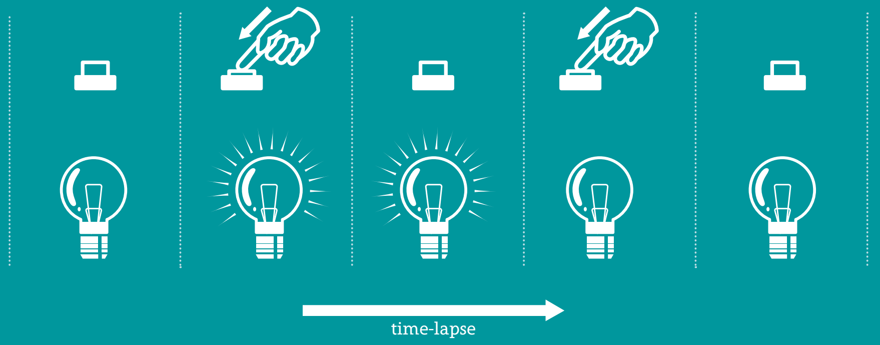
alternating switch

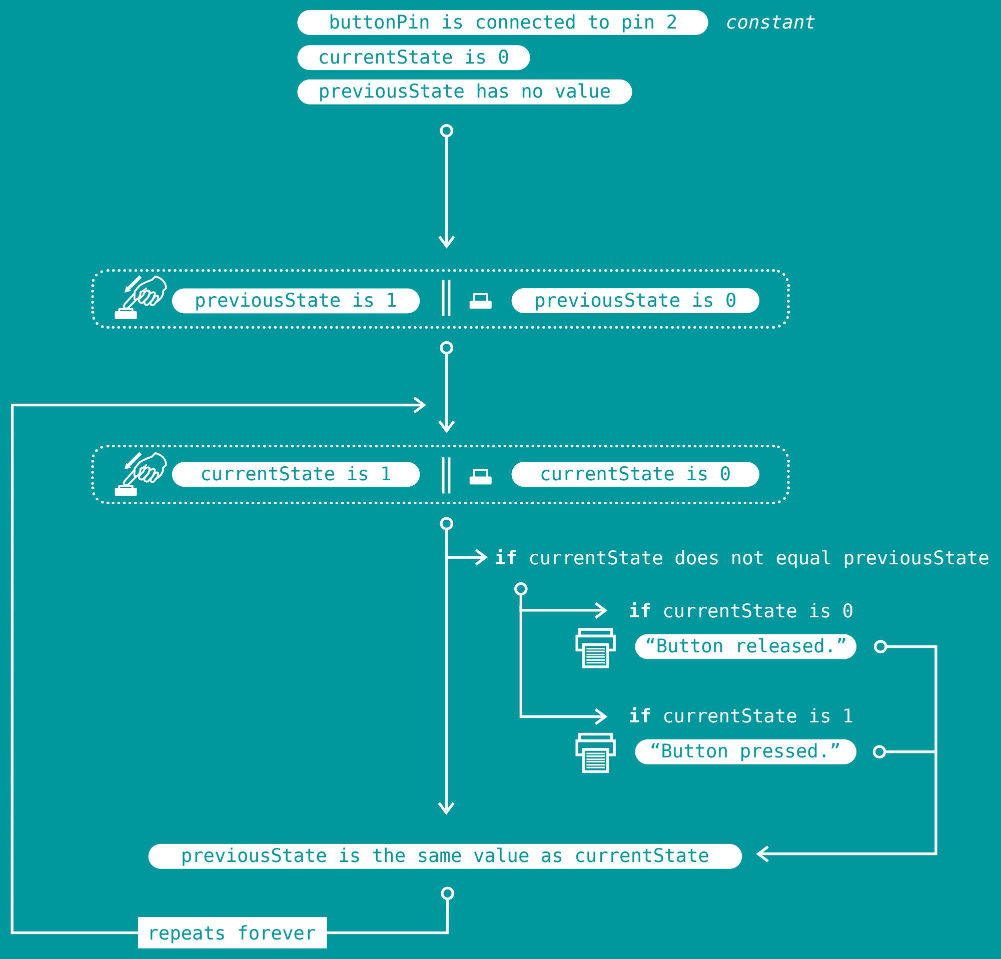
An alternating switch is a switch that remembers its previous state. Instead of pressing a button or switch to activate something continuously, the alternating switch only needs to be switched once.
This alternating switch uses a variable called previousState to store its previous state. It compares the current state of the switch with its previous state; only if they are different the switch activates.

examples: *light switch, on/off button, remote control, activation button*



```
int buttonPin = 2;                                          data
int currentState = 0;                                       data
int previousState;                                          data

void setup() {
  Serial.begin(9600);                                       output
  pinMode(buttonPin, INPUT);                                input
  digitalWrite(buttonPin, LOW);                             transformer
  previousState = digitalRead(buttonPin);                   transformer
}

void loop() {

  currentState = digitalRead(buttonPin);                    input

    if (currentState != previousState) {                    condition

      if (currentState == 0) {                              condition
        Serial.println("Button released.");                 output
      }
      else {                                                condition
        Serial.println("Button pressed.");                  output
      }
    }
  previousState = currentState;                             transformer
}
repeats forever
```

