as to know what the program did or didn’t do before exiting.

Encryption and decryption intro only.

**Week 3: Algorithm**

* Algorithm = series of steps
* Big-Oh for worst case

Big-omega for best case

Big-theta for average case (tightly bound)

* User defined data type:

**typedef struct**

**{**

**//collection of data**

**}**

**name\_of\_data\_type;**

Eg:

typedef struct

{

string name;

string number;

}

person;

Now you can use new data type called person as:

person person1;

To access data of the person:

person1.name for name

person1.number for number

* You can create array of structure too.
* Selection sort, bubble sort, merge sort

O(n2) for selection sort. (same for omega)

O(n2) for bubble sort. (n for omega)

O(nlogn) for merge sort. (same for omega)

**Week 4: Memory**

Pointers: Variable that contains address of some value.

Eg:

int n = 50;

int \*p = &n; // p is pointer that holds address of a integer

//&n gives address of n

print(“%p\n”,p); // %p for address so it prints address of n

printf(“%p\n”,&n); // This works the same way

printf(“%i\n”,p); // not this not this

printf(“%d\n”,p); // this doesn’t either

printf(“%d\n”,\*p); // this gives 50

Pointers take 8 bytes in modern computer. (DAMN!!)

String is actually a pointer to first character of a character array.

i.e. typedef char \*string; // i.e. char \* renamed to string.

& (reference) operator gives address of the variable, \* (dereference) operator gives value at address contained by the variable.

Now string:

int main(void)

{  
 char \*s = “HI!”; // is same as using string s = “HI!” with string.h

printf(“%p\n”,s); // This displays address of s (i.e. first character)

printf(%p\n”,&s[0]); //This does the same

printf(%c\n”, s[0]); // This prints H (first character)

}

Pointer arithmetic:

int main(void)

{  
 char \*s = “HI!”; // is same as using string s = “HI!” with string.h

printf(%c\n”, s[0]); // This prints H (first character)

printf(“%c\n”, \*s); //This also prints H(using dereference operator

printf(“%c\n”, s[1]); //This prints I (second character)

printf(“%c\n”, \*(s+1)); //This also prints the second character

}

danger of using string is that you could use pointer arithmetic to access beyond the string. e.g.: \*(s+1000000) for example

Try this:

int main(void)

{

char \*s = “HI!”;

printf(“%s\n”, s); //This displays HI!

printf(“%s\n”, s+1); //This displays I!

}

You could also do this for integer array too.

malloc and free:

malloc allows you to allocate memory of size you provide and return starting memory address.

i.e. char \*t = malloc(4); will allocate 4 bytes of memory to you and return starting address.

free frees memory.

i.e. free(t); //Always free what you get.

You can also use:

int \*x = malloc(3 \* sizeof(int)); for integer array of size 3

**valgrind:**

This looks at your code as it runs and finds memory related bug in the code.

e.g. valgrind ./program\_name (use compiled program)

**Garbage value:** If variables are not initialized but declared, they still have values.

**Some problematic code:**

int main(void)

{  
 int \*x;

int \*y;

x = malloc(sizeof(int));

\*x = 42;

\*y = 13; //This line is a problem. //Y isn’t pointing anywhere so you can’t put a value there

y = x;

}

This could be solved by swapping last two lines.

**Memory is divided to segments:**

Machine code

Globals (Global variables)

Heap (big chunk of memory that can be used) (malloc grabs memory from here)

(Grows downward)

Stack (grows upward) (this is used when function is called.

Heap overflow and stack overflow can occur.

**Pass by value and Pass by reference:**

Consider the function:

void swap(int a, int b)

{

int temp = a;

b = a;

b = temp;

}

When called as swap(x, y), It doesn’t actually swap the values as the function gets own copy of variables (a and b) with values of x and y. So, after function returns, there is no change in value of x and y.

This could be done instead:

void swap( int \*a, int \*b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;  
}

And use swap (&x, &y) to call the function. By doing this, we are passing actual value’s address which is called pass by reference.

User input:

#include <stdio.h>

int main(void)

{  
 int x;

printf(“x: “);

scanf(“%i”, &x); // This is passing value by reference too.

printf(“x: %i\n”, n);

}

For string:

int main(void)

{

char \*s = NULL;

printf(“s: “);

scanf(“%s”, s); // S Is already address

printf(“s: %s\n”, s);

}

This doesn’t work as we haven’t used malloc() so damn.

int main(void)

{

char s[4];

printf(“s: “);

scanf(“%s”, s); // S Is already address

printf(“s: %s\n”, s);

}

This works (properly) for just 3 letter word. So, problem.

**Week 5: Data Structures**

Queue: First in first out line usual and fair queue in real life.

