TANGIBLE MEDIA & PHYSICAL COMPUTING

ESSENTIAL ARDUINO

AGENDA

WHAT IS ARDUINO
WHY ARDUINO
ARDUINO TOOLS
ARDUINO MICROCONTROLLER
ARDUINO I/O
ARDUINO LANGUAGE

WHAT IS ARDUINO

ARDUINO WAS BORN AT THE IVREA INTERACTION DESIGN INSTITUTE AS AN EASY TOOL FOR FAST PROTOTYPING, AIMED AT STUDENTS WITHOUT A BACKGROUND IN ELECTRONICS AND PROGRAMMING.

IT SOON REACHED A WIDER COMMUNITY, THE ARDUINO BOARD ADAPTED & DIFFERENTIATED FROM SIMPLE 8-BIT BOARDS TO PRODUCTS FOR IOT, WEARABLES AND EMBEDDED ENVIRONMENTS.

WHAT IS ARDUINO

ARDUINO IS AN OPEN-SOURCE ELECTRONICS PLATFORM BASED ON EASY-TO-USE HARDWARE AND SOFTWARE. ARDUINO BOARDS ARE ABLE TO READ INPUTS AND TURN IT INTO AN OUTPUT.

TO DO SO YOU USE THE ARDUINO PROGRAMMING LANGUAGE (BASED ON WIRING), AND THE ARDUINO SOFTWARE (IDE), BASED ON PROCESSING.

WHY ARDUINO

ARDUINO IS A KEY TOOL TO LEARN NEW THINGS.

ANYONE, DUE TO ITS SIMPLE AND ACCESSIBLE USER EXPERIENCE, CAN START TINKERING BY FOLLOWING THE STEP BY STEP INSTRUCTIONS OF A KIT, OR SHARING IDEAS ONLINE WITH OTHER MEMBERS OF THE ARDUINO COMMUNITY.

WHY ARDUINO

OVER THE YEARS ARDUINO HAS BEEN THE BRAIN OF THOUSANDS OF PROJECTS, FROM EVERYDAY OBJECTS TO COMPLEX SCIENTIFIC INSTRUMENTS.

A WORLDWIDE COMMUNITY OF MAKERS HAVE GATHERED AROUND THIS OPEN-SOURCE PLATFORM AND THEIR CONTRIBUTIONS HAVE ADDED UP TO AN INCREDIBLE AMOUNT OF ACCESSIBLE KNOWLEDGE.

WHY ARDUINO

THERE ARE MANY OTHER MICROCONTROLLER PLATFORMS AVAILABLE FOR PHYSICAL COMPUTING.

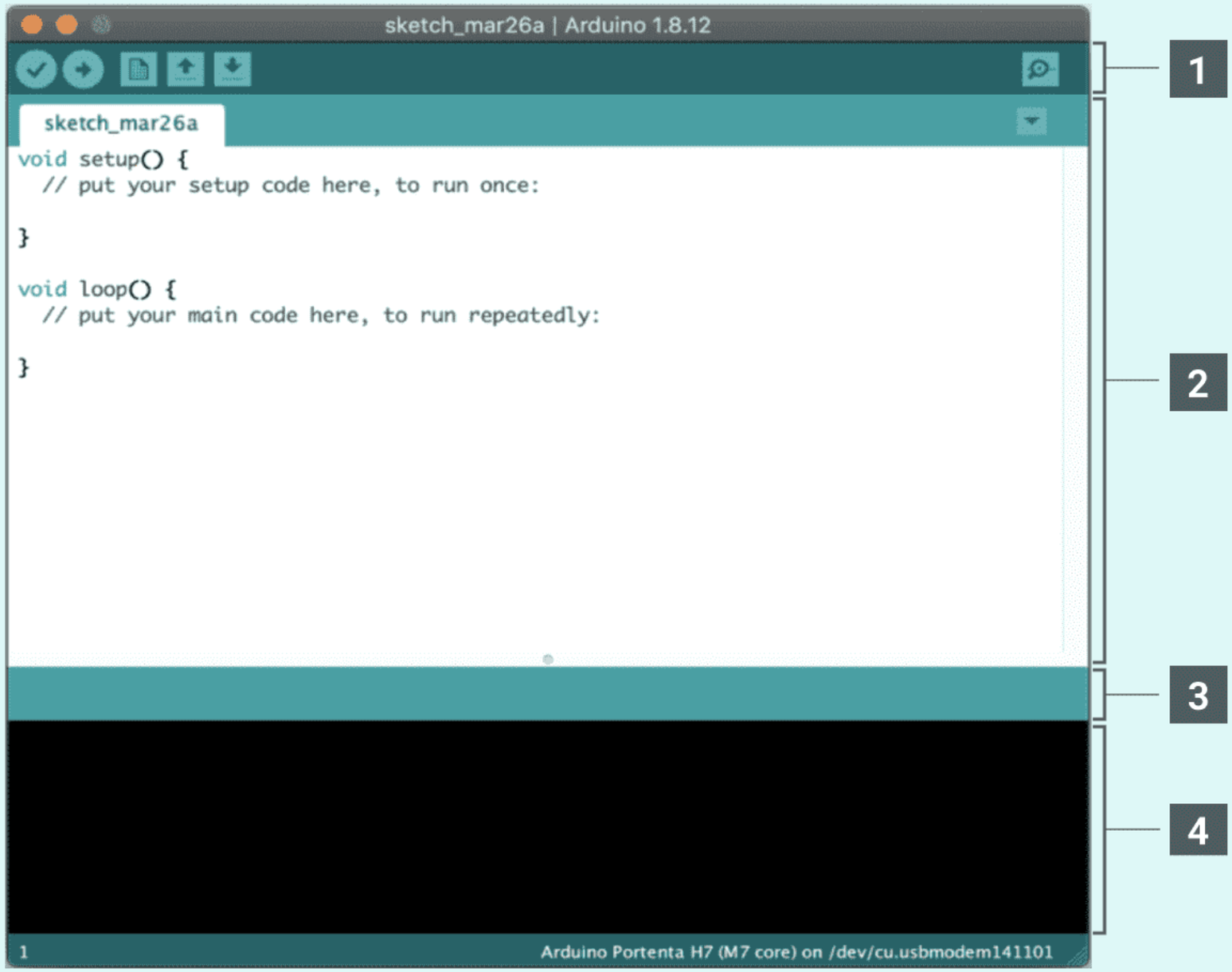
THE MAJORITY OF MICROCONTROLLER PLATFORMS SIMPLIFY THE PROCESS OF WORKING WITH MICROCONTROLLERS, ALTHOUGH ARDUINO OFFERS SOME ADVANTAGE: INEXPENSIVE HARDWARE, CROSS-PLATFORM IDE, OPEN SOURCE EXTENSIBLE SOFTWARE.