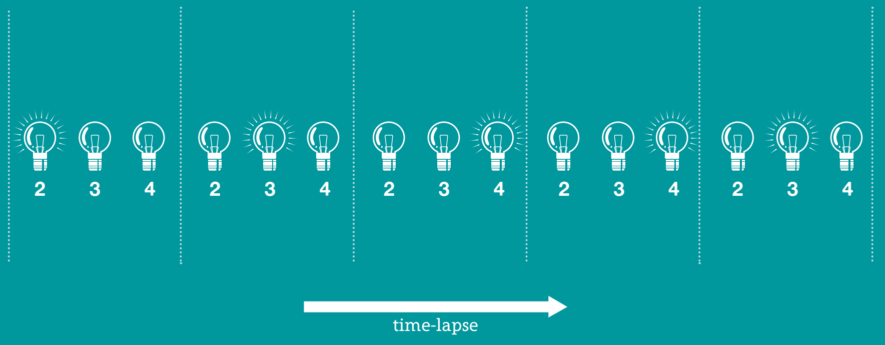
blueprint equivalents



array

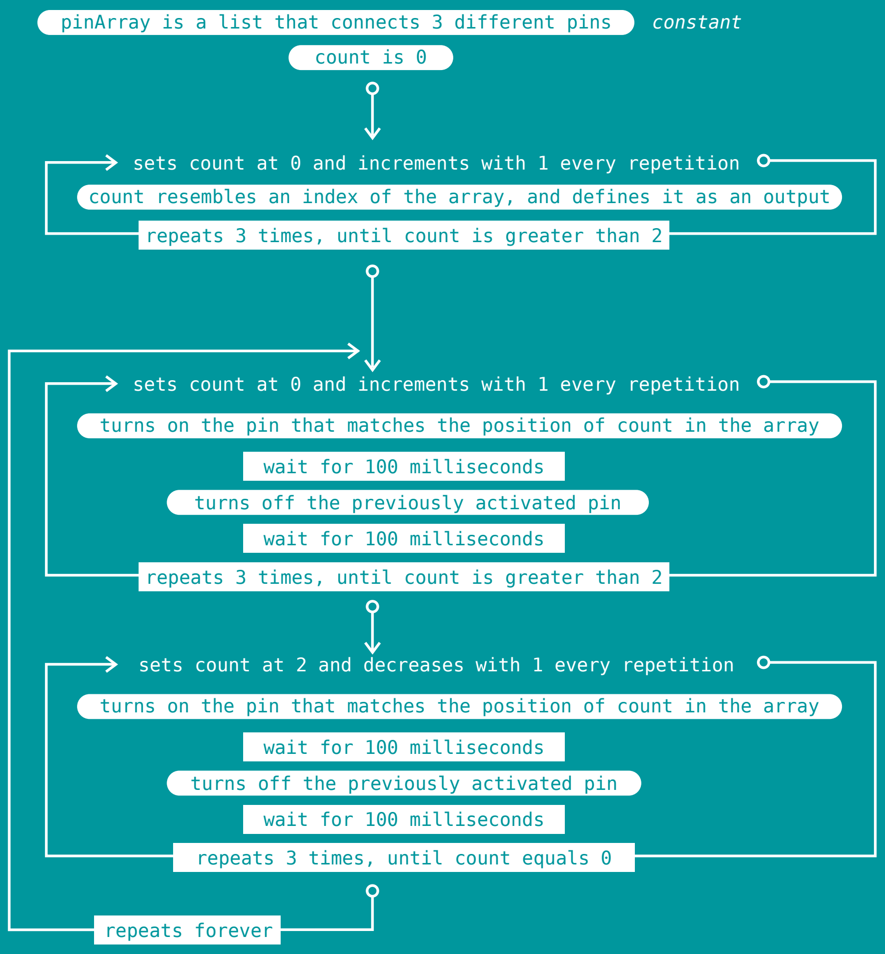
An array is a collection of variables that are accessed with an index number. It is a list of values, can be accessed through an index number. Often arrays are used for creating lists or table with a single reference.
The array is formulated in the following way: arrayname[indexnumber]. Arrays are zero indexed, which means the first position in the array is accessed by using 0 as index.

examples: *knight rider lights, tables, grid, lists, ring buffer*



```
int pinArray[] = {2, 3, 4};                                   data
int count = 0;                                                data

void setup(){
  for (count = 0; count < 3; count++) {                       condition
    pinMode(pinArray[count], OUTPUT);                         output
  repeats 3 times, until count is greater than 2
  }
}

void loop() {
  for (count = 0; count < 3; count++) {                       condition
    digitalWrite(pinArray[count], HIGH);                      output
    delay(100);                                               condition
    digitalWrite(pinArray[count], LOW);                       output
    delay(100);                                               condition
  repeats 3 times, until count is greater than 2
  }

  for (count = 2; count >= 0; count--) {                      condition
    digitalWrite(pinArray[count], HIGH);                      output
    delay(100);                                               condition
    digitalWrite(pinArray[count], LOW);                       output
    delay(100);                                               condition
    repeats 3 times, until count equals 0
  }
}
repeats forever
```

blueprint equivalents

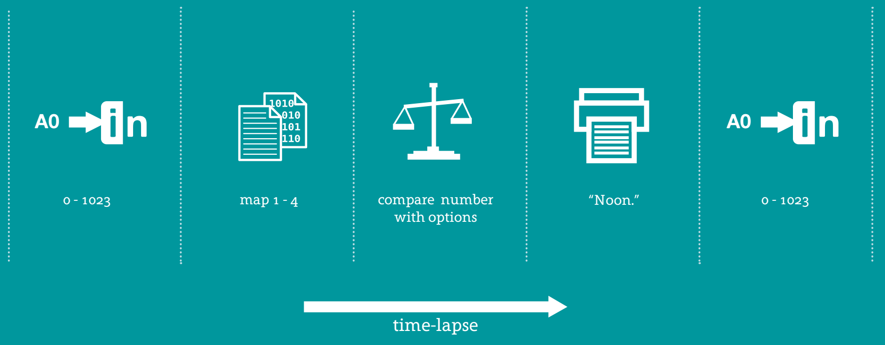


pinArray is a list that connects 3 different pins
constant
count is 0
sets count at 0 and increments with 1 every repetition
count resembles an index of the array, and defines it as an output
repeats 3 times, until count is greater than 2
sets count at 0 and increments with 1 every repetition
turns on the pin that matches the position of count in the array
wait for 100 milliseconds
turns off the previously activated pin
wait for 100 milliseconds
repeats 3 times, until count is greater than 2
sets count at 2 and decreases with 1 every repetition
turns on the pin that matches the position of count in the array
wait for 100 milliseconds
turns off the previously activated pin
wait for 100 milliseconds
repeats 3 times, until count equals 0
repeats forever

