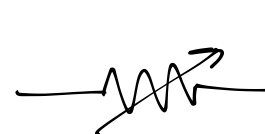
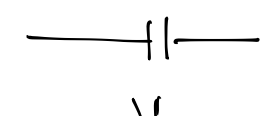


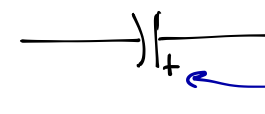
Arduino Intro 1

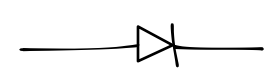
Basic circuits review

 - resistor - measured in Ohms (Ω), but typical values are $k\Omega$ (100k, 330k)

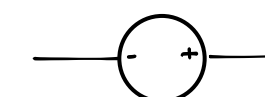
 - variable resistor (often called potentiometer or pots) and $M\Omega$ (1M, 1.5M)


 - capacitor - measured in farad
- typical values μF or pF

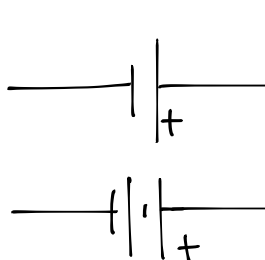
 - some capacitors are polarized (must be connected in the correct direction)


 - diode - only allow current in one direction (in the direction the triangle is pointing)

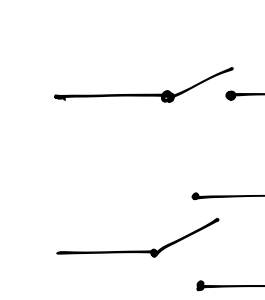
 - Light Emitting Diode (LED)

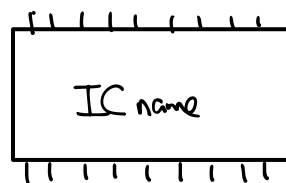
 - DC voltage source

 - AC voltage source (we won't use much in this class)

