Tecnologia da Informática

Licenciatura em Engenharia Informática

Universidade de Coimbra

2023/2024

Exercícios recomendados (da ficha de exercícios)

Exercício 1.9 - Monte um circuito com 8 LED (ligados nos pinos 13-6). Faça um programa que conte números utilizando uma variável inteira do tipo byte (de 1 em 1) e acenda os LED conforme o equivalente binário (LED 6=LSB -> LED 13=MSB). Se não tiver LEDs suficientes poderá usar o Tinkercad.

(Conselho: recorde o processo de conversão manual – maneiras mais eficientes serão abordadas nas próximas aulas, mas este método servirá por agora).

Exercícios recomendados (da ficha de exercícios)

Exercício 2.1 – Escreva um programa que implemente a potência de 2^n usando o operador de deslocamento binário. O programa deve iniciar quando for premido o botão de pressão e deve fazer as seguintes potências 2^0, 2^1, 2^2, 2^3, 2^4. Utilize a função `Serial.println`() para mostrar o resultado em decimal e em binário. Não precisa de fazer *debounce*.

Exercício 2.3 – Resolva o exercício 2.1 utilizando ciclos. Altere-o de forma a imprimir as potências de 2 inferiores a 100.

Exercícios recomendados (da ficha de exercícios)

Exercício 2.2 - Escreva um programa que permita verificar se um dado número é uma potência de dois usando os operadores binários. Use variáveis com o valor 5, 8 e 10 para validar o seu programa. Sugestão: utilize o AND binário entre o próprio número e o número menos uma unidade. Utilize a função `Serial.println`() para mostrar o resultado.

Exercício 2.11 - Como sabe, a leitura da porta série é feita sob a forma de carateres em código ASCII. Faça um programa que leia um dígito entre 0 e 9 e o converta para o respetivo valor numérico (para armazenar numa variável do tipo byte). Acrescente validações para que outros carateres não sejam aceites.

Conteúdo

- Funções

Funções

Sobre as funções

- As funções são mecanismos que permitem modularizar e reutilizar o código.
- Podem ser utilizadas para abstrair funcionalidade.

Recordando….

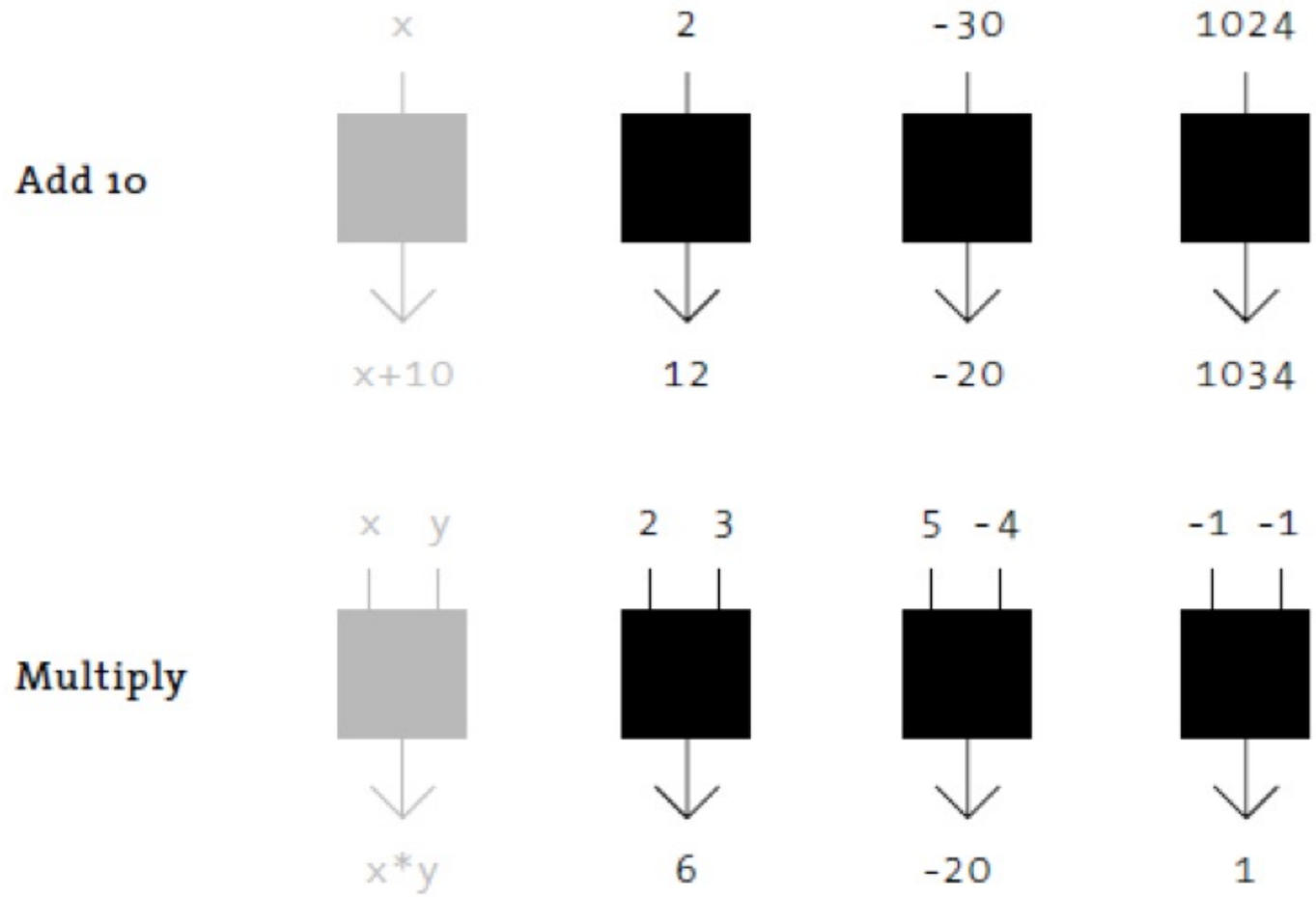
Funções: um conceito versátil.

