Parâmetros

Dar parâmetro a uma função, seria a mesma coisa dizer o número de vezes (numFlashes) que o LED dos exemplo abaixo deve piscar e que sua duração deve ser a duração (d).

```
// sketch 04-02
int ledPin = 13;
int delayPeriod = 250;

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

void loop()
{
   flash(20, delayPeriod);
   delay(3000);
}

void flash(int numFlashes, int d)
{
   for (int i = 0; i < numFlashes; i ++)
   {
      digitalWrite(ledPin, HIGH);
      delay(d);
      digitalWrite(ledPin, LOW);
      delay(d);
   }
}
```

Lembre-se de declarar as variáveis na função:
Int numFlshes

numFlashes = 20
d = delayPeriod
ou que d = 250

Parâmetros, Variáveis globais, locais e estáticas

- Variáveis que não são parâmetro e que serão usadas apenas dentro da função são chamadas de variáveis locais.

```
// sketch 04-02
int ledPin = 13;
int delayPeriod = 250;

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

void loop()
{
   flash(20, delayPeriod);
   delay(3000);
}

void flash(int numFlashes, int d)
{
   for (int i = 0; i < numFlashes; i ++)
   {
      digitalWrite(ledPin, HIGH);
      delay(d);
      digitalWrite(ledPin, LOW);
      delay(d);
   }
}
```

Variável Global

Pode ser usada em todo o Sketch

Parâmetro

Aqui temos o parâmetro “numFlashes” e “d”. Ou seja, a função flash aceita 2 números inteiros como parâmetro.

Obs.:

- Variáveis globais vão contra o princípio do encapsulamento. A ideia do encapsulamento é que você deve criar um pacote, embrulhando em um único volume tudo o que tem a ver com alguma coisa em particular (Monk, 2013)

Parâmetros, Variáveis globais, locais e estáticas

```
// sketch 04-03
int ledPin = 13;
int delayPeriod = 250;
void setup()
{
   pinMode(ledPin, OUTPUT);
}

void loop()
{
  int count = 0;
  digitalWrite(ledPin, HIGH);
  delay(delayPeriod);
  digitalWrite(ledPin, LOW);
  delay(delayPeriod);
  count ++;
  if (count == 20)
  {
     count = 0;
     delay(3000);
  }
}
```

Variável Local

Ela só é válida dentro da função

Problema: A contagem aqui vai sempre ficar em 0.

Solução: Palavra chave "static"

```
// sketch 04-04
int ledPin = 13;
int delayPeriod = 250;
void setup()
{
   pinMode(ledPin, OUTPUT);
}

void loop()
{
  static int count = 0;
  digitalWrite(ledPin, HIGH);
  delay(delayPeriod);
  digitalWrite(ledPin, LOW);
  delay(delayPeriod);
  count ++;
  if (count == 20)
  {
     count = 0;
     delay(3000);
  }
}
```

"static" usada na frente de uma declaração de variável, tem o efeito de inicializar a variável apenas na primeira vez em que a função é executada.

Obs.:

- Poderia também como solução utilizar a variável count do problema inicial como global ao invés de local, teria solucionado a questão também.

Retornando Valores

Todas a funções até este momento eram “void” (vazio, no sentido de que nada devolvem)

```
void setup() {

}
void loop() {

}
```

- Agora para uma função retornar algum valor, ela deve conter o comando de return e deve ser especificado o tipo de valor para o retorno.

$$\begin{cases} int\ x \\ return\ x \end{cases}$$

Retornando Valores

Todas a funções até este momento eram “void” (vazio, no sentido de que nada devolvem)

```
void setup() {

}
void loop() {

}
```

- Agora para uma função retornar algum valor, ela deve conter o comando de return e deve ser especificado o tipo de valor para o retorno.

$$\begin{cases} int\ x \\ return\ x \end{cases}$$

Exemplo 7

Faça uma função que receba 2 valores e retorne a multiplicação entre esses 2 valores.