ARDUINO TOOLS

THE ARDUINO SOFTWARE (IDE) MAKES IT EASY TO WRITE CODE AND UPLOAD IT TO YOUR ARDUINO HARDWARE.

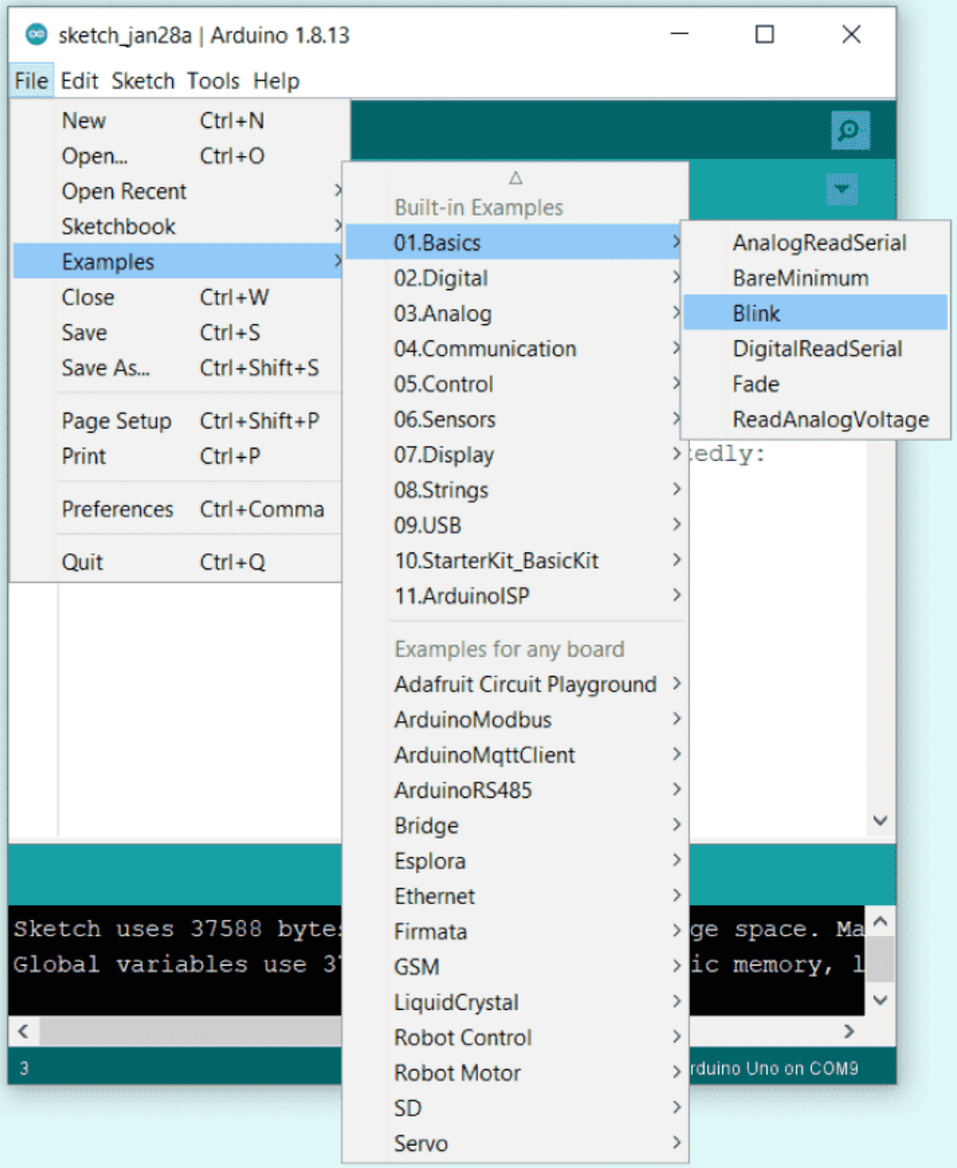
ALONG WITH THE CONVENIENT CODE EDITING FUNCTIONS, THE ARDUINO SOFTWARE (IDE) IS EQUIPPED WITH A LIST OF LIBRARIES THAT PROVIDE EXTRA FUNCTIONALITY FOR USE IN SKETCHES, MAKING IT EASIER FOR YOU TO CONNECT SENSORS, DISPLAYS, MODULES, ETC.

ARDUINO TOOLS: IDE



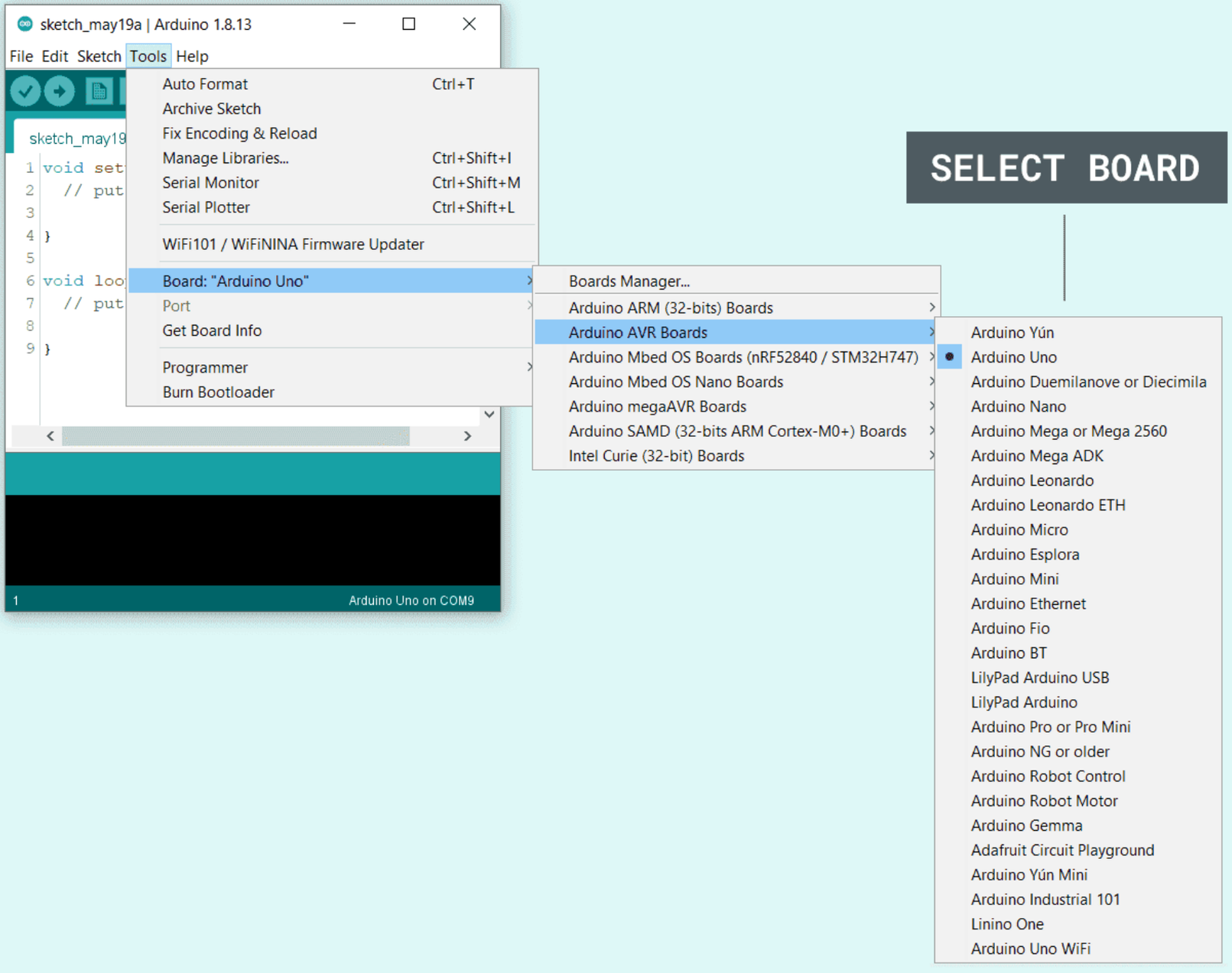
[1] TOOLBAR [2] CODE EDITOR [3] MESSAGE AREA [4] TEXT CONSOLE

