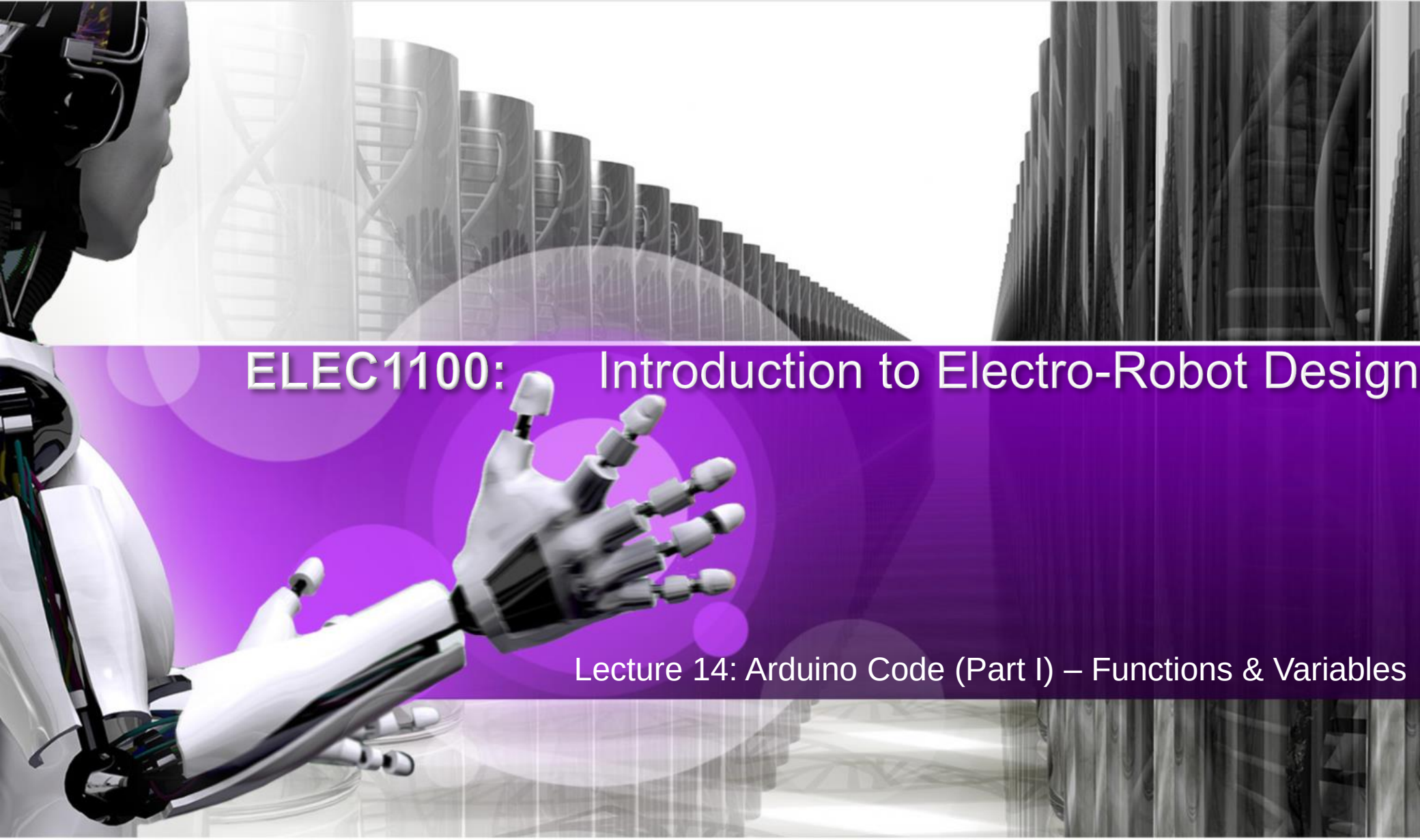
