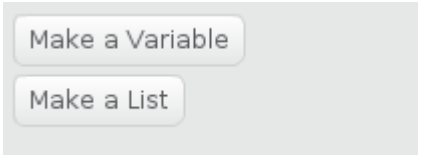
Week 3 Lecture 1

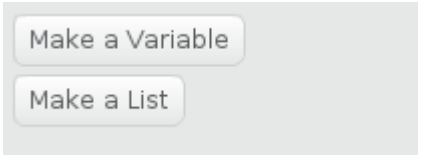
Basic C



Data in C

```
/* Data */
int integer = 1;
char character = 'a';
char *string = "Hello world\n";
```

- Scratch is a modern Object-Oriented language
  - It hides the details of the computer
  - Scratch has variables and lists
    - The language remembers what kind of thing is in the variable or list
- C is an older languages
  - It's computational model is closer to the computer's
  - C has data types representing different interpretations of the bits.
    - You need to tell C what is in the variable or list



Types

```
/* Data */
int integer = 1;
char character = 'a';
char *string = "Hello world\n";
```

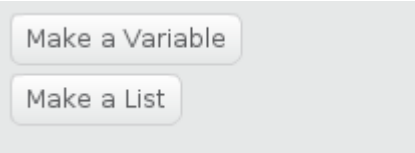
- Two basic data types are int and char
  - An int is like an integer, it has no fractional part
  - A char is interpreted as a character
    - a single letter, punctuation mark, or digit.

Variables

- Variable are locations that hold values
- Variables (e.g. int x) have:
  - A name (e.g. x)
  - A type (e.g. int)
  - A value (e.g. after x = 5; x will have the value five)
  - A location in memory
    - some 32 bit binary number

Variables are confusing

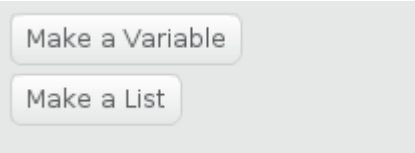
- Variables are one of the most confusing and difficult parts of the C programming language.
- They are confusing because they do not abstract away from the hardware much.
- It will take the full semester to understand them fully, so we had better start now.



Initialization

```
/* Data */
int integer = 1;
char character = 'a';
char *string = "Hello world\n";
```

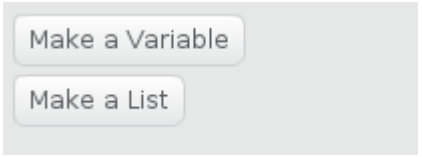
- In C, you can give a variable an initial value. Here:
  - The variable integer gets the value 1.
  - The variable character gets the value a.
  - The variable string gets the value Hello world<newline>



Constants

```
/* Data */
int integer = 1;
char character = 'a';
char *string = "Hello world\n";
```

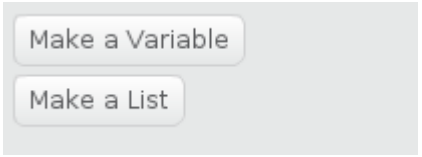
- To initialize, you specify the value using a constant:
  - The constant 105 has the value one hundred and five.
    - A string of digits is a int constant.
  - The constant 'a' has the value a.
    - A character (or occasionally two) surrounded by single quotes is a char constant
  - The constant “Hello world\n” has the value Hello world<newline>
    - A string of characters surrounded by double quotes is a char * constant



Char

```
/* Data */
int integer = 1;
char character = 'a';
char *string = "Hello world\n";
```

- The char data type may be
  - A letter (e.g. 'A', 'a', 'Q', 'z')
  - A punctuation mark (e.g. '.', '*', '%', '_')
  - A digit (e.g. '1', '5', '0')
  - A control character such as
    - '\n': newline
    - '\0': null
    - '\t': tab
    - '\'': single quote (i.e., ')
    - '\\': backslash (i.e. \)



Strings

```
/* Data */
int integer = 1;
char character = 'a';
char *string = "Hello world\n";
```