ARDUINO TOOLS: IDE



LOAD A SKETCH

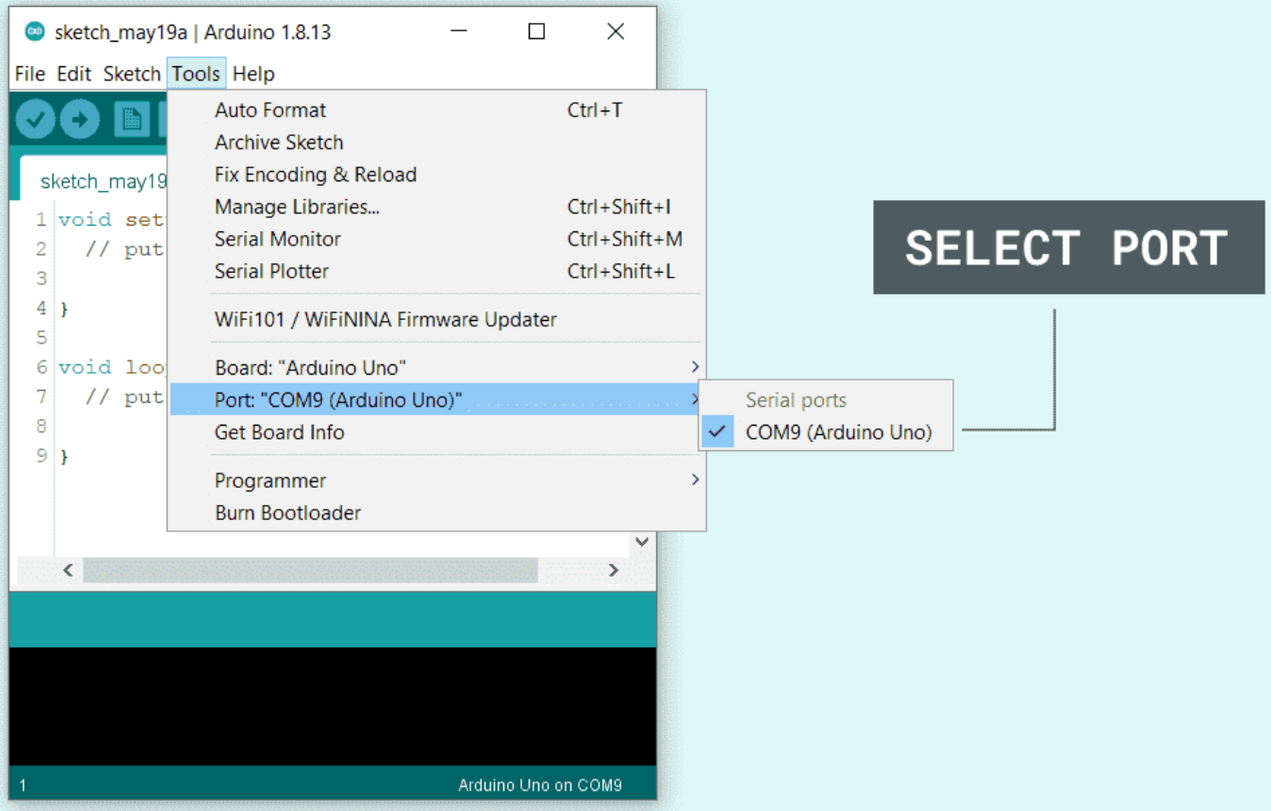
ARDUINO TOOLS: IDE



SELECT BOARD

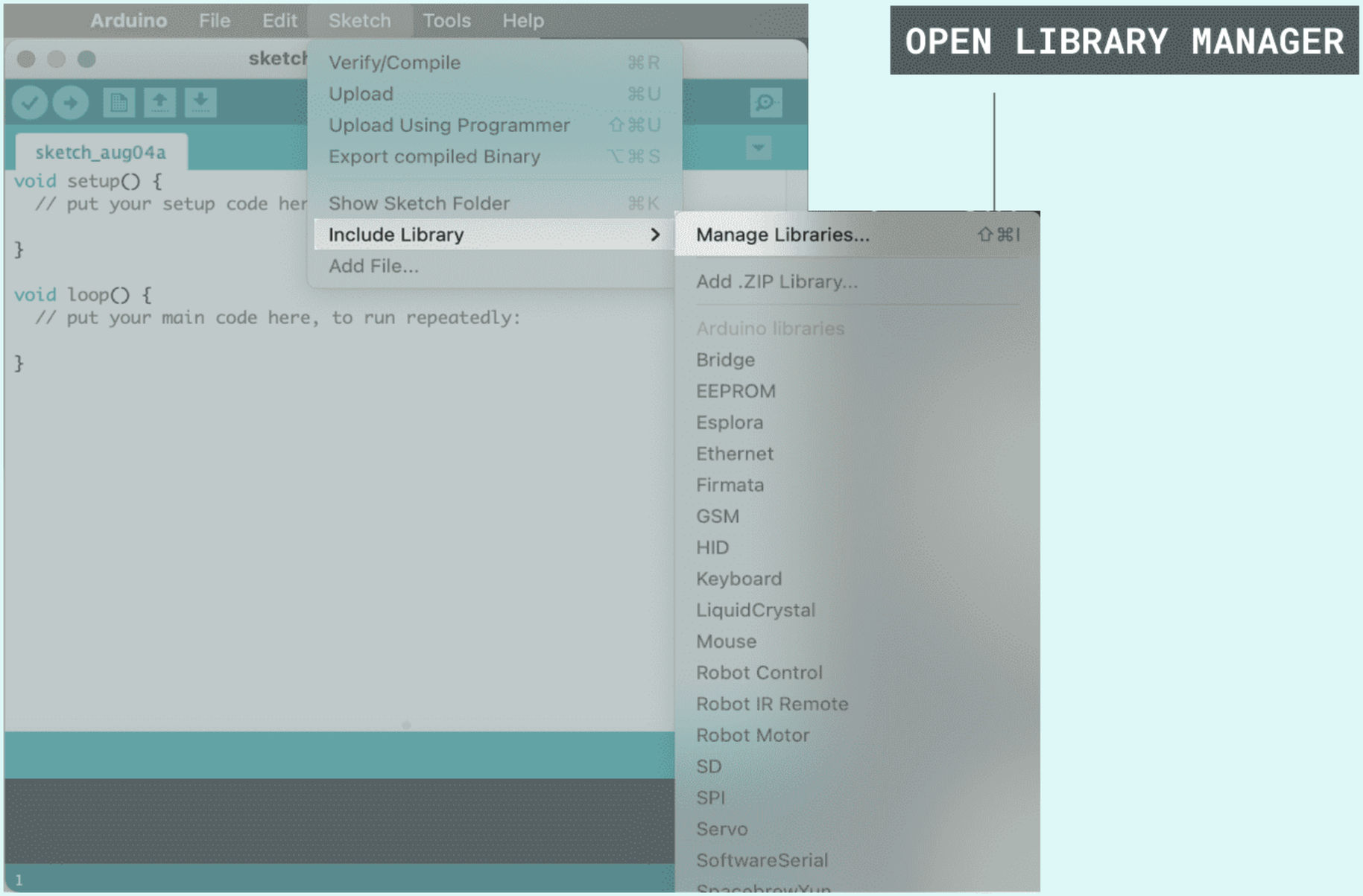
ARDUINO BOARDS & PLATFORMS

ARDUINO TOOLS: IDE



ARDUINO COMMUNICATION PORT

ARDUINO TOOLS: IDE

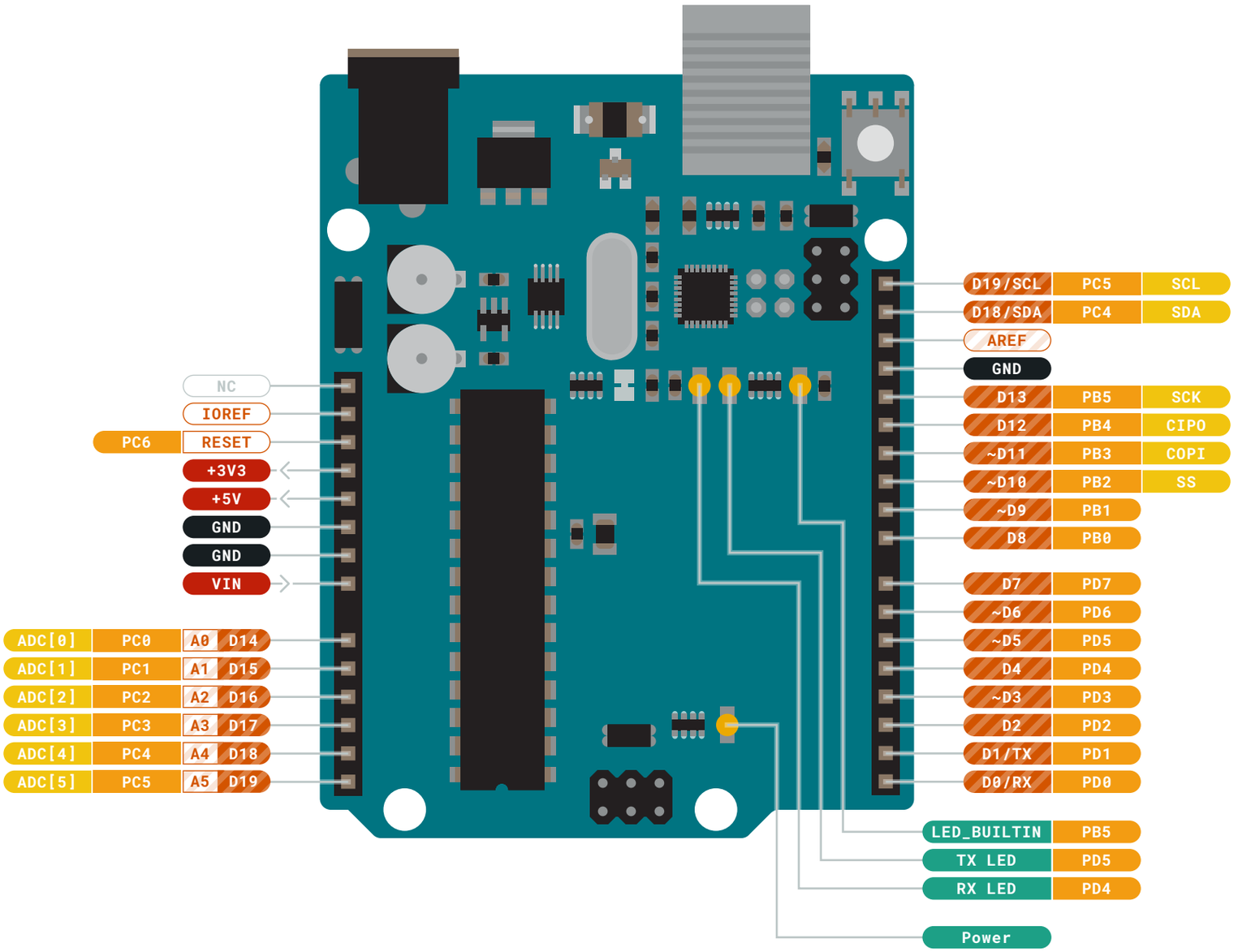


ARDUINO ENVIRONMENT CAN BE EXTENDED WITH THE USE OF LIBRARIES.

ARDUINO TOOLS

YOU CAN WRITE PROGRAMS AND UPLOAD THEM TO YOUR BOARD WITH THE BROWSER IDE (ARDUINO WEB EDITOR), THE DESKTOP IDE (ARDUINO SOFTWARE IDE), VIA THE CLI OR USE THE ARDUINO IOT CLOUD WEB PLATFORM.