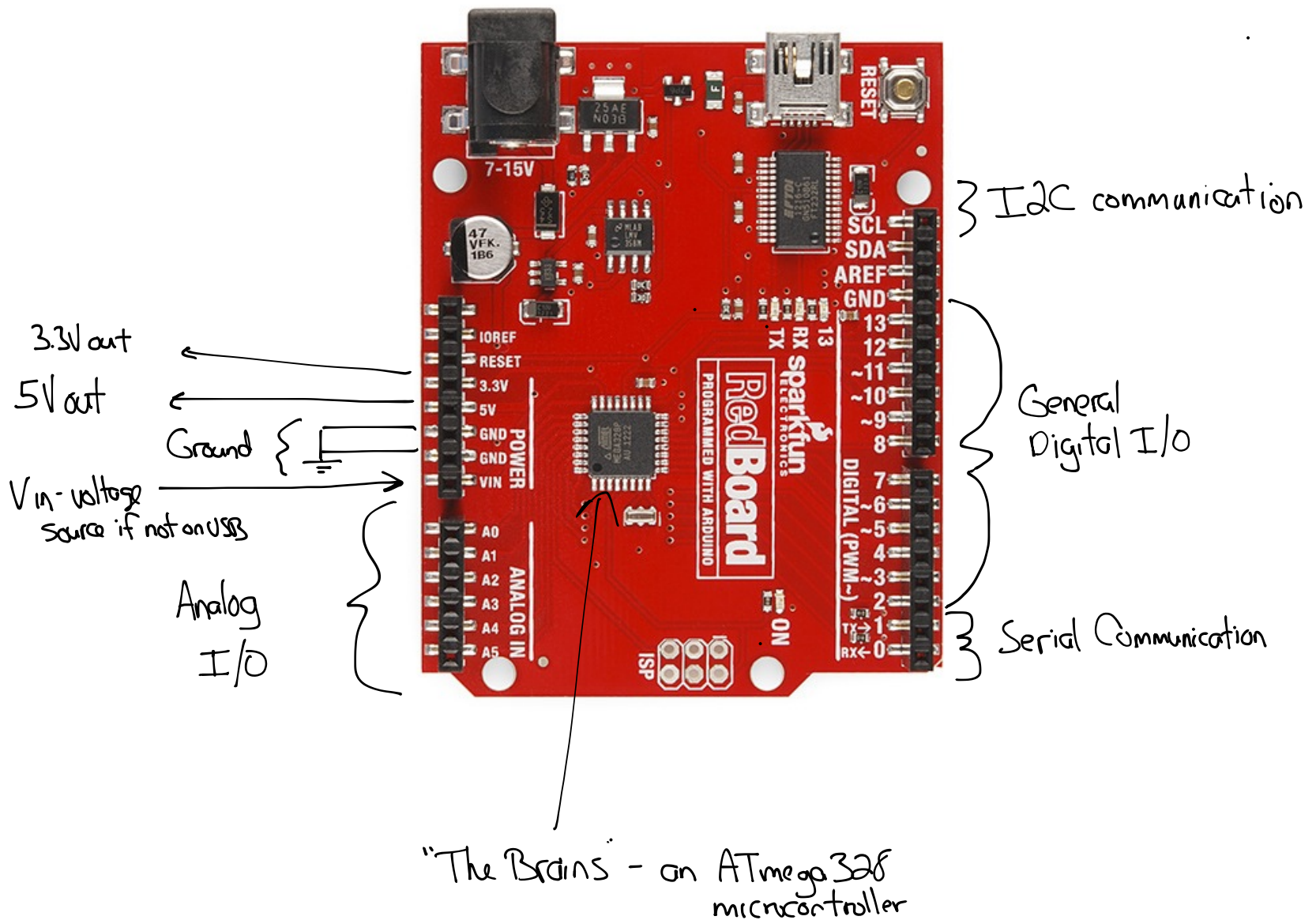
 } Batteries

 - Ground (almost always draw pointing down)

 } switches (typically drawn in the default state, if there is one)

 - Integrated Circuit (the "chips" in your kit)

The Redboard layout



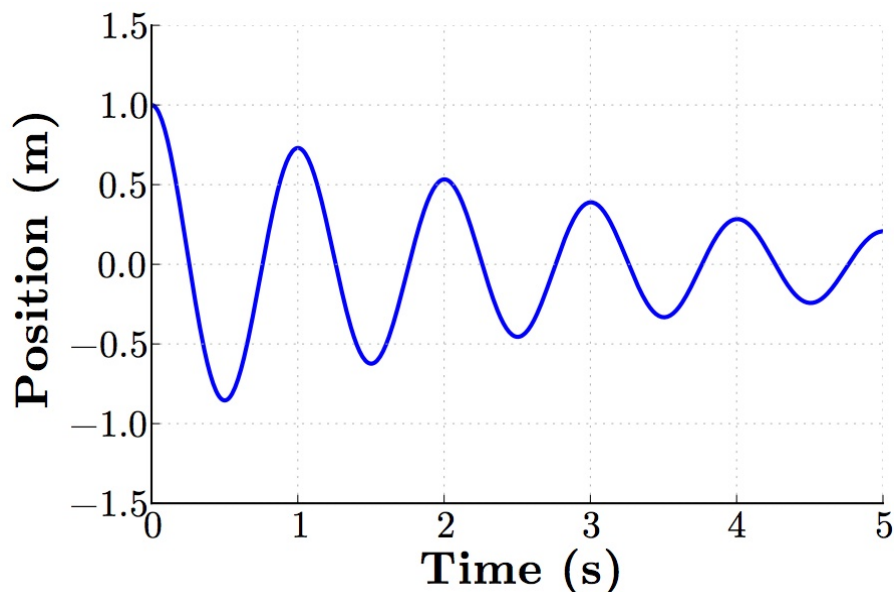
This board is very similar to the Arduino Uno R3 (the "official" Arduino release)