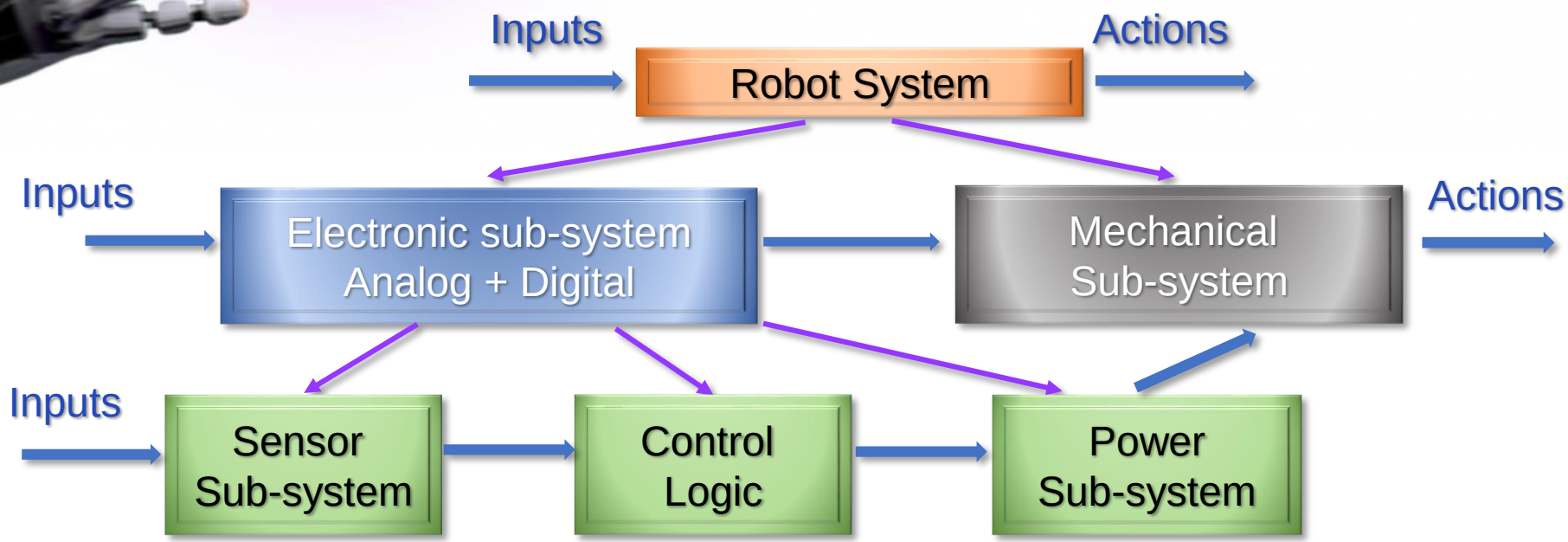