- In C, Strings are lists of type char
  - A list is represented either by char *<name> or char <name>[];
    - E.g., char *string is exactly the same thing as char string[]
  - The name of the variable (e.g. string) points to the beginning of the list.
  - The end of the list is designated by the null character, written '/0'
    - If the bits of the null character were interpreted as an int, it's value would be 0

Strings are lists

```
/* Data */
int integer = 1;
char character = 'a';
char *string = "Hello world\n";
```

- Remember, lists are *indexed*.
  - You access an element of the list using square brackets.
- E.g., given *string = “Hello world/n”
  - The value of string[0] is 'H'
  - The value of string[1] is 'e'
  - The value of string[11] is '/n'
  - The value of string[12] is '/0'
  - The value of string[13] is whatever was left in that memory location. It is unknown.

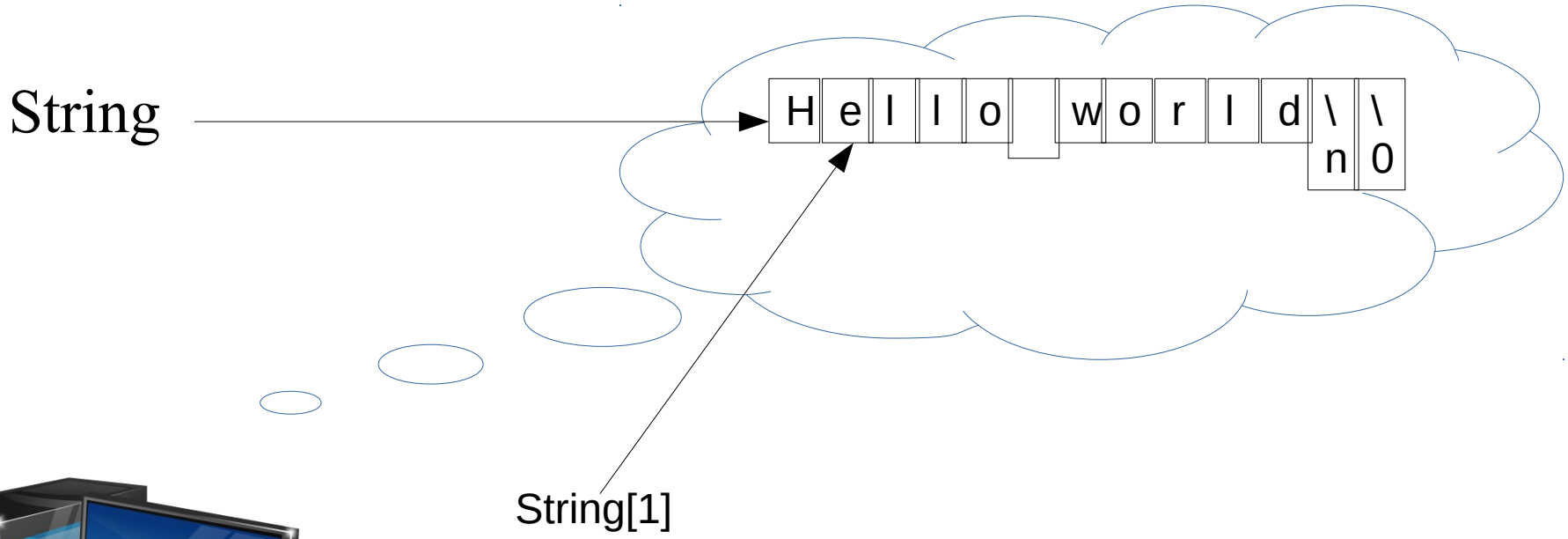
Lists in C

```
/* Data */
int integer = 1;
char character = 'a';
char *string = "Hello world\n";
```

- In C, lists are called *arrays*.
- A number in brackets that selects an item from an array is called an *index*, plural *indices*.
  - The index of the first element is always 0.
  - The index of the last element is always one less than the length of the array.
  - Because strings have a null character at the end, a string, interpreted as an array has the same number of elements as the length of the array.

