Object Oriented Programming

Object Oriented Programming is a next-level advance in programming. This complex skill allows you to create objects, that have certain functions or “methods” associated with them. Every Library in Arduino (which you will learn about in a bit) uses object oriented programming to get things done, and you may start to recognize the structure in the Arduino libraries you are already using. This is a very abstract concept, and it will be difficult to understand at first. You will very very likely need to try things several times before you get them right.

Coding in general started with simple instructions that are followed in sequential order. This is called “Procedural code” and it is how to describe all the code you've written so far in this course. Procedural code is the name for code that follows a set and linear procedure to operate. “Object Oriented code” is a little more non-linear, because you need to create objects and then ask them to do things by calling methods, which has the logic of your program jumping all over the place, instead of sequentially moving top-to-bottom through all the lines of code you’ve written. There is also “Functional code” which is also different, but will not be covered here.

# Some Supporting Documents

There are many new vocabulary words that come with Object Oriented Programming. There is a list of these terms summarized in the table below, but it is best to learn these by reading the delightful tutorial written by the good people at Adafruit:

<https://learn.adafruit.com/multi-tasking-the-arduino-part-1/a-classy-solution>

The vocabulary here is important, so you can correctly refer to all the parts of an OOP program and search relevant terms if you ever need any help when not in class/ able to ask a teacher.

|  |  |
| --- | --- |
| Class | What an OOP object is called, the code indicator you’re using OOP |
| Member Variable | A variable that is used within the Class. It can be public (accessible outside the class) or private (only accessible within the class) |
| Constructor | The code in a class that actually ‘builds’ the object |
| Member Function/ Class Method | A custom function that exists within the class. It can be called by the program, and these are what make OOP powerful. |
| instance | The name for a single use of an object/ class. In any given program you can have one or many instances of a class, but it will work the same each time you “instantinize” it in the program, or call a class method. |

# Example Sketch 1

Below is an example OOP program. It doesn’t do much, but it is a first look at how this can work. The stuff written in side the class curly brackets defines the object, for object oriented programming. The

**class Thinggy{** //the keyword ‘class’ tells Arduino you want OOP

**int favNum;** //any variables, scoped for the whole class

**public:** //anything below this are functions of the OOP

**Thinggy(int num){** //the constructor: same name as class

**favNum = num;** //any startup operations can happen here

**}**

**void setFavNumber(int x){** //a ‘setter’ function or method

**favNum = x;**  //this function sets the value of favNum

**}**

**int getFavNumber(){** //a ‘getter’ function or method

**return favNum;** //this function returns the value of favNum

**}**

**};** // end of OOP instructions

//================= Program starts here =======================

**Thinggy num1(7);** // construct a ‘Thinggy’ named ‘num1’

//^instantiationpass in the value 12 to Thinggy ‘num1’

**void setup(){** // void setup happens after constructor

**Serial.begin(9600);** // because you can’t do this within a class

**}**

**long p = 0;** // a global variable for ‘void loop()’

**void loop(){**

**Serial.print("Your Favorite Number is: ");**

**Serial.println(num1.getFavNumber());** //use the ‘getter’ func

**delay(1000);**

**p = p + 1;**

**if((p%10)==3){**

**num1.setFavNumber(p);**  //use the ‘setter’ function

// see how similar this is to the syntax of ‘Serial.print()’?

**Serial.print("Your new Favorite Number is: ");**

**Serial.println(num1.getFavNumber());** // use the ‘getter’

**}**

**}**

Explain the Serial Monitor behavior of the above Example Sketch 1 with written sentences:

Copy/ Paste this example program into your own Arduino IDE and test it. Then describe the output of this program on a serial monitor. (What text does it print and time intervals are there between the printed outputs?)