Existem vários tipos de funções. Todas fazem alguma coisa, mas:

- Algumas aceitam parâmetros e não devolvem valores
  Exemplo: ***digitalWrite()***
- Algumas aceitam parâmetros e devolvem alguma coisa
  Exemplo: ***digitalRead()***
- Algumas não aceitam parâmetros nem devolvem nada,
  Exemplo: ***noTone()***

E podemos criar as nossas próprias funções….

Sobre as funções



Declaração de uma função

```
returnType   functionName (arguments ) {
        // Code body of function
}
```

Declaração do tipo de dados retornado ou “void”, se nada for devolvido

Nome da função

Argumentos passados para a função: nome e tipo de dados (pode estar vazio)

```
int minhaFuncaoMaravilha(int x, int y){
    int resultado;

    resultado=x*y+random(100);
    return resultado;
}
```

Variável local

Instrução de retorno: dados devolvidos compatíveis com tipo declarado na função

Uso

```
void setup(){
  Serial.begin(9600);
}

void loop() {
  int i = 2;
  int j = 3;
  int k;
  k = minhaFuncaoMaravilha(i, j);
  Serial.println(k);
  delay(500);
}

int minhaFuncaoMaravilha(int x, int y){
    int resultado;

    resultado=x*y+random(100);
    return resultado;
}
```

A invocação usa o resultado retornado (um *int*) numa atribuição

Como é que os argumentos funcionam ?

```
int minhaFuncaoMaravilha(int x, int y){
    int resultado;

    resultado=x*y+random(100);
    return resultado;
}
```

Quando a função é invocada, x e y contém os **valores** dos argumentos que foram passados

Dentro da função, os argumentos funcionam como se fossem variáveis

Há contudo uma limitação: se houver alguma alteração aos valores passados como argumento, essas alterações não serão propagadas para fora da função.

Uso

```
void setup(){
  Serial.begin(9600);
}

void loop() {
  int i = 2;
  int j = 3;
  int k;

  k = minhaFuncaoMaravilha(i, j);
  Serial.println(k);
  Serial.println(i);
  delay(500);
}

int minhaFuncaoMaravilha(int x, int y){
    int resultado;

    x=10;
    resultado=x*y+random(100);
    return resultado;
}
```

Esta alteração apenas existe dentro do contexto da função. Valor de variável usada na invocação (i) não é alterado.

E se a função fosse do tipo *void ?*

```
void setup(){
  Serial.begin(9600);
}

void loop() {
  int i = 2;
  int j = 3;
  minhaFuncaoMaravilha(i, j);
  delay(500);
}

void minhaFuncaoMaravilha(int x, int y){
    int resultado;

    resultado=x*y+random(100);
    Serial.println(resultado);
    return;
}
```