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

void loop() {
  int a = 2;
  int b = 3;
  int c;

  c = funcaoMultiplicacao(a, b); // c = 6.
  Serial.println(c);
  delay(500);
}

int funcaoMultiplicacao(int x, int y){
  int resultado;
  resultado = x * y;
  return resultado;
}
```

Função recebe 2 valores

Retorna o resultado da multiplicação dos 2 valores

Tipos de Variáveis

boolean	1	verdadeiro ou falso (0 ou 1)	
char	1	–128 até +128	Usado para representar um código de caractere ASCII (American Standard Code for Information Interchange)*. Por exemplo, *A* é representado como 65. Normalmente, os seus números negativos não são usados.
byte	1	0 até 255	Frequentemente usado na comunicação serial de dados. Veja o Capítulo 9.
int	2	–32768 até +32767	
unsigned int	2	0 até 65536	Pode ser usado para ter uma precisão extra, quando não há necessidade de números negativos. Use com cautela, porque a aritmética que usa o tipo **int** pode produzir resultados inesperados.
long	4	–2,147,483,648 até 2,147,483,647	Necessário apenas para representar números muito grandes.
unsigned long	4	0 até 4,294,967,295	Veja **unsigned int**.
float	4	–3.4028235E+38 até + 3.4028235E+38	
`double`	4	O mesmo que **float**	Normalmente, seriam 8 bytes com uma precisão mais elevada que **float** e um intervalo de representação maior. No Arduino, entretanto, é o mesmo que **float**.

Arrays (estruturando os dados)

Um Array é uma maneira de organizar uma lista de valores. Um Array contem uma lista de valores onde pode ser acessado qualquer um destes valores de acordo com a posição nesta lista.

Obs.: Até agora só havia sido usado 1 valor por variável.

- Para criar um Array consiste em usar chaves que contém os valores, separados entre si por vírgulas.

  Ex.:

```
int durations[] = {200, 200, 200, 500, 500, 500, 200, 200, 200};
```

- Para acessar os valores desta array, deve-se usar notação de colchete.

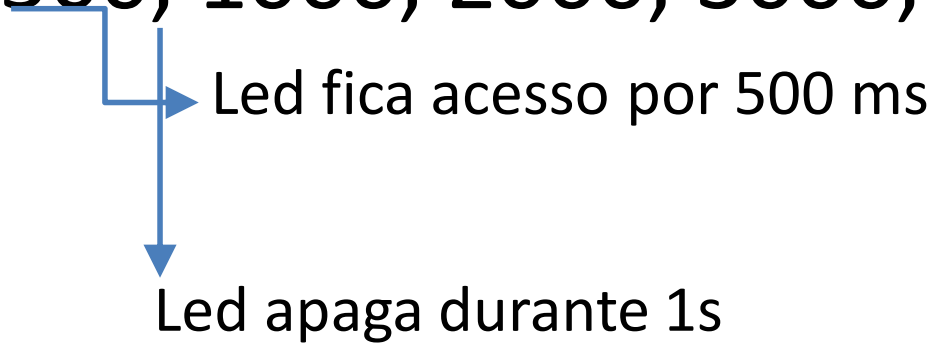
  Ex.: Para acessar o primeiro elemento desta lista, deve-se usar o seguinte comando:

  durations [0]

Exemplo 8

Elabore uma lista de 5 valores mostrado abaixo, onde a cada loop da função “void loop” o programa mostre o valor de uma posição desta lista e utilize este valor para deixar o led da porta 13 do Arduino acesso numa duração igual ao valor da posição da lista. A cada mudança de valores da lista, o led do Arduino deve ser apagado durante 1s.

Lista = { 500, 1000, 2000, 3000, 4000}



Exercício 8

Resultado:

```
int ledPin = 13;
int duracao[] = {500, 1000, 2000, 3000, 40

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop()
{
for (int i = 0; i < 9; i++)
  {
    Serial.println(duracao[i]);
    digitalWrite(ledPin, HIGH);
    delay(duracao[i]);
    digitalWrite(ledPin, LOW);
    delay(1000);
  }
}
```

500

1000

2000

3000

4000

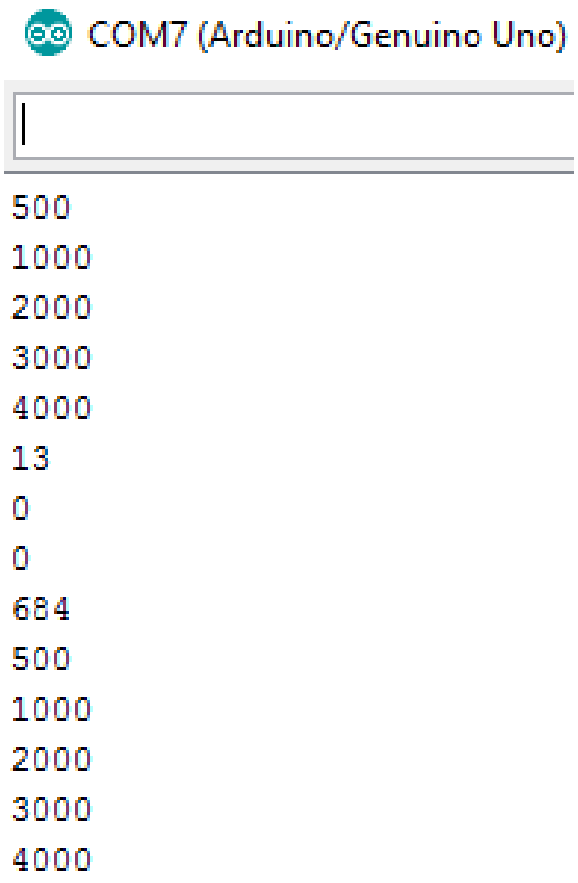
? 13

? 0

? 0

? 684

Problema:



Obs.: Exercício 8

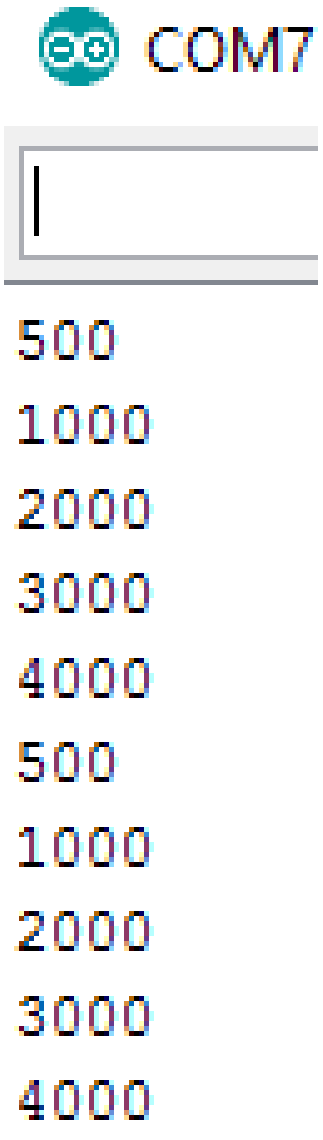
Neste caso, está sendo acessado memórias além do objetivo deste exercício, pois mesmo o programa sendo compilado, o compilador não impede que seja acessado dados que estão além do final do Array.
O computador mantêm os seus dados, tanto os comuns como os arrays, na memória.

Solução Exercício 8

```
int ledPin = 13;
int duracao[] = {500, 1000, 2000, 3000, 4000};

void setup()
{
  Serial.begin(9600);
  pinMode(ledPin, OUTPUT);
}

void loop()
{
for (int i = 0; i < 5; i++)
  {
    Serial.println(duracao[i]);
    digitalWrite(ledPin, HIGH);
    delay(duracao[i]);
    digitalWrite(ledPin, LOW);
    delay(1000);
  }
}
```



Referência

Monk, Simon; Programação com Arduino, começando com sketches. Porto Alegre: Bookman, 2013.