Power

- We'll almost always power via USB
- Can use 7-15VDC through the barrel jack or Vin
- 5V and 3.3V pins on board are low-current OUTPUTS

Analog Input and Output (I/O)

Analog - continuously varying signals (infinitely many states)



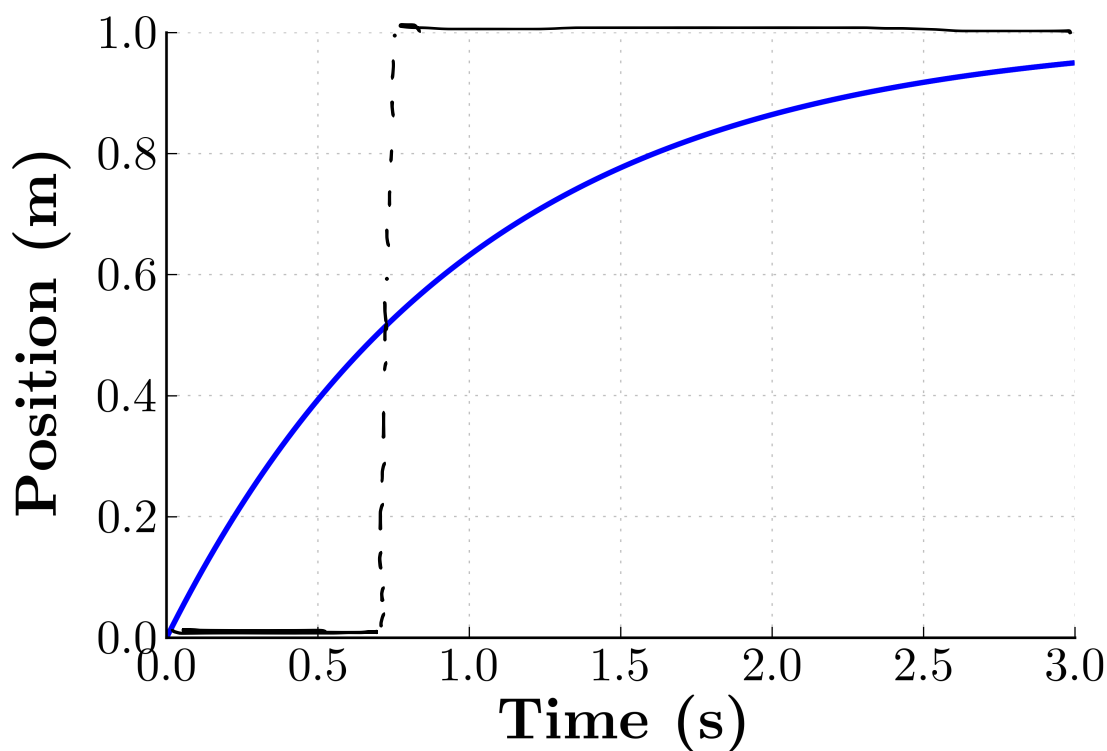
The position response of a mass-spring-damper system is analog

Q: What else?

- temperature
- sound
- most "natural" processes

Computers are not good at analog, so we need:

Analog to Digital Converters (ADC)



Q: How can this be represented digitally?

Digital \leftrightarrow 0 or 1... How about...

i) Analog $< 0.5 \rightarrow 0$

a) Analog $\geq 0.5 \rightarrow 1$

Q: Problems?

Resolution is terrible!

So, add more 0/1 (bits)

The RedBoard (like most Arduinos) has 10 bit ADC

10 bit ADC - Q: How many "levels" or "buckets" are available to approximate the analog signal?

