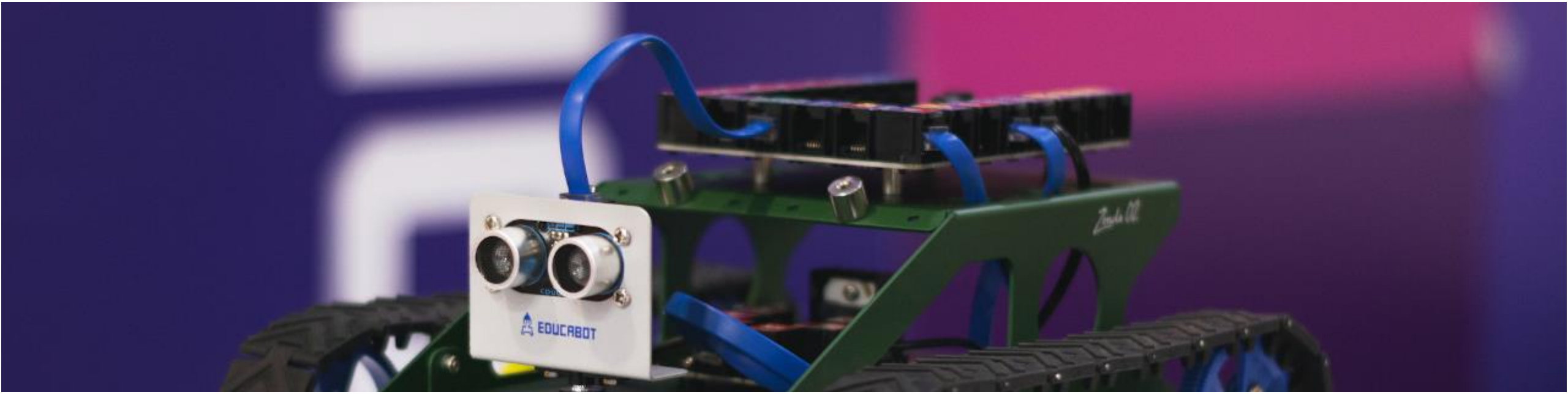


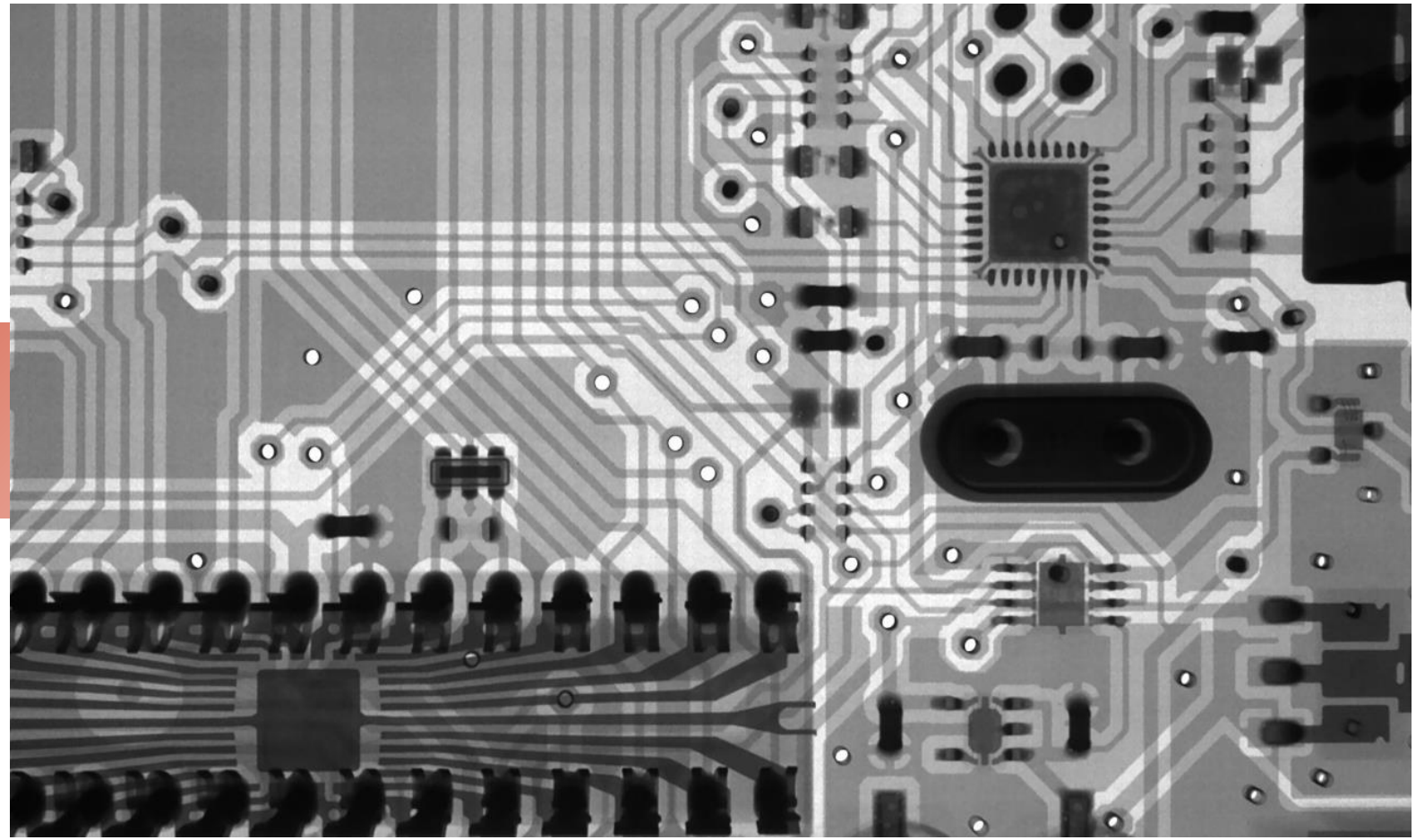
Arduino SYS-STEM for Schools

Training Methodology

SYS-STEM – Arduino SYS-STEM for Schools
Erasmus + Key Action 2 Strategic partnership - 2019-1-ES01-KA201-064454

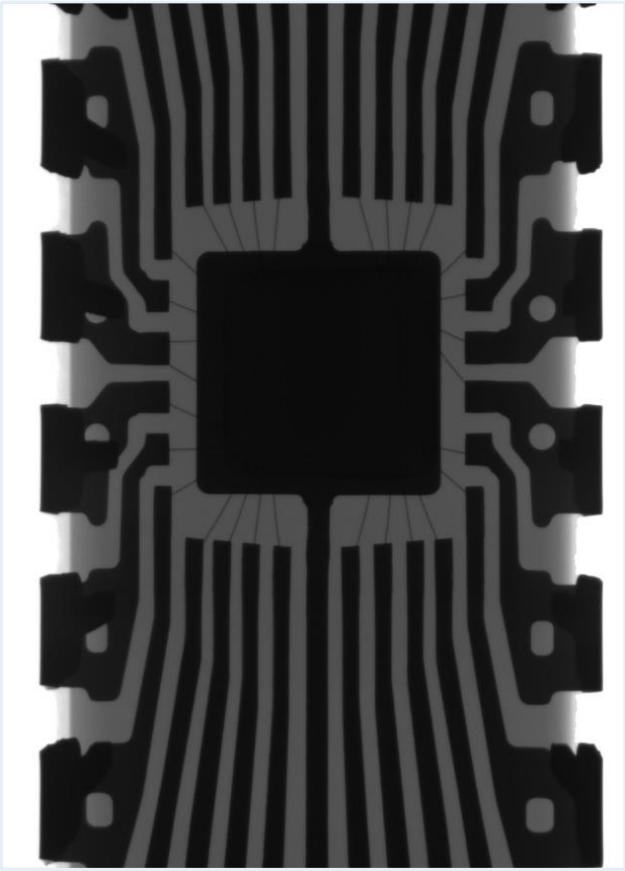
MODULE 4

Variables and Expressions



TRAINING MODULE CONTENTS

- Objectives
- Learning Outcomes
- Unit 1
    a. Variables and Constants
    b. Variable types
    c. Variable Scope and qualifiers
    d. Constants
- Unit 2
    a. Expressions
    b. Operations and operators
- Additional Reading Materials
- Exercises



OBJECTIVE

Module 4 will present the use of variables and constants in Arduino, their scope of use and variable qualifiers. Mathematical expressions, operators and their use in programming are also presented.