ELEC1100: Introduction to Electro-Robot Design

Lecture 14: Arduino Code (Part I) – Functions & Variables

SONG Shenghui and MURCH Ross, Dept. of ECE, HKUST

ELEC1100 ROADMAP



Sensor Basics:
- Wk6: Sensor Basic – Sensor/Line/ADC

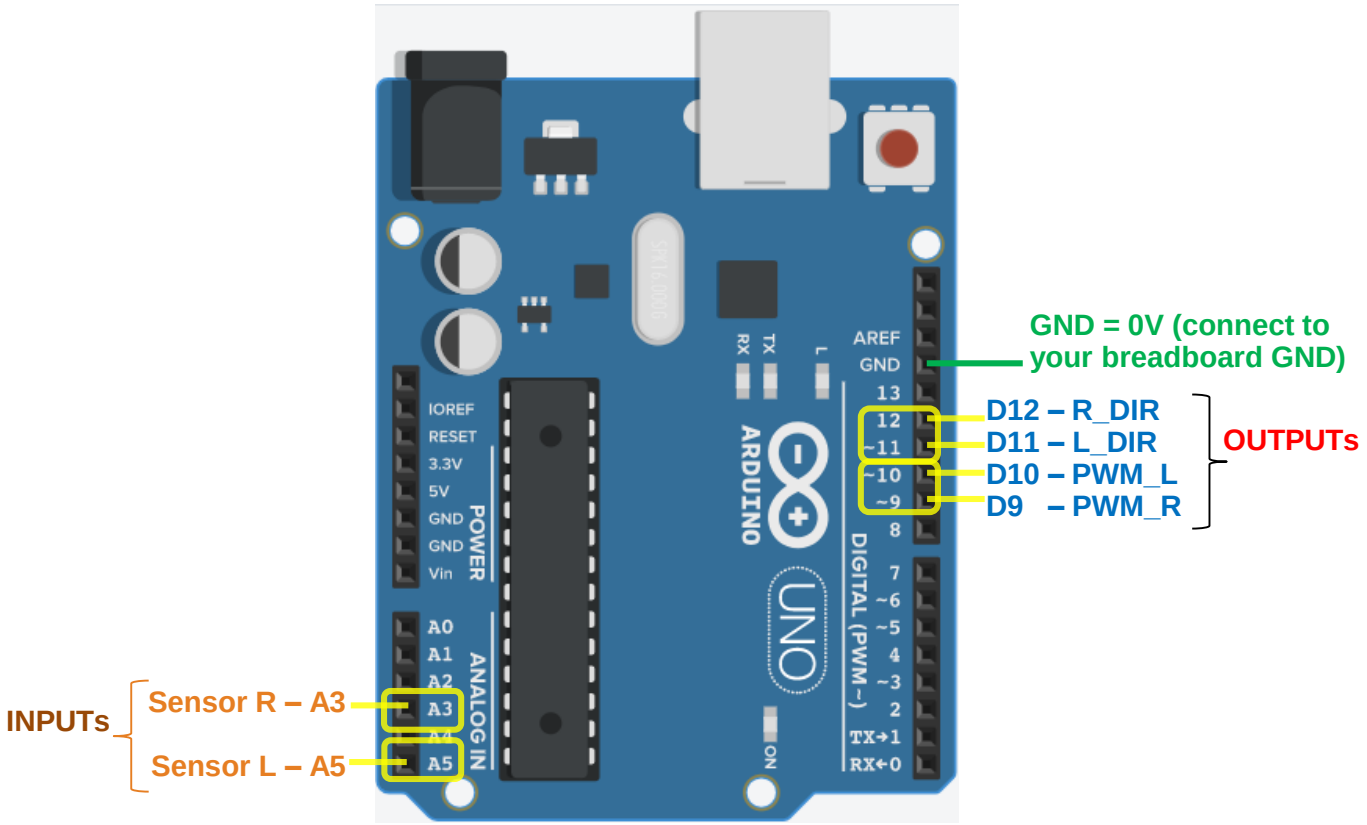
Combinational/Sequential Logic:
- Wk7: Robot Brain: Logic Gate and Logic Operation
- Wk8: MCU & Arduino
- Wk8-9: Programming Language

Basic electronics:
- Wk1: Basic Electronics - Charge/Current/Voltage/Resistor
- Wk2: Energy/Power and DC Sources

Motor Power Supply:
- Wk3: Pulse Signal and PWM Control
- Wk4: Transistor and H-Bridge

FROM LAST LECTURE

- You need to use Arduino programming language to write programs.