compare

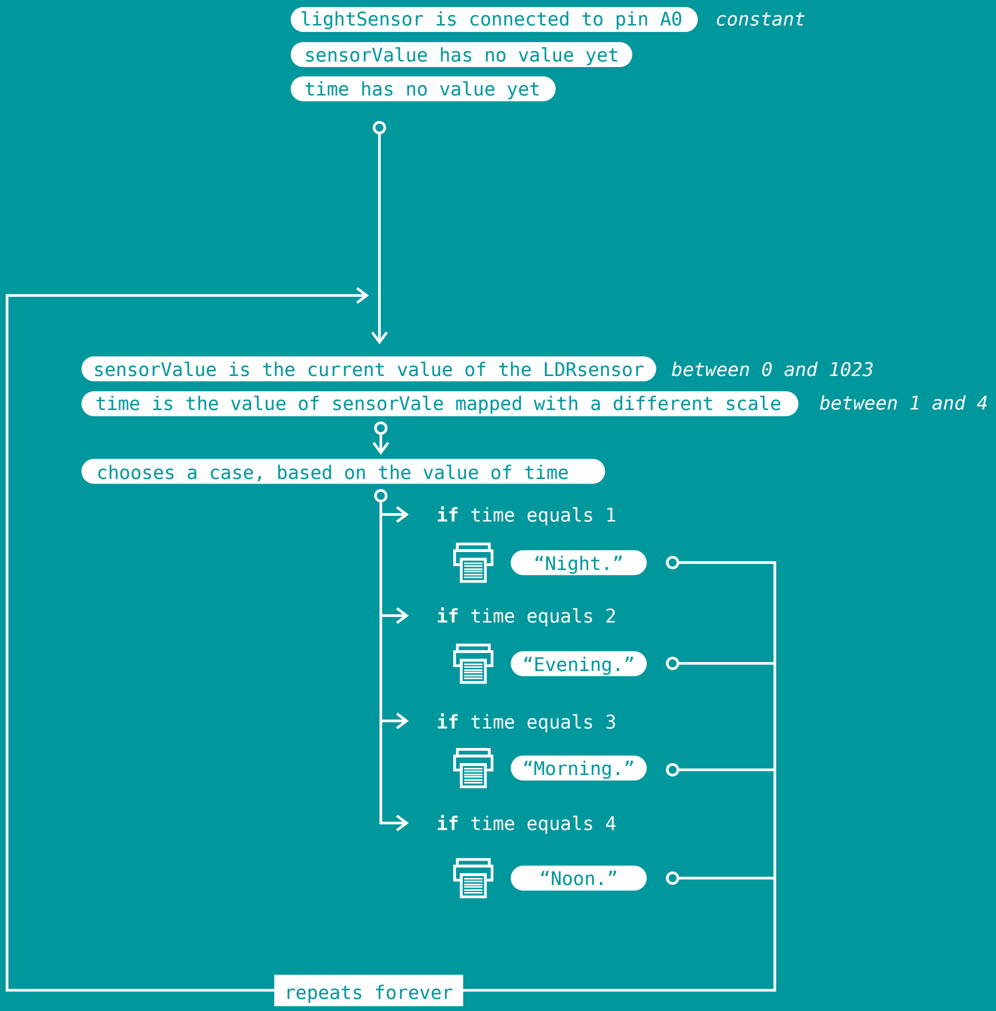
Compare is the activity of comparing two or more things and produce an action, based on this comparison. Comparing is an activity of condition, as it determines when and what should happen.
This pattern uses a switch case, to select a specific case based upon a numeral input. It maps an analog input to a scale of 1 to 4, and compares this number to four cases. If the number matches the case, it will execute that specific case.

examples: *events based on different conditions, verifying something*



```
int lightSensor = A0;                                  data
int sensorValue;                                       data
int time;                                              data

void setup() {
  Serial.begin(9600);                                  output
  pinMode(lightSensor, INPUT);                         input
}

void loop() {
  sensorValue = analogRead(lightSensor);               input
  time = map(sensorValue, 0, 1023, 1, 4);              transformer

  switch (time) {                                      condition
     case 1:
         Serial.println("Night.");                     output
         break;                                        condition
     case 2:
         Serial.println("Evening.");                   output
         break;                                        condition
     case 3:
         Serial.println("Morning.");                   output
         break;                                        condition
     case 4:
         Serial.println("Noon.");                      output
         break;                                        condition
  }
}
        repeats forever
```

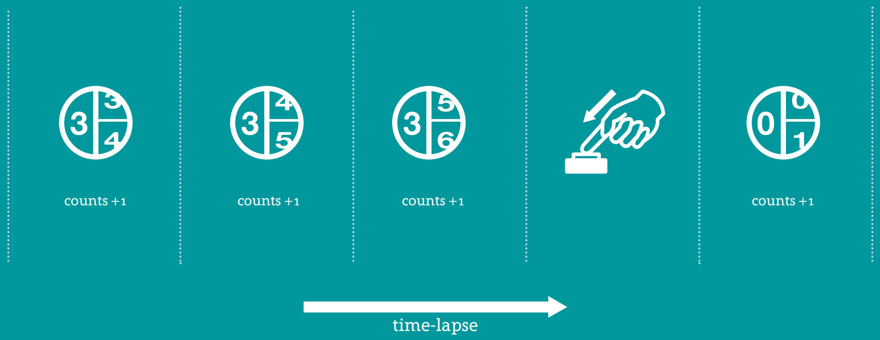
blueprint equivalents


lightSensor is connected to pin A0
constant
sensorValue has no value yet
time has no value yet
sensorValue is the current value of the LDRsensor
between 0 and 1023
time is the value of sensorVale mapped with a different scale
between 1 and 4
chooses a case, based on the value of time
if time equals 1
"Night."
if time equals 2
"Evening."
if time equals 3
"Morning."
if time equals 4
"Noon."
repeats forever