Stack: Last in first out like a stack of plates.

Lets recall:

int \*tmp = malloc( 4 \* sizeof(int)); allocates memory space to store 4 int values and returns the address.

free(tmp); frees the allocated memory

You can use tmp[0] to access first element of the array, temp[1] for second and so on.

You can create a new pointer new: int \*new

and use new = tmp to point to same thing as tmp.

malloc() will return NULL if it can’t assign memory due to some reason (like memory full).

Memory leak => not freeing up memory that you allocated.

. for accessing members of a structure variable.

\* for dereference (also declare) pointers. (i.e. go to a pointer)

. and \* combined can be done using -> symbol.

typedef struct node

{

int number;

struct node \*next;

}

node;

By doing this you can create struct node and use node pointer within that.

typedef struct

{

int number;

node \*next;

}

node;

This wouldn’t work.

(\*n).number = 1; Take you to structure pointed by n and changes number of that struct to 1.

This is same as:

n -> number = 1;

You should use n->next = NULL for termination in next node.

**Week 6: Python**

python filename.py will run the program.

print(“Hello, world!”)

I.e. Semicolon gone. New line by default. It’s not printf anymore.

def function\_name(parameters):

…..

return sth\_to\_return

i.e. You don’t need data type

file = open(dictionary, “r”) #opening a file in read mode

close(file) to close the file

Python is slower because work is being done for you and most of the time that means not really optimized to your needs.

if x < y :

print(“x is less than y.”)

Indentation (4 spaces as norm) is necessary.

i ++ doesn’t work in python

i += 1 does work.

i = 0

while i<3:

print(“Hello”)

for i in [0, 1, 2]:

print(“Hello”)

for i in range(3):

print(“Hello”)

for i in range(0,4):

print(“Hello!)

while True:

print(“meow”)

floating point imprecision is an issue.

Integer overflow is not a thing

if condn1:

code

elif condn2:

code

else:

code

You can use (or, and) in condition.

You can use single or double quote in python.

if answer in [“answer1”, “answer2”]:

code

Some functions are built in to data types. They are called methods.

eg:

if answer.lower() in [“y”,”yes”]:

print(“Yes”)

Eg:

def meow():

print(“meow”)

for i in range(3):

meow()

It’s kinda tradition to make your own main function like:

def main():

#main block of code

otherfunction()

#remaining part of code

def otherfunction():

#other block of code

main() #you need to call as main is treated as normal function too.

With parameter:

def main():

meow(4)

def meow(n):

for i in range(n):

print(“meow”)

main()

**Week 7: SQL**

Lambda function in python:

lambda name\_of\_argument : return\_Value (not return keyword)

SQL: (sequel or ess-cue-ell)

C – Creating data

R – Reading data

U – Updating data

D – Deleting data

Creating a new database file:

sqlite3 table\_name.db

To turn csv file to database:

.mode csv

.import file\_name.csv table\_name

.schema returns schema (i.e. description of all the tables blah blah)

Now for sql commands:

SELECT columns FROM table; # use capital for sql keywords (convention)

SELECT \* FROM table; # open a database and print all the stuff

SELECT language from table //for language column only

SELECT COUNT(\*) FROM table //for count

SELECT DISTINCT(languages) from table //no repetition

SELECT COUNT(DISTINCT(languages)) from table // damn this is handy

SELECT COUNT(DISTINCT(languages)) AS n FROM table // woo

OTHER STUFF:

AVG, COUNT, DISTINCT, LOWER, MAX, MIN …

Some Predicates are: (condition kinda thing)

WHERE, LIKE, ORDER BY, LIMIT, GROUP BY …

Example:

SELECT COUNT(\*) FROM table WHERE language = ‘C’; //use single quote

SELECT COUNT(\*) FROM table WHERE language = ‘C’ AND problem = ‘Mario’;

SELECT language, COUNT(\*) FROM table GROUP BY language; //whoah

SELECT language, COUNT(\*) FROM table GROUP BY language ORDER BY COUNT(\*) ASC; //DESC for descending order

SELECT language, COUNT(\*) FROM table GROUP BY language ORDER BY COUNT(\*) DESC LIMIT 1; // just show one (top)

To insert data:

INSERT INTO table (column, ….) VALUES(value,….);

INSERT INTO favorites (language, problem) VALUES(‘SQL’, ‘Fiftyville’);

To update:

UPDATE table SET column = value WHERE condition;

UPDATE table SET language = ‘C++’ WHERE language = ‘C’;

To delete:

DELETE FROM table //deletes all

DELETE FROM favorites WHERE problem = ‘Tideman’; //delete data with predicate

Entity relationship diagram.