This module also helps to understand memory saving strategies, namely with the use of constants and static values.

EXPECTED LEARNING OUTCOMES

Knowledge

Upon completion of this module, the student will be able to:

- Understand how variables work and how they are handled in a Arduino sketch and in memory
- Know the the difference between variables and constants
- Understand variable scope and variable qualifiers

Competences and Skills

Upon completion of this module, the student will be able to:

- Correctly define variables and constants in a Arduino sketch
- Adopt memory saving strategies

UNIT 1.a.

Variables

Variables

- A variable is a way of naming and storing a value for later use by the program, such as
  - data from a sensor
  - intermediate value used in a calculation
- Before they are used, all variables have to be declared
- Declaring a variable means defining its type
- Optionally, you can set an initial value when declaring (initializing the variable)
- Uninitialized variables have a random values
- When variable contents exceeds their maximum capacity they “roll over” back to their minimum capacity (and viceversa)

Variable Types

- C/C++ defines several data types (more information https://www.arduino.cc/reference/en/#data-types)

Type	Description
bool	Can be **true** or **false**, but occupies 1 byte (**boolean** is an Arduino alias**)**
char	1 byte, from -**128** to **127** (use only for storing characters)
byte	1 byte unsigned number, from **0** to **255**
short	2 bytes, from **-32,768** to **32,767**
int	2 bytes or 4 bytes, depending on CPU architecture from **-32,768** to **32,767** (16bit) or **-2,147,483,648** to **2,147,483,647** (32bit)
unsigned int *or* **word**	from **0** to **65,535** (16bit) or **0** to **4,294,967,295** (32bit)
long	4 bytes from **-2,147,483,648** to **2,147,483,647**
float/double	4 bytes (8 bytes on Arduino DUE), from **-3.4028235E+38** to **3.4028235E+38**

Arrays

- Arrays are variables collections, that are accessed using an index number
- Arrays have a fixed dimension
  - Can be defined between brackets e.g. `[10]`
  - Can be inferred by the quantity of elements initialized e.g. `[]`

Declaring Arrays

- Arrays can be created without initialization

```
int myInts[10];
Double myReal[22];
```

- Or can be initialized with values between { }
  - Dimension obtained by quantity of initialization elements

```
int myPins[] = {2, 4, 8, 3, 6};
```

  - Dimension specified

```
int mySensVals[6] = {2, 4, -8, 3, 2};
```

String arrays

- Strings are arrays of characters

```
char str[255];
```

- Strings need a end character `'\0'`, so the can only can be `n-1` characters long
- Strings can be initialized
  - Using a string between `""`

```
char message[6]="Hello";
```

  - Using a list of characters (needs to be null terminated with `'\0'`)

```
char message[6]={'H','e','l','l','o','\0'};
```

Accessing an array

- The first index of an array is always **0**
- The last index of and array is **`size-1`**
- To assign a value to an array:

```
myInts[0] = 10;
```

- To retrieve a value from an array:

```
x = myInts[4];
```

- A variable can be used as an array index

```
x = myInts[i];
```

Arduino Memory

- There are three pools of memory in the microcontrollers used on Arduino boards (e.g. the Uno ATmega328P)

  Flash memory (or program space) — where the Arduino sketch is stored

  SRAM (Static Random Access Memory) — where the sketch creates and manipulates runtime variables

  EEPROM — memory space that programmers can use to store long-term information

- **Flash memory** and **EEPROM** memory are non-volatile (the information persists after the power is turned off)
- **SRAM** is volatile and is lost when the power is cycled

