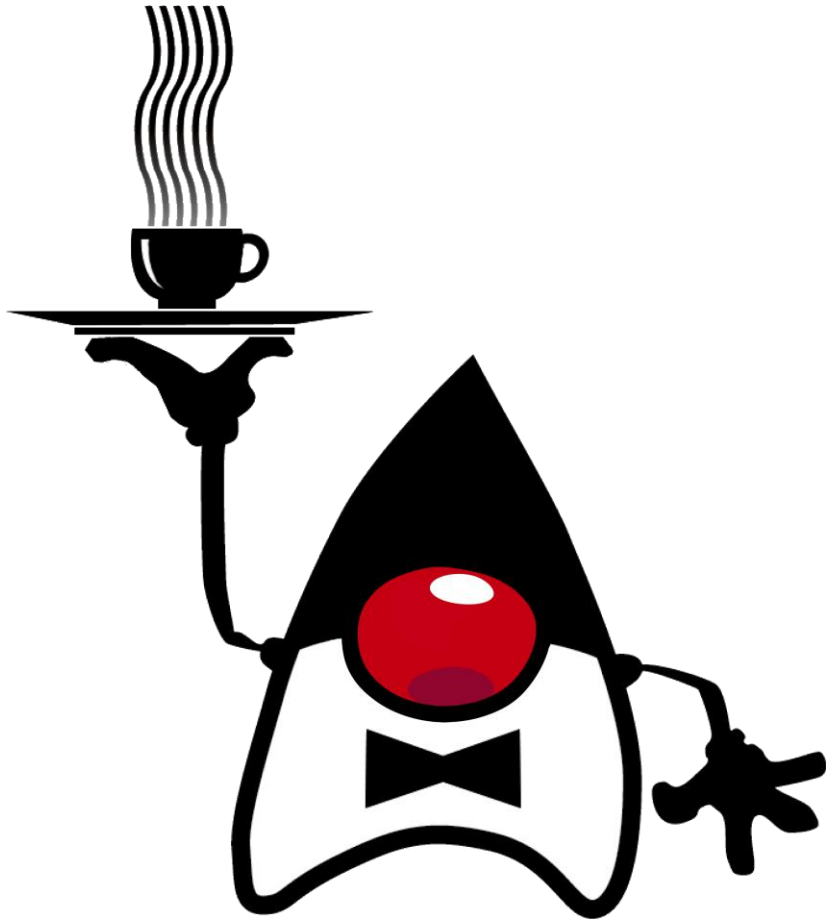
Ciência da Computação

Programação Orientada a Objetos
Prof. Luciano Rodrigo Ferretto

Vamos tomar um café???
Mas só se for um café “forte”, ou seja, um
Java Coffe



Linguagens Intepretadas e Compiladas

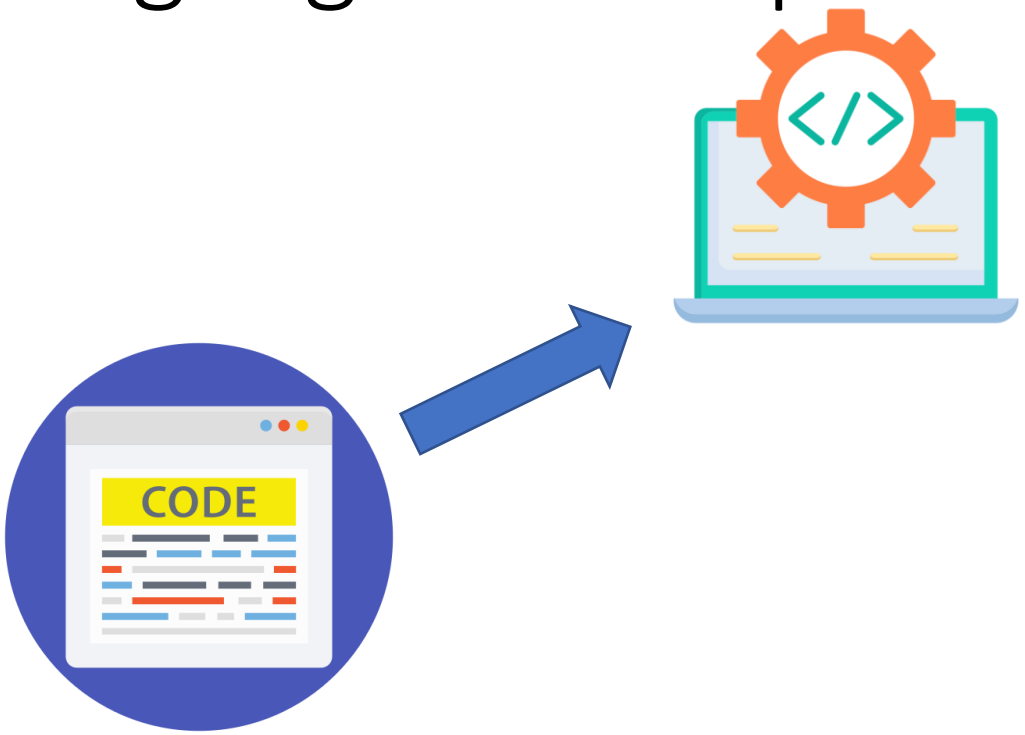
- **Linguagens interpretadas** são aquelas em que o código-fonte é executado linha por linha diretamente pelo interpretador, sem a necessidade de um processo prévio de compilação.
  - Exemplos: Python, JavaScript, PHP, Ruby, etc.
  - Linguagens interpretadas geralmente oferecem mais **portabilidade** e são mais fáceis de depurar
- **Linguagens compiladas** requerem um passo de compilação antes da execução. Durante esse processo, o código-fonte é traduzido para linguagem de máquina, produzindo um arquivo executável.
  - Exemplos: C, C++, Swift, Rust, etc.
  - Linguagens compiladas tendem a ser mais **rápidas e eficientes**, uma vez que o código é otimizado antes da execução, sem falar que evita-se muitos erros em tempo de **Execução**.