ARDUINO MICROCONTROLLER



- Ground
- Power
- LED
- Internal Pin
- SWD Pin
- Digital Pin
- Analog Pin
- Other Pin
- Microcontroller's Port
- Default

⚠ **MAXIMUM** current per I/O pin is 20mA

⚠ **MAXIMUM** current per +3.3V pin is 50mA

VIN 6-20 V input to the board.

NOTE: CIPO/COPI have previously been referred to as MISO/MOSI

ARDUINO.CC

Last update: 17/06/2020

ARDUINO MICROCONTROLLER

THERE ARE THREE POOLS OF MEMORY IN THE MICROCONTROLLER USED ON THE ARDUINO BOARD:

32KB FLASH MEMORY (PROGRAM SPACE)
2KB SRAM (OPERATING MEMORY)
1KB EEPROM (LONG TERM STORAGE)

FLASH MEMORY AND EEPROM MEMORY ARE NON-VOLATILE (PERSISTANT STORAGE). SRAM IS VOLATILE AND WILL BE LOST WHEN THE POWER IS CYCLED.

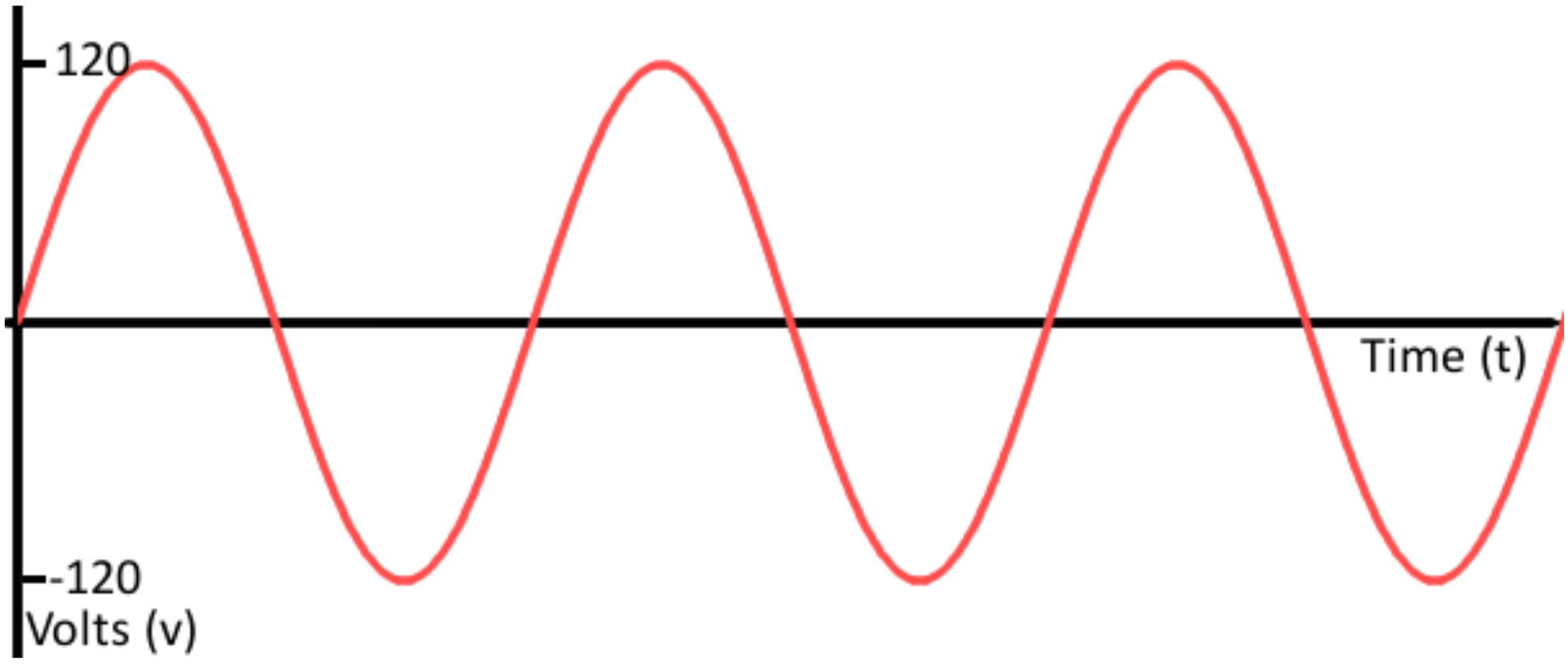
ARDUINO I/O

WE LIVE IN AN ANALOG WORLD IN WHICH ANALOG SIGNALS HAVE INFINITE POSSIBILITIES, WHEREAS DIGITAL SIGNALS AND SENSORS OPERATE IN THE REALM OF THE DISCRETE SETS OF SIGNALS.

WORKING WITH ELECTRONICS MEANS DEALING WITH BOTH ANALOG AND DIGITAL SIGNALS BOTH INPUT AND OUTPUT.