Arduino Memory

- Different microcontrollers have different memory sizes

	ATMega168	ATMega328P	ATmega1280	ATmega2560
Flash (1 kB used by bootloader)	16 kB	32 kB	128 kB	256 kB
SRAM	1 kB	2 kB	8 kB	8 kB
EEPROM	512 bytes	1 kB	4 kB	4 kB

- The ATMega328P is the microcontroller used by the Arduino Uno R3

Storing Variables in Flash Memory

- For large projects **2KB** is not a lot of memory
- Large unchanging structures or strings can be stored in flash memory
- **`PROGMEM`** is the variable modifier to store variables in flash (use before the type or after the name)

```
const PROGMEM uint16_t years[] = {1950, 1960, 1970, 1980, 1990, 2000, 2010};

const char string[] PROGMEM = {"Create a string and store it in flash memory"};
```

- Version 1.0 of the Arduino IDE introduced the **`F()`** macro to store static strings in flash memory instead of RAM

```
Serial.println(F("String to be stored in flash memory"));
```

Retrieving Variables from Flash Memory

- **`PROGMEM`** is part of the **`<avr/pgmspace.h>`** library and is automatically included by current Arduino IDE
- To access variables we need to use special functions included in the **`pgmspace`** library

  `pgm_read_byte` (char), **`pgm_read_word`** (int), **`pgm_read_float`** (float)

- Example to read a year in position **`i`** from previous slide **`years[]`** array (**`&`** is needed to access variable memory location)

```
readValue = pgm_read_word(&years[i]);
```

- **`pgmspace`** library reference http://www.nongnu.org/avr-libc/user-manual/group__avr__pgmspace.html

UNIT 1.b.

Variable Scope and Qualifiers

Variable Scope

- All variables in Arduino programing language (C++) have a property called scope
- Depending on where in the program you declare Variables, they may be “global” or “local”
  - **Global variables** are defined outside the functions, and can be used everywhere in a program
  - **Local variables** are declared inside functions and can only be used inside that function
- Variables can also be defined inside blocks (e.g. Inside a **`for`** loop)

```
for (int i = 0; i < 10; i++) {
    // variable i can only be accessed inside the for-loop brackets
}
```

- Local variables have precedence over global variables with the same name

Variable qualifiers — static

- The **`static`** keyword is used to create variables that are visible to only one function
- **`static`** variables persist across multiple function calls (unlike local variables which are created and destroyed on each function call)
- **`static`** variables are created and initialized only the first time the function is called

```
int staticTest () {
    static int value=0;
    value+=10;
    return value;
}
```

First time the function is called, value is initialized to 0

In every function call the value is incremented 10 and returned, i.e., 10 … 20 … 30 … 40 … etc.

Variable qualifiers — volatile

- The **`volatile`** keyword is used to create variables that has to be read from memory every time, even if it is already on the processor registers
- Variables changed outside the normal program flow (for instance inside an **interrupt function**) need to be declared **`volatile`**
- On a 8-bit microcontroller **`volatile`** variables bigger than one byte cannot be read in one step, so they may render random values if an interrupt occurs between first and second bytes reading. To avoid this problem disable interrupts or use an **`ATOMIC_BLOCK`** (see: https://www.arduino.cc/reference/en/language/variables/variable-scope-qualifiers/volatile/)

```
volatile byte x=123;
```

UNIT 1.c.

Constants

CONSTANTS

- Constants are predefined expressions in the Arduino language
- They are used to make the programs easier to read
- We can classify the Arduino predefined constants in four groups:
  - Defining Logical Levels: **`true`** and **`false`** (Boolean Constants)
  - Defining Pin Levels: **`HIGH`** and **`LOW`**
  - Defining Digital Pins modes: **`INPUT`**, **`INPUT_PULLUP`**, and **`OUTPUT`**
  - Defining built-ins: **`LED_BUILTIN`**

DEFINING CONSTANTS (with #define)

- We can use the C++ **`#define`** construct to create our own constants

```
#define constantName Value
```