$2^{10}!$ \rightarrow 1024 levels (0-1023)

The ADC gives a ratio to max voltage, we have to convert

In a 5V system,

5V \rightarrow ADC \rightarrow 1023

Some math for in between \rightarrow

0V \rightarrow ADC \rightarrow 0

$$\frac{\overbrace{\text{Resolution of ADC}}^{2^{10} \text{ for us}}}{\underbrace{\text{Sys. Voltage}}_{5\text{V for us}}} = \frac{\text{ADC value} \leftarrow \text{from AD converter}}{\text{(Analog Voltage)} \leftarrow \text{Actual Analog Voltage}}$$

Arduino/RedBoard ADC

Pins A0-A5 are analog

```
// Define pin A2 as an input
pinMode(A2, INPUT);

// Read the analog value of A2 into x
// x can be between 0 and 1023
int x = analogRead(A2);
```

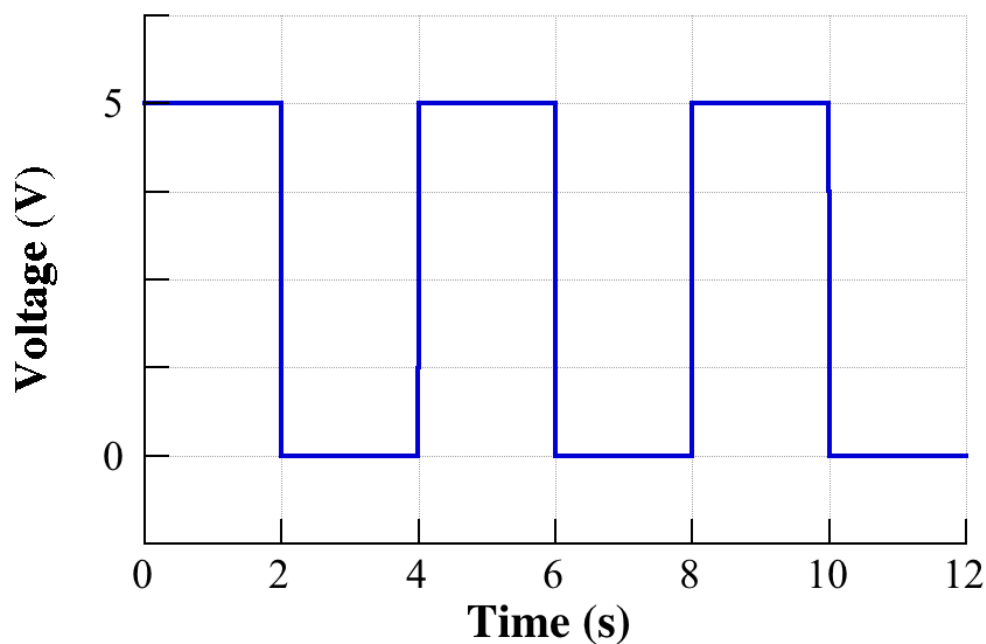
More on this later

Q: What about analog out? How can we do that?

- "true analog"
- Approximate the analog with digital I/O ← This is what the RedBoard (and other Arduinos do)

Digital Input and Output

Discrete - either on or off (for the RedBoard this means either 5V or 0V)



Q: How can we approximate an analog signal with just on/off?

Q: What is the "average" of the signal?
2.5V

Q: How could other values be approximated?
If we can switch on/off quickly enough, we can approximate voltages between 0 + 5V