counter

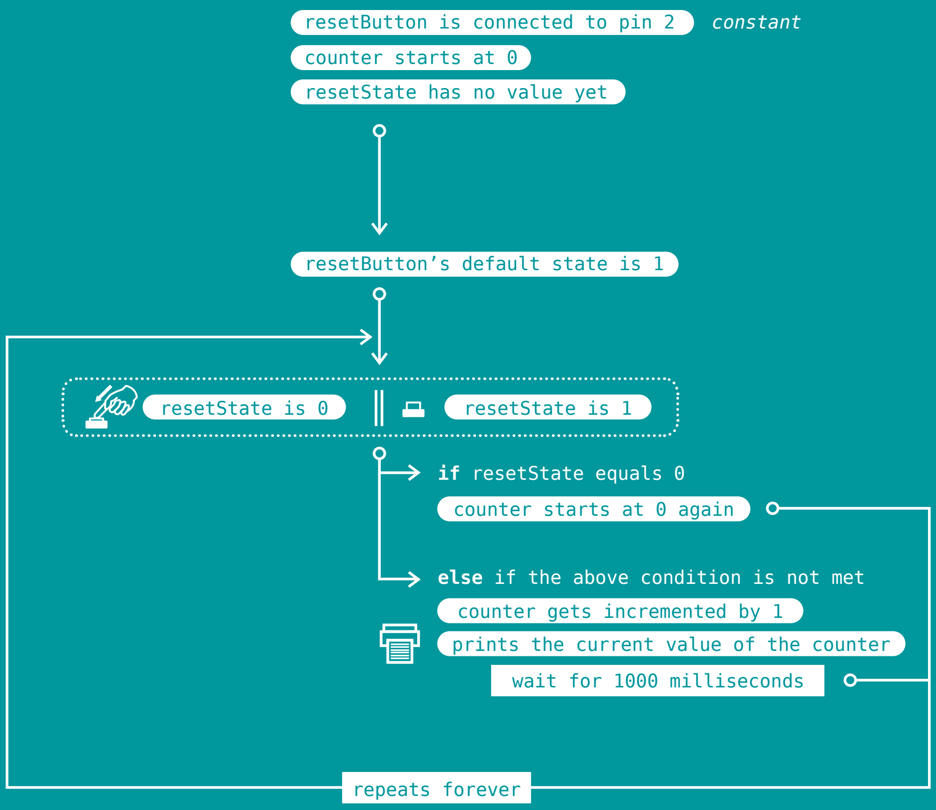
A counter is a pattern that counts how often something occurs. It allows counting of actions or behaviors. Counting works by reassigning a variable with its own value +1.
This pattern uses the looping nature of the Arduino to increment the counter, as long as the button is not pressed. It increments by 1, before delaying for 1 second. This will be repeated until the button is pressed, which resets the counter to 0.

examples: *pedometer, clock, specific amount of iterations of something*



```
int resetButton = 2;                                   data
int counter = 0;                                       data
int resetState;                                        data

void setup() {
  Serial.begin(9600);                                  output
  pinMode(resetButton, INPUT);                         input
  digitalWrite(resetButton, HIGH);                     transformer
}

void loop() {
  resetState = digitalRead(resetButton);               input

     if (resetState == 0) {                            condition
            counter = 0;                               transformer
     }
     else {                                            condition
            counter += 1;                              transformer
            Serial.println(counter);                   output
            delay(1000);                               condition
      }
}
repeats forever
```

blueprint equivalents



resetButton is connected to pin 2 *constant*
counter starts at 0
resetState has no value yet

resetButton's default state is 1

resetState is 0 || resetState is 1

if resetState equals 0
counter starts at 0 again

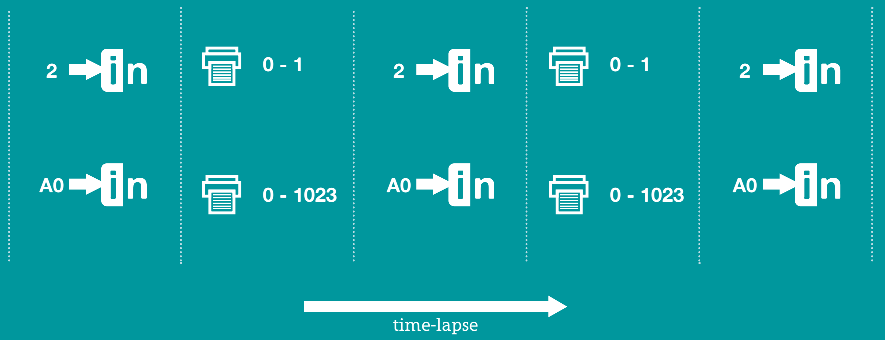
else if the above condition is not met
counter gets incremented by 1
prints the current value of the counter
wait for 1000 milliseconds

repeats forever

input

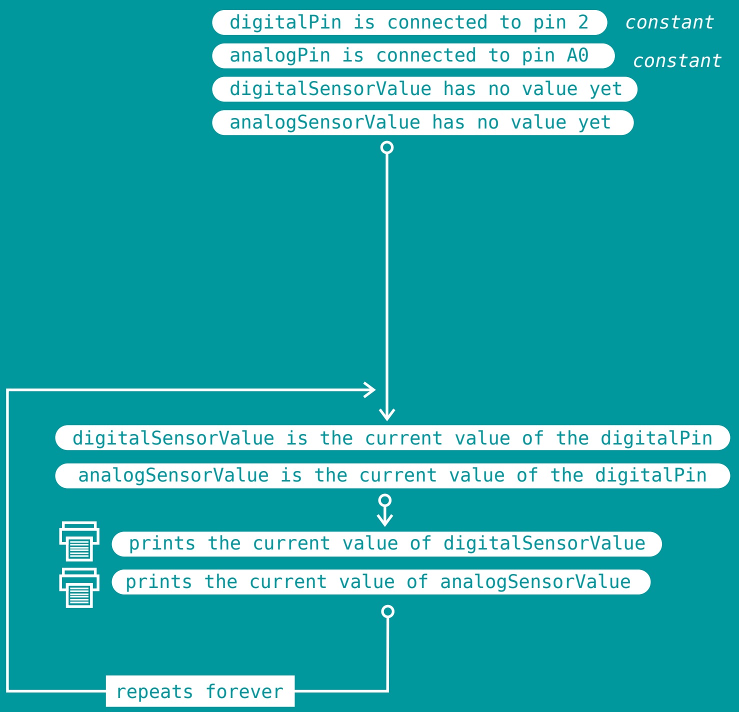
Input is one of the fundamental patterns of Arduino. It is the action of converting electrical inputs into digital values. These values in turn, are used to make interaction or behaviors happen.
Digital input is registered only as HIGH or LOW, or 1 and 0. Analog input is registered as a value between 0 and 1023. These values can come from all sorts of inputs, such as buttons or sensors.