Invocação não devolve nada, logo não existe valor de retorno para utilizar

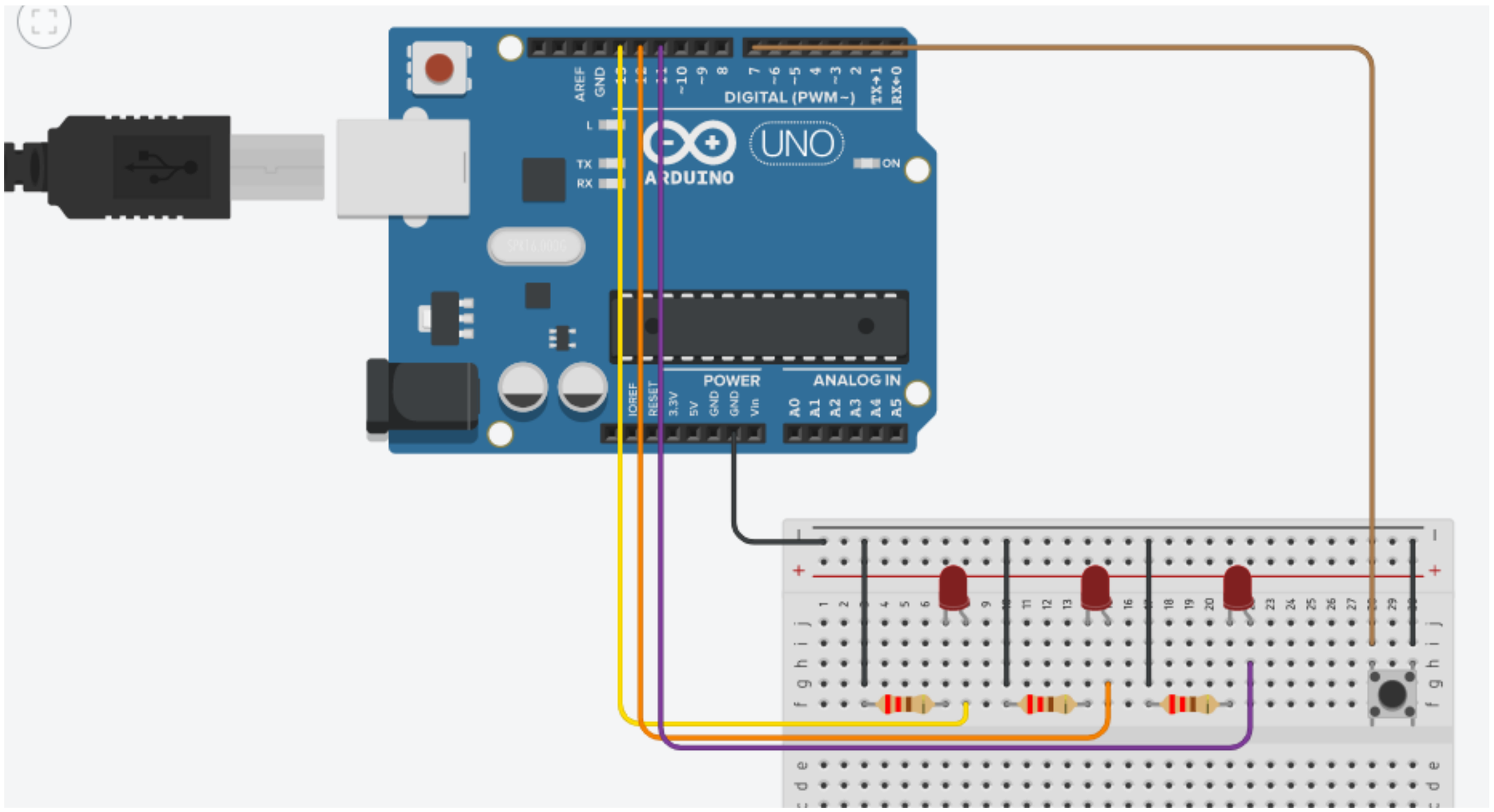
Return apenas assinala término da execução

Exercícios

- Escreva uma função *check_even* que, dado um valor passado como argumento devolva *true* ou *false*, conforme este seja par ou não.

- Resolva o exercício 2.6, incluindo uma função da sua autoria que se encarregue de acender os LEDs correspondentes ao código binário de um valor do tipo *byte* passado como argumento.

Diagrama 2.6


AREF
GND
~10
~9
8
7
~6
~5
4
~3
2
TX→1
RX←0
DIGITAL (PWM~)
L
TX
RX
ARDUINO
UNO
ON
POWER
ANALOG IN
IOREF
RESET
3.3V
5V
GND
GND
Vin
A0
A1
A2
A3
A4
A5

Solução 2.6, com funções

(com debouncing)

```
/* buttonReading armazena a leitura
do botaoprevStatus armazena o estado
anterior do botao buttonStatus contem
o estado assumido para o botao, depois
do deboucing */

boolean buttonReading=false;
boolean prevStatus=false;
boolean buttonStatus=false;
unsigned long debounceDelay = 20;
int count=0;
long ellapsed;

void setup()
{
  for (int res=11;res<=13;res++)
    pinMode(res,OUTPUT);
  pinMode(7, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop()
{
  buttonReading=!digitalRead(7);

  //Reset ao contador do debouncer
  if (buttonReading!=prevStatus)
     ellapsed=millis();
```

```
//Se passou um intervalo de tempo seguro
 if ((millis()-ellapsed)>debounceDelay) {
   //e o estado do botao realmente mudou
   //(sem que tenha havido resets)
   //entao assume o estado novo para o botao
   if (buttonStatus!=buttonReading) {
     buttonStatus=buttonReading;

     if (buttonStatus) count=(count+1)%8;
   }
   if (buttonStatus && (millis()-ellapsed>2000)) {
         count=0;
         ellapsed=millis();
   }
  }

  gen3LEDBin(count,11);
  prevStatus=buttonReading;
}

void gen3LEDBin(byte number, byte LEDoffset) {
 for (int digit=0;digit<3;digit++)
     digitalWrite(LEDoffset+digit,(number>>digit)&1);
}
```