PROGRAMMING LANGUAGE

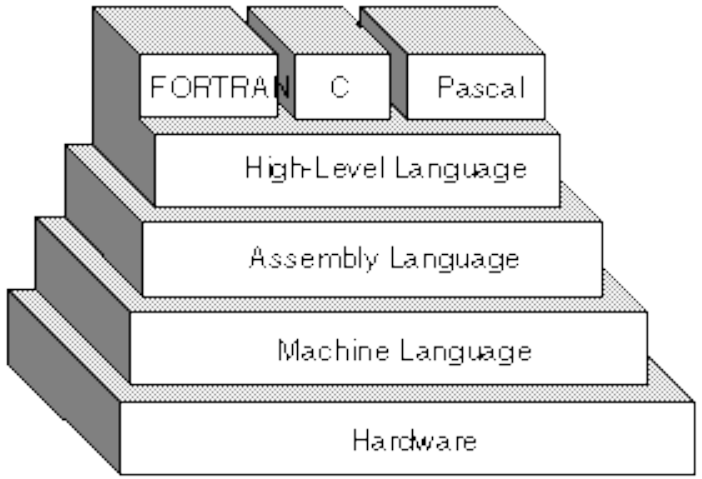
- A programming language is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks.
- The term *programming language* usually refers to high-level languages, such as PYTHON, Java, C, C++, FORTRAN and Pascal.
- Each programming language has a unique set of keywords (words that it understands) and a special syntax for organizing program instructions.

HIGH-LEVEL PROGRAMMING LANGUAGE

- High-level programming languages, while simple compared to human languages, are more complex than the languages the computer actually understands, called machine languages.

- Assembly languages are similar to machine languages, but they are much easier to program in because they allow a programmer to substitute names for numbers. Machine languages consist of numbers only.

- Regardless of what language you use, eventually you need to convert your program into machine language so that the computer can understand it.



EVOLUTION OF PROGRAMMING LANGUAGE [1]

- Early computers programmed in machine languages
  - All binary numbers (all 0's and 1's)
  - bits (**BI**nary digi**T**s)
- Data and commands stored in binary
  - Bits are arranged into bytes and words that represent different elements
  - 8 bits in a byte
  - 2 bytes is called a word
  - ASCII character stored in a byte to represent letters
  - Integers stored in 2 or 4 bytes
  - Commands
  - Address
  - …

EVOLUTION OF PROGRAMMING LANGUAGE [2]

- Assembly language organized the 1 and 0 into simple commands
  - Codes translated into machine language by a program called the **"assembler"**

Assembly Language	Machine Language
LOAD	100100
STOR	100010
MULT	100110
ADD	100101
SUB	100011

EVOLUTION OF PROGRAMMING LANGUAGE [3]

- High-level languages read like combination of English and algebra
  - They will be translated into machine language by a program called a **"compiler"**

```
void loop()
{
  // read the input on analog pin 0:
  int sensorValue = analogRead(A0);

  // Convert the analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
  float voltage = sensorValue * (5.0 / 1023.0);

  // print out the value you read:
  Serial.println(voltage);
}
```

- Many high-level languages are similar enough that programmers can easily understand source code written in multiple languages.

ARDUINO PROGRAMMING LANGUAGE

- Arduino programming language can be divided in three main parts: functions, values (variables and constants), and structure.

  FUNCTIONS

  - For controlling the Arduino board and performing computations.

  VARIABLES

  - Arduino data types and constants.

  STRUCTURE

  - The elements of Arduino (C++) code.

- We will only introduce some key components for your project.

FUNCTIONS

- Digital I/O
  - pinMode()
  - digitalRead()
  - digitalWrite()
- Time
  - delay()
  - micros()
  - millis()
- Math
  - abs()
  - constrain()
  - map()
  - max()
  - min()
  - pow()
  - sq()
  - sqrt()
- Random Numbers
  - random()
  - randomSeed()
- Communication
  - Serial