examples: *buttons, switches, motion sensor, infra-red sensor, other sensors*



```
int digitalPin = 2;                                          data
int analogPin = A0;                                          data
int digitalSensorValue;                                      data
int analogSensorValue;                                       data

void setup() {
  Serial.begin(9600);                                        output
  pinMode(digitalPin, INPUT);                                input
  pinMode(analogPin, INPUT);                                 input
}

void loop() {
  digitalSensorValue = digitalRead(digitalPin);              input
  analogSensorValue = analogRead(analogPin);                 input

  Serial.println(digitalSensorValue);                        output
  Serial.println(analogSensorValue);                         output
}
repeats forever
```

blueprint equivalents

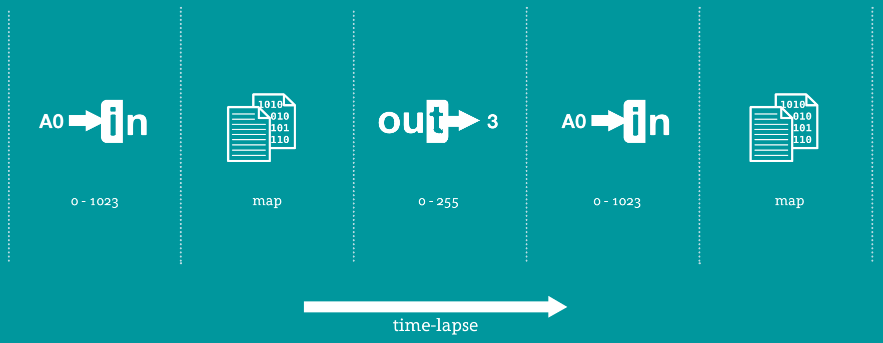


```
digitalPin is connected to pin 2              constant
analogPin is connected to pin A0              constant
digitalSensorValue has no value yet
analogSensorValue has no value yet

digitalSensorValue is the current value of the digitalPin
analogSensorValue is the current value of the digitalPin

prints the current value of digitalSensorValue
prints the current value of analogSensorValue
repeats forever
```

map

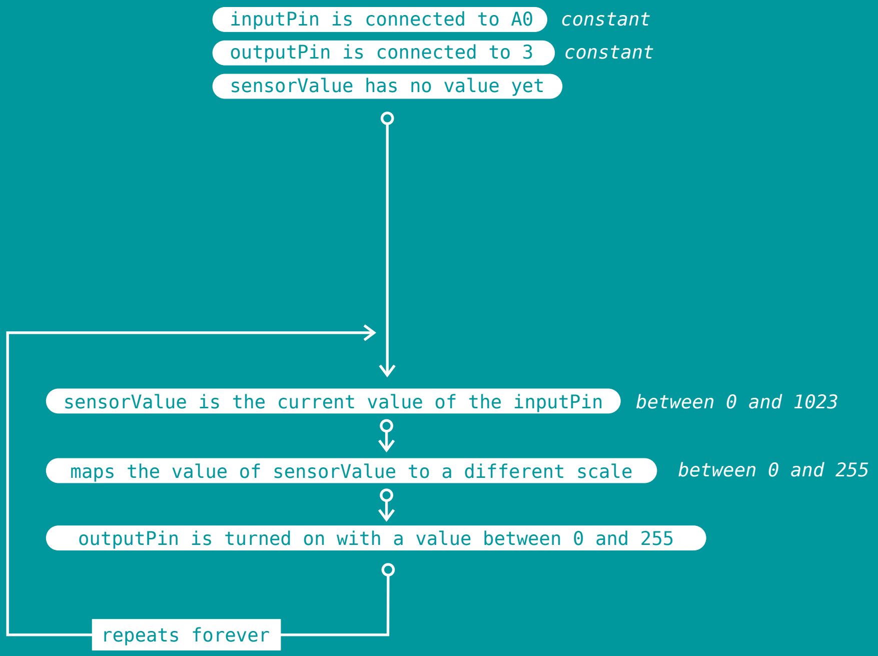
Mapping the action of remapping a value from one range to another. This means a range can be redefined to a different range. Mapping is a useful pattern, because often a different range of values is required to achieve the desired behavior or interaction.
In the pattern below, an analog input value can be between 0 and 1023, depending on what the sensor registers, is remapped to a range of 0 to 255.

examples: *diffent scales, Celsius to Fahrenheit, conversion*



```
int inputPin = A0;                                          data
int outputPin = 3;                                          data
int sensorValue;                                            data

void setup() {
  pinMode(inputPin, INPUT);                                 input
  pinMode(outputPin, OUTPUT);                               output
}

void loop() {

  sensorValue = analogRead(inputPin);                       input

  sensorValue = map(sensorValue, 0, 1023, 0, 255);    transformer

  analogWrite(outputPin, sensorValue);                      output
}
repeats forever
```