Solução funcionalmente equivalente em: https://www.tinkercad.com/things/2KmxmUfdzLA

Trabalho para casa

1. Fazer os exercícios 2.4, 2.5, 2.9 e 2.10

2. Instalar o modulo *pySerial* no vosso ambiente python (anaconda?)

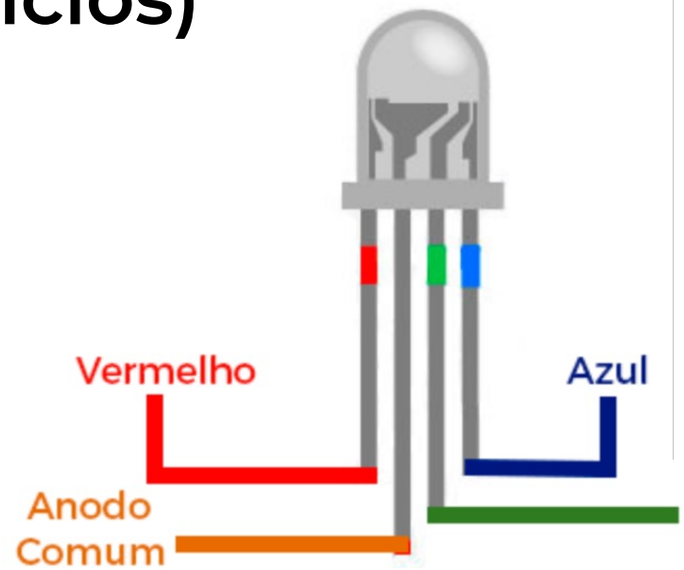
3. Fazer algum trabalho de pesquisa para averiguar como se usa. Tente responder às seguintes perguntas:

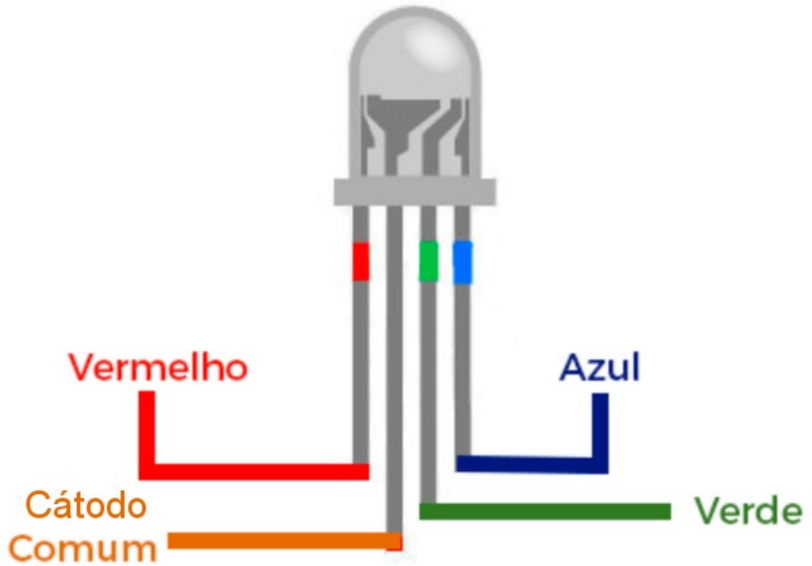
    1. Como se inicializa a comunicação com um porto série ?
    2. Que parâmetros são utilizados na comunicação – vê paralelos que com os que conhece do Arduino ?
    3. Como se podem ler carateres ou cadeiras de carateres do porto série ?

Exercícios recomendados (da ficha de exercícios)

Exercício 2.4

Exercício 2.5





NOTA: Para o exercício 2.5 assuma que o led RGB possui uma das anatomia ilustrada pela figura, sendo que todas as ligações aos leds devem ser protegidas (ou seja, precisam de colocar um resistência de 220 Ohm para todos os pinos, à exceção do ânodo ou cátodo comum). O ânodo será ligado aos +5V e no caso do cátodo comum, este será ligado ao GND

Se não tiver LEDs RGB, pode implementar com o tinkercad.

Exemplo de uso de LED RGB: https://www.tinkercad.com/things/f2LMFdpY9qG