Ciência da Computação

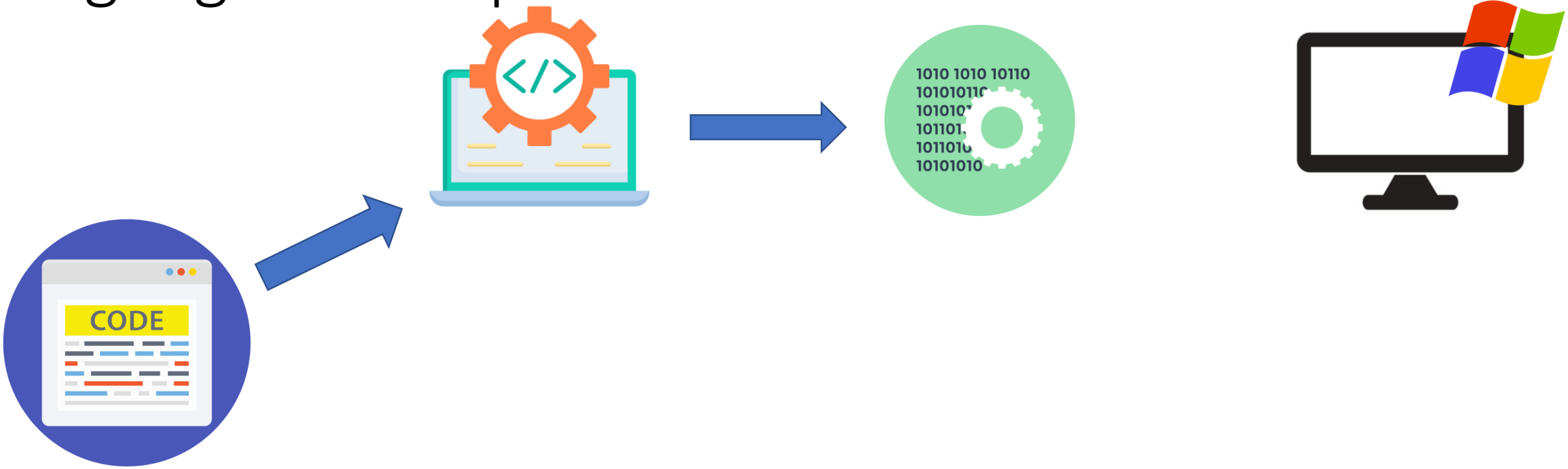
Linguagens Compiladas

Linguagens Compiladas

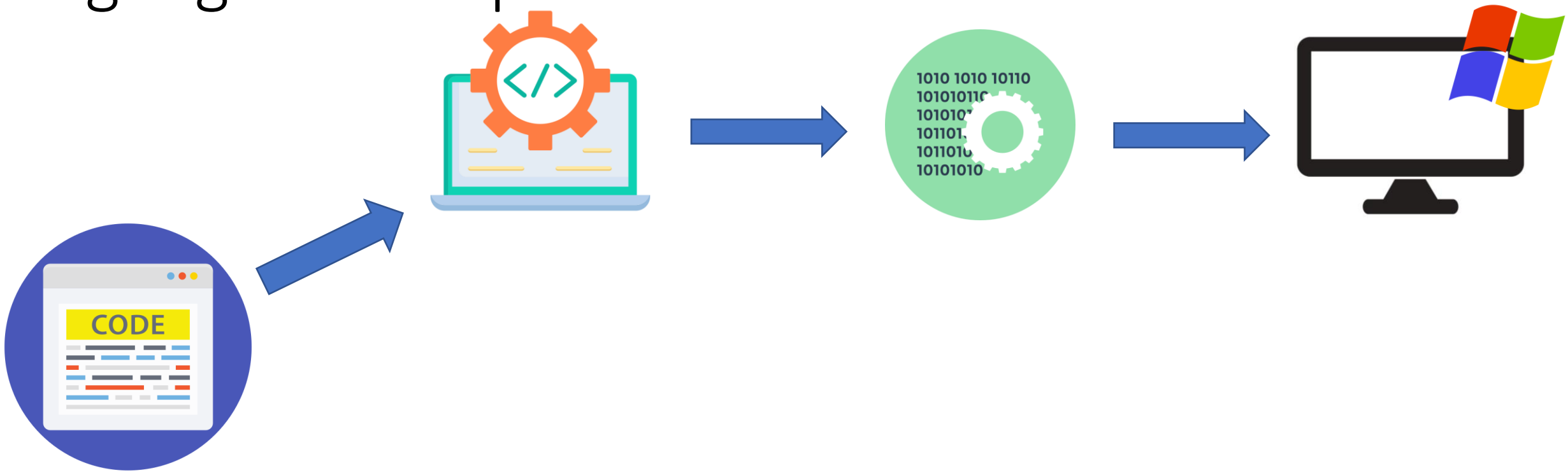