Strings

```
/* Data */
int integer = 1;
char character = 'a';
char *string = "Hello world\n";
```



Pointers in C

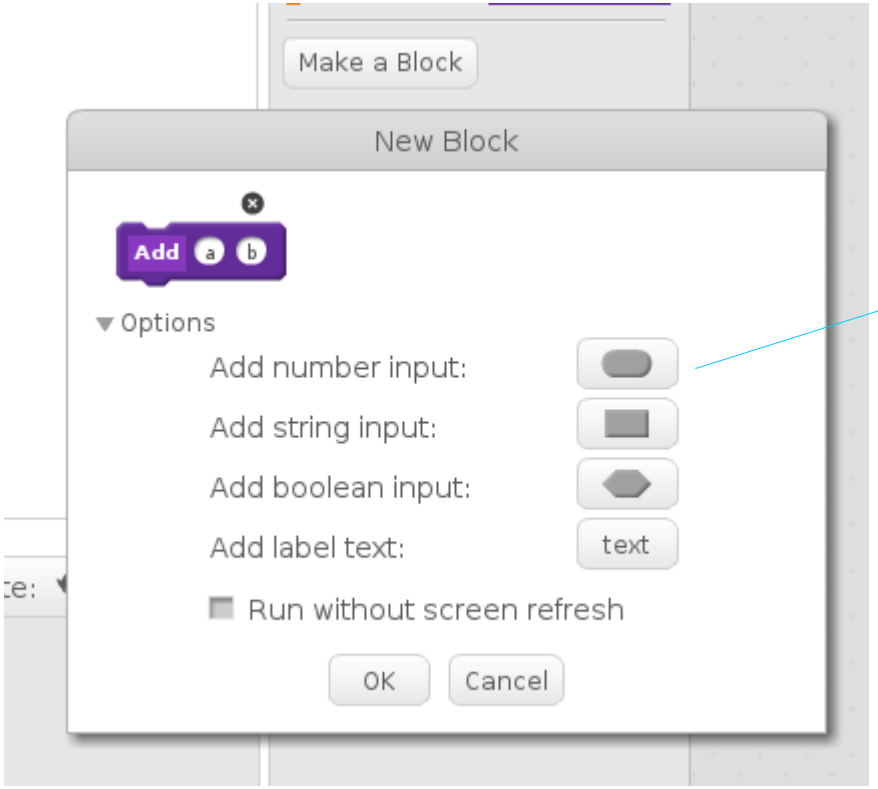
- The variable *string is the same a string[0].
  - The variable string is a pointer that points to the beginning of the array.
    - The syntax *string emphasizes that it is a pointer; string[0] emphasizes that it is an array
- A pointer is a variable that points to another variable.
  - It's value is the address of the variable to which it points

Places and Addresses

- My house
  - There are letters in my house
- Address of my house
  - To put a letter in my house you need the address

24 Calumet St
Rochester, NY

In C, new blocks are functions




```
int add(int a, int b)
{

}
```

C Functions: declare, define, call

- Declaration states the function name and parameters
- Definition states the functions action
- Call states that the functions actions should be done

Example: declare, define, call

- Declaration
  - Ends in a semi-colon ';'
- Definition
  - Followed by instructions
  - Uses parameters as variables
  - Return value is value of function call
- Call
  - Provides initial values for parameters

```
/*
 * Function Declaration
 */
int add(int a, int b);

/*
 * Function Definition
 */
int add(int a, int b)
{
  return a + b;
}

/*
 * Function Call
 */

int main()
{
  add(1, 2);
}
```

Parameters

- The inputs to functions are called *parameters* in C
- Parameters are brand new variable every time the function is called.
  - Parameters act like variables inside the function
- Variables defined inside functions are created when the function is called.
  - They are destroyed when the function ends.
    - Barring a few exceptions.