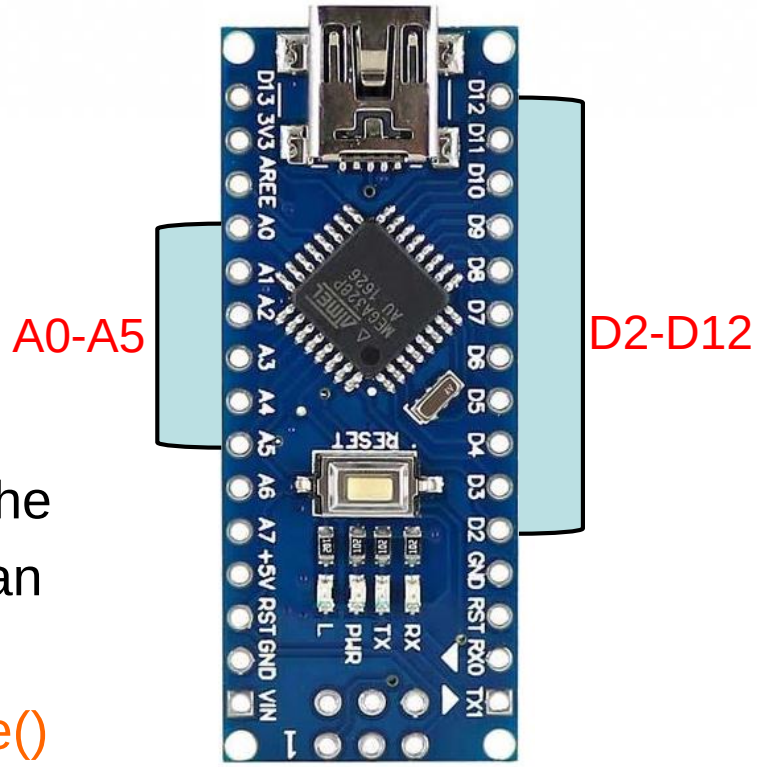
ARDUINO NANO: INPUT AND OUTPUT

- Digital I/O

  pinMode()

  digitalRead()

  digitalWrite()



- Each of the pins A0-A5 and D2-D12 on the Nano-board/Uno-board can be used as an digital input or output (Digital I/O), using pinMode(), digitalRead(), and digitalWrite() functions.

FUNCTIONS: DIGITAL I/O [1]

- pinMode()
  - Configure a specific pin to behave either as an input or an output pin
  - Syntax: pinMode(pin, mode)
  - "pin": the Arduino pin number you want to set
  - "mode": INPUT or OUTPUT

For digital pins (D2-D12), we write:

```
pinMode(11, OUTPUT);
digitalWrite(11, HIGH);
```

That means we use pin D11 as logic output and set to logic high. Notice "D" is omitted in the statement.

For analog pins (A0-A5), we write:

```
pinMode(A3, OUTPUT);
digitalWrite(A3, HIGH);
```

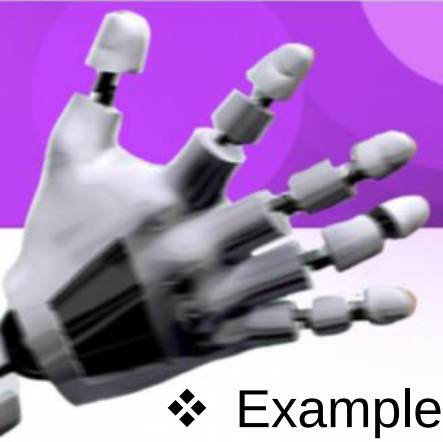
That means we use pin A3 as logic output and set to logic high. Here "A" is included in the statement.

Arduino uses this way to distinguish between digital and analog pins when writing sketches.

FUNCTIONS: DIGITAL I/O [2]

- digitalRead()
  - Read the value from a “INPUT” pin, either HIGH or LOW
  - Syntax: digitalRead(pin)
  - “pin”: the configured Arduino pin number you want to read
  - Returns: HIGH or LOW

- digitalWrite()
  - Write a HIGH or a LOW value to a “OUTPUT” pin
  - Syntax: digitalWrite(pin, value)
  - “pin”: the configured Arduino pin number you want to write
  - “value”: HIGH or LOW



FUNCTIONS: DIGITAL I/O [3]

- Example Code: Sets pin 13 (build-in LED pin) to the same value as pin 7, declared as an input.

```
int ledPin = 13;  // LED connected to digital pin 13
int inPin = 7;    // pushbutton connected to digital pin 7
int val = 0;      // variable to store the read value

void setup() {
  pinMode(ledPin, OUTPUT);  // sets the digital pin 13 as output
  pinMode(inPin, INPUT);    // sets the digital pin 7 as input
}

void loop() {
  val = digitalRead(inPin);   // read the input pin
  digitalWrite(ledPin, val);  // sets the LED to the button's value
}
```