Linguagens Compiladas



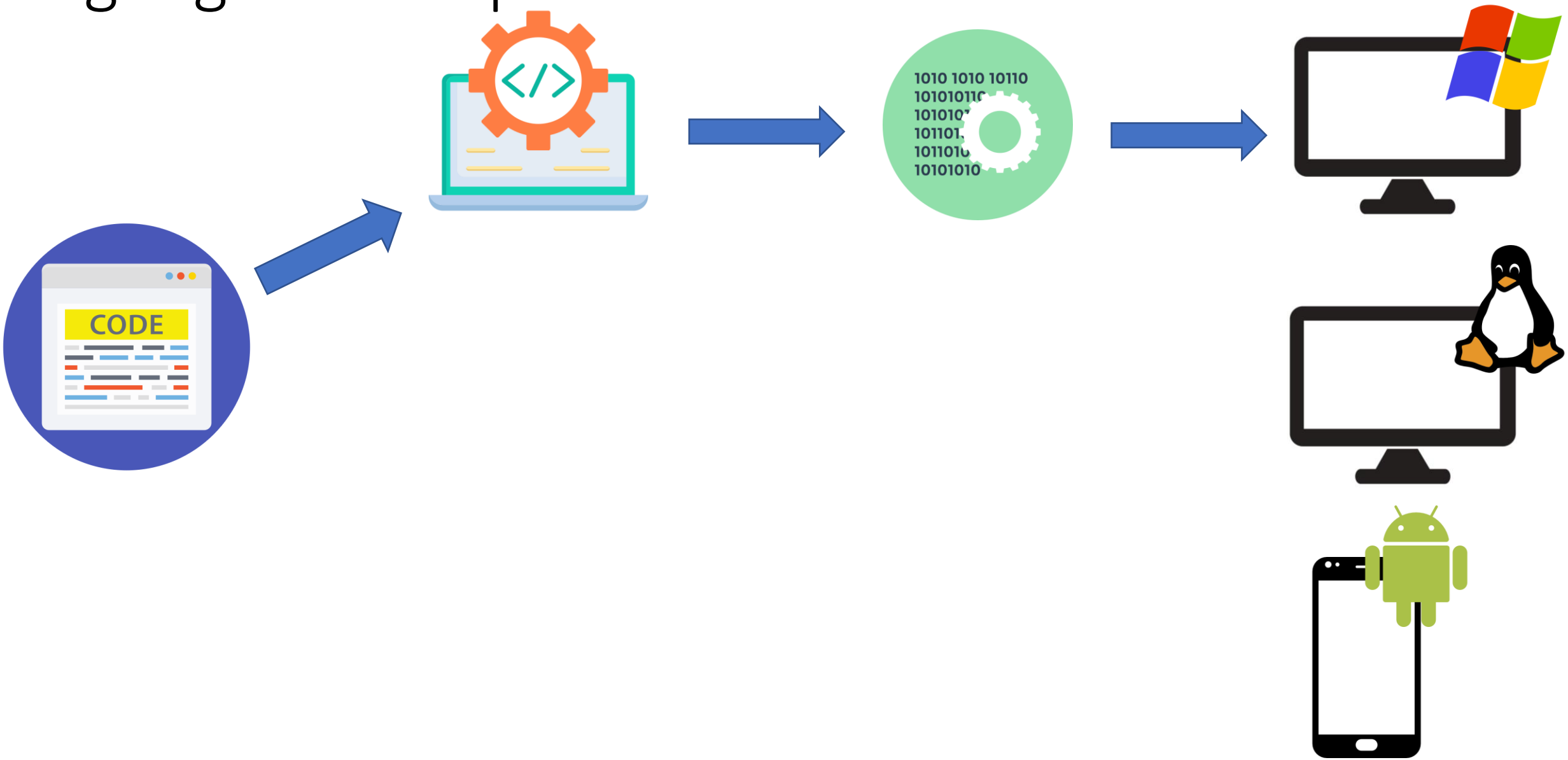
Linguagens Compiladas



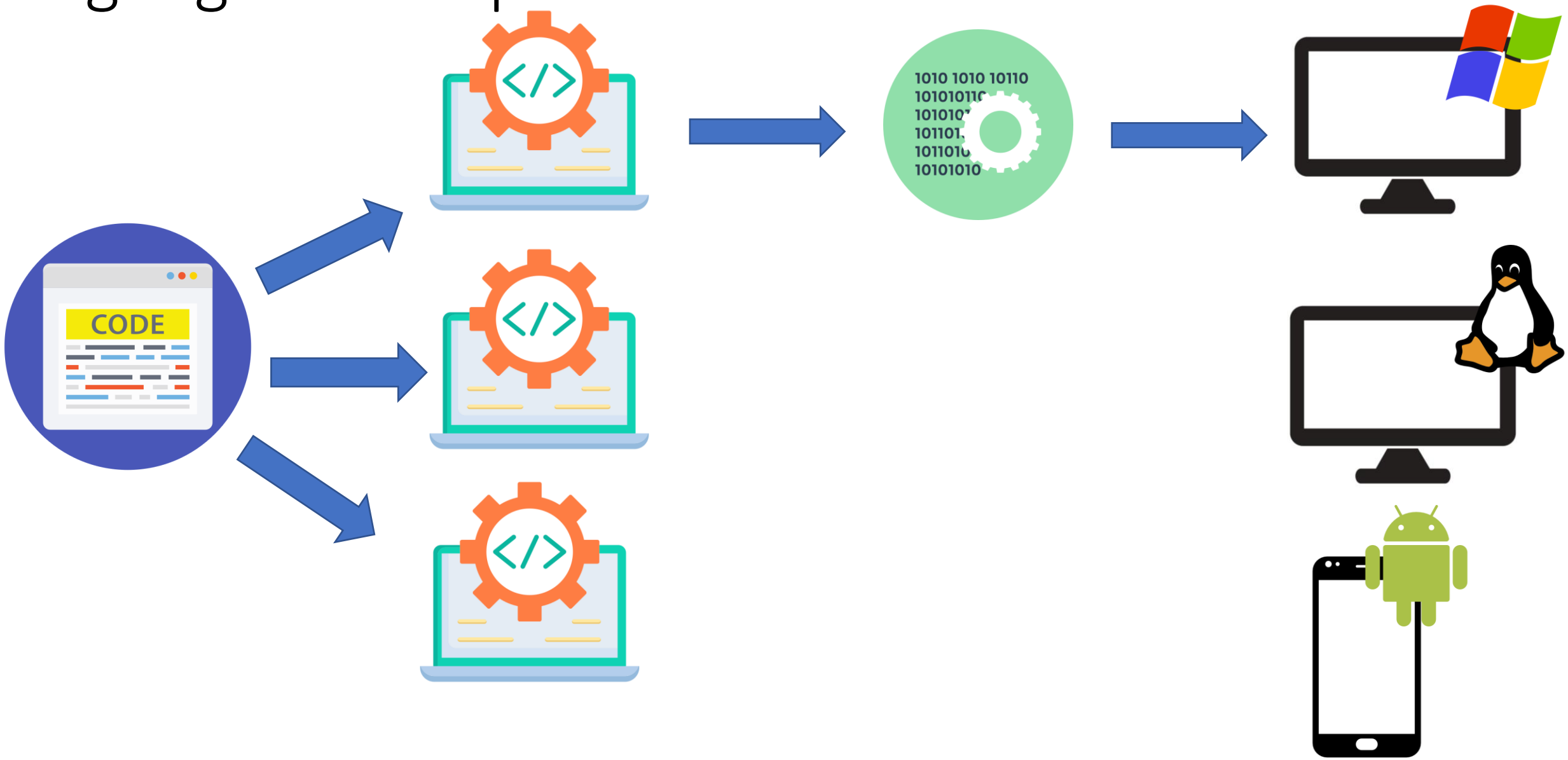
Linguagens Compiladas



