ALGORITMOS E TÉCNICAS DE PROGRAMAÇÃO

FABER XAVIER

INFORMAÇÃO É O NOVO PETRÓLEO

TIPOS DE DADOS

- Textual
  - Caracteres
  - Textos
- Números
  - Inteiros
  - Fracionados
- Booleanos
- Estruturais
- Tempo

ARQUITETURA DE COMPUTADORES

O que é ? Onde vive? Como se alimentam?

www.comm

COMMODORE BN/E

CARTRIDGE EXPANSION (44P N FEMALE) CN6

[5] R8 RESISTOR SHOULD BE REMOVE FOR 8500 SID.

[4] FRANCE VERSION
R21 200 RESISTOR & C60 2200pF CAPACITOR

[6] POWER SUPPLY
8501 SID VERSION:
CRT - 12.6V SHOULD BE USED
8580 SID VERSION:
CR3 - 9.6V SHOULD BE USED

[5] JUMPER CONNECTION J3 OPEN NTSC
J3 SHORT PAL

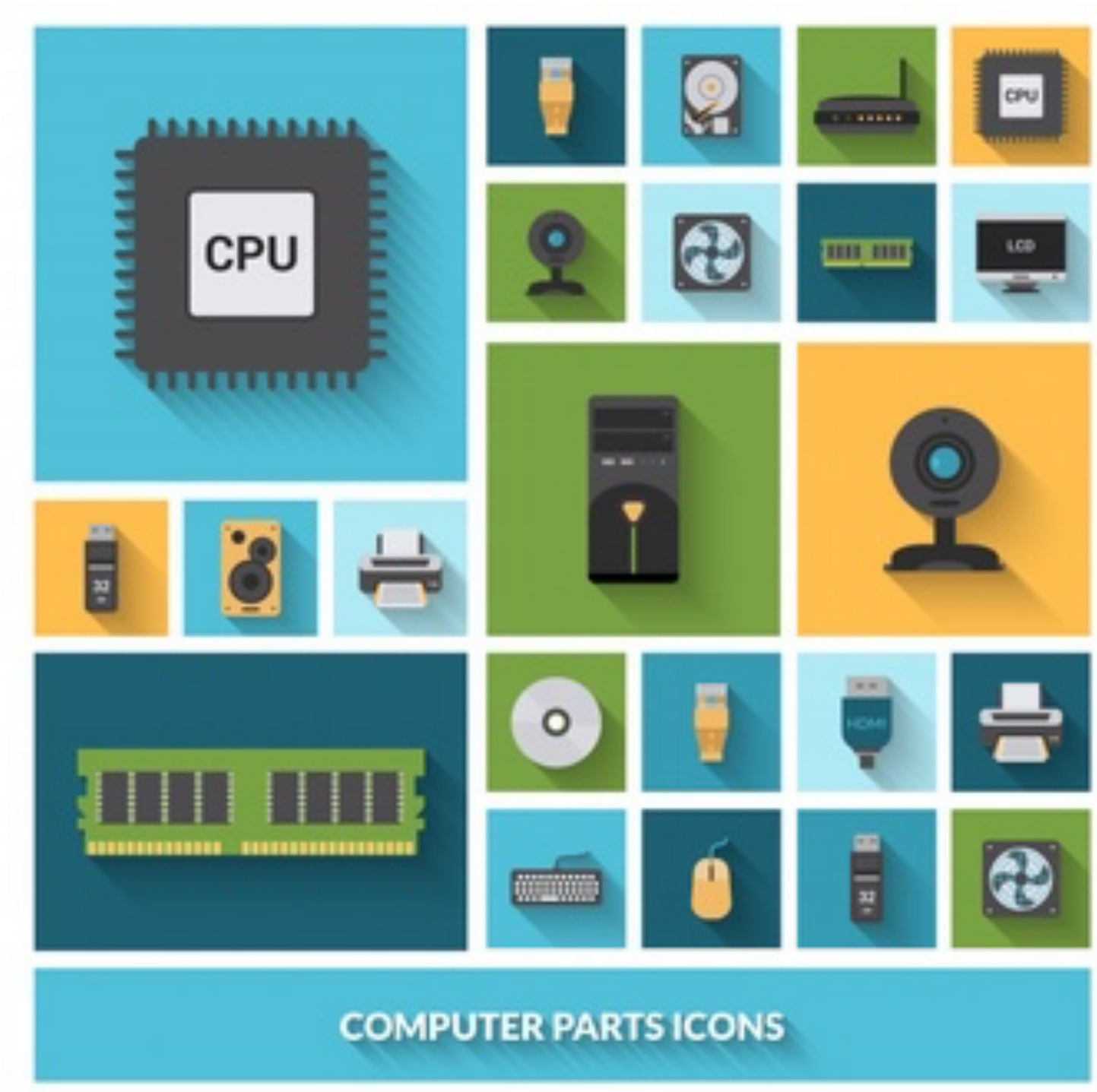
[4] DOT CLOCK: 8,18 MHz FOR NTSC
7,88 MHz FOR PAL

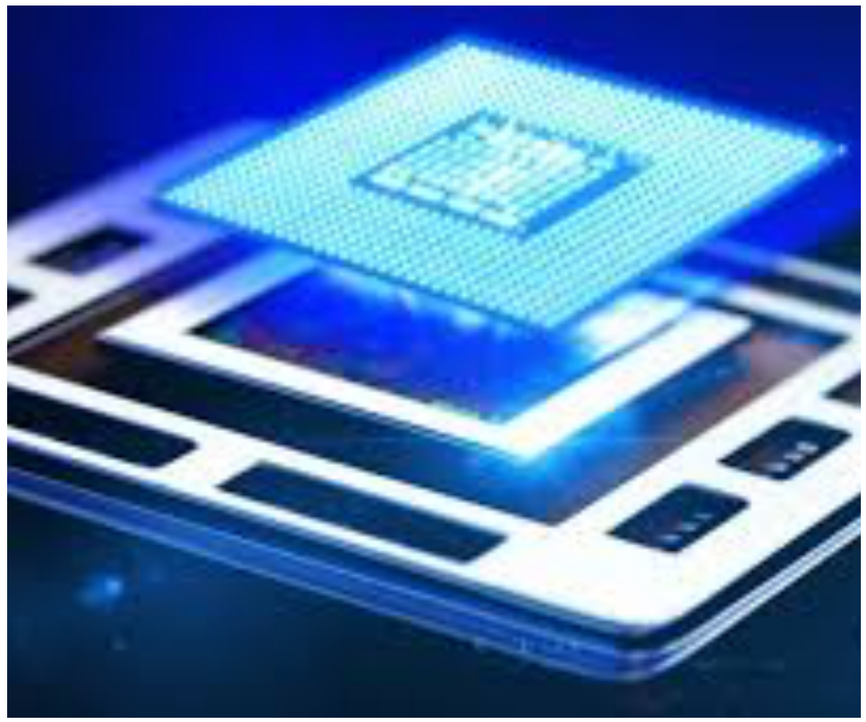
[3] Y1 14,31818 MHz FOR NTSC C_L = 18P
17,734472 MHz FOR PAL C_L = 18P

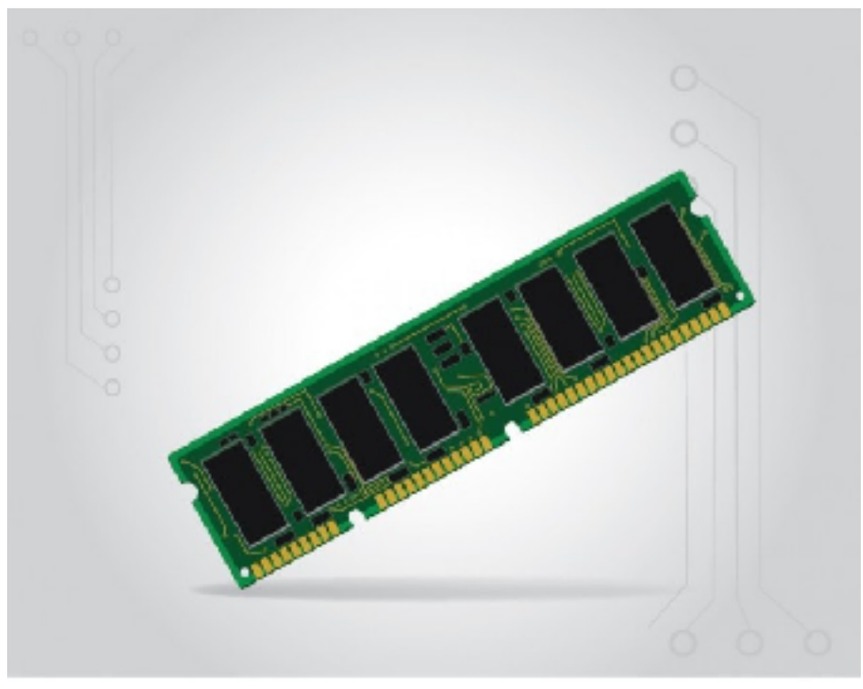
2 ALL CAPACITOR VALUES ARE IN µF

1 ALL RESISTORS ARE 1/4 WATT 5 %

NOTES: UNLESS OTHERWISE SPECIFIED:

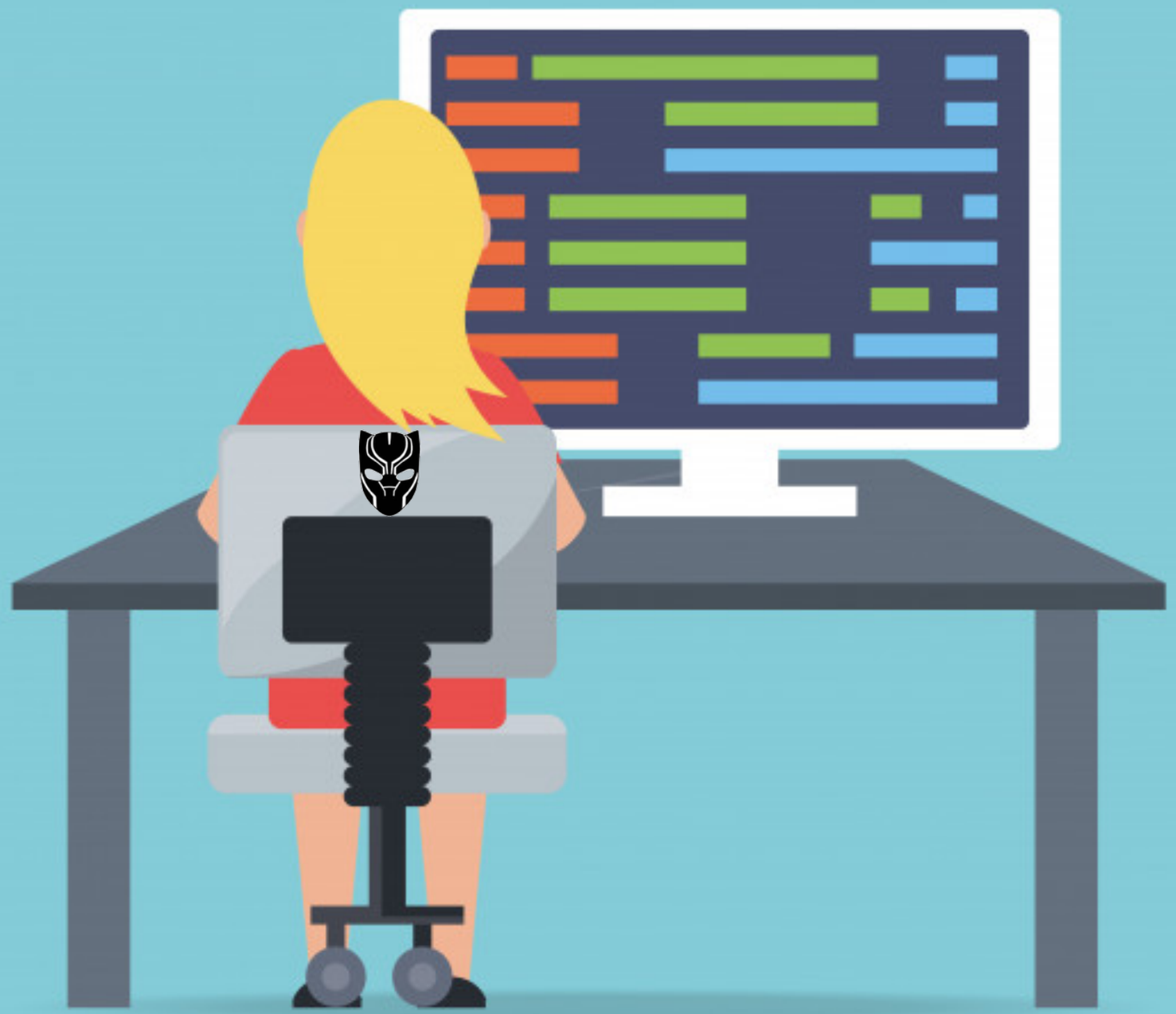

CPU
CPU
LCD
HDMI
COMPUTER PARTS ICONS





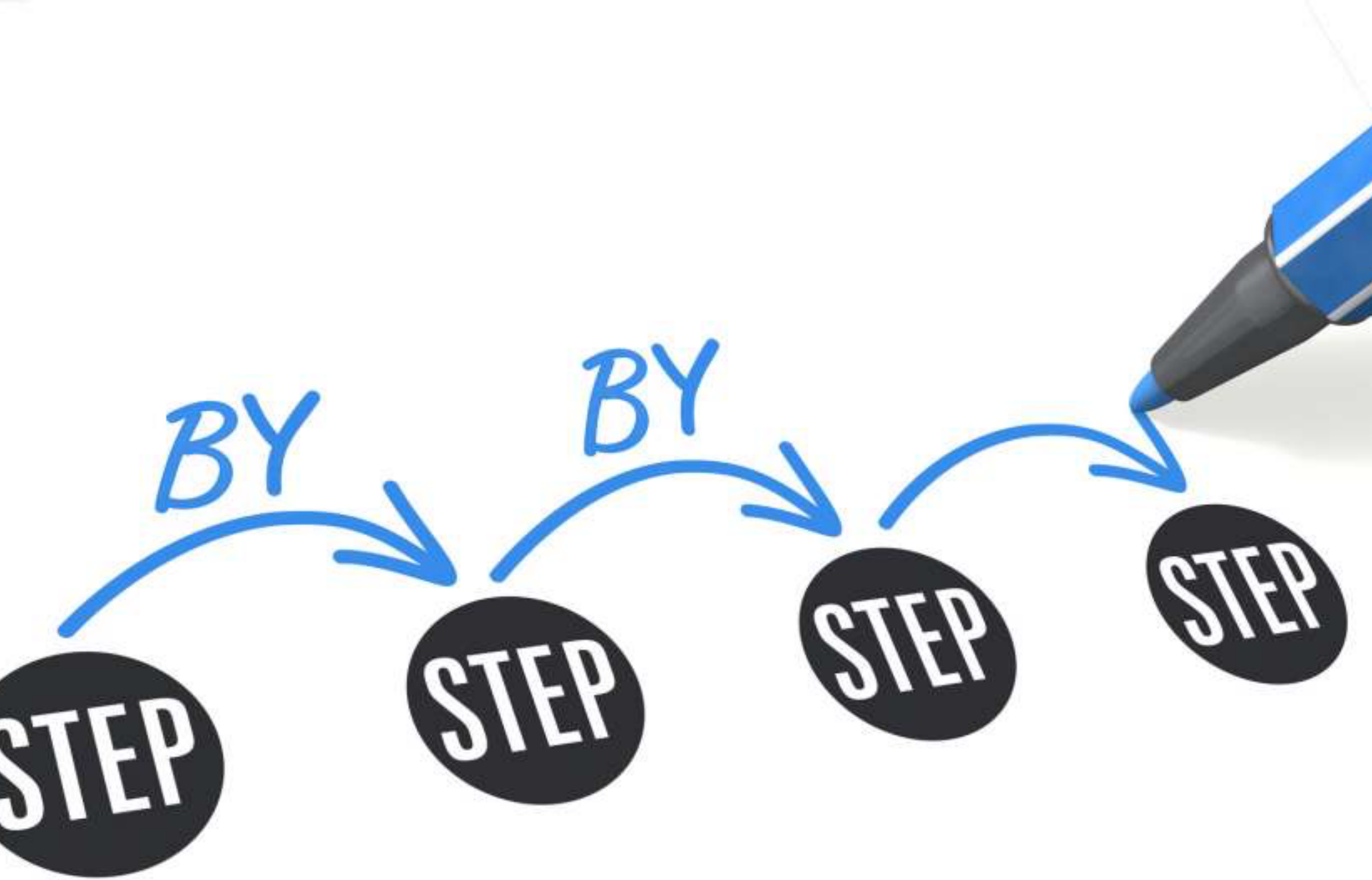
TIPOS DE DADOS

- Textual
  - Caracteres
  - Textos
- Números
  - Inteiros
  - Fracionados
- Booleanos
- Estruturais
- Tempo



Let's CODE

VARIÁVEIS / COMANDOS DE ENTRADA E SAÍDA


BY
BY
STEP
STEP
STEP
STEP

TIPOS DE DADOS

sbyte	SByte	8		signed byte	–128 to 127
byte	Byte	8		unsigned byte	0 to 255
short	Int16	16		short integer	-32,768 to 32,767
ushort	UInt16	16		unsigned short integer	0 to 65535
int	Int32	32		integer	-2,147,483,648 to 2,147,483,647
uint	UInt32	32	u	unsigned integer	0 to 4,294,967,295
long	Int64	64	L	long integer	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	UInt64	64		unsigned long integer	0 to 18,446,744,073,709,551,615
char	Char	16		Unicode character	any valid character (e.g., 'a' , '*', '\x0058' [hexadecimal], or '\u0058' [Unicode])
float	Single	32	F	floating-point number	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$
double	Double	64	d	double floating-point number	Range $\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$
bool	Boolean	1		logical true/false value	true/false
decimal	Decimal	128	m	used for financial and monetary calculations	from approximately 1.0×10^{-28} to 7.9×10^{28} with 28 to 29 significant digits

OPERADORES

Category (by precedence)	Operator(s)	Associativity
Primary	x.y f(x) a[x] x++ x-- new typeof default checked unchecked delegate	left
Unary	+ - ! ~ ++x --x (T)x	left
Multiplicative	* / %	left
Additive	+ -	left
Shift	<< >>	left
Relational	< > <= >= is as	left
Equality	== !=	right
Logical AND	&	left
Logical XOR	^	left
Logical OR	|	left
Conditional AND	&&	left
Conditional OR	||	left
Null Coalescing	??	left
Ternary	?:	right
Assignment	= *= /= %= += -= <<= >>= &= ^= |= =>	right

O USUÁRIO

O USUÁRIO

```
0 references
static void Main(string[] args)
{
    int vari
    int vari
    // Quand
    //Testan
    Console.
     //Testa
    Console.
    //Testan
    Console.
    //Testan
    Console.
    Console.ReadLine();
```

Reads the next line of characters from the standard input stream.

The next line of characters from the input stream, or null if no more lines are available.

Exceptions:

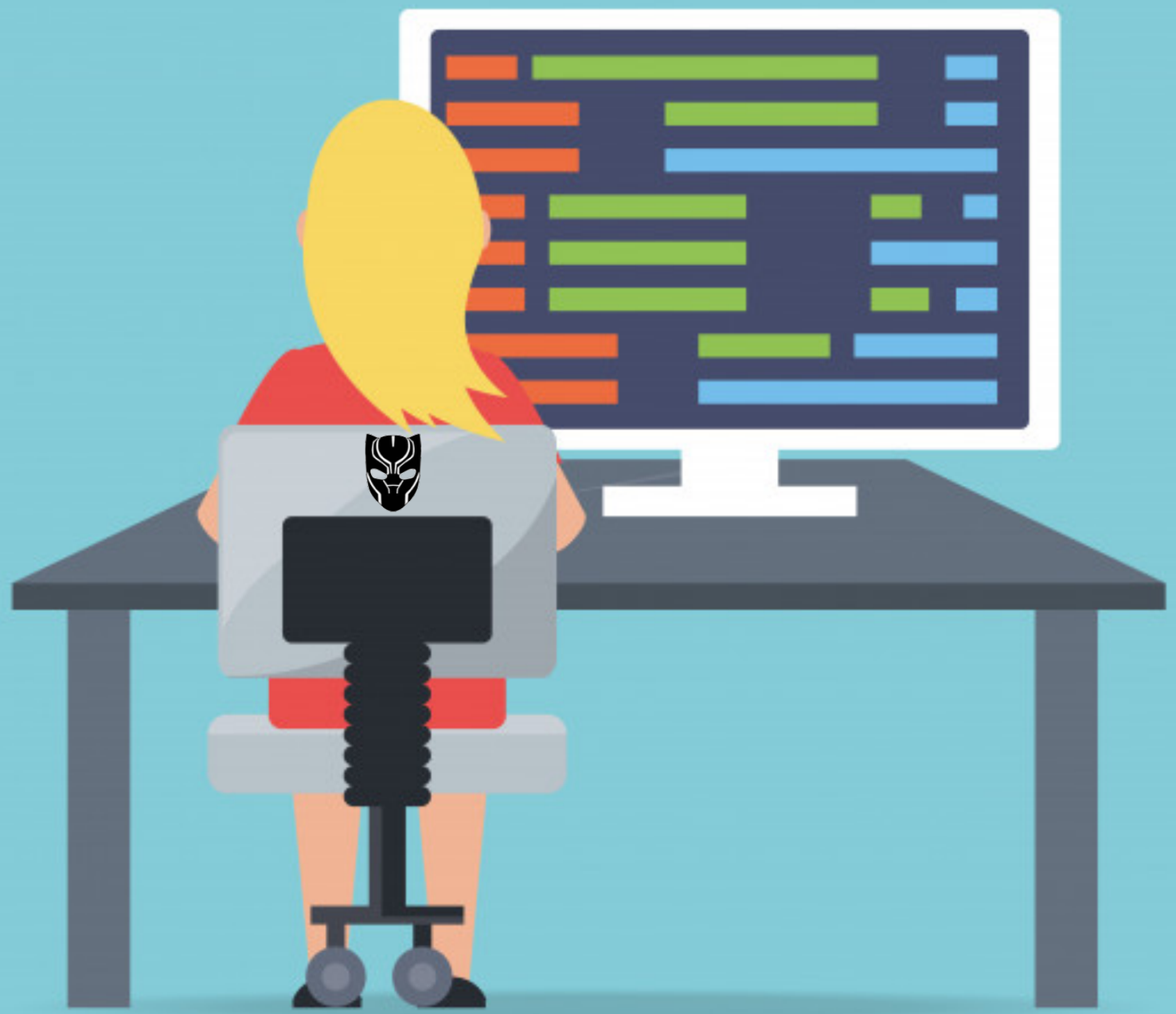
    System.IO.IOException: An I/O error occurred.
    System.OutOfMemoryException: There is insufficient memory to allocate a buffer for the returned string.
    System.ArgumentOutOfRangeException: The number of characters in the next line of characters is greater than System.Int32.MaxValue
    .

```
string Console.ReadLine()
```

TIPOS DE DADOS

sbyte	SByte	8		signed byte	–128 to 127
byte	Byte	8		unsigned byte	0 to 255
short	Int16	16		short integer	-32,768 to 32,767
ushort	UInt16	16		unsigned short integer	0 to 65535
int	Int32	32		integer	-2,147,483,648 to 2,147,483,647
uint	UInt32	32	u	unsigned integer	0 to 4,294,967,295
long	Int64	64	L	long integer	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
ulong	UInt64	64		unsigned long integer	0 to 18,446,744,073,709,551,615
char	Char	16		Unicode character	any valid character (e.g., 'a' , '*', '\x0058' [hexadecimal], or '\u0058' [Unicode])
float	Single	32	F	floating-point number	$\pm 1.5 \times 10^{-45}$ to $\pm 3.4 \times 10^{38}$
double	Double	64	d	double floating-point number	Range $\pm 5.0 \times 10^{-324}$ to $\pm 1.7 \times 10^{308}$
bool	Boolean	1		logical true/false value	true/false
decimal	Decimal	128	m	used for financial and monetary calculations	from approximately 1.0×10^{-28} to 7.9×10^{28} with 28 to 29 significant digits

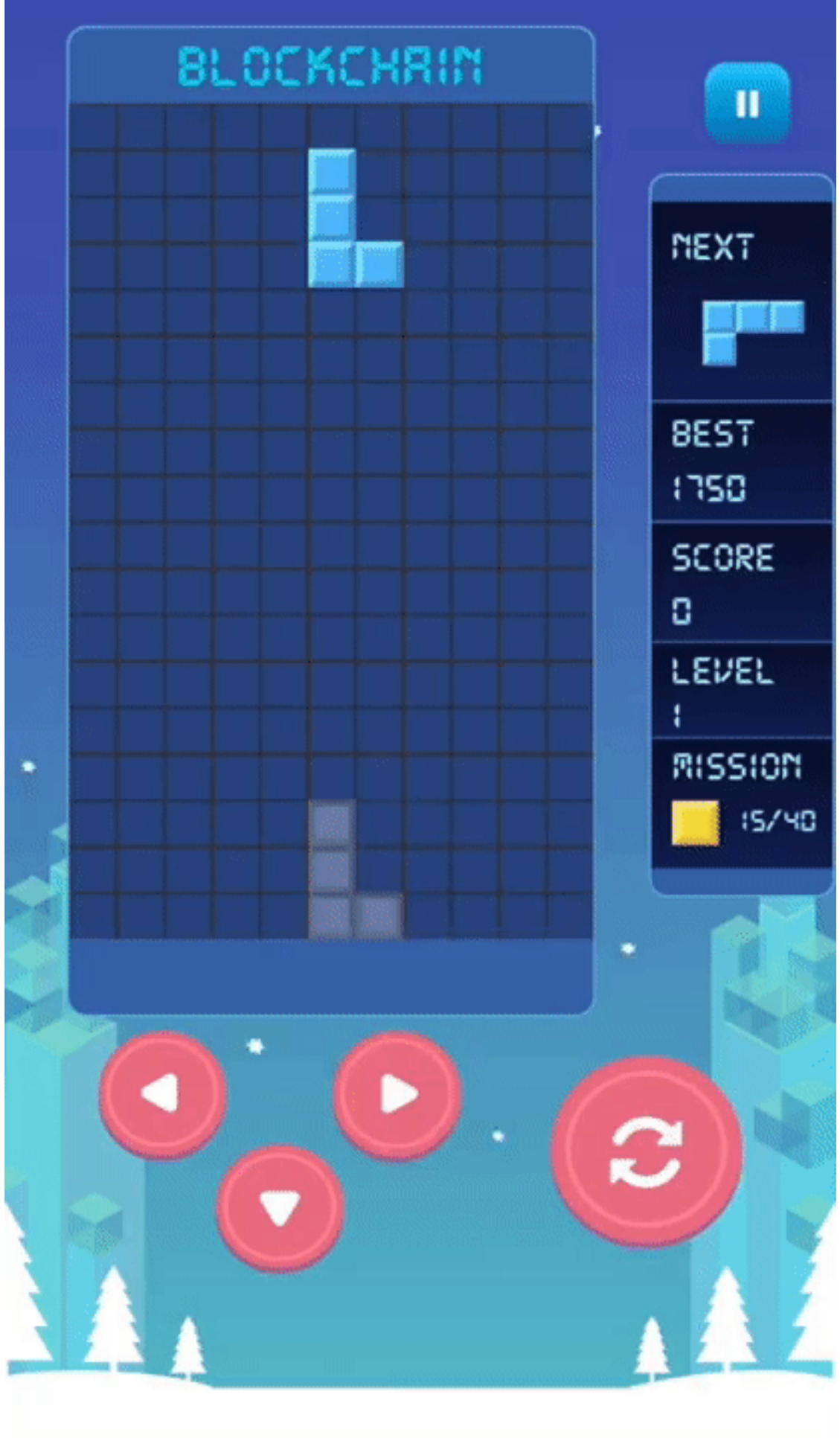
TIPOS DE DADOS



Let's CODE

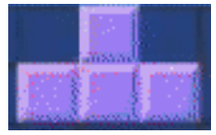
ESTRUTURAS CONDICIONAIS

VARIÁVEIS / COMANDOS DE ENTRADA E SAÍDA

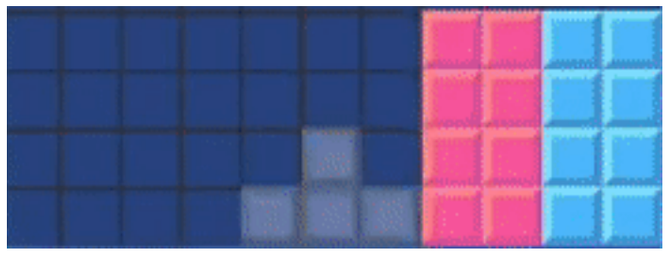


CÓDIGO LINEAR

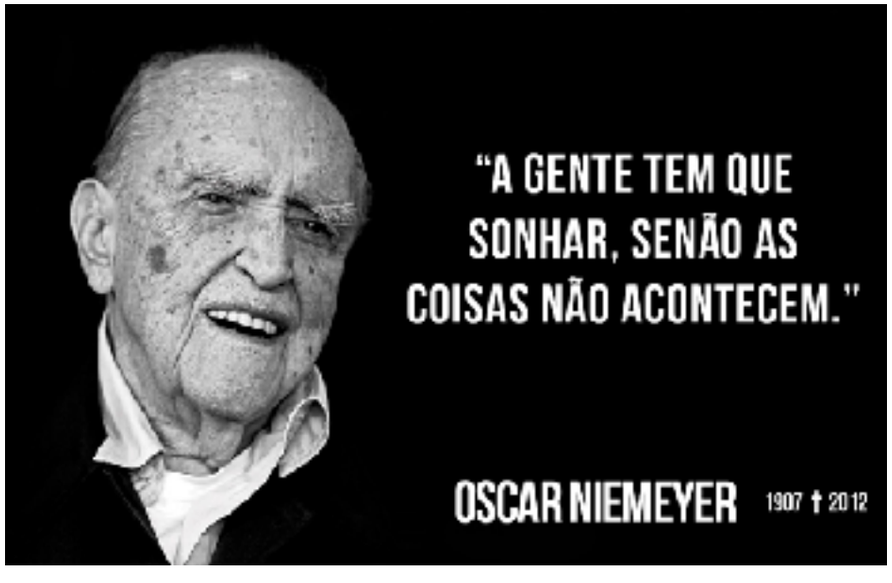
- Tarefas até agora sempre foram executadas
- Onde colocar a próxima peça?



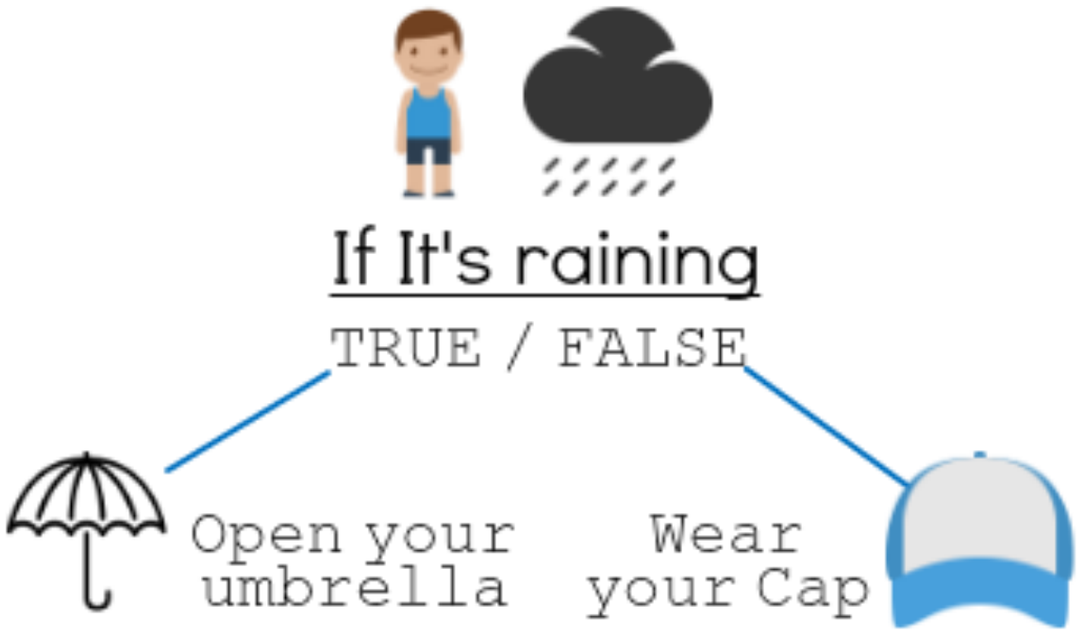
- Sabendo que o estágio atual é:

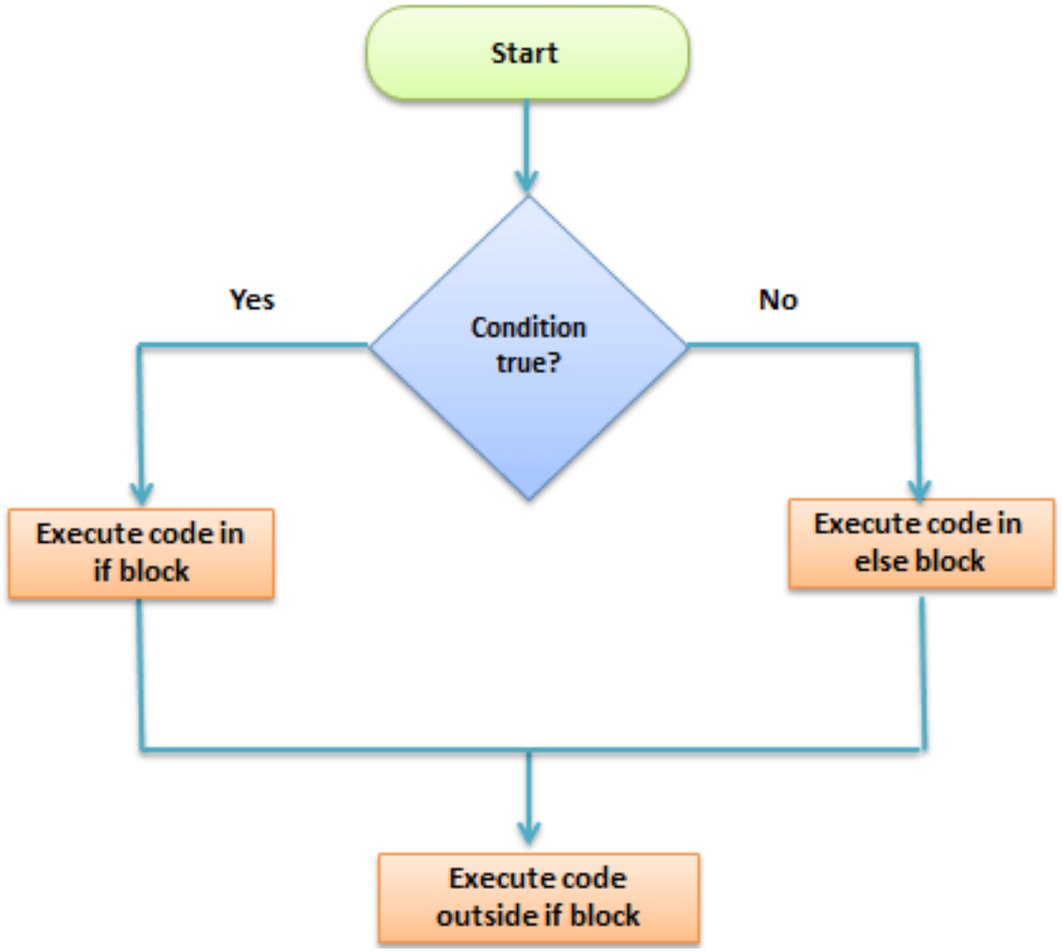


ESTRUTURAS CONDICIONAIS

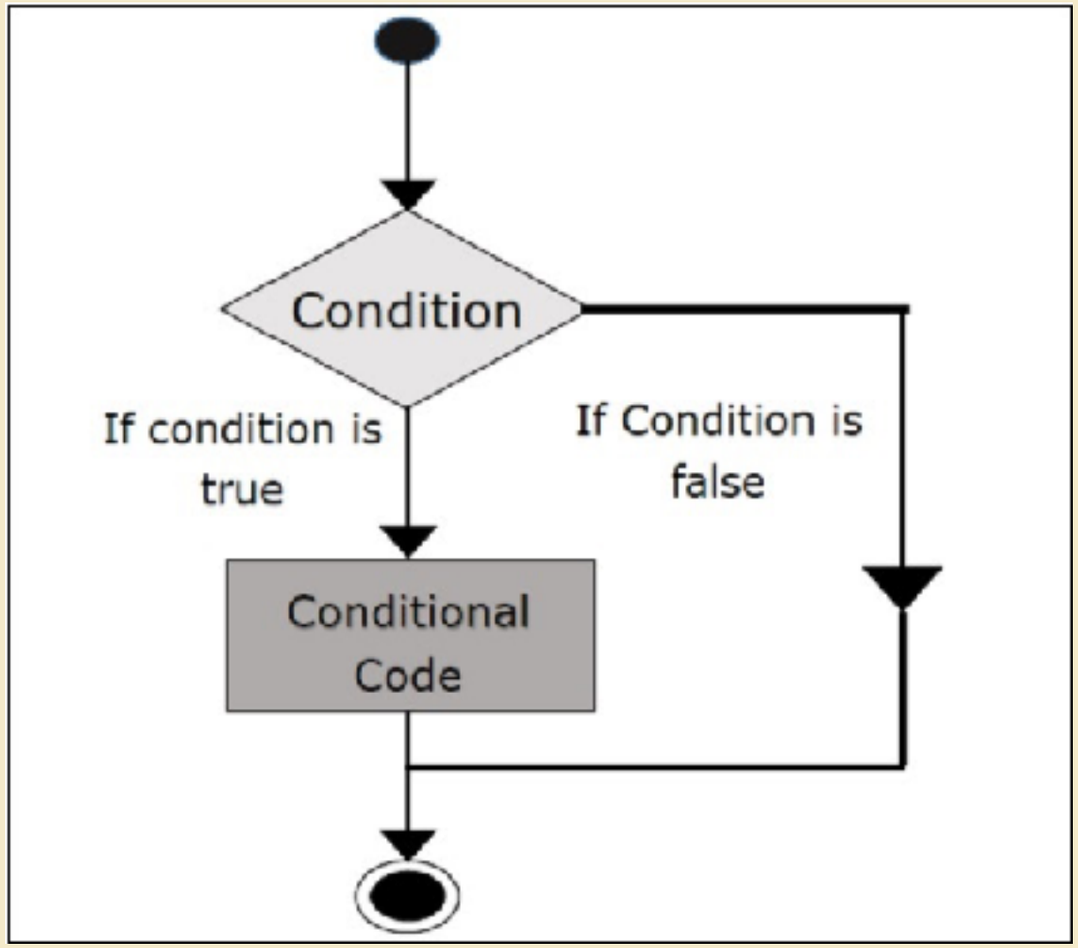


ESTRUTURAS CONDICIONAIS



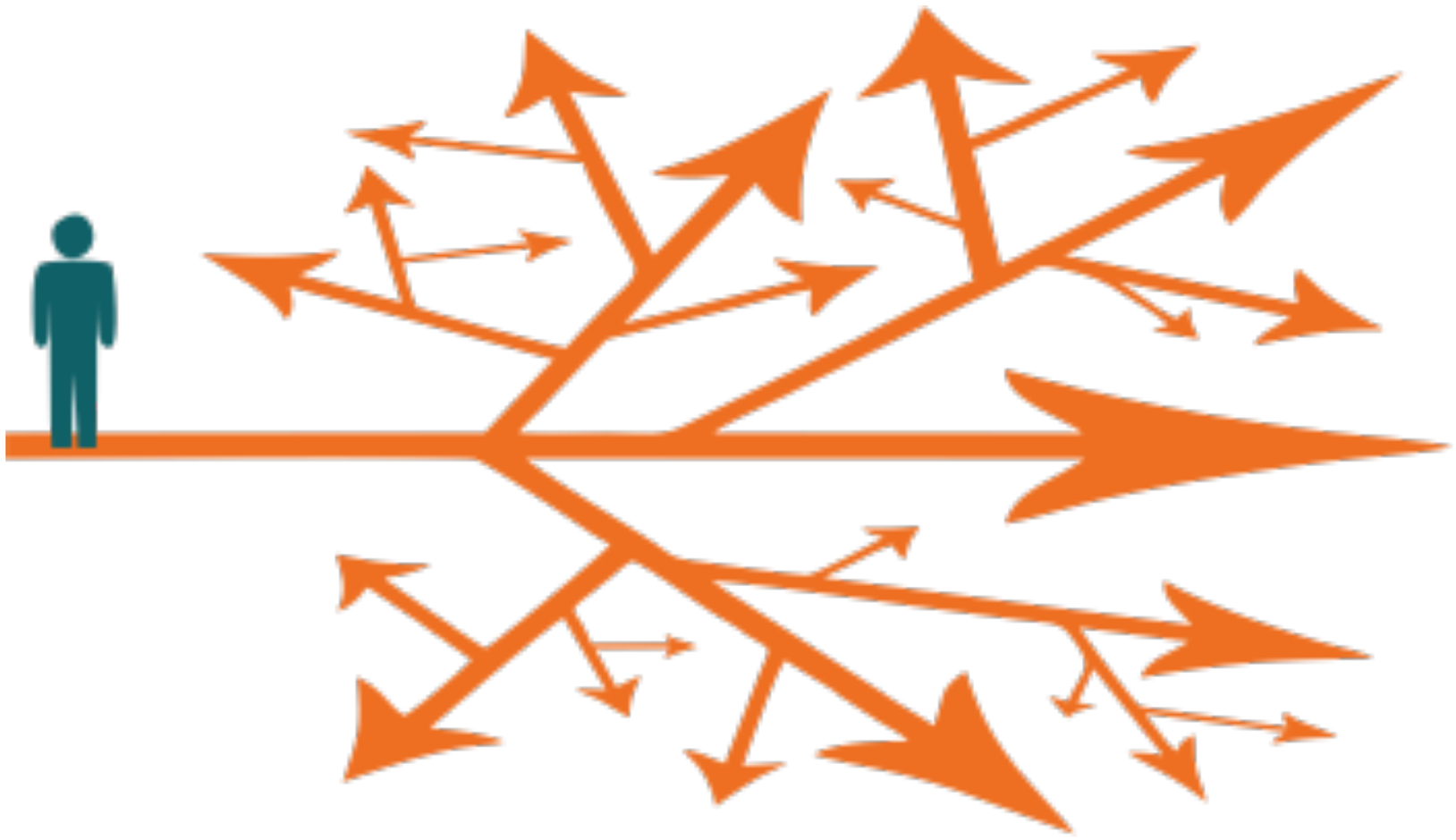


COMANDO IF


Condition
If condition is true
If Condition is false
Conditional Code

Let's CODE

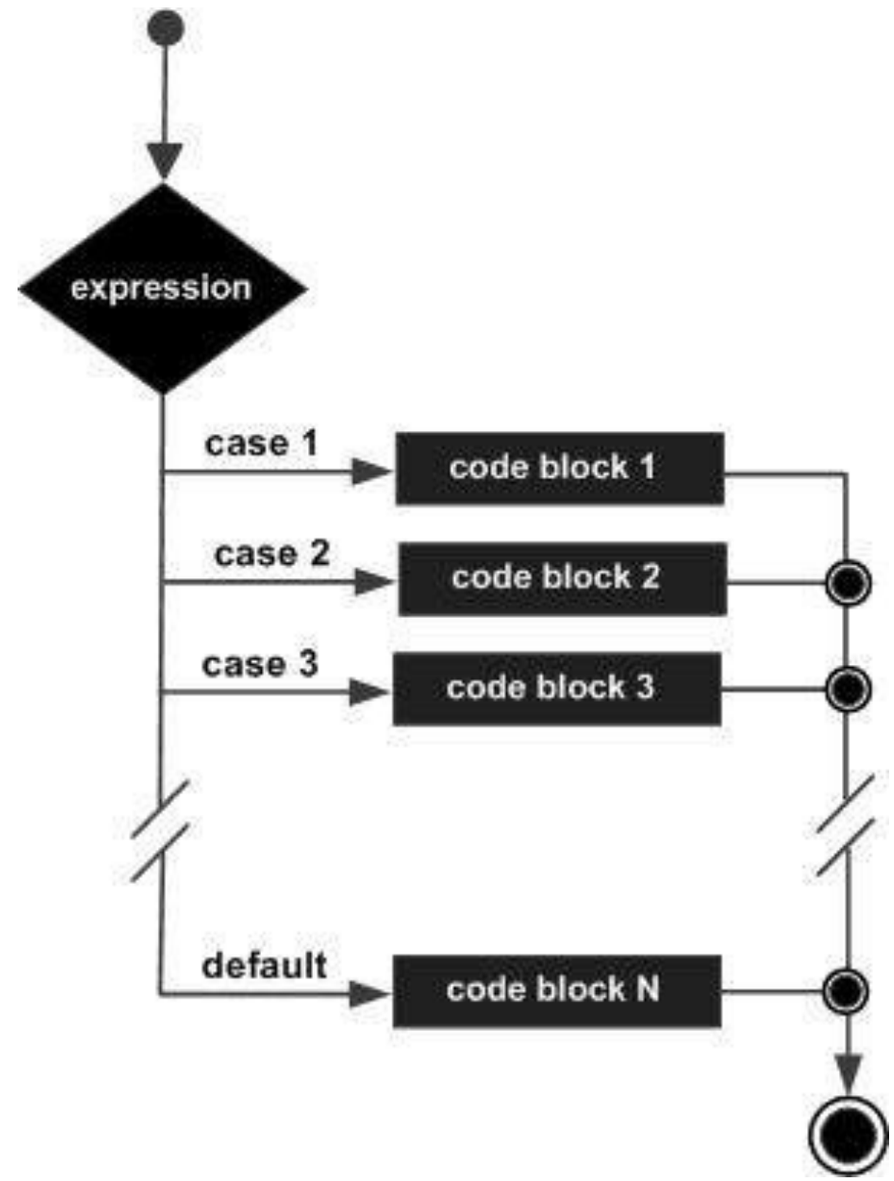
CONDIÇÕES



COMANDO SWITCH

WHATEVER YOUR CHOICE IS,
I WILL ALWAYS STAND BY YOU

WHALE LAB



Let's CODE

13. Faça um programa que apresente o menu a seguir, permita ao usuário escolher a opção desejada, receba os dados necessários para executar a operação e mostre o resultado. Verifique a possibilidade de opção inválida e não se preocupe com restrições, como salário negativo.

Menu de opções:
1. Imposto
2. Novo salário
3. Classificação

Digite a opção desejada.

Na opção 1: receber o salário de um funcionário, calcular e mostrar o valor do imposto usando as regras a seguir:

SALÁRIO	PERCENTUAL DO IMPOSTO
Menor que R$ 500,00	5%
De R$ 500,00 a R$ 850,00	10%
Acima de R$ 850,00	15%

Na opção 2: receber o salário de um funcionário, calcular e mostrar o valor do novo salário, usando as regras a seguir:

SALÁRIO	AUMENTO
Maior que R$ 1.500,00	R$ 25,00
De R$ 750,00 (inclusive) a R$ 1.500,00 (inclusive)	R$ 50,00
De R$ 450,00 (inclusive) a R$ 750,00	R$ 75,00
Menor que R$ 450,00	R$ 100,00

Na opção 3: receber o salário de um funcionário e mostrar sua classificação usando a tabela a seguir:

SALÁRIO	CLASSIFICAÇÃO
Até R$ 700,00 (inclusive)	Mal remunerado
Maiores que R$ 700,00	Bem remunerado

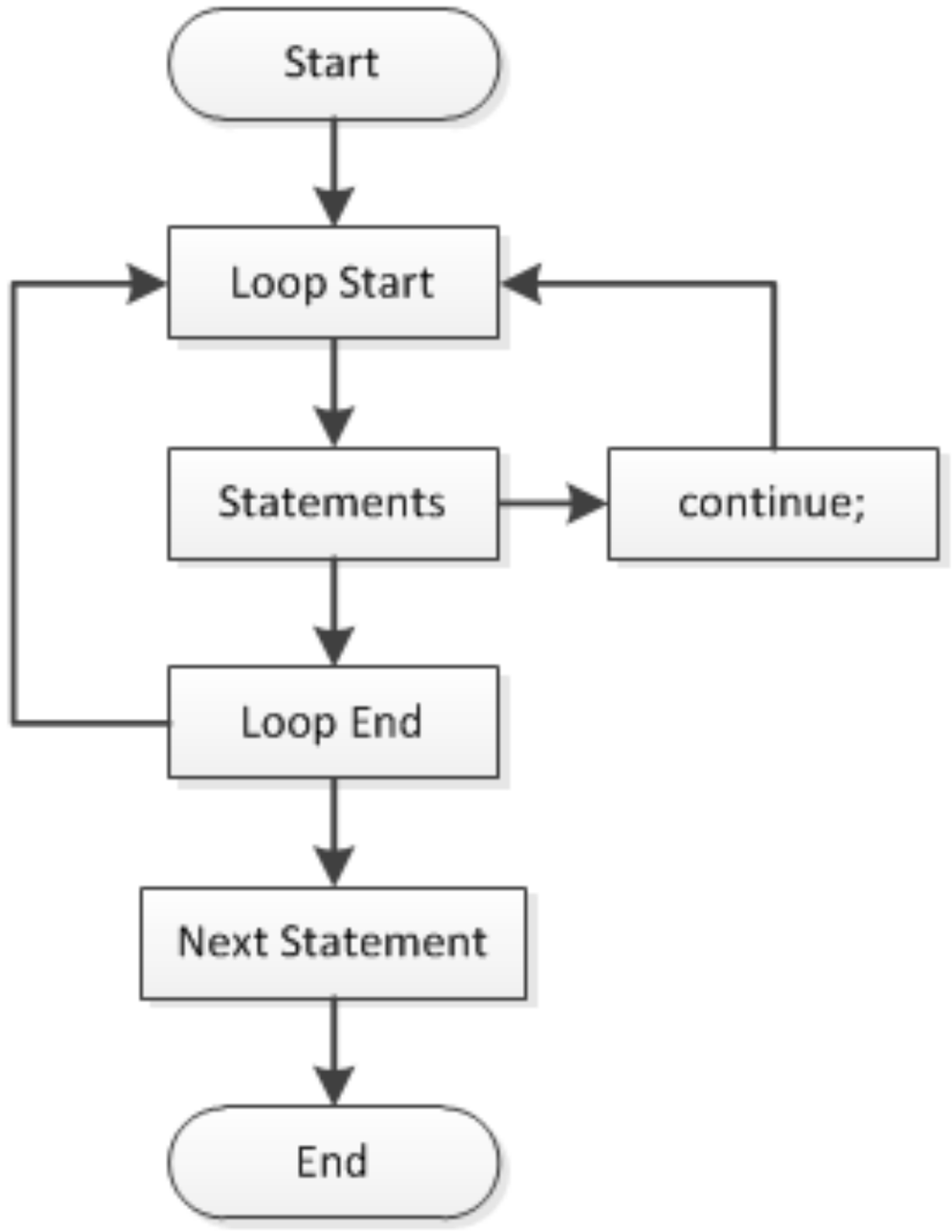
ESTRUTURAS REPETIÇÃO

ESTRUTURAS CONDICIONAIS

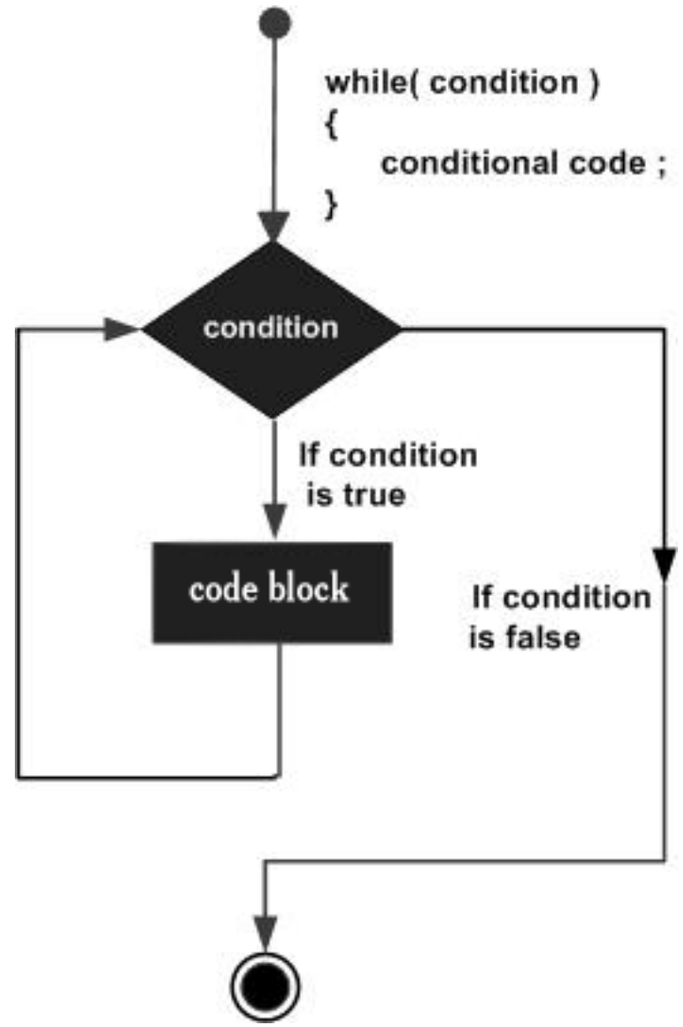
VARIÁVEIS / COMANDOS DE ENTRADA E SAÍDA

ESTRUTURAS DE REPETIÇÃO

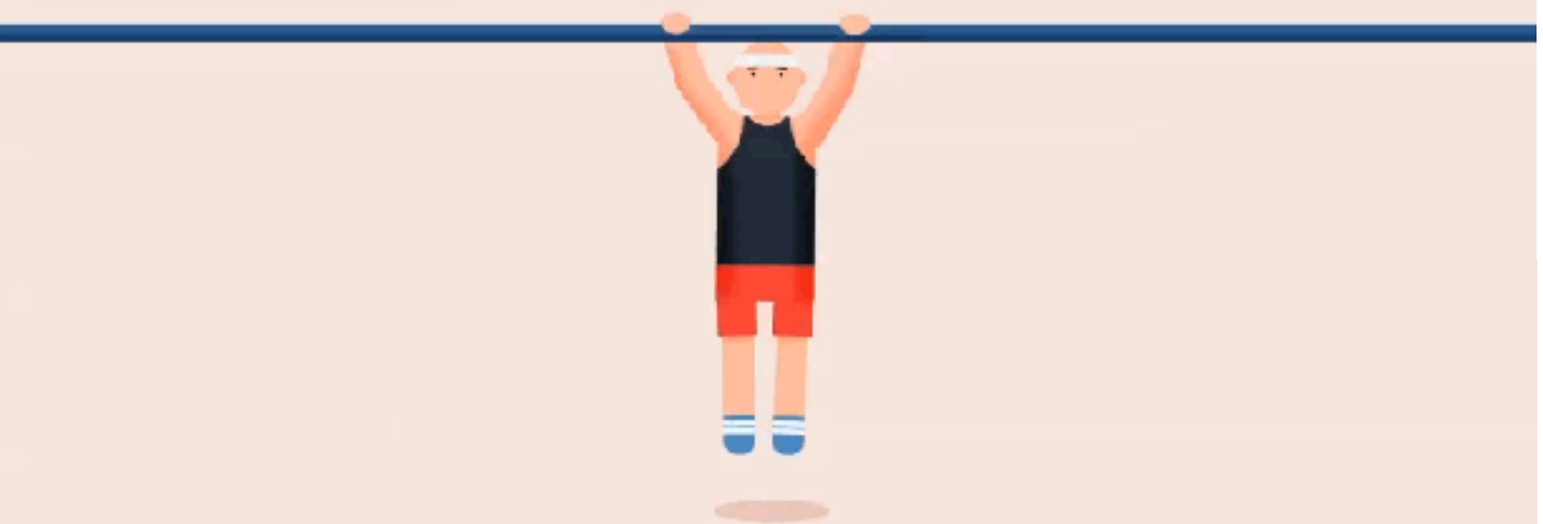
ESTRUTURA DE REPETIÇÃO

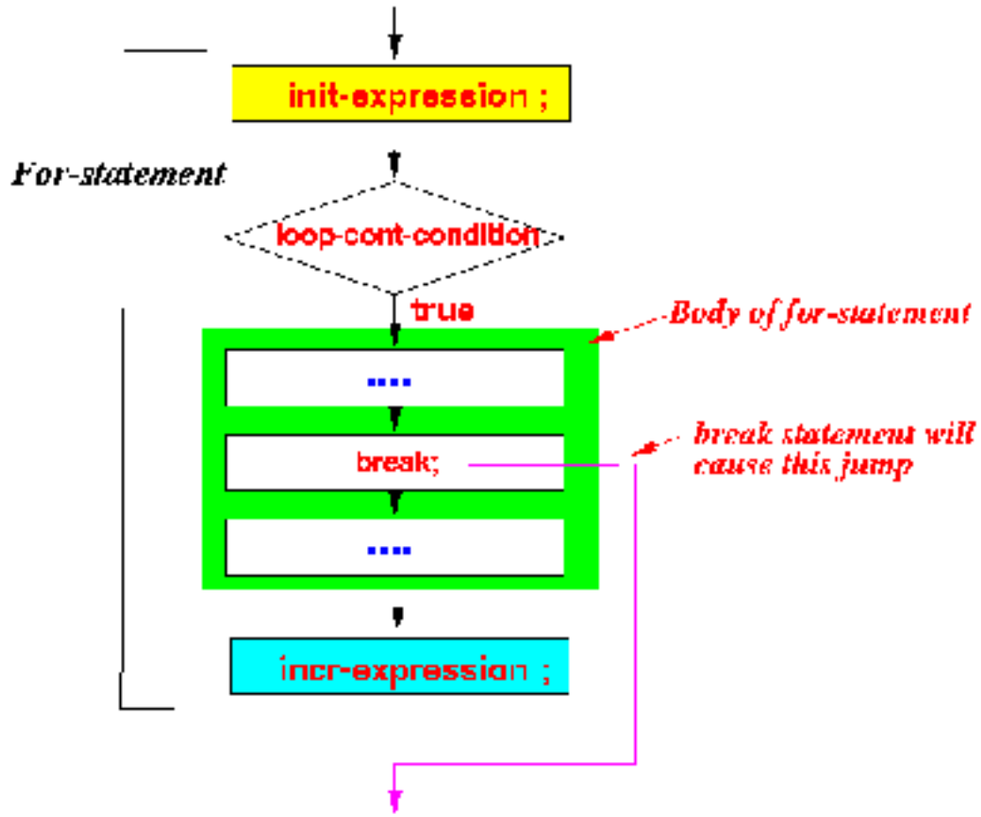


COMANDO WHILE



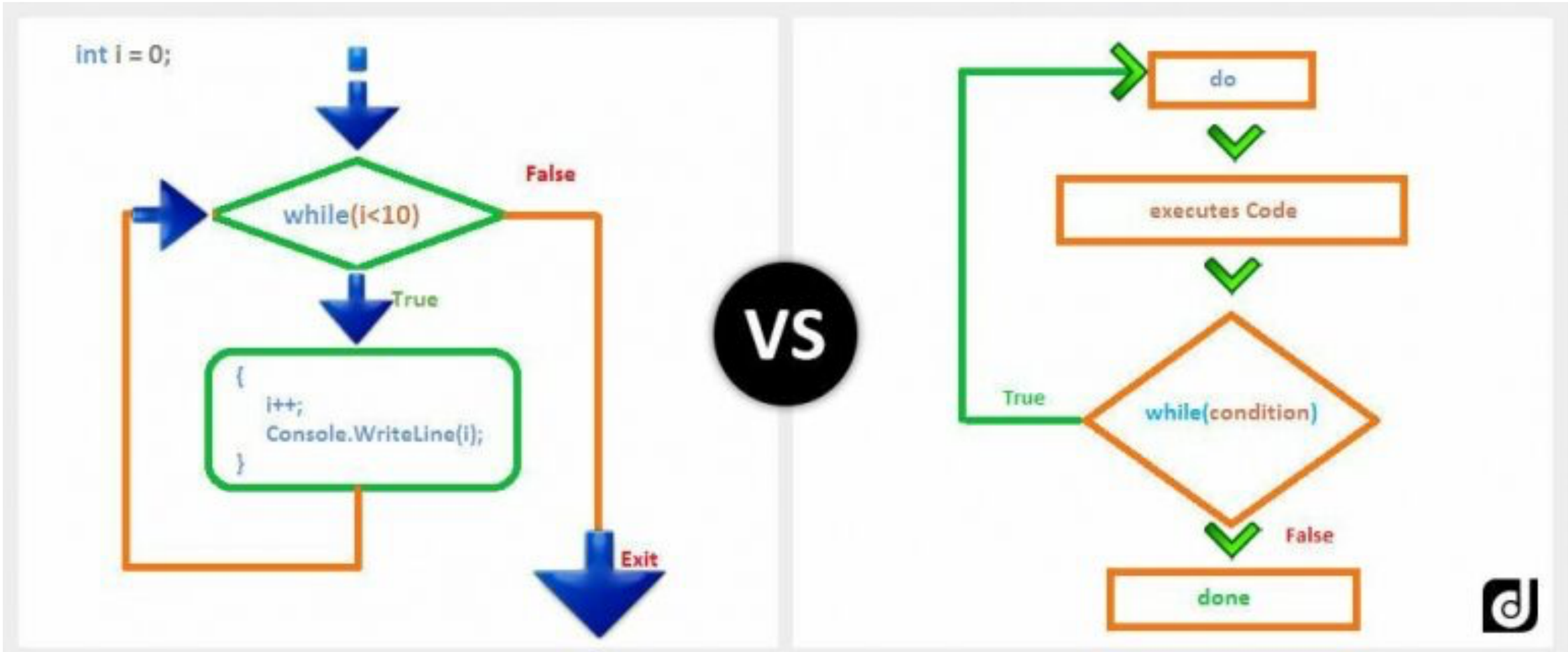
COMANDO FOR




init-expression ;
For-statement
loop-cont-condition
true
Body of for-statement
....
break;
break statement will cause this jump
....
incr-expression ;

ESTRUTURAS DE REPETIÇÃO

COMANDO DO-WHILE



Let's CODE

MODULARIZAÇÃO / FUNÇÕES

ESTRUTURAS REPETIÇÃO

ESTRUTURAS CONDICIONAIS

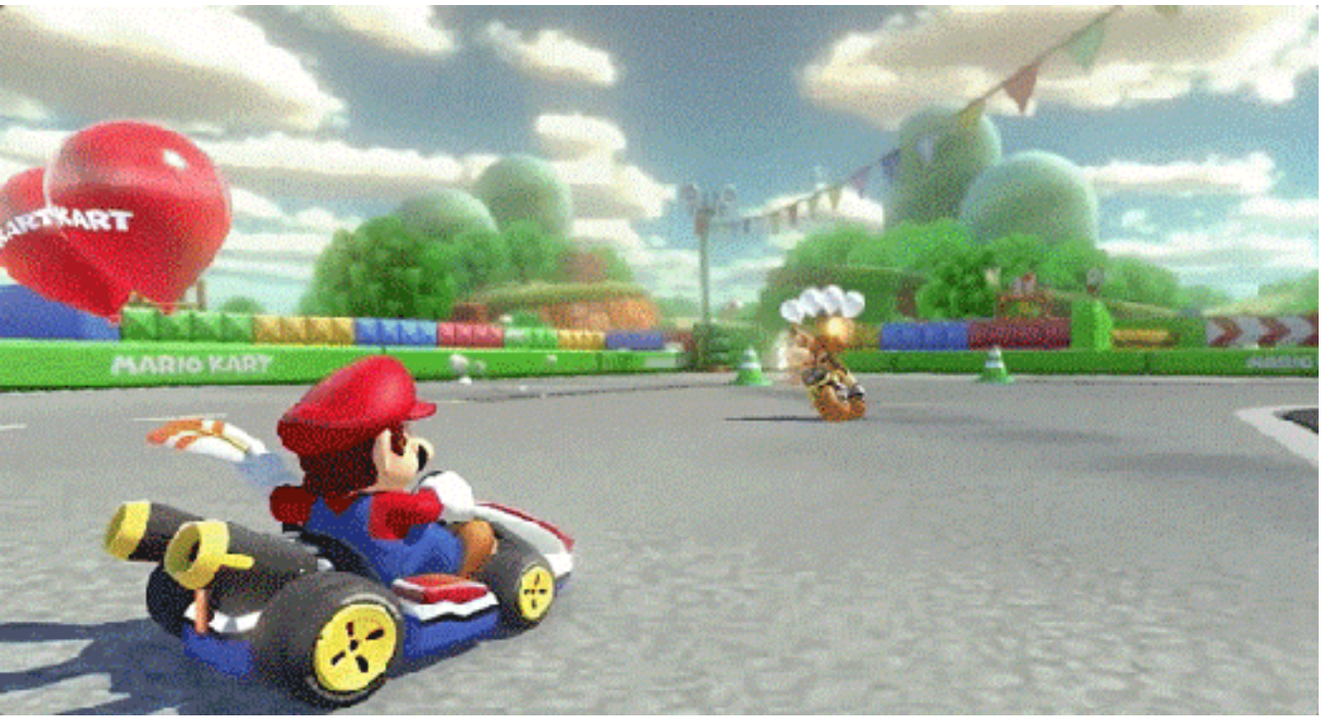
VARIÁVEIS / COMANDOS DE ENTRADA E SAÍDA

EXCUSE ME BUT...YOUR CODE SMELLS

MODULARIZAÇÃO

Vamos deixas as coisas menos complexas

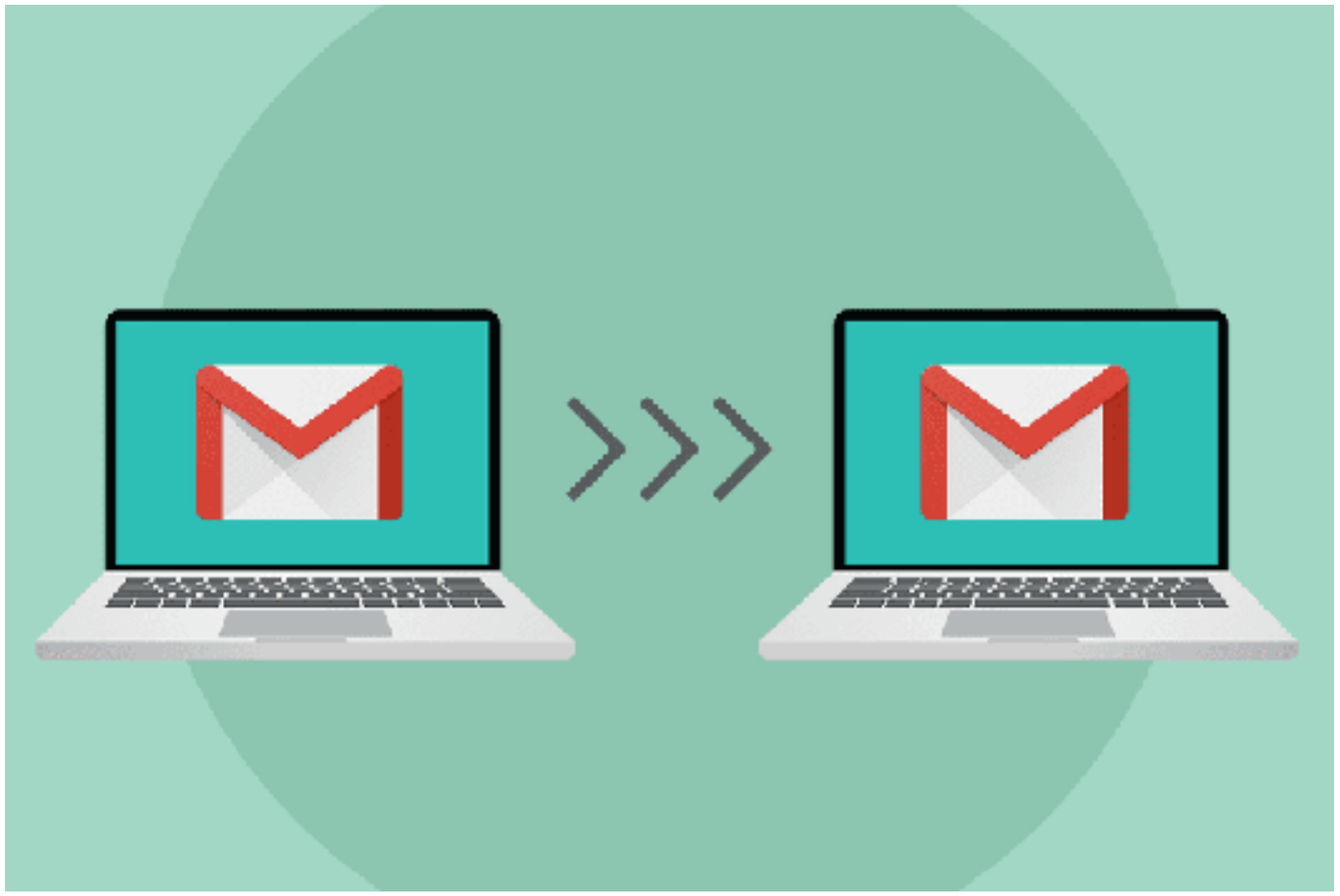
FUNÇÕES



FUNÇÕES

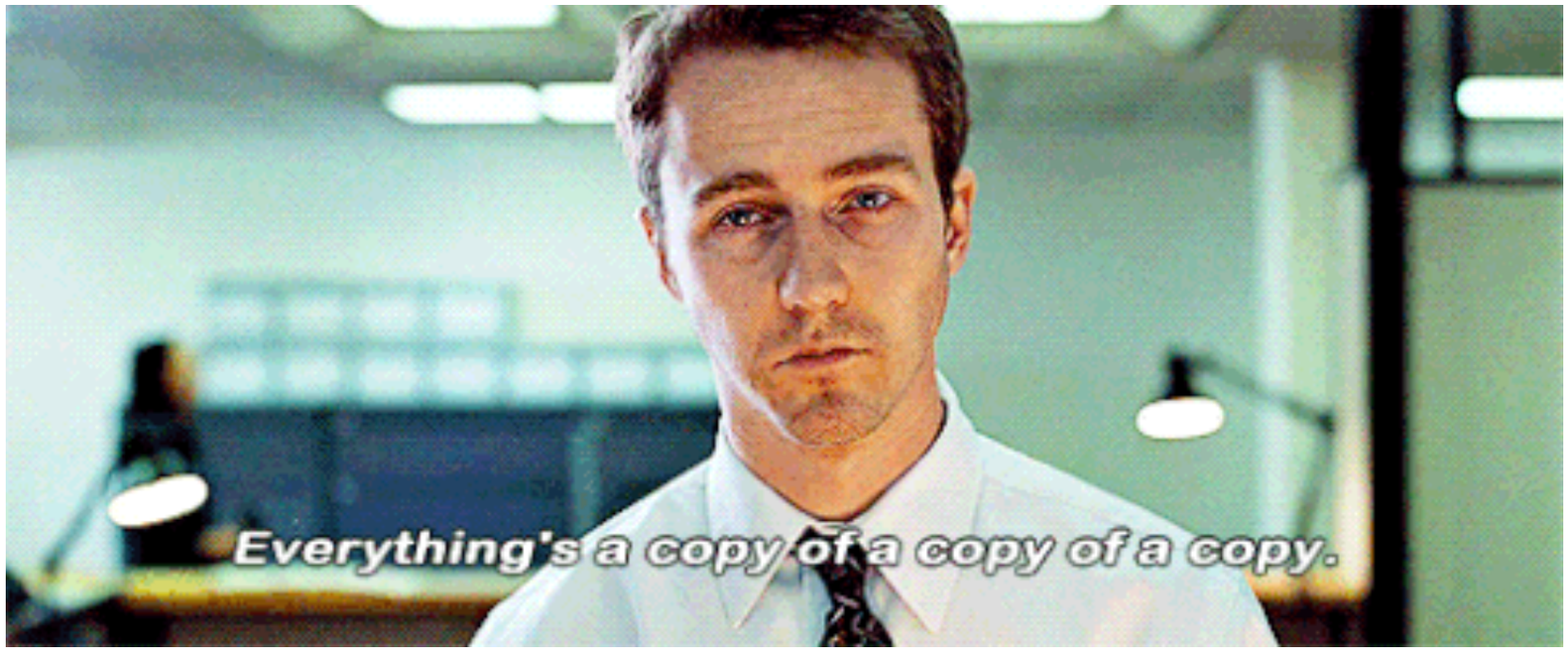
- Melhor organizações
- Ações relacionadas devem ficar juntas
- Melhor entendimento do código
- Menor possibilidade de erros

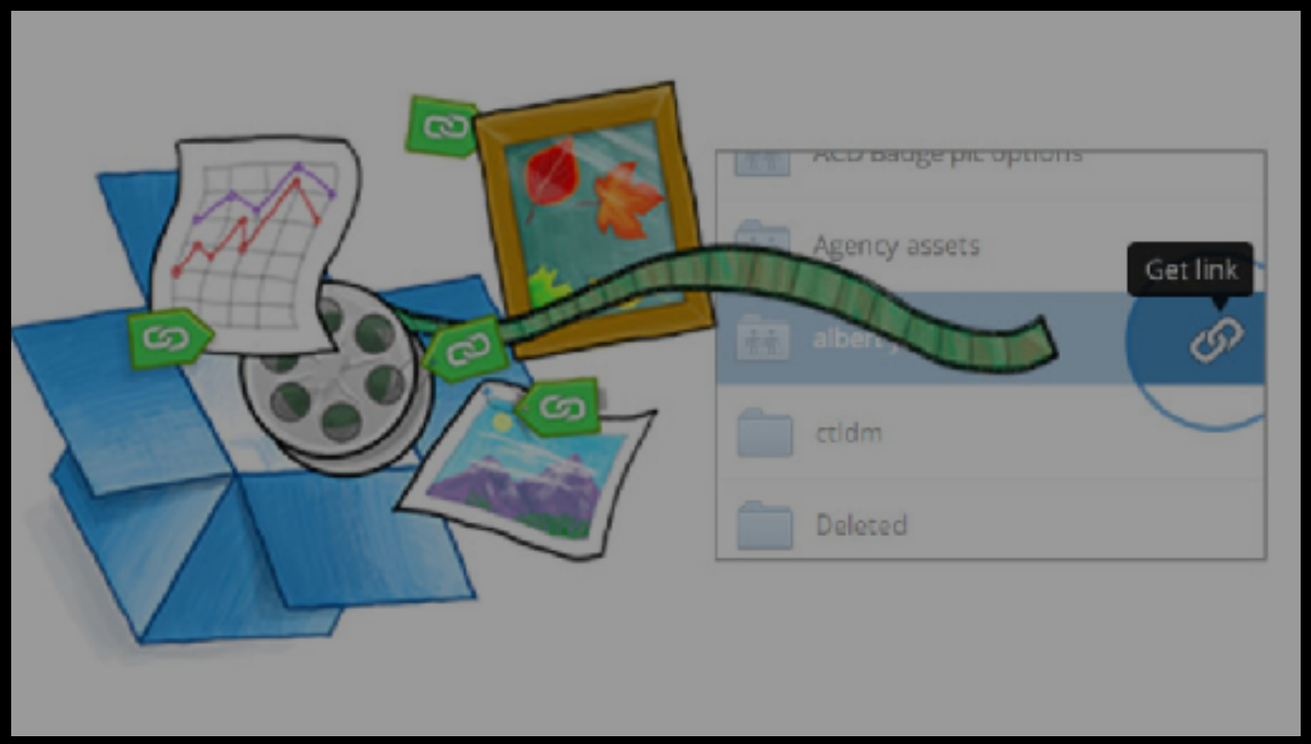
PASSAGEM DE DADOS



PASSAGEM POR VALOR

- Forma mais comum
- Proteção pelo Escopo
- Informação é copiada para a função de destino

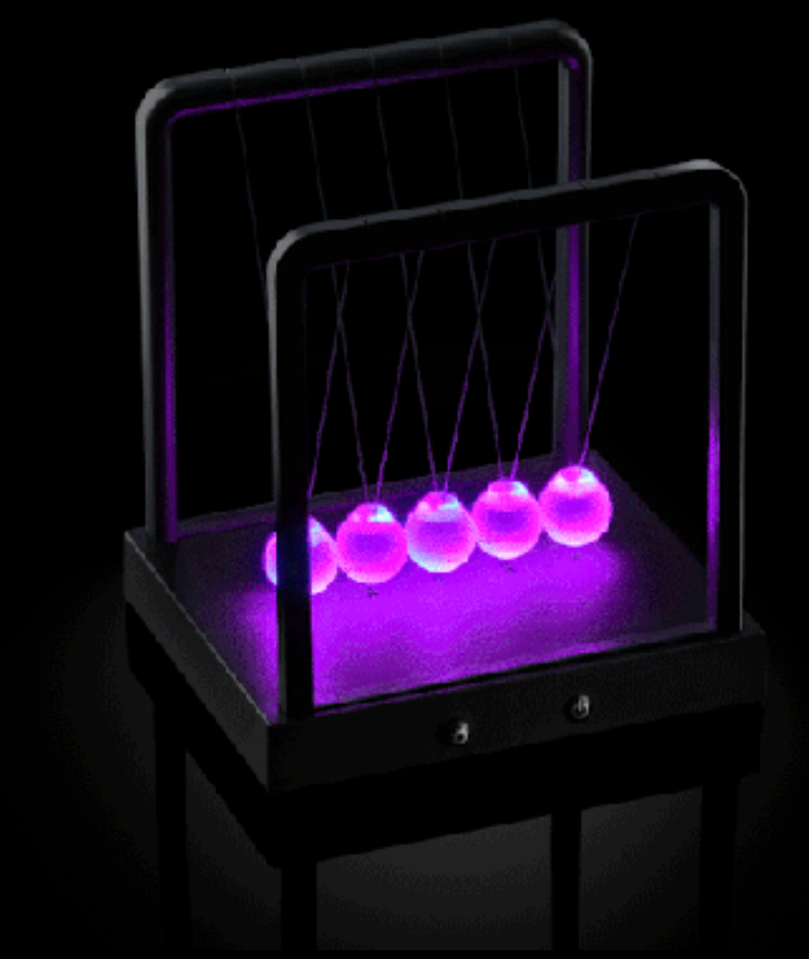




PASSAGEM POR REFERÊNCIA

- Forma mais arriscada de tratar os dados

- Permite “retornar” mais de 1 valor



- Informação é linkada para a nova função (Não existe copia aqui)

Let's CODE

ESTRUTURA DE DADOS

MODULARIZAÇÃO / FUNÇÕES

ESTRUTURAS REPETIÇÃO

ESTRUTURAS CONDICIONAIS

VARIÁVEIS / COMANDOS DE ENTRADA E SAÍDA

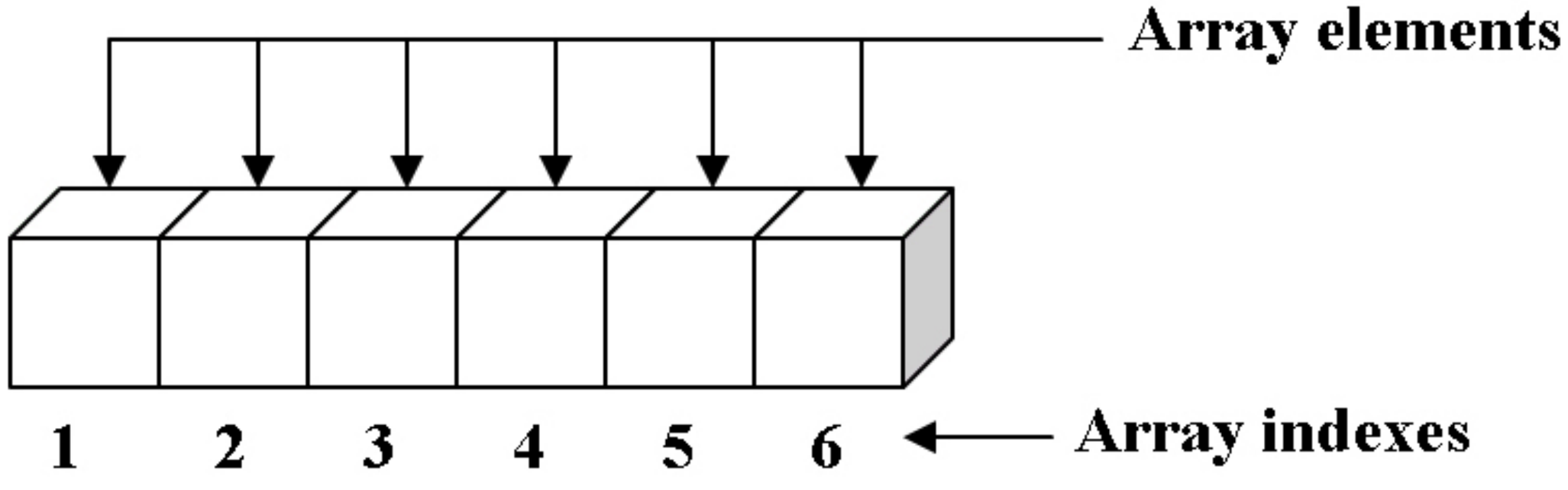
TIPOS DE DADOS

- Textual
  - Caracteres
  - Textos
- Números
  - Inteiros
  - Fracionados
- Booleanos
- **Estruturais**
- Tempo

VETORES E MATRIZES

- Vetores
  - Estruturas lineares
  - Podem ser compostos de todos os tipos básicos
  - **ARMAZENA APENAS 1 TIPO**
- Matrizes
  - Estruturas bidimensionais
  - Podem ser compostos de todos os tipos básicos
  - **ARMAZENA APENAS 1 TIPO**

VETORES E MATRIZES



One-dimensional array with six elements

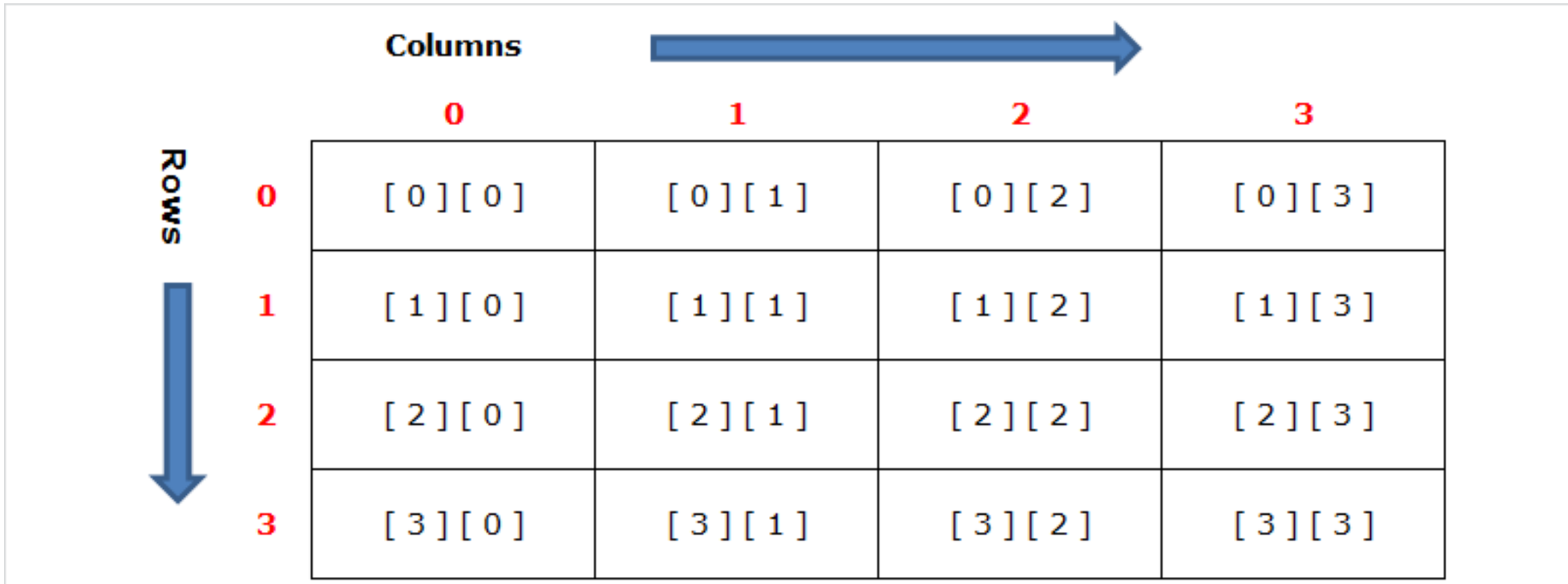
```
int[] stock = {12,8,34,19,26,31,-5,78};
```

	0	1	2	3	4	5	6	7
stock	12	8	34	19	26	31	-5	78

LET`S CODE

BORA CODA!!

VETORES E MATRIZES




```
string[,] MyArray = new string[3,3];
MyArray[0, 0] = "Bob";
MyArray[0, 1] = "Steven";
MyArray[0, 2] = "Jim";

MyArray[1, 0] = "Alice";
MyArray[1, 1] = "Jane";
MyArray[1, 2] = "Jill";

MyArray[2, 0] = "Tom";
MyArray[2, 1] = "Andrew";
MyArray[2, 2] = "Joe";
```

LET`S CODE

BORA CODA!!


ARQUIVOS
ESTRUTURA DE DADOS
MODULARIZAÇÃO / FUNÇÕES
ESTRUTURAS REPETIÇÃO
ESTRUTURAS CONDICIONAIS
VARIÁVEIS / COMANDOS DE ENTRADA E SAÍDA

ARQUIVOS

C#

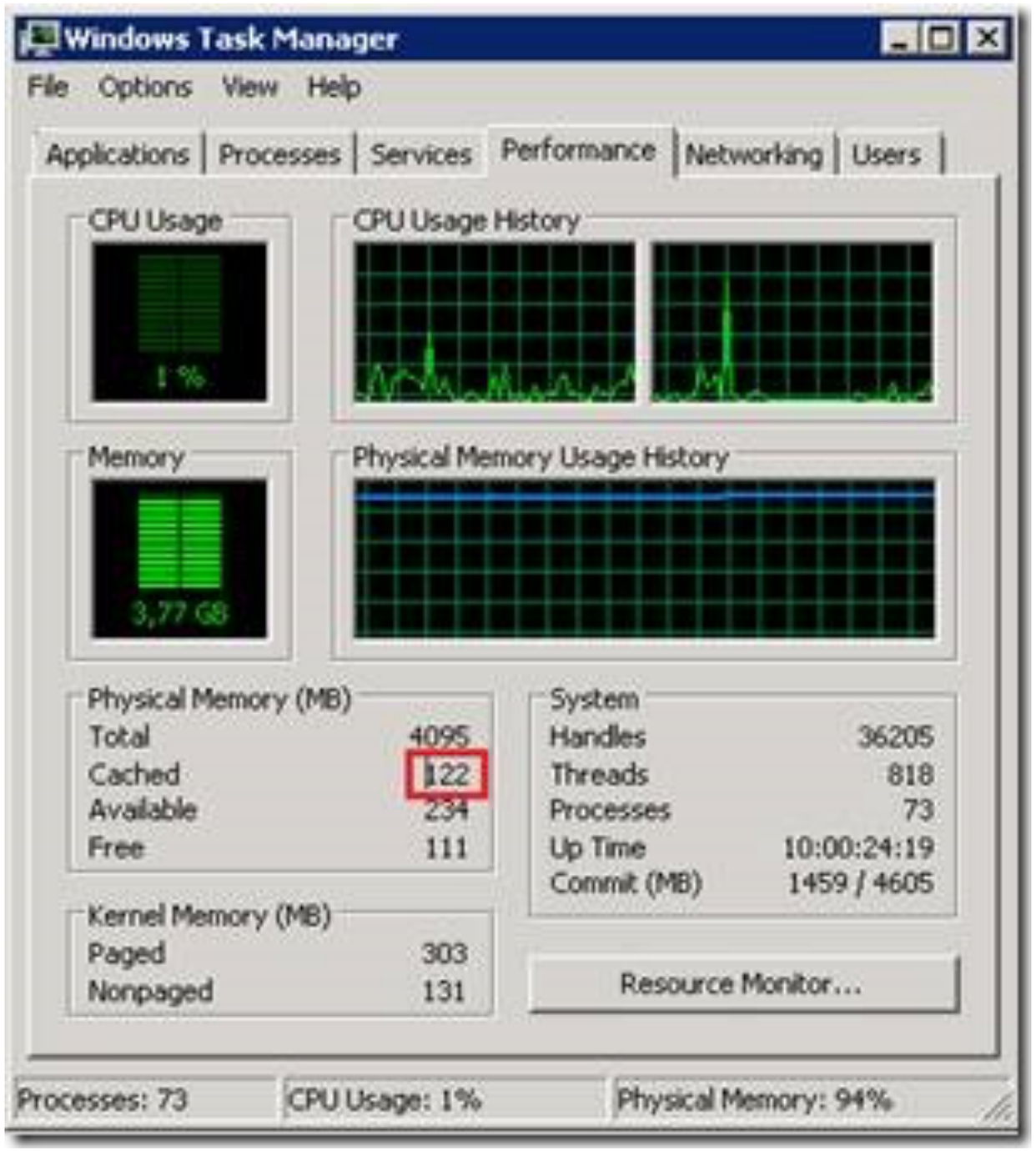
Copiar

```
using System;
using System.IO;
using System.Text;

class Test
{
    public static void Main()
    {
        string path = @"c:\temp\MyTest.txt";

        // This text is added only once to the file.
        if (!File.Exists(path))
        {
            // Create a file to write to.
            string createText = "Hello and Welcome" + Environment.NewLine;
            File.WriteAllText(path, createText);
        }

        // This text is always added, making the file longer over time
        // if it is not deleted.
        string appendText = "This is extra text" + Environment.NewLine;
        File.AppendAllText(path, appendText);

        // Open the file to read from.
        string readText = File.ReadAllText(path);
        Console.WriteLine(readText);
    }
}
```

File read and Write

docs.microsoft

ARQUIVOS


Windows Task Manager
File Options View Help
Applications Processes Services Performance Networking Users
CPU Usage
1 %
CPU Usage History
Memory
3,77 GB
Physical Memory Usage History
Physical Memory (MB)
Total 4095
Cached 122
Available 234
Free 111
System
Handles 36205
Threads 818
Processes 73
Up Time 10:00:24:19
Commit (MB) 1459 / 4605
Kernel Memory (MB)
Paged 303
Nonpaged 131
Resource Monitor...
Processes: 73
CPU Usage: 1%
Physical Memory: 94%

LET`S CODE

BORA CODA!!