Data types in SQL:

BLOB – binary large objects

INTEGER - integer

NUMERIC – numbers formatted specially (Eg: date)

REAL - float

TEXT – text

Constraints NOT NULL, UNIQUE

Primary key: unique identifier of data

Foreign key: connection between two table

SELECT title FROM shows WHERE id IN(SELECT show\_id FROM genres WHERE genre = ‘Comedy’) ORDER BY title LIMIT 10;

Indexes: For faster access

.timer on //for timing

SELECT \* FROM shows WHERE title = ‘The Office’;

CREATE INDEX title\_index ON shows (title); //shows is name of table (Creates B-tree)

//Only optimize common cases (memory needs money bro)

from cs50 import SQL

db = SQL(“sqlite:///favorites.db”) //open favorites.db using sqlite

favorite = input(“Favorite: “)

rows = db.execute(“SELECT \* FROM favorites WHERE problems = ‘Mario’”) executes sequel query

for row in rows:

print(row[“Timestamp]”)

rows = db.execute(“SELECT COUNT(\*) AS n FROM favorites WHERE problems = ‘Mario’”) executes sequel query //db.execute returns list of dictionary

for row in rows:

print(row[“n”])

favorite = input(“Favorite: “)

rows = db.execute(“SELECT COUNT(\*) AS n FROM favorites WHERE problem = ?”, favorite) //? is placeholder

print(rows[0][“n”])

race conditions: transactions solves this

SQL injection attacks: Solution is to not use f-string but a placeholder (?)

Little Bobby Table

.shell clear //to clear

Week 7 reference:

.tables //gives you list of tables

.schema //for table detail

sqlite3 database\_name.db //opens the database

SELECT \* FROM database\_name.db; //gives all the data

//I.e. select all columns from the database

SELECT \* FROM database\_name.db LIMIT 1; //gives only 1 row

SELECT column\_name FROM database\_name.db; //give that one column

SELECT column\_name FROM database\_name.db ORDER BY column\_name2 ASC;

//list the values with column\_name ordered using column\_name2 columns value

SELECT column\_name FROM database\_name.db WHERE condition\_here;

//You can use AND, OR to connect conditions

SELECT AVG(column\_name) FROM database\_name.db;

//Gives average of values from that column

SELECT column\_name FROM database\_name. db WHERE condition;

condition can be of form artist\_id = (another\_query\_here)

//nesting queries

SELECT name FROM songs WHERE name LIKE '%feat.%';

//to find names with feat. in it

ORDER BY rating DESC, title ASC;

//Sort by rating first.

//If same, sort by title

SELECT col1, col2 FROM table1

JOIN table2

ON table1.col1 = table2.col3

// joins two tables making table1.col1 corresponding to table2.col3

// you will have access to all the columns of both table

// col1, and col2 will be displayed

**Lecture 8: HTML, CSS and JavaScript**

TCP/IP: Transmission control protocol and Internet Protocol allows data communication.

DNS: Domain Name System.

HTTP(S): Hypertext Transfer Protocol (secure) /scheme.

What to do while accessing the webpage.

HTTP can be considered an internet service. (It’s like application level Protocol)

[www.example.com](http://www.example.com)/path

www is host name

example.com is called domain name

.com is Top level domain(TLD)

Inspect mode:

Network tab allows you to look at what message the browser is actually sending.

curl –I url on your terminal to see the communication.

Error codes:

200, 301, 302, 304, 307, 401, 403, 404, 418, 500, 503

HTML:

Hyper Text Markup Language.

Tags + Attributes

<!DOCTYPE html>

<html lang = “en”>

<head>

<title>

hello,title

</title>

</head>

<body>

hello,body

</body>

</html>

code hello.html and the above code.

Web Server.

In CS50 you can do:

http-server in cs50. Server is actually a program that’s on powerful device.

CSS = Cascading Style Sheets

Inside style tag or different file

Bootstrap – framework for CSS and JavaScript (just use link tag)

JavaScript:

Programming language.

Can listen for events and do the task.(Event listener)

Inside script tag or different file.

Section:

HTTP : Hyper Text transfer protocol.

Hyper Text: It referred to text with links. Interactive text.

Transfer Protocol: Rules for transfer

HTTP:

Request is sent to server, the request is sent in proper form by HTTP.

Now server sends something back.

Error 404: The requested webpage was not found.

Error 500: Internal Server Error (Server ran into error)

http-server creates own server in cs50 codespace.