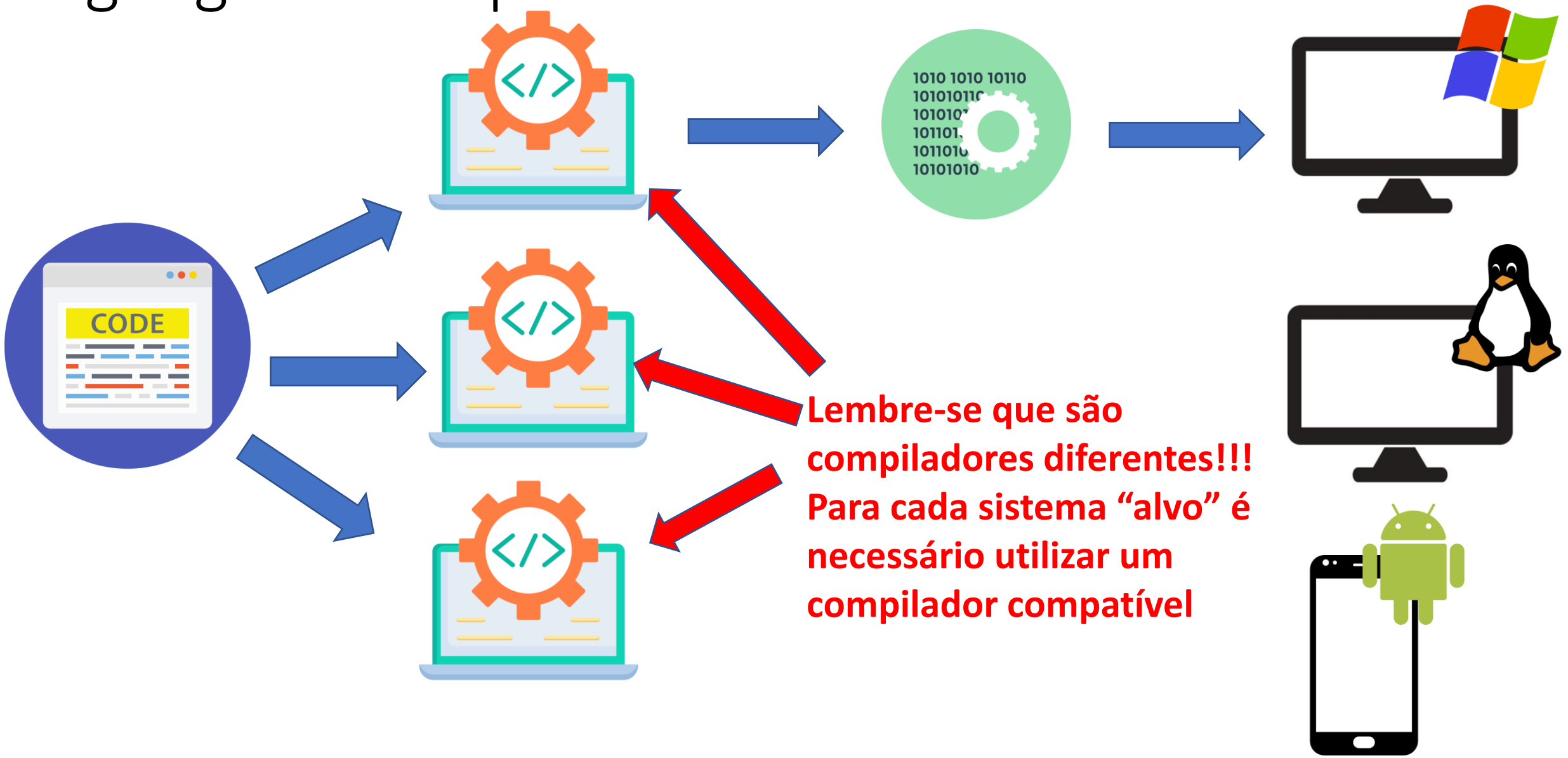
Linguagens Compiladas



Linguagens Compiladas

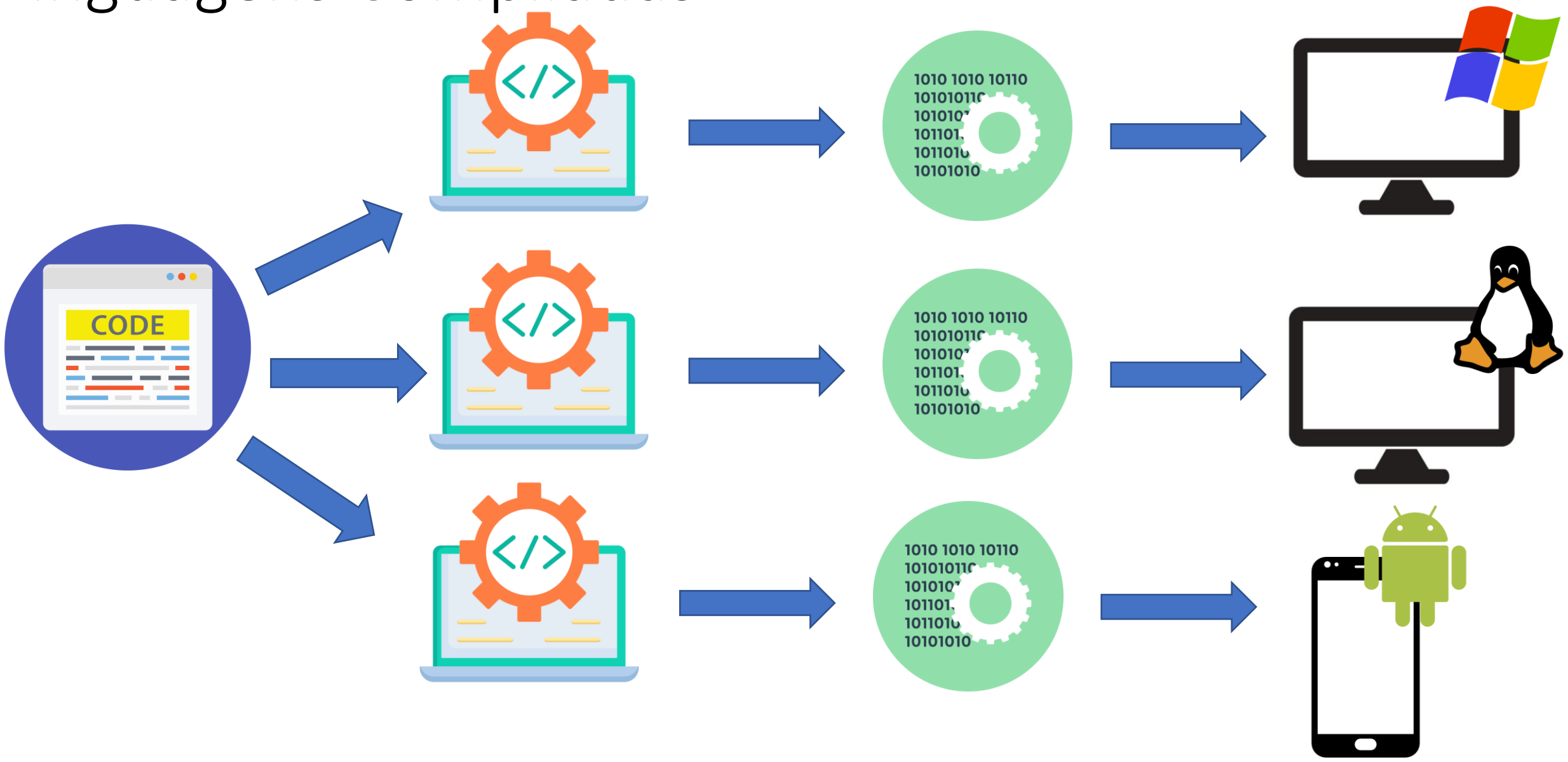


Linguagens Compiladas