Pulse-Width Modulation

Pulse-width Modulation

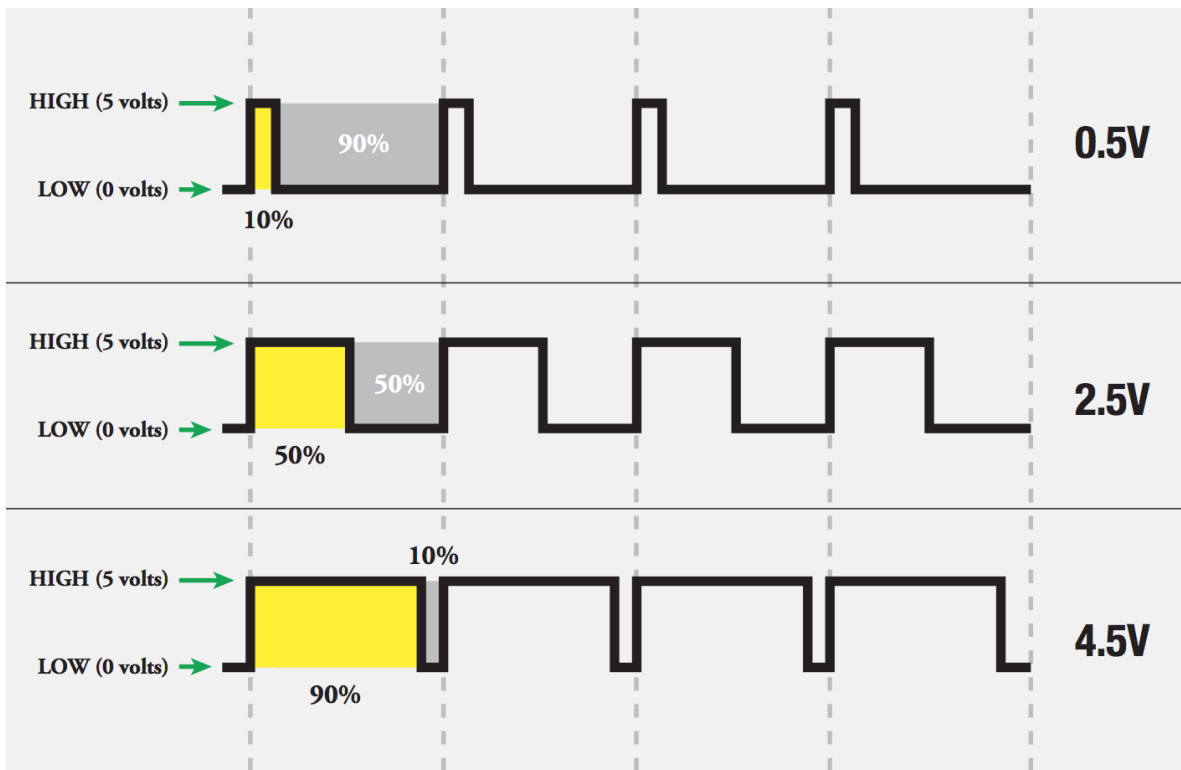


Image from SparkFun Inventor's Kit Guide

To get 10% of "on" value, stay "on" 10% of the time

The case we just saw

To get 90% of the "on" value, stay on 90% of the time

The "on time" defines the duty cycle

Arduino/RedBoard PWM

- PWM is only available on pins 3, 5, 6, 9, 10, 11

```
void setup()
```

```
{
```

```
  pinMode(9, OUTPUT);
```

```
}
```

```
void loop()
```

```
{
```

```
  // analogWrite values from 0 to 255
```

```
  // Here, we have approx. 50% duty cycle
```

```
  analogWrite(9, 127);
```

```
}
```

More on this later

Serial Communication

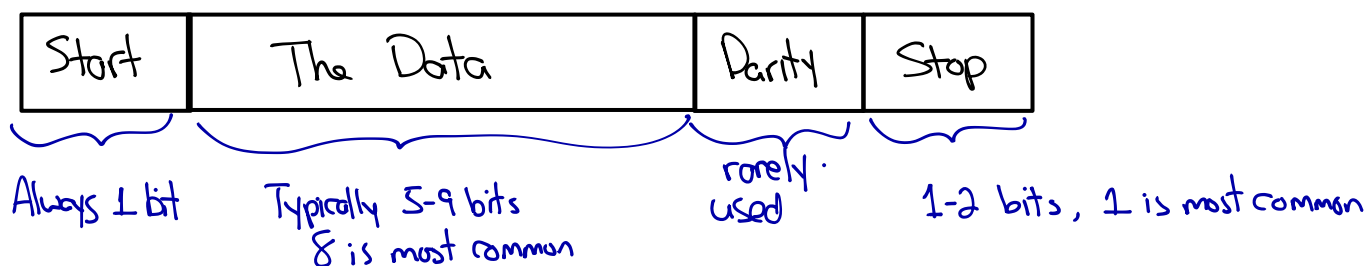
Q: Can we communicate with digital signals? How?