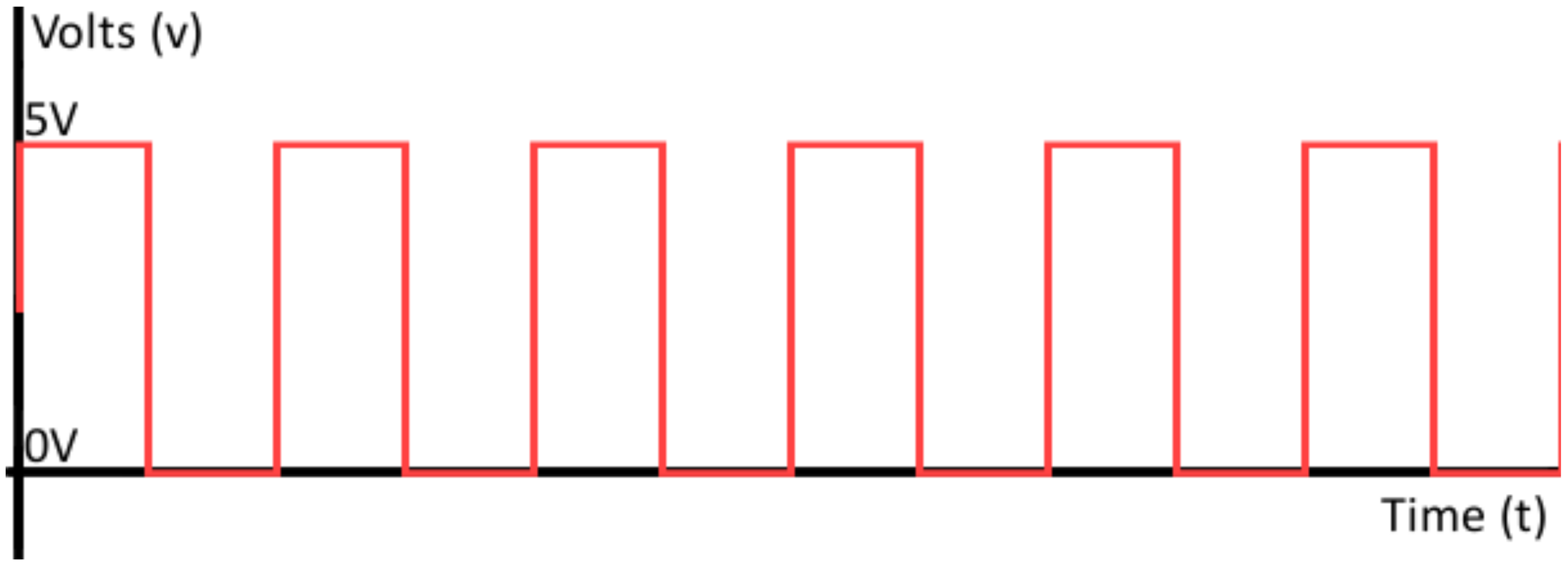
ARDUINO I/O: ANALOG SIGNALS

A TIME-VERSUS-VOLTAGE GRAPH OF AN ANALOG SIGNAL SHOULD BE SMOOTH AND CONTINUOUS.



ARDUINO I/O: DIGITAL SIGNALS

DIGITAL SIGNALS WILL BE ONE OF TWO VALUES – EITHER 0V OR 5V. TIMING GRAPHS OF THESE SIGNALS ARE STEPPING, SQUARE, AND DISCRETE.



ARDUINO I/O: DIGITAL INPUT

BY DEFAULT, ARDUINO I/O PINS ACT AS DIGITAL INPUTS, SO THEY DON'T NEED TO BE EXPLICITLY DECLARED AS DIGITAL INPUTS WITH PINMODE().

```
#define BUTTON_PIN 2; //DIGITAL INPUT

 void setup() {
    pinMode(BUTTON_PIN, INPUT);
    // pinMode(BUTTON_PIN, INPUT_PULLUP);
 }

void loop() {
    int val = digitalRead(BUTTON_PIN);
}
```

ARDUINO I/O: DIGITAL OUTPUT

DIGITAL PINS CONFIGURED AS OUTPUT WITH PINMODE() CAN PROVIDE A SUBSTANTIAL AMOUNT OF CURRENT TO OTHER CIRCUITS.

ARDUINO PINS CAN SOURCE OR SINK UP TO 40 MA (MILLIAMPS) OF CURRENT TO OTHER DEVICES. THIS IS ENOUGH CURRENT TO LIGHT UP AN LED OR ENABLE MANY SENSORS. BUT NOT ENOUGH CURRENT TO RUN MOST RELAYS, SOLENOIDS OR MOTORS.

ARDUINO I/O: DIGITAL OUTPUT

DIGITAL PINS CONFIGURED AS OUTPUT WITH PINMODE() CAN PROVIDE A SUBSTANTIAL AMOUNT OF CURRENT TO OTHER CIRCUITS.

```
#define MOTOR_1 2; //DIGITAL OUT

 void setup() {
    pinMode(MOTOR_1, OUTPUT);
 }

void loop() {
    digitalWrite(MOTOR_1);
}
```

ARDUINO I/O: ANALOG INPUT

ARDUINO HAS AN ONBOARD 6 CHANNEL ANALOG-TO-DIGITAL (A/D) CONVERTER. THE CONVERTER HAS 10 BIT RESOLUTION, RETURNING INTEGERS FROM 0 TO 1023.

THE MAIN FUNCTION OF THE ANALOG PINS IS TO READ ANALOG SENSORS, ALTHOUGH THE ANALOG PINS ALSO HAVE ALL THE FUNCTIONALITY OF GENERAL PURPOSE INPUT/OUTPUT PINS (THE SAME AS DIGITAL PINS 0 – 13).

```
void loop() {
   int val = analogRead(A0);
}
```

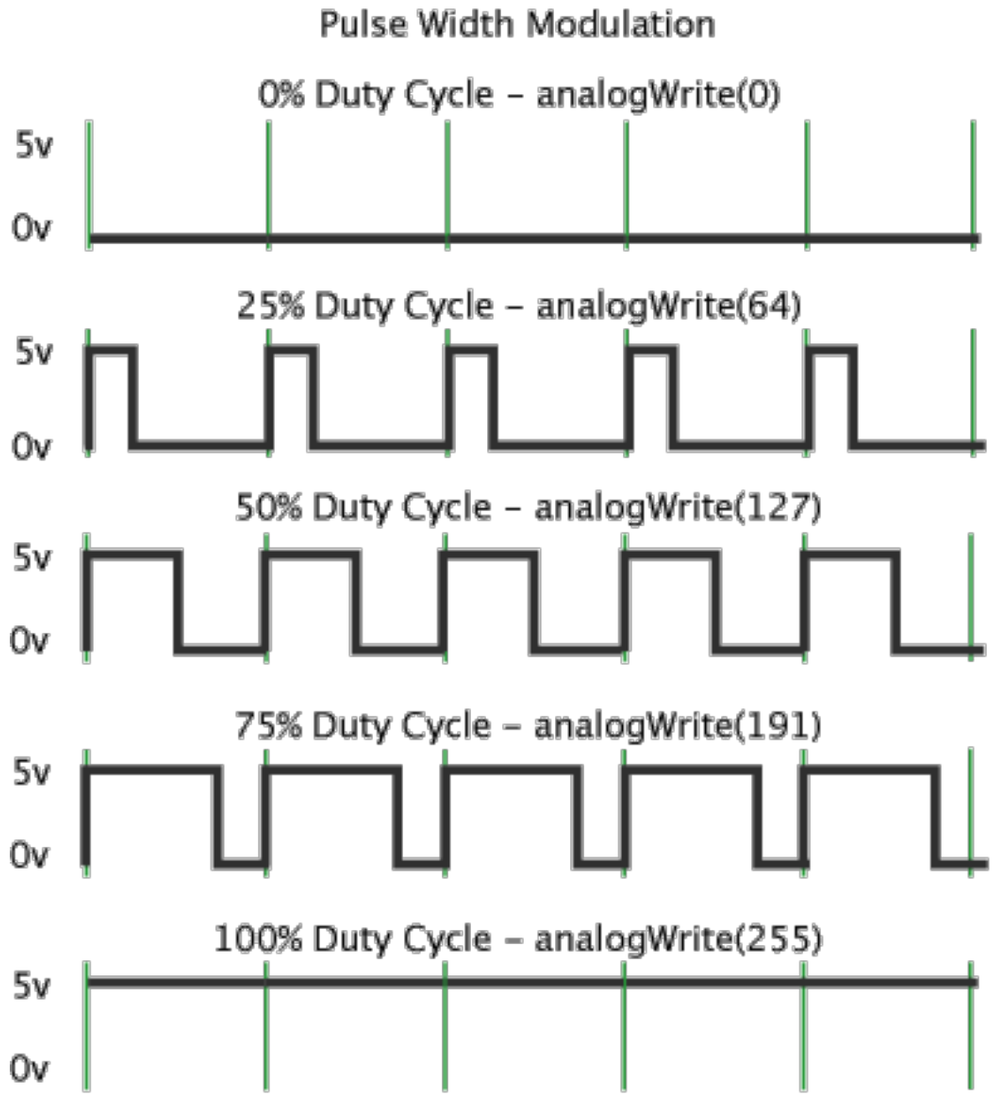
ARDUINO I/O: PWM (ANALOG OUTPUT)

PULSE WIDTH MODULATION, OR PWM, IS A TECHNIQUE FOR GETTING ANALOG RESULTS WITH DIGITAL MEANS.