Example

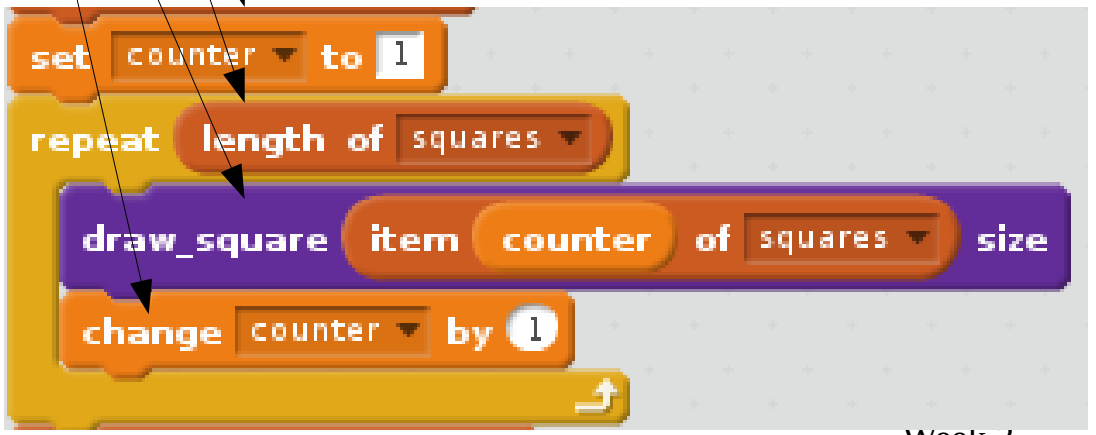
```c
#include <stdio.h>

int add(int a, int b)
{
  printf("add: a = %d, b = %d\n", a, b);
  return a + b;
}

int main(int argc, char *argv[])
{
  int a = 1;
  int b = 2;
  printf("%d\n", add(100, 200));
  printf("main: a = %d, b = %d\n", a, b);
}
```

```
student> gcc -o functions functions.c
student> ./functions
add: a = 100, b = 200
300
main: a = 1, b = 2
```

Looping through a list

- Initialize counter
- Repeat to end of list
- Body
- Increment counter

```
for (int i = 0; i < 2; i++) {
  draw_square(squares[i]);
}
```



Loops

```
for (int i = 0; i < 2; i++) {
  draw_square(squares[i]);
}
```

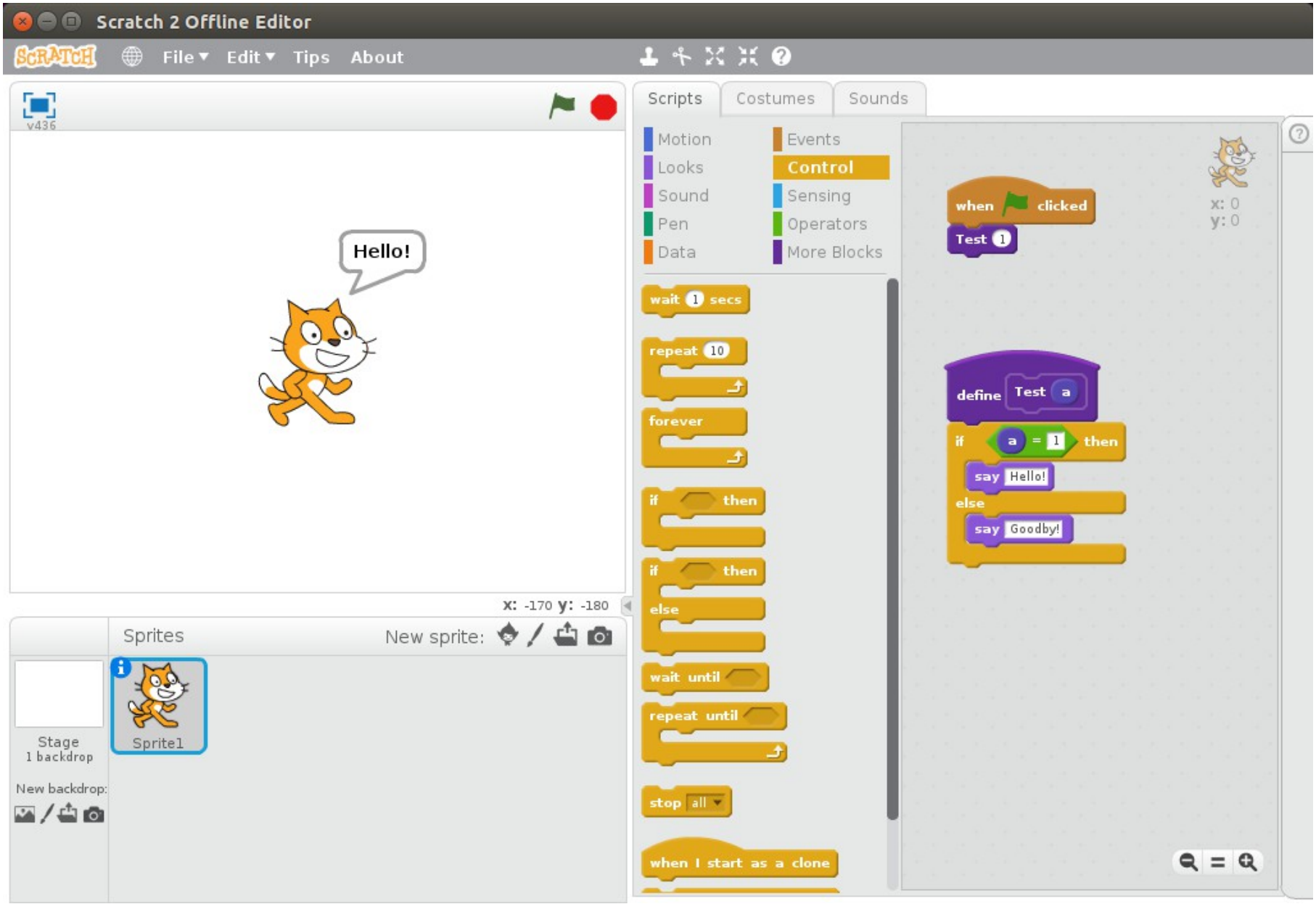
- Loops have three parts
  - Initialization (e.g., int i = 0)
    - Done once before the loop starts
  - Test (e.g. i < 2)
    - Done every time just before doing the body of the loop
  - Increment
    - Done every time just after doing the body of the loop