FUNCTIONS: TIME [1]

- delay()
  - Pauses the program for the amount of time (in milliseconds) specified as parameter (*There are 1000 milliseconds in a second*).
  - Syntax: delay(ms)
  - "ms": the number of millisecond to pause.
- Example Code: Blinking LED

```
#define LED_PIN 13                  // Pin number attached to LED.

void setup() {
    pinMode(LED_PIN, OUTPUT);       // Configure pin 13 to be a digital output.
}

void loop() {
    digitalWrite(LED_PIN, HIGH);    // Turn on the LED.
    delay(1000);                    // Wait 1 second (1000 milliseconds).
    digitalWrite(LED_PIN, LOW);     // Turn off the LED.
    delay(1000);                    // Wait 1 second.
}
```

FUNCTIONS: TIME [2]

- micros()
  - Returns the number of microseconds since the Arduino board began running the current program.
  - Syntax: time = micros()

- millis()
  - Returns the number of milliseconds since the Arduino board began running the current program.
  - Syntax: time = millis()

Notes: There are 1,000 microseconds in a millisecond and 1,000,000 microseconds in a second.

FUNCTIONS: MATH

- abs(): Calculates the absolute value of a number.
- constrain(): Constrains a number to be within a range.
- map(): Re-maps a number from one range to another.
- max(): Calculates the maximum of two numbers.
- min(): Calculates the minimum of two numbers.
- pow(): Calculates the value of a number raised to a power.
- sq(): Calculates the square of a number: the number multiplied by itself.
- sqrt(): Calculates the square root of a number.

- You may go to Arduino Language Reference Web page for details if necessarily to use above.

https://www.arduino.cc/reference/en/#functions

FUNCTIONS: RANDOM NUMBERS

- random()
  - The random function generates pseudo-random number.
  - Syntax: random(max), random(min, max)
  - “min”/”max” – lower/upper bound of the random value.
  - Returns: A random number between min and max-1.

- randomSeed()
  - Initializes the pseudo-random number generator, causing it to start at an arbitrary point in its random sequence. This sequence, while very long, and random, is always the same.
  - Syntax: randomSeed(seed)
  - “seed”: number to initialize the pseudo-random sequence.

FUNCTIONS: COMMUNICATION [1]

- Serial.begin()
  - Set the data rate in bits per second (baud) for serial data transmission. For communicating with the computer, use one of these rates: 300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, or 115200.
  - Syntax: Serial.begin(speed), Serial.begin(speed, config)
  - “Serial”: serial port object.
  - “speed”: in bits per second (baud)
  - “config”: sets data, parity, and stop bits.

FUNCTIONS: COMMUNICATION [2]

- Example Code: Generates random numbers and displays them.

```
long randNumber;

void setup() {
  Serial.begin(9600);

  // if analog input pin 0 is unconnected, random analog
  // noise will cause the call to randomSeed() to generate
  // different seed numbers each time the sketch runs.
  // randomSeed() will then shuffle the random function.
  randomSeed(analogRead(0));
}

void loop() {
  // print a random number from 0 to 299
  randNumber = random(300);
  Serial.println(randNumber);

  // print a random number from 10 to 19
  randNumber = random(10, 20);
  Serial.println(randNumber);

  delay(50);
}
```

USER-DEFINED FUNCTIONS [1]

- Segmenting code into functions allows a programmer to create modular pieces of code that perform a defined task and then return to the area of code from which the function was "called". The typical case for creating a function is when one needs to perform the same action multiple times in a program.

- Format

```
<data> function(<arg1>,<arg2>, …)
```

Output data-type
(can be void)

arguments (input/output)

- Data type of output and arguments need to be defined inside the function
- Try to avoid using arguments as output variable

USER-DEFINED FUNCTIONS [2]

- Example Code:

To "call" our simple **multiply function**, we pass it parameters of the datatype that it is expecting.

