Intro to the IOT: Design and Applications
2017 – Demonstration 1

The goal of this exercise is to get a first hands-on experience with the Arduino UNO development environment (IDE) that allows you to quickly build applications involving digital and analog I/O, as well as a debugging console.

1. Serial Monitor

The Arduino IDE provides a built-in serial monitor capable of displaying text sent from the board's serial port (integrated with USB). This monitor is particularly useful when debugging applications, therefore we will use it as our main output during the rest of this demo.
The first step is to configure the IDE to use the correct USB port for the monitor. Connect the board to the PC using the USB cable, then in the IDE, go to *Tools→ Port* and select the port the board is connected to. For Linux, typically the Arduino board will be on `/dev/ttyACM0`. The port currently used by Arduino is shown in the bottom right corner of the IDE window.
In order to use the board's serial port, an application must first initialize it. This is done within the `setup()` function of the application by using `Serial.begin(baudrate)`, where `baudrate` is an integer that specifies the baud rate the port must use. Once the serial port has been initialized, the application can send text to it by using `Serial.print()` within the `loop()` function. `print()` is a very versatile function that allows printing numerical values, characters and strings, as shown next:

- `Serial.print(78)` gives "78"
- `Serial.print(1.23456)` gives "1.23"
- `Serial.print('N')` gives "N"
- `Serial.print("Hello world.")` gives "Hello world."

More details about Arduino's `print()` function can be found in *http://arduino.cc/en/Serial/Print*.

Hands-on:
- Create a sketch with the necessary `setup()` and `loop()` functions in it. Use the `Serial.begin()` and `Serial.print()` to initialize the serial port and send a message to the Serial Monitor.
Verify and upload your sketch to the board. In the IDE, go to *Tools→Serial Monitor*, this will open the monitor window. Verify the baud rate of the monitor (bottom-right corner) is the same as in the application, in our case 9600 bps. You should see your message displayed on the monitor.

2. Timing

The Arduino IDE provides a series of functions that allow more control over the timing of an application's execution. The most common one is the `delay(ms)` function which pauses the execution for `ms` milliseconds. However, `delay()` has the drawback that no other processing can be done while waiting for the time period to complete, therefore other timing functions are available:

- `millis()` returns the number of milliseconds since the program started as an `unsigned long`.
- `micros()` returns the number of microseconds since the program started as an `unsigned long`.

Hands-on:
- Modify the previous sketch so the “Hello world!” message is printed on the serial monitor every 3 seconds.
- Assume a system has 3 messages that require to be sent every 3, 7, and 15 seconds respectively. Create a sketch that prints the messages at their required frequencies on the serial monitor. (HINT: You can keep a counter per each message and update the counters at a common frequency, let's say 1 second. Then it is only necessary to detect when a counter reaches the correct value to print the message and reset the counter.)
- The values returned by the `millis()` and `micros()` functions are internally kept by a counter, which means soon or later they will reset to 0. If they both return an unsigned 4-byte width value, how long is the period each one of them can measure? Typical embedded and cyber physical systems must operate continuously for months or even years, how would you keep track of such long time periods?

3. Digital Pins

The Arduino UNO has 14 digital pins (numbered 0-13) that can be configured either as input or output. The functionality of a digital pin is defined in the `setup()` function by calling `pinMode(pin,mode)`, where `pin` selects the pin to be configured and `mode` is either `INPUT` or `OUTPUT`.

Once a digital pin is configured, the `digitalWrite(pin,value)` and `digitalRead(pin)` functions are used within `loop()` to write or read the pin. When writing to a pin, `value` must be either `HIGH` or `LOW`.

This way we have that:

- `pinMode(5,OUTPUT)` configures digital pin 5 as an output.
- `digitalWrite(5,HIGH)` sets digital pin 5 to a `HIGH` level.
- `pinMode(6,INPUT)` configures digital pin 6 as input.
- `pinval = digitalRead(6)` reads digital pin 6 an assigns its value to the variable `pinval`.

More details about Arduino's digital pins can be found at *http://arduino.cc/en/Tutorial/DigitalPins*

4. Analog Read

The Arduino Uno board contains a 6 channel, 10-bit analog to digital converter. This means that it will map input voltages between 0 and 5 volts into integer values between 0 and 1023. The function `analogRead(pin)` is used within `loop()` to read the value from the specified analog pin. It gets the analog pin name or pin number and returns the converted value. Analog pin names are `A0` to `A5`.

- `analogRead(0)` reads analog value from pin 0.
- `analogRead(A1)` reads analog value from pin 1.

More details about Arduino's analog inputs can be found at *http://arduino.cc/en/Tutorial/AnalogInput*

The `analogRead()` function returns a number between 0 and 1023. The function `map()` can be used to convert this value to corresponding voltage.

- `map(value, fromLow, fromHigh, toLow, toHigh)` re-maps a number from one range to another. That is, a value of `fromLow` would get mapped to `toLow`, a value of `fromHigh` to `toHigh`, values in-between to values in-between.
- `map(value, 0, 1023, 0, 5)` converts read values to corresponding voltages.

More details about `map()` function can be found at *http://arduino.cc/en/reference/map*

Hands-on: Light Meter Gage

Your task is to make a light meter gage with a light sensor, a servo, Grove shield and Arduino Uno board. Servo is a motor which you can locate it at any position between 0 to 179 degrees. It uses the "Servo.h" library and operates with `write` command:

```
#include <Servo.h>
Servo myservo;  // create servo object to control a servo

void setup()
{
  myservo.attach(3);  // attaches the servo on D3 to the servo object
}

void loop()
{
    myservo.write(50);  // fixes the servo at 50 degrees
}
```

The Light Meter Gage has two phases:

- **Calibration:** in this phase you measure the maximum and minimum available light. This phase takes 10 seconds long and the onboard LED (attached to pin 13) starts blinking. During this phase you should face the light sensor to the maximum available light around yourself (e.g. your PC display) and also to the minimum light (you can put your finger on the sensor). (HINT: find and store the maximum and minimum `analogRead` values of light sensor in two variables.)

- **Working:** after 10 seconds calibration phase the onboard LED stops blinking and the Servo starts moving like a gage according to amount of light. It should move to 0 degree position for darkest condition and to 179 degree position in the lightest condition and somewhere in between for the current room light. (HINT: map the maximum and minimum `analogRead` values to 0 and 179, and then move the servo to mapped value.)