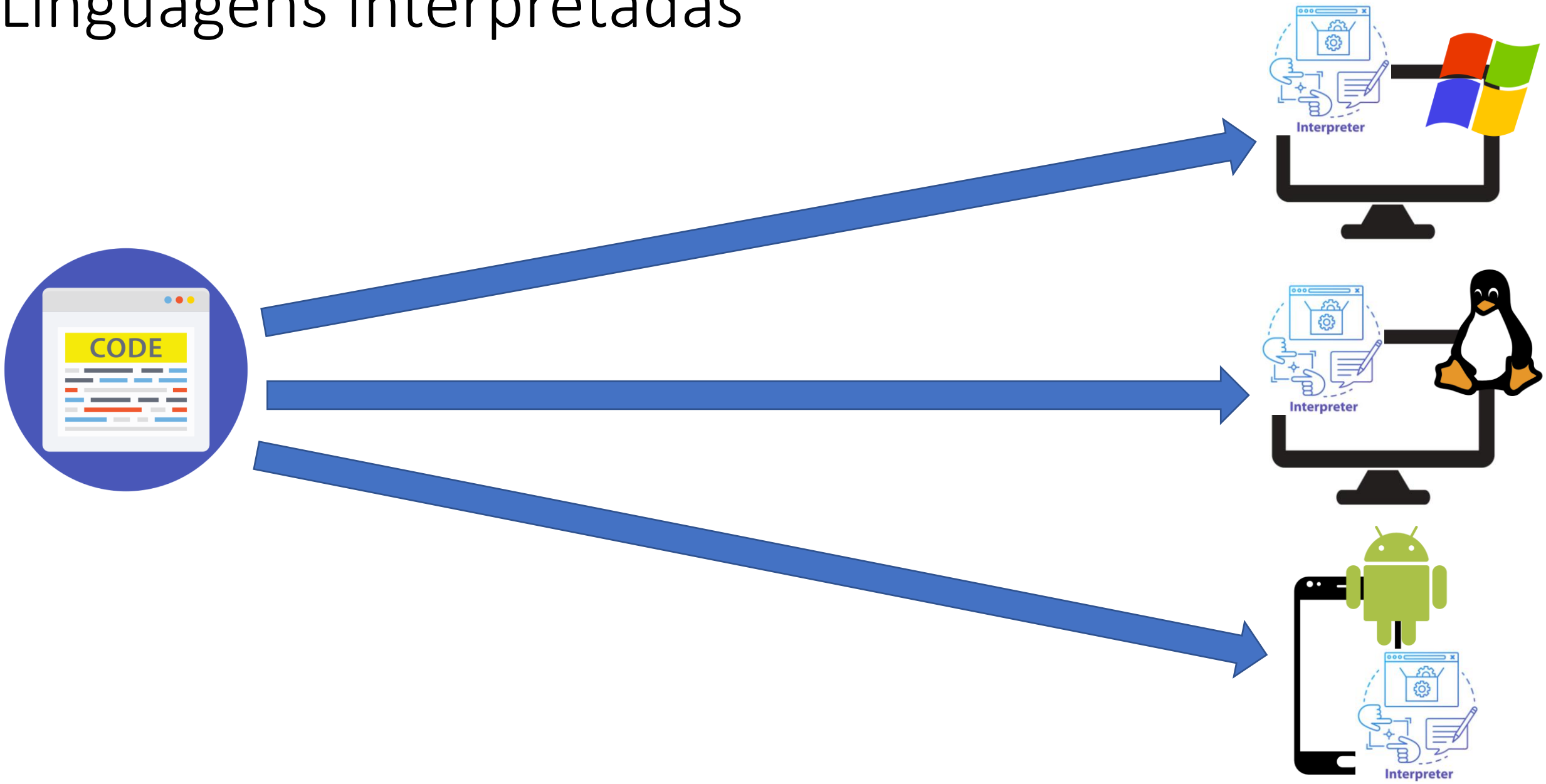


Lembre-se que são compiladores diferentes!!! Para cada sistema "alvo" é necessário utilizar um compilador compatível

Linguagens Compiladas



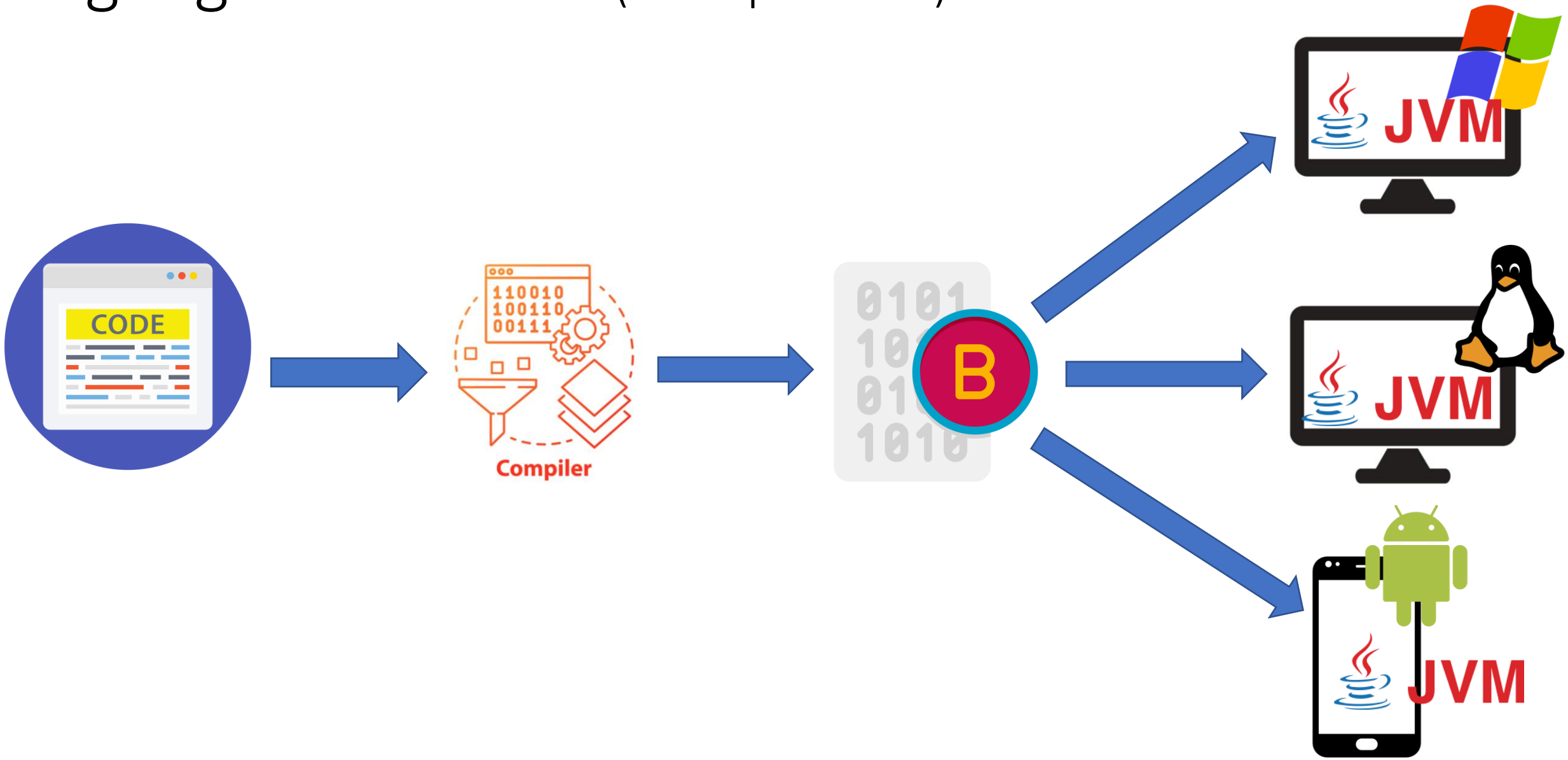
Linguagens Interpretadas



Mas e o Java???

- O Java é uma linguagem de programação que combina características de linguagem **interpretada e compilada**.
- O código-fonte Java é primeiro **compilado** pelo compilador Java (javac) em **bytecode** Java, que é uma representação intermediária independente de plataforma. Esse processo de compilação gera arquivos **.class** contendo o bytecode.
- Em seguida, o bytecode Java é **interpretado** pela máquina virtual Java (JVM) durante a execução do programa. O interpretador JVM lê o bytecode linha por linha e executa as ações apropriadas. Essa etapa de interpretação fornece a **portabilidade** do Java, pois o mesmo bytecode pode ser executado em qualquer plataforma que possua uma JVM compatível.