Our function needs to be **declared** outside any other function, so "**myMultiplyFunction()**" can go either above or below the "loop()" function.

```
void setup(){
  Serial.begin(9600);
}

void loop() {
  int i = 2;
  int j = 3;
  int k;

  k = myMultiplyFunction(i, j); // k now contains 6
  Serial.println(k);
  delay(500);
}

int myMultiplyFunction(int x, int y){
  int result;
  result = x * y;
  return result;
}
```

USER-DEFINED FUNCTIONS [3]

- Advantages:
  - Functions help the programmer stay organized. Often this helps to conceptualize the program.
  - Functions codify one action in one place so that the function only has to be thought out and debugged once.
  - This also reduces chances for errors in modification, if the code needs to be changed.
  - Functions make the whole sketch smaller and more compact because sections of code are reused many times.
  - They make it easier to reuse code in other programs by making it more modular, and as a nice side effect, using functions also often makes the code more readable.

VARIABLES

- Constants
  - INPUT / OUTPUT
  - HIGH / LOW
  - true / false
- Data Types
  - char
  - int
  - bool
  - float
  - void

VARIABLES: CONSTANTS [1]

- Constants are predefined expressions in the Arduino language.
  - They are used to make the programs easier to read.
  - Easier to make modifications to the program.

- Defining Logical Levels: true and false (Boolean Constants)
  - False is defined as 0 (zero)
  - True is often said to be defined as 1. In a Boolean sense, any non-zero integer is true.

Note: the “true” and “false” constants are typed in lowercase unlike “HIGH”, “LOW”, “INPUT” and “OUTPUT”.

VARIABLES: CONSTANTS [2]

- Defining Digital Pins Mode: INPUT and OUTPUT

- INPUT
  - Arduino pins configured as INPUT with pinMode()
  - Pins configured as INPUT will draw very small power.
  - This makes them useful for reading a sensor.

- OUTPUT
  - Arduino pins configured as OUTPUT with pinMode()
  - They can provide a high current to other devices/circuits, up to 40mA (for powering LEDs).
  - Loads greater than 40mA (e.g. motors) will require a transistor or other interface circuitry.

Note: Pins configured as outputs can be damaged or destroyed if they are connected to either the ground or positive power rails.

VARIABLES: CONSTANTS [3]

- Defining Pins Levels: HIGH and LOW

- HIGH
  - If a pin is configured as **INPUT** with pinMode() and read with digitalRead(), Arduino will report HIGH if
    - a voltage greater than 3.0V is present (5V boards)
    - a voltage greater than 2.0V is present (3.3V boards)
  - If a pin is configured as **OUTPUT** and set to HIGH with digitalWrite(), the pin is at
    - 5.0V (5V boards)
    - 3.3V (3.3V boards)
- LOW
  - If a pin is configured as **INPUT** with pinMode() and read with digitalRead(), Arduino will report LOW if
    - a voltage less than 1.5V is present (5V boards)
    - a voltage less than 1.0V is present (3.3V boards)
  - If a pin is configured as **OUTPUT** and set to LOW with digitalWrite(), the pin is at
    - 0V (both 5V and 3.3V boards)

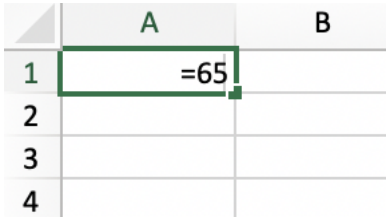
VARIABLES: DATA TYPES [1]

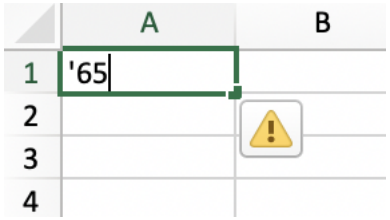
- Data Type: classification of data which tells the compiler or interpreter how the programmer intends to use the data.

- WHY ARE DATA TYPES IMPORTANT?

  What does “01000001” present?