THE ARDUINO IS USED TO CREATE A SQUARE WAVE THAT CAN SIMULATE VOLTAGES IN-BETWEEN 5 V & 0 V. BY CHANGING THE PORTION OF THE TIME THE SIGNAL SPENDS ON VERSUS THE TIME THAT THE SIGNAL SPENDS OFF. THE DURATION OF "ON TIME" IS CALLED THE PULSE WIDTH.

ARDUINO I/O: PWM (ANALOG OUTPUT)

ARDUINO'S PWM FREQUENCY IS ABOUT 500HZ, EACH PULSE OCCURS ON A 2 MILLISECOND INTERVAL.



ARDUINO I/O: PWM (ANALOG OUTPUT)

PULSE WIDTH MODULATION, OR PWM, IS A TECHNIQUE FOR GETTING ANALOG RESULTS WITH DIGITAL MEANS.

```
// analogRead() and PWM Output
#define RELAY 3 // RELAY PWM PIN

 void setup() {
    // ONLY ANALOG I/O
  }

void loop() {
    int val = analogRead(A0);
    analogWrite(RELAY, val / 4);
    delay(5);
}
```

ARDUINO LANGUAGE

THE CORE FOUNDATION OF THE ARDUINO PROGRAMMING LANGUAGE IS C/C++ (AVR-C). ALTHOUGH, FOR MAJORITY OF THE TIME YOU WILL BE PROGRAMMING PROCEDURALLY USING ARDUINO-C – A SUBSET OF C++ AND EMPLOYING EXTERNAL LIBRARIES TO DO THE HEAVY LIFTING.

ARDUINO LANGUAGE

EXTERNAL LIBRARIES ARE WRITTEN USING THE C++ OBJECT ORIENTATED PARADIGM. OVER TIME, YOU WILL NEED TO UNDERSTAND HOW TO INSTANTIATE OBJECTS FROM THESE LIBRARIES AND INVOKE THEIR METHODS.

YOU WILL PROBABLY NOT HAVE TO CREATE YOUR OWN CLASS DEFINITIONS FROM SCRATCH – RATHER WE WILL FOCUS ON PROGRAM FLOW, FUNCTIONS AND OPTIMISATION STRATEGIES.

ARDUINO LANGUAGE

EVERY PROGRAMMING LANGUAGE, LIKE ARDUINO-C, HAS A DEFINED GRAMMAR AND STYLE WHICH CONSISTS OF VARIABLES, CONDITIONALS, FUNCTIONS, OPERATORS, CONSTRUCTORS, DATA STRUCTURES AND ADVANCED OPERATORS.

ARDUINO LANGUAGE: SKETCH

THESE FUNCTIONS MUST EXIST IN YOUR SKETCH IN ORDER FOR THE PROGRAM TO COMPILE CORRECTLY.

VOID SETUP() { /*PUT YOUR SETUP CODE HERE */ }

THIS FUNCTION IS EXECUTED ONCE (WHEN PROGRAM STARTS)

VOID LOOP() { /*MAIN CODE HERE */ }

THIS FUNCTION IS EXECUTED REPEATEDLY UNTIL YOU RESET OR CUT POWER.

ARDUINO LANGUAGE: VARIABLES

A VARIABLE IS A PLACE TO STORE A PIECE OF DATA. IT HAS A NAME, A VALUE, AND A TYPE. ARDUINO-C IS A STRONGLY TYPED LANGUAGE. VARIABLES HAVE SCOPE – DETERMINED BY WHERE YOU DECLARE IT AND WHERE THE VARIABLE CAN BE USED.

BOOLEAN	1 BYTE	TRUE/FALSE
CHAR	1 BYTE	-127 TO 127 (ASCII)
UNSIGNED CHAR	1 BYTE	0 TO 255 (ASCII)
BYTE	1 BYTE	0 TO 255
INT/SHORT	2 BYTES	-32768 TO 32767
UNSIGNED INT	2 BYTES	0 TO 65535
LONG	4 BYTES	-2^{*31} TO $2^{*31}-1$
UNSIGNED LONG	4 BYTES	0 TO 2^{*32} -
FLOAT	4 BYTES	6-7 DECIMALS

ARDUINO LANGUAGE: VARIABLE SCOPE

```
int valA;
int valB = 6; // GLOBALLY SCOPED

void setup(){
   int valA = 4; // LOCALLY SCOPED
}

void loop(){
    int res = sumFunction(valA,valB);
}

int sumFunction (int a, int b) {
   int valA = a + b;
   return valA;
}
```

ARDUINO LANGUAGE: CONSTANTS

CONSTANTS ARE PREDEFINED EXPRESSIONS IN THE ARDUINO LANGUAGE. THEY ARE USED TO MAKE THE PROGRAMS EASIER TO READ.

HIGH | LOW | INPUT | INPUT_PULLUP | OUTPUT | LED_BUILTIN

ARDUINO LANGUAGE: CONST

THE CONST KEYWORD STANDS FOR CONSTANT. IT IS A VARIABLE QUALIFIER THAT MODIFIES THE BEHAVIOUR OF THE VARIABLE, MAKING A VARIABLE "READ-ONLY".