Linguagens Híbridas (exemplo: Java)



ByteCode – Funcionamento do Java

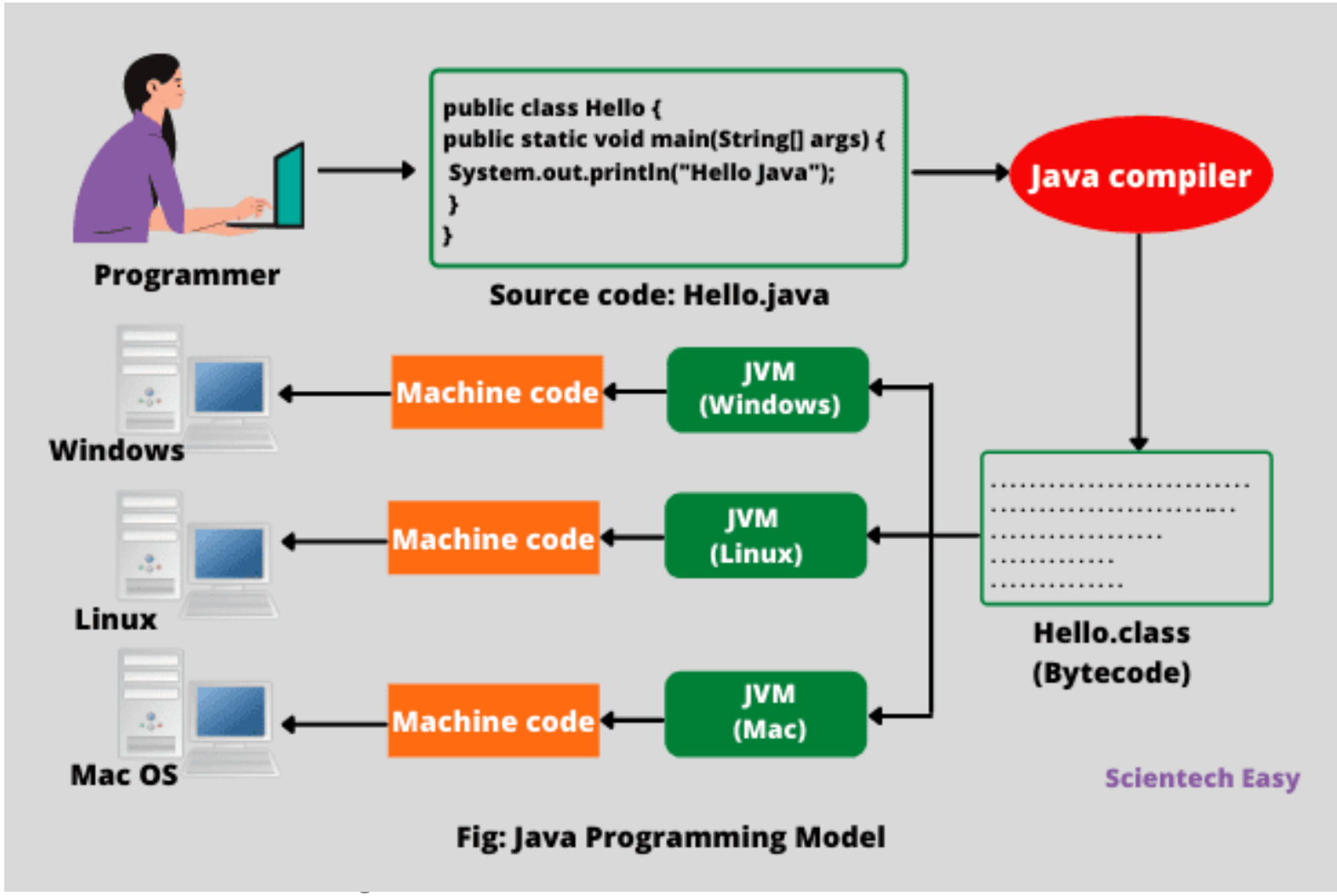


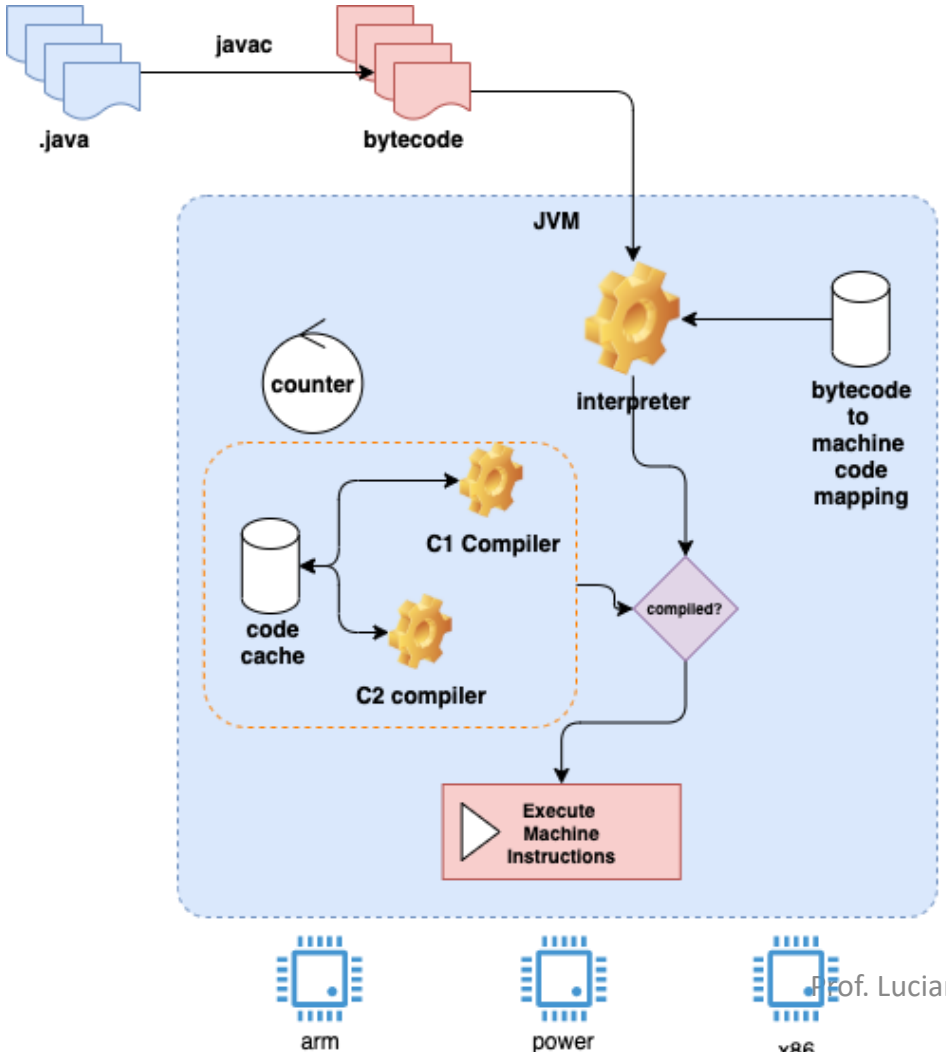
Fig: Java Programming Model

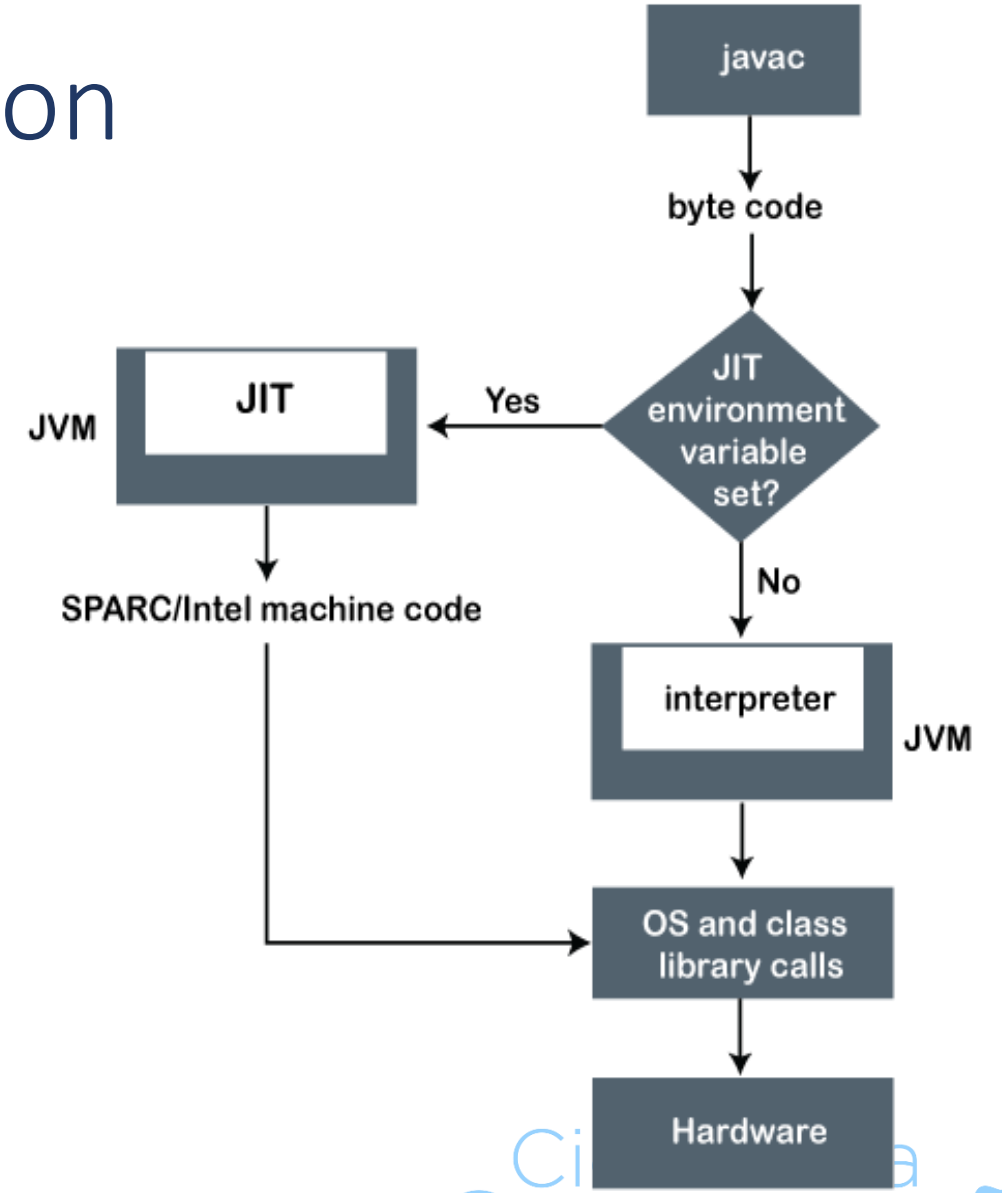
Mas e o Java???

- No entanto, o Java não para por aí. A JVM também inclui um **JIT (Just-In-Time) Compiler**. O JIT identifica partes críticas do código (com base no perfil de execução) e as **compila em código de máquina nativo** específico da arquitetura do processador em que a JVM está sendo executada.