Answer here

How is this output achieved by the program? (Explain the logic of the program)

Answer Here

## 

# Example Sketch 2

**class Blinker{**

**int ledPin;** //member variables to be used in the class

**long waitTime;** //these are declared, but not given values

**unsigned long previousMillis;**

**bool pinOnOff;** // the values will come in the constructor

**public:**

**Blinker(int num, long wait){** //this is the constructor

**ledPin = num;** //set the member variable value

**waitTime = wait;** //set the member variable value

**pinMode(ledPin, OUTPUT);** //set up the pin as an output

**previousMillis = millis();** //needed for tracking time

**pinOnOff = true;** //start with the LED turned on

**digitalWrite(ledPin, pinOnOff);** //turn the LED on to start

**}**

**void blinkit(){**

**if((millis()-previousMillis)>=waitTime){** //if enough time...

**previousMillis = millis();** //reset the timer/counter

**pinOnOff = !pinOnOff;** //change the value of the LED output

**digitalWrite(ledPin, pinOnOff);** //change the LED

**}** //end if logic

**}** //end blinkit() function

**};** // end class declaration

**Blinker light1(3,1000);** //constructor #1

**Blinker light2(4,910);** //constructor #2

**Blinker light3(5,130);**  //each constructor takes 2 inputs

**Blinker light4(6,340);** //first input is the pin#

**Blinker light5(7,1520);** //second input is the blink delay

**void setup(){** // don’t need to do anything here...

**}** // but Arduino requires it in a program

**void loop(){**

**light1.blinkit();** //call the blinkit method on light1 object

**light2.blinkit();** //call the blinkit method on light2 object

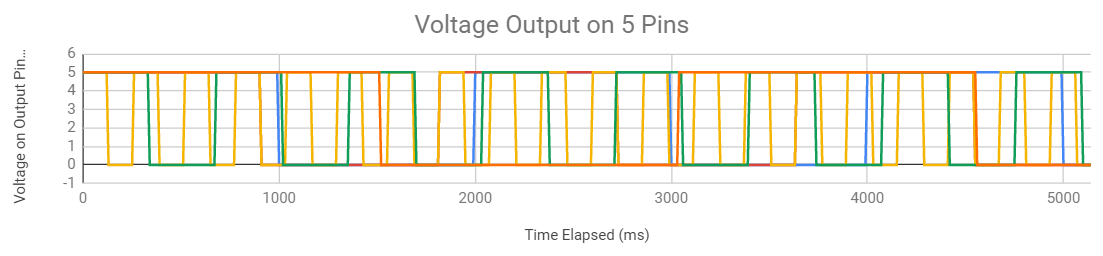
**light3.blinkit();** //call the blinkit method on light3 object

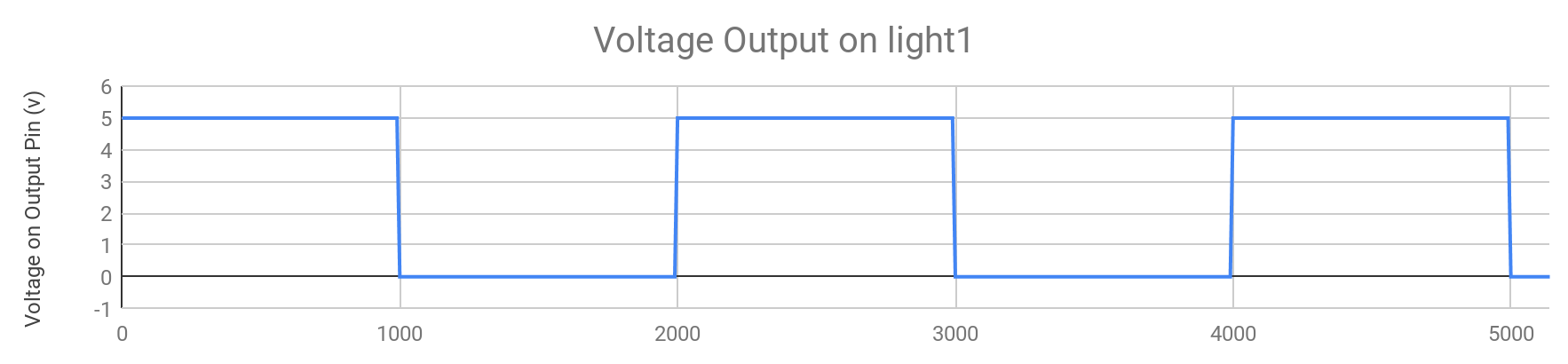
**light4.blinkit();** //call the blinkit method on light4 object