- parallel comm. (each input/output pair has meaning)
- morse code (just on/off... order and timing dictate message)
- serial communication
 - a series of 1's and 0's
 - both sender & receiver have to know what protocol is being used

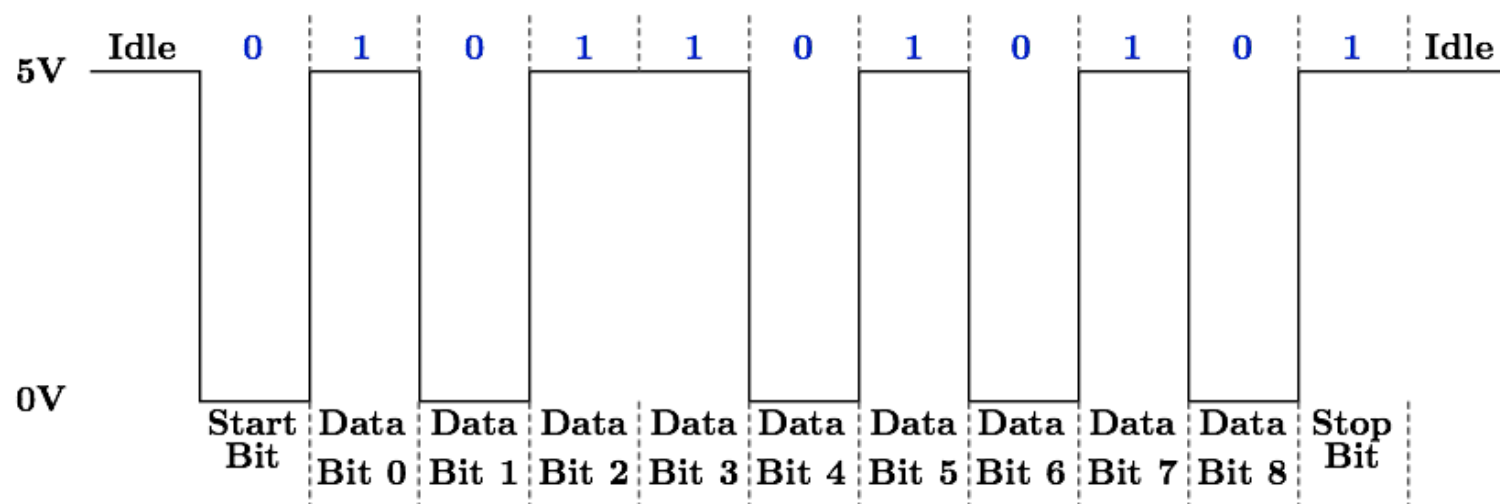
Serial Communication Parameters

- Baud rate - speed the data is being sent, measured in bits per second (bps)
- Stop bit - let's the receiver know the message is finished, either 1 or 2 bits
- Parity - not often used... For our purposes, assume "no parity"
- Number of Data bits - how much actual data is being sent, 8 is most common

An example serial data packet



A Serial Comm. at 9600 8N1 (9600 bps, 8 bits of data, no parity, 1 stop bit)



Arduino/RedBoard Serial Communication

```
void setup()
{
  // This sends text back to the main computer.
  // The Serial.begin() function initializes the port
  and sets the comm. speed.

  Serial.begin(9600);
}

void loop()
{
  // We just "print" a string of characters to the
  serial port
  // The low-level comm. is handled for us
  Serial.print("Hello World!");
}
```