- **`ConstantName`** is the constant name (by convention should be all caps)
- **`Value`** is the constant value
- Never include the equal sign or a semicolon at the end
- Examples

```
#define GREEN_LED      10
#define NUMBER_LEDS    6
```

DEFINING CONSTANTS (const qualifier)

- We can use the **`const`** qualifier to declare constants

```
const type Name = Value;
```

- Works like a variable declaration

- **`Value`** is the constant value
- Examples

```
const double pi=3.1415926536;
const int numLinha = 10;
```

#define *vs* const

- `#define` is a preprocessing directive
  - All entries after the `#define` are replaced prior to compilation (can be annulled with a `#undef` directive)
  - Unlike variables does not use runtime memory
  - One error in the `#define` directive is presented as an error in the usage lines
- `const` qualifier
  - Creates a read only variable (in some circumstances, compiler optimization produce a inline substitution)
  - Must have a type and observe variable scopes (we can have local constants)

Integer Constants

- We can use integer values in different bases

Base	Example	Formatter	Comment
10	123	none	Decimal values
2	B1010101	leading B	Max length 1 byte (8bits); Valid characters are: 0 and 1
8	0177	leading 0	Valid characters are: 0-7
16	0xA1F0	leading 0x	Valid characters are: 0-9, A-F or a-f

Integer Constants

- An integer constant is treated as an `int` (with their inherent value limitations)
- Formatters can be used to specify an integer constant with another data type

Type	Example	Formatter
unsigned int	123u	U or u
long	100000L	L or l
unsigned long	12345678UL	UL or ul

UNIT 2.a

Expressions

Expressions

- In in the C++ programming language, a single equal sign = is called the assignment operator
- The arithmetic operators in C++ are:

Operator	Description
%	Remainder of the integer division
+	Addition
-	Subtraction
*	Multiplication
/	Division (integer division if both operands are integers)

Expressions

- Expressions are evaluated left to right (multiplication and division have priority over addition and subtraction)
- Multiple expressions can be separated with a comma (useful inside **`for`** initializations or conditions)

```
x=2+3,y=4*2-z;
```

Is equivalent to:

```
x=2+3;
y=4*2-z;
```

- Expressions can be used in function parameters

```
delay(mill*1000)
```

UNIT 2.b

Compound operators

Compound operators

- C++ has convenient shorthands to perform usual actions on variables

Operator	Description
+=	Compound Addition: `x+=y*2` is equivalent to `x=x+y*2`
-=	Compound Subtraction: `x-=y` is equivalent to `x=x-y`
*=	Compound Multiplication: `x*=2` is equivalent to `x=x*2`
/=	Compound Division: `x/=y` is equivalent to `x=x/y`
++	Increment: `x++` is equivalent to `x=x+1`
--	Decrement: `x--` is equivalent to `x=x-1`

- The `++/--` operator, can be used before or after the variable: `a[++x]` increments `x` before using it for index; `a[x++]` increments `x` after using it for index

EXTRA READING MATERIALS

Online Resources

- Arduino Reference on variables, constants, variable scope & qualifiers
  - https://www.arduino.cc/reference/en/#variables
- Arduino Reference on operators (arithmetic, comparison, boolean, compound, etc.)
  - https://www.arduino.cc/reference/en/#structure

EXERCISES / TESTS / QUIZZES

Exercises

1. Define the variables/constants needed to store the following values:

   `a` = the area of a circle

   c = the circumference of a circle

   `d` = the diameter of a circle

   π = 3.141592

2. Assign to **a** and **c** respectively:

   a) The area of the circle, defined by the formula πr^2 (**r** is the circle radius, calculate from d)

   b) The circumference of the circle, defined by the formula $2\pi r$ (**r** is the circle radius , calculate from d)

Exercises

3. Define the variable(s) to easily read/change the signal of 10 consecutive pins
4. We have defined a global and a local variable (on the `loop()` function) with the same name. Which variable is used on:
   a) The `setup()` function
   b) The `loop()` function
   c) Other functions

CONGRATULATIONS

You have completed SYS-STEM Module 4