  - It can be a integer value of 65
  - It can be the ASCII code of “A”
  - It can be a command?
  - If type is not defined, what is “01000000”+”01000001”?





e.g. in EXCEL

VARIABLES: DATA TYPES [2]

- char:
  - A data type used to store a **character** value. Character literals are written in single quotes, like this: 'A'.
  - Characters are stored as numbers however. You can see the specific encoding in the ASCII chart. This means that it is possible to do arithmetic on characters, in which the ASCII value of the character is used (e.g. 'A' + 1 has the value 66, since the ASCII value of the capital letter A is 65).
  - Syntax: char var = val
  - "var" - your "char" variable name
  - "val" – the value you assign to that variable

- Example Code:

```
char myChar = 'A';
char myChar = 65; // both are equivalent
```

VARIABLES: DATA TYPES [3]

- int:
  - **Integers** are your primary data-type for number storage.
  - Syntax: int var = val
  - “var” - your “int” variable name
  - “val” – the value you assign to that variable

- bool:
  - Holds one of two values: true or false.
  - Syntax: bool var = val
  - “var” - your “bool” variable name
  - “val” – the value you assign to that variable

VARIABLES: DATA TYPES [4]

- Data Types: “int” and “bool”
- Example Code

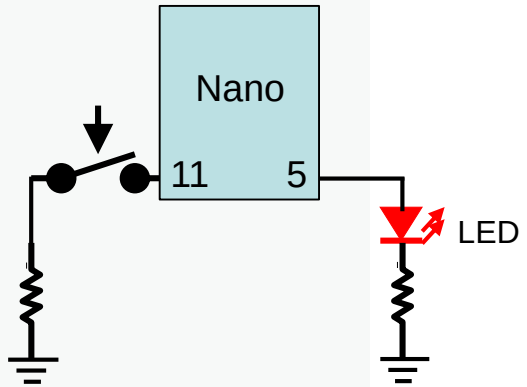
```
int LEDpin = 5;      // LED on pin 5
int switchPin = 11; // momentary switch on 11, other side connected to ground

bool running = false;

void setup() {
  pinMode(LEDpin, OUTPUT);
  pinMode(switchPin, INPUT_PULLUP); // set INPUT mode and
                                    // turn on pullup resistor
}

void loop() {
  if (digitalRead(switchPin) == LOW) {
    // switch is pressed - pullup keeps pin high normally
    delay(100);                     // delay to debounce switch
    running = !running;             // toggle running variable
    digitalWrite(LEDpin, running);  // indicate via LED
  }
}
```

Circuit Connection



VARIABLES: DATA TYPES [5]

- float:
  - A number that has a decimal point. Floating-point numbers are often used to approximate analog and continuous values because they have greater resolution than integers.
  - Syntax: float var = val

- Example Code:

```
float myfloat;
float sensorCalbrate = 1.117;

int x;
int y;
float z;

x = 1;
y = x / 2;           // y now contains 0, ints can't hold fractions
z = (float)x / 2.0; // z now contains .5 (you have to use 2.0, not 2)
```

VARIABLES: DATA TYPES [6]

- void:
  - This keyword is used only in function declarations. It indicates that the function is expected to return no information to the function from which it was called.

- Example Code:

```
// actions are performed in the functions "setup" and "loop"
// but  no information is reported to the larger program

void setup() {
  // ...
}

void loop() {
  // ...
}
```

SUMMARY

- Arduino programming language can be divided in three main parts: functions, values (variables and constants), and structure. We only introduce some key components for your project.

- Functions:

Digital I/O
- pinMode()
- digitalRead()
- digitalWrite()

Time
- delay()
- micros()
- millis()

Math
- abs()
- constrain()
- map()
- max()
- min()
- pow()
- sq()
- sqrt()

Random Numbers
- random()
- randomSeed()

Communication
- Serial

- Variables:

Constants
- INPUT / OUTPUT
- HIGH / LOW
- true / false

Data Types
- char
- int
- bool
- float
- void

NEXT LECTURE

- Arduino Code:
  - Structure
  - Conditional Statement

QUESTIONS?