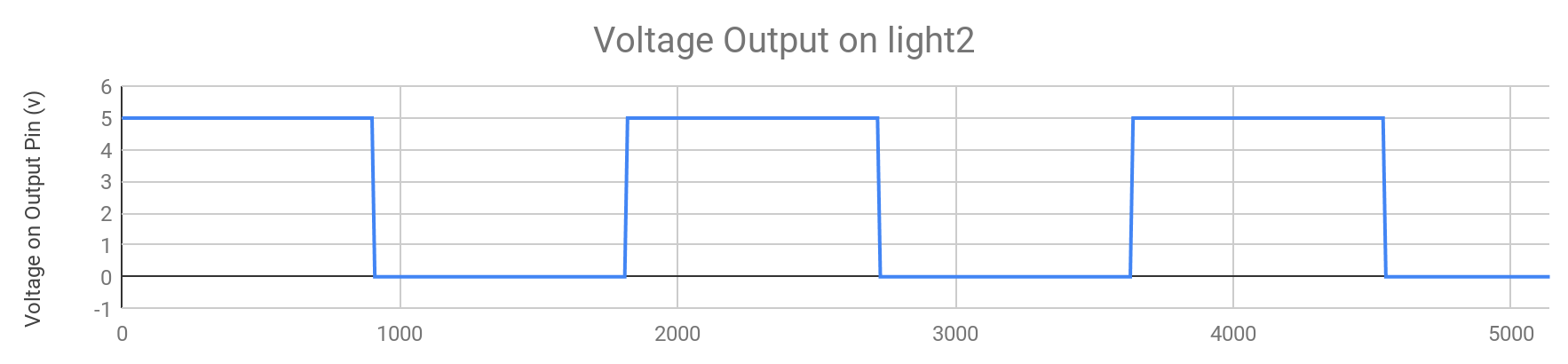
**light5.blinkit();** //call the blinkit method on light5 object

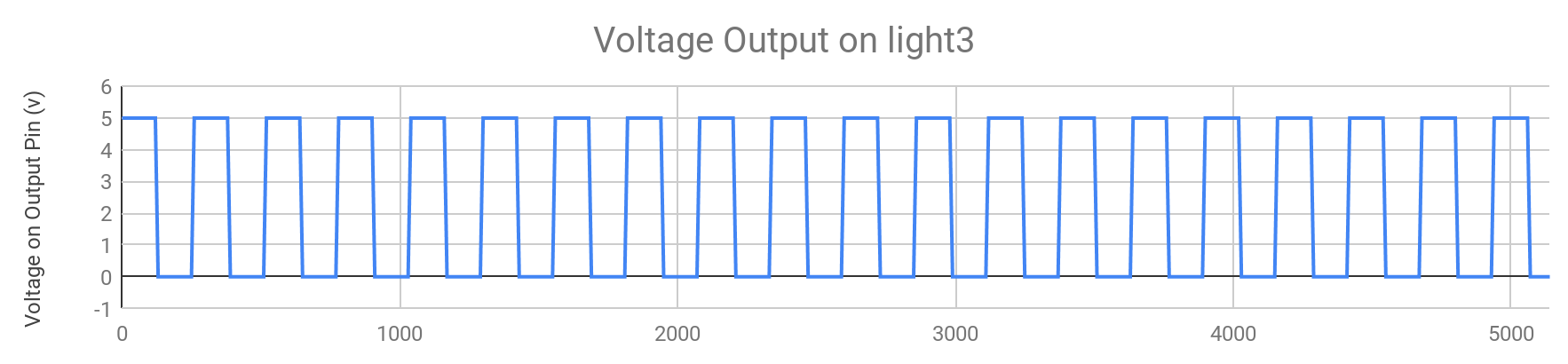
**}**

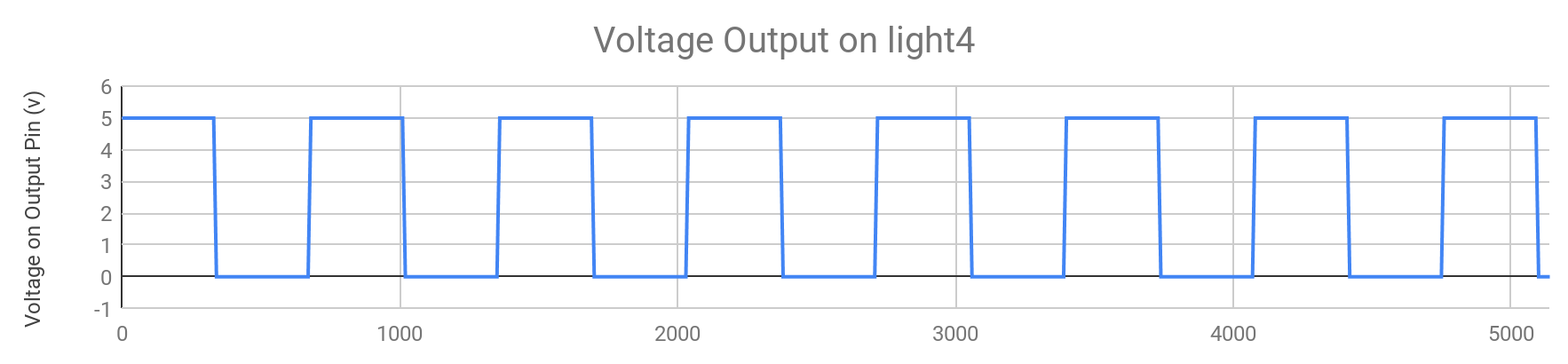
The Example Sketch 2 above would output the voltage shown in the graphs below:

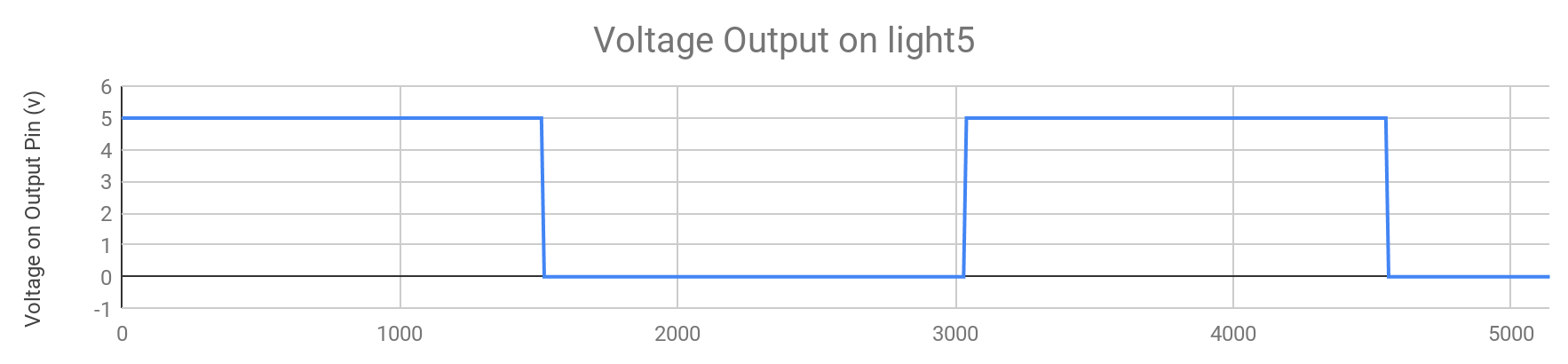












Understanding how this voltage relates to the program is an important part of understanding microcontrollers. Please explain in at least one (6 sentence) paragraph how and why the voltage output shown above comes from this program.

Answer here

# 

# Explanations From across the Internet

[A Classy Solution](https://learn.adafruit.com/multi-tasking-the-arduino-part-1/a-classy-solution) << Adafruit maintains the most cohesive explanation of OOP for Arduino

[Arduino, the Object Oriented Way](http://paulmurraycbr.github.io/ArduinoTheOOWay.html) << some person wrote this and put it on GitHub

[Tutorial: Object Oriented Programming with the Arduino](https://youtu.be/S_uaROFnWSg) << someone on YouTube

# Create your own OOP Sketches

*You don't know if you understand it, until you can create it from nothing...*

Using what you have seen above, in the example sketches and from Arduino’s reference materials. Create your own sketch that implements a while loop in some way. You have plenty of creative license in this goal, but you must make a sketch that successfully uses some OOP. A screenshot of your sketch and output is half of your response to these. A written explanation of the logic is also required to prove that you understand what you are doing with this work. Primary tasks are required for all students. Secondary tasks are required to get a top grade.

* Primary tasks:
  + successfully implement an OOP object in a program

Screenshot & Explanation here

* Moderate understanding secondary tasks:
  + Use ‘setter’ methods to change the properties of an object within your program

Screenshot & Explanation here

* + Use ‘getter’ methods to get information about your object's properties (variables)

Screenshot & Explanation here

* Advanced understanding secondary tasks:
  + Get real creative with some OOP in your programming….

Screenshot & Explanation here

# 