Example

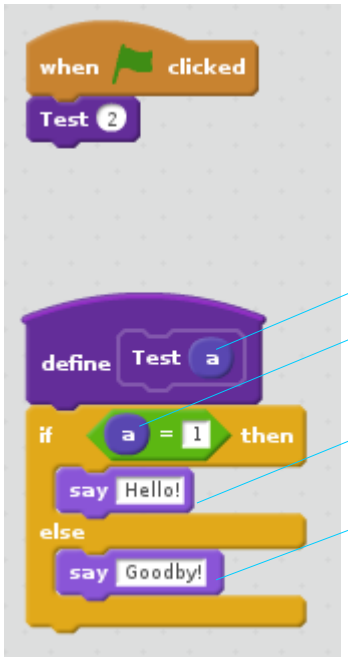
```c
#include <stdio.h>

int main(int argc, char *argv[])
{
  for (int i = 0; i < 5; i++) {
    printf("iteration %d: i = %d\n", i+1, i);
  }
}
```

```
student> gcc -std=c11 -o loops loops.c
student> ./loops
iteration 1: i = 0
iteration 2: i = 1
iteration 3: i = 2
iteration 4: i = 3
iteration 5: i = 4
```

Selection (if else)



Selection in C




```c
#include <stdio.h>

void test(int a)
{
  if (1 == a) {
    printf("Hello!\n");
  } else {
    printf("Goodby!\n");
  }
}

int main(int argc, char *argv[])
{
  test(1);
  test(2);
  test(3);
  test(4);
}
```

Example

```
#include <stdio.h>

void test(int a)
{
  if (1 == a) {
    printf("Hello!\n");
  } else {
    printf("Goodby!\n");
  }
}

int main(int argc, char *argv[])
{
  test(1);
  test(2);
  test(3);
  test(4);
}
```

```
student> ./selection
Hello!
Goodby!
Goodby!
Goodby!
```

N.B.: In C = is not ==

- = puts a value in a variable
- == tests weather two expressions have the same value.