THIS MEANS THAT THE VARIABLE CAN BE USED JUST AS ANY OTHER VARIABLE OF ITS TYPE, BUT ITS VALUE CANNOT BE CHANGED.

```
const <datatype> <NAME> = <value>;
```

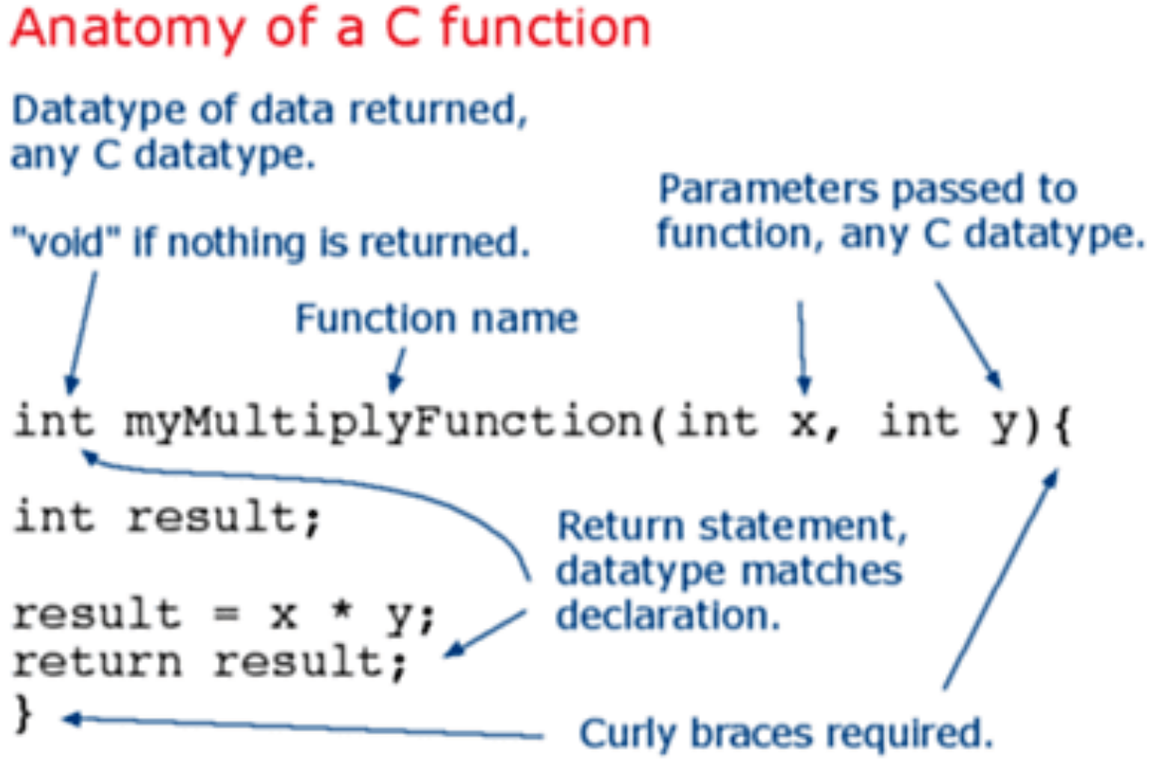
ARDUINO LANGUAGE: OPERATORS

ARDUINO SUPPORTS UNARY AND BINARY OPERATIONS.

Arithmetic:	+ , - , * , / , %
Compound:	+= ,-= ,*= , /=
Inc & Dec:	++ , - -
Comparison:	== , != , < , > , ≥ ,≤
Logical:	! , && , ||
Bitwise:	<< , >> , | , &,^;

ARDUINO LANGUAGE: FUNCTIONS

FUNCTIONS ORGANISE YOUR SKETCH, MAKE IT MORE MODULAR & COMPACT AS SECTIONS OF CODE ARE REUSED, AND AS A NICE SIDE EFFECT, USING FUNCTIONS ALSO MAKES THE CODE MORE READABLE.



ARDUINO LANGUAGE: CONDITIONALS

```
if ( boolean expr is true )
{
   /*perform instructions in this clause*/

}
else if ( another boolean expr is true )
{
   /* perform instructions in this clause*/
}

else
{
   /*previous statements were false*/
}
```

ARDUINO LANGUAGE: SWITCH

1: TESTVAL IS THE VARIABLE THAT WE TEST WITH
2: EXECUTE CORRESPONDING SWITCH STATEMENT
3: "BREAK": JUMP OUT OF THE SWITCH CONSTRUCT
4: "DEFAULT": IF NO OTHER CASE IS MATCHED

```
switch ( testVal )
{
   case 1:   { // do this; ... break; }
   case 2:   { // do this; ... break; }
   case n:   { //do this; ... break; }
   default:   { // no other cases were true; }
}
```

ARDUINO LANGUAGE: FOR …

1: COUNTER START
2: TERMINATING CONDITION
3: HOW MUCH TO CHANGE COUNTER EACH TIME

```
for ( int i = 0; i < max ; i++ )
{
     /* do whatever repeatedly until
    counter(here: var named i) reaches
    max. */
}
```

ARDUINO LANGUAGE: WHILE …

1: COME UP WITH BOOLEAN EXPRESSION
2: STATEMENTS INSIDE WHILE CLAUSE WILL EXECUTE UNTIL BOOLEAN EXPRESSION IS FALSE
3: MAY NOT EVEN EXECUTE ONCE
4: BEWARE OF INFINITE LOOPS

```
while ( boolean expression is true )
{
  /* do whatever repeatedly until boolean
   expression is false */
}
```

ARDUINO LANGUAGE: DO … WHILE

1: COME UP WITH BOOLEAN EXPRESSION
2: STATEMENTS INSIDE WHILE CLAUSE WILL EXECUTE UNTIL BOOLEAN EXPRESSION IS FALSE
3: WILL EXECUTE AT LEAST ONCE
4: BEWARE OF INFINITE LOOPS

```
do
{
  /* do whatever repeatedly until boolean
  expression is false */
}
while ( boolean expression is true );
```

ARDUINO LANGUAGE: ARRAYS

A COLLECTION OF VARIABLES (SAME DATATYPE) THAT ARE ACCESSED WITH AN INDEX NUMBER.

```
int numbers[6];
int myPins[] = {2, 4, 8, 3, 6};
int mySensVals[5] = {2, 4, -8, 3, 2};
char message[6] = “hello”;
```

ARDUINO LANGUAGE: ARRAYS

THE FIRST ELEMENT OF THE ARRAY IS AT INDEX 0, AND THE LAST ELEMENT IS ACCESSED BY ARRAYSIZE-1:

```
int testArray[ 10 ] = { 9,3,2,4,3,2,7,8,9,11 };
// testArray [ 0 ] contains 9
// testArray [ 9 ]  contains 11
// testArray [ 10 ]
```

ASSIGN AND RETRIEVE:

```
testArray [ 0 ] = 10;
int test = testArray [ 3 ];
```

ARDUINO LANGUAGE: ARRAYS

TO DETERMINE ARRAY SIZE, IT DEPENDS ON THE DATATYPES OF ELEMENTS STORED IN THE ARRAY:

```
int testInts [ ] = { 1,2,3,4,5,6 };
int arraySize = sizeof(testInts )/sizeof(int);
arraySize == 6
```

ITERATE THROUGH AN ARRAY:

```
for (int i = 0; i<sizeof(testInts)/sizeof(int); i++)
{
   // do something with testInt[i] ...;
}
```

REFERNECE

ARDUINO LANGUAGE REF
BUILTIN EXAMPLES