- Portanto, podemos dizer que o Java **combina características de linguagem interpretada (portabilidade do bytecode) com características de linguagem compilada (otimização JIT)**, tornando-o uma linguagem híbrida que visa o equilíbrio entre portabilidade e desempenho.

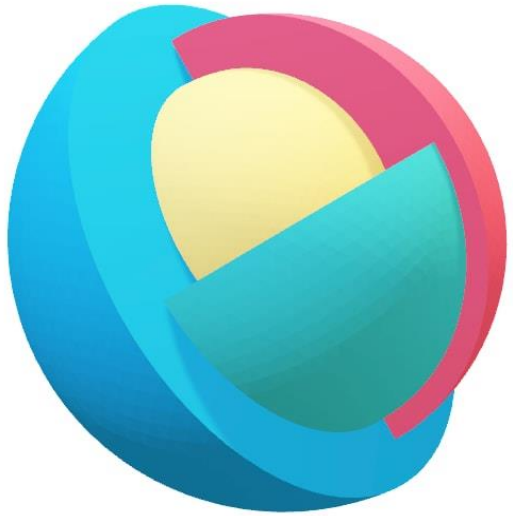
JIT – Jut-in-time compilation





JIT Compilation Process

JVM – JRE – JDK



JVM

Converts bytecode to machine-specific code.