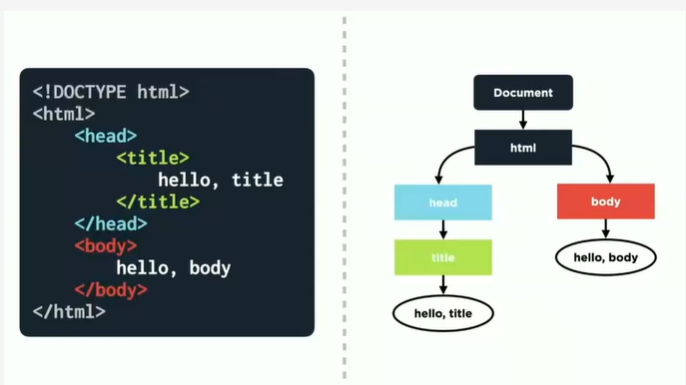
HTML: Hyper Text Markup Language.

Markup language specifies structure.

<!DOCUMENT html> tag to denote it’s html document.

<html> </html>

First one is opening tag. Second one is closing tag.



<head></head> tag contains some description about the page which isn’t actually shown in the page.

<table></table> to create data

<tr></tr> to add row. inside it <td></td> for table data.

You can put <h1> </h1> tag inside <head> tag but that wouldn’t be good practice.

Tags like <main></main> are semantic tags that let you to kind of categorize the elements.

CSS: Cascading Style Sheet

Cascading => Inheritance i.e. style to parents is applied to children. (with exception)

Structure:

selector

{

property: value;

}

For id you need to use # and for class you need to use . (dot)

Use id for individual and class for grouping

JavaScript:

<script>

let team1 = 0;

let team2 = 0;

document… = function(){

…

}

document… = function(){

…

}

</script>

example:

document.querySelector(“#add1”).onclick = function()

{

team1 = team1 + 1;

document.querySelector(‘#team1\_score’).innerHTML = team1;

}

document.querySelector(“#add1”).addEventListener(‘click’, function() {});

Week 9: Flask

Flask is a framework. Jinja is syntax.

flask run in terminal to start server.

you must have:

app.py file

templates / folder contains html file for now

requirements.txt contains third party libraries required

static/ static content goes here by convention (images,js file, css file etc)

Example: (inside app.py)

from flask import Flask, render\_template, request

app = Flask(\_\_name\_\_)

@app.route(“/”)

def index():

if “name” in request.args:

name = request.args[“names”]

else:

name = “world”

return render\_template(“index.html”, namehere = name)

in html file use hello, {{namehere}}

After this, if you execute flask run in terminal, add ?name=Bishal in the url, you get hello, Bishal

OR,

from flask import Flask, render\_template, request

app = Flask(\_\_name\_\_)

@app.route(“/”)

def index():

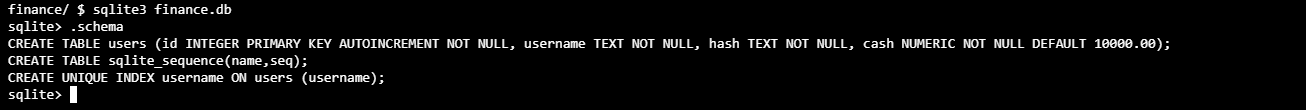
name = request.args.get(“name”, “world”)

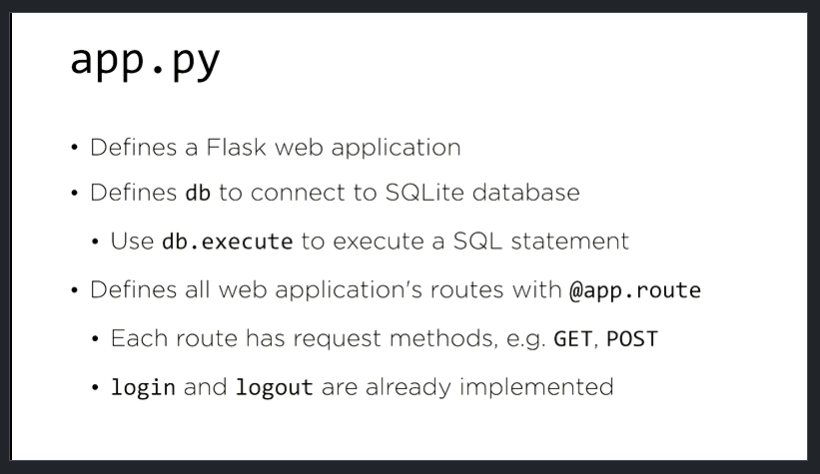
return render\_template(“index.html”, namehere = name)

You can create a layout using jinja syntax

AJAX: no submission required

JSON: JavaScript Object Notation

Problem set 9:

* register (done)
* quote (done)
* buy (done)
* index
* sell
* history (done)
* personal touch (not doing I guess)

HELLO