blueprint equivalents

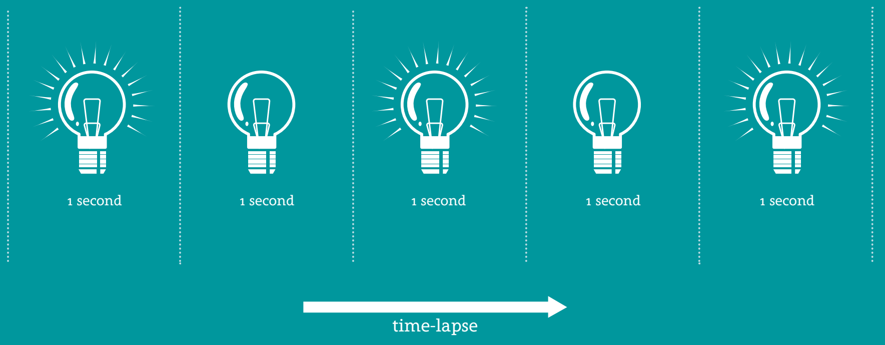


```
inputPin is connected to A0  constant
outputPin is connected to 3  constant
sensorValue has no value yet

sensorValue is the current value of the inputPin  between 0 and 1023
maps the value of sensorValue to a different scale  between 0 and 255
outputPin is turned on with a value between 0 and 255
repeats forever
```

output

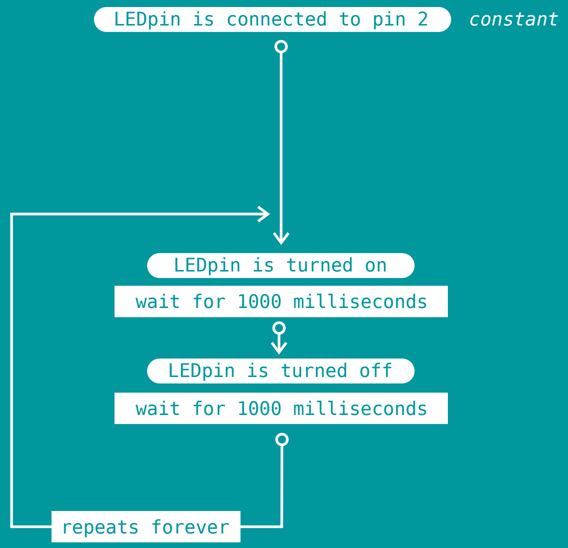
Output is one of the fundamental patterns of Arduino. It is the action of converting digital values into electrical outputs, which allows the Arduino to be able to control electrical hardware.
Outputs can be a variety of things, such as LEDs, servo motors or buzzers. The pattern below shows how to control an LED as a digital output, which means it can be in two states: on or off.

examples: *LEDs, servo motors, buzzers, LCD displays, other actuators*



```
int LEDpin = 2;                                  data

void setup() {
  pinMode(LEDpin, OUTPUT);                       output
}

void loop() {
  digitalWrite(LEDpin, HIGH);                    output
  delay(1000);                                   condition

  digitalWrite(LEDpin, LOW);                     output
  delay(1000);                                   condition
}

repeats forever
```

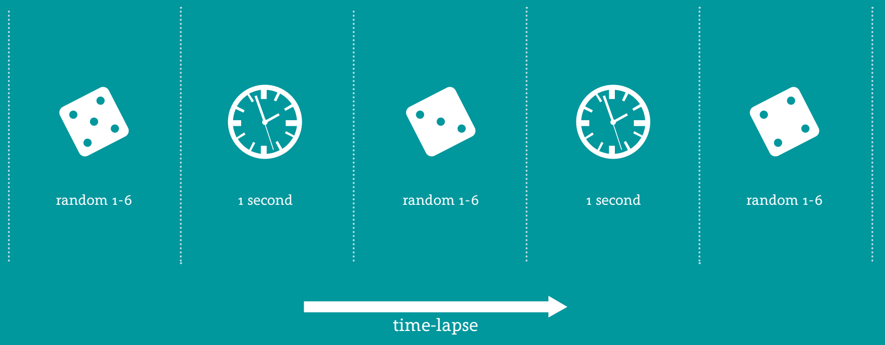
blueprint equivalents


LEDpin is connected to pin 2
constant
LEDpin is turned on
wait for 1000 milliseconds
LEDpin is turned off
wait for 1000 milliseconds
repeats forever

random

Arduino does not posses the ability to create truly random numbers. It has a very long sequence of numbers, however it always starts at the beginning of this sequence (and thus is always the same).
This pattern uses randomSeed during setup, which is a function that reads the current value of pin A0. In turn, it uses this value to randomly start somewhere on this long sequence of numbers, which makes the output appear to be random.

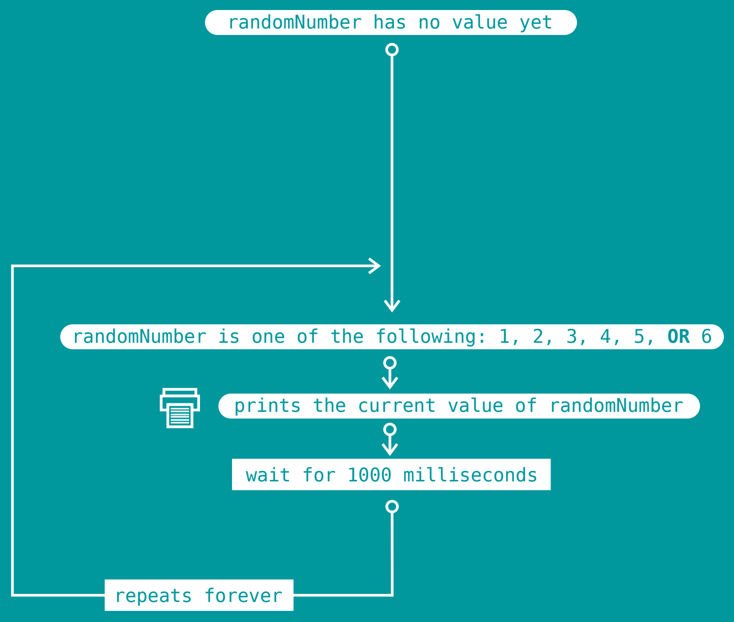
examples: *dice, shuffle, unplanned events, unpredictable events*



```
long randomNumber;                                  data

void setup() {
  Serial.begin(9600);                               transformer
  randomSeed(analogRead(A0));                       input
}

void loop() {

  randomNumber = random(1, 7);                      transformer

  Serial.println(randomNumber);                     output

  delay(1000);                                      condition
}
```

repeats forever

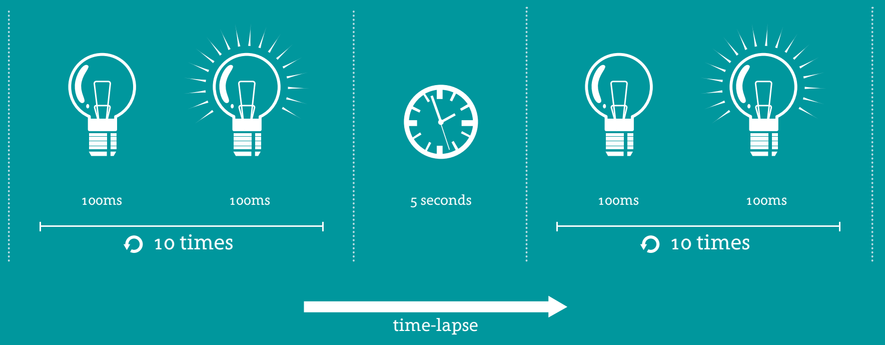
blueprint equivalents


randomNumber has no value yet
randomNumber is one of the following: 1, 2, 3, 4, 5, OR 6
prints the current value of randomNumber
wait for 1000 milliseconds
repeats forever

repeat

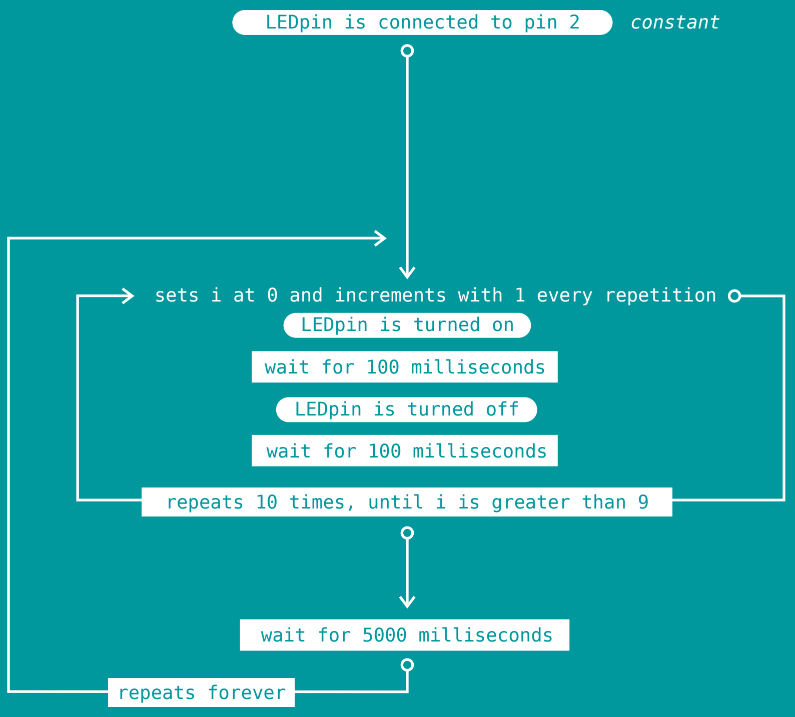
Repeat is one of the fundamental patterns of Arduino. It is the action of repeating a specific set of instructions a number of times. By nature, Arduino already repeats its loop section, however to repeat a specific section, instead of the entire code, another function is required.
This pattern uses a for loop, to repeat a set of instructions. The for loop has a begin condition and end condition, and instructions of what it should do every loop.

examples: *repetition, same action a lot of times, similar behaviors*



```
int LEDpin = 2;                                         data

void setup() {
  pinMode(LEDpin, OUTPUT);                              output
}

void loop() {

  for(int i = 0; i < 10; i++) {                         condition

    digitalWrite(LEDpin, HIGH);                         output
    delay(100);                                         condition

    digitalWrite(LEDpin, LOW);                          output
    delay(100);                                         condition
   repeats 10 times, until i is greater than 9
  }
  delay(5000);                                          condition
}
repeats forever
```

blueprint equivalents

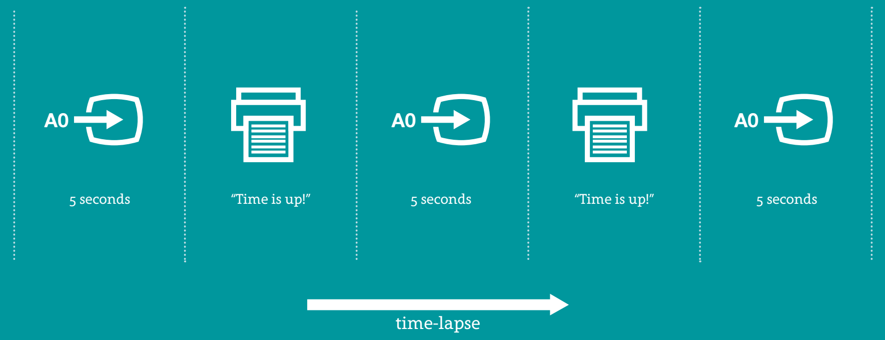


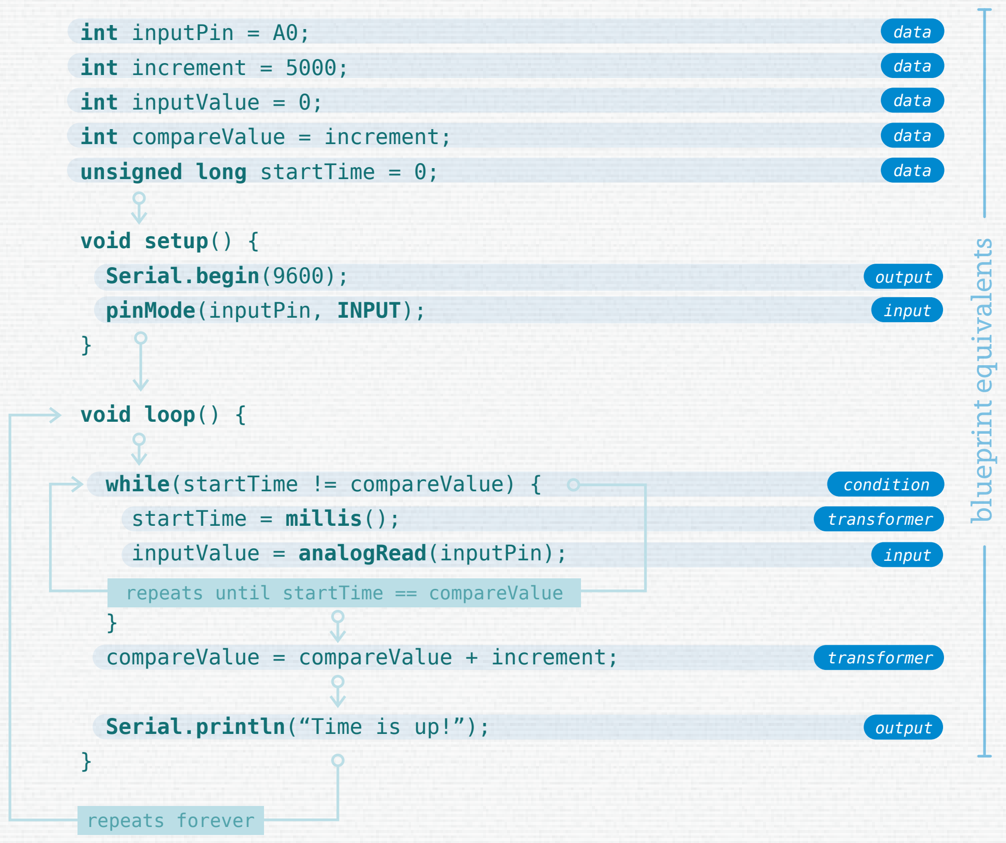
LEDpin is connected to pin 2
constant
sets i at 0 and increments with 1 every repetition
LEDpin is turned on
wait for 100 milliseconds
LEDpin is turned off
wait for 100 milliseconds
repeats 10 times, until i is greater than 9
wait for 5000 milliseconds
repeats forever

timer

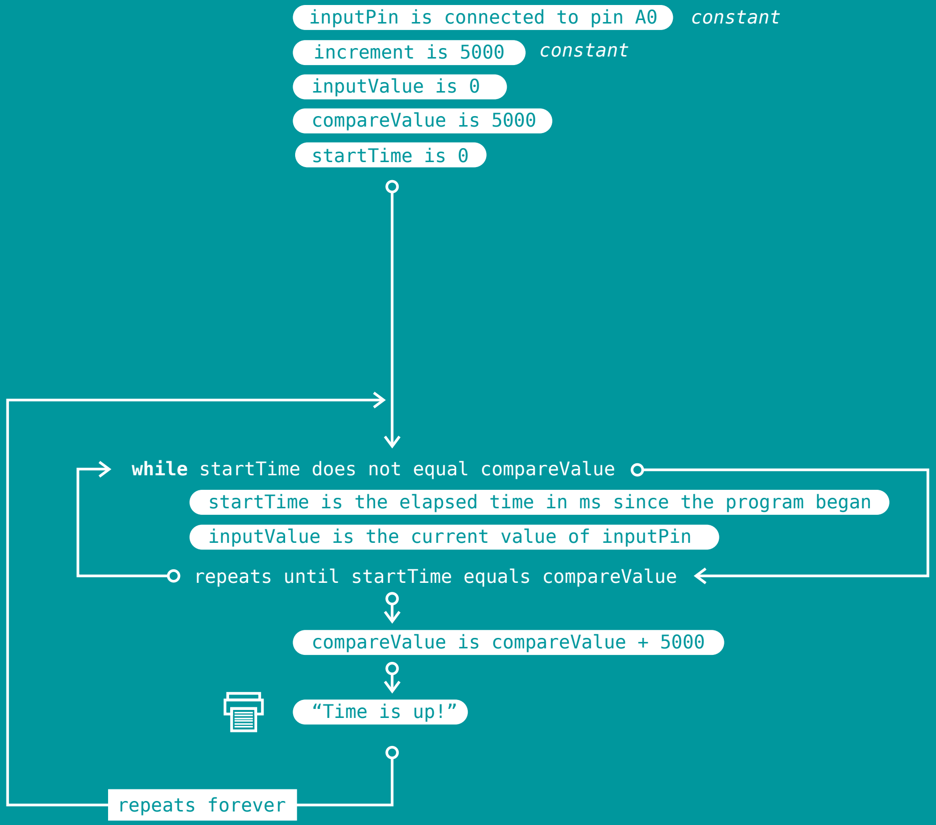
The timer is a pattern that allows actions to happen only during or after a defined amount of time. The Arduino has no built-in clock, and the delay function only stalls the whole program for the time of the delay.
This timer uses the mills function, which returns the number of milliseconds from the point of when the Arduino began to run the program. By comparing the start value with an increment, this function create a reference point that allows to keep track of time.

examples: *alarm, game elements, phased events, time related challenges*





```
int inputPin = A0;                                    data
int increment = 5000;                                 data
int inputValue = 0;                                   data
int compareValue = increment;                         data
unsigned long startTime = 0;                          data

void setup() {
  Serial.begin(9600);                                 output
  pinMode(inputPin, INPUT);                           input
}

void loop() {

  while(startTime != compareValue) {                  condition
    startTime = millis();                             transformer
    inputValue = analogRead(inputPin);                input
  repeats until startTime == compareValue
  }
  compareValue = compareValue + increment;            transformer

  Serial.println("Time is up!");                      output
}
repeats forever
```



```
inputPin is connected to pin A0   constant
increment is 5000   constant
inputValue is 0
compareValue is 5000
startTime is 0

while startTime does not equal compareValue
    startTime is the elapsed time in ms since the program began
    inputValue is the current value of inputPin
repeats until startTime equals compareValue
compareValue is compareValue + 5000
"Time is up!"
repeats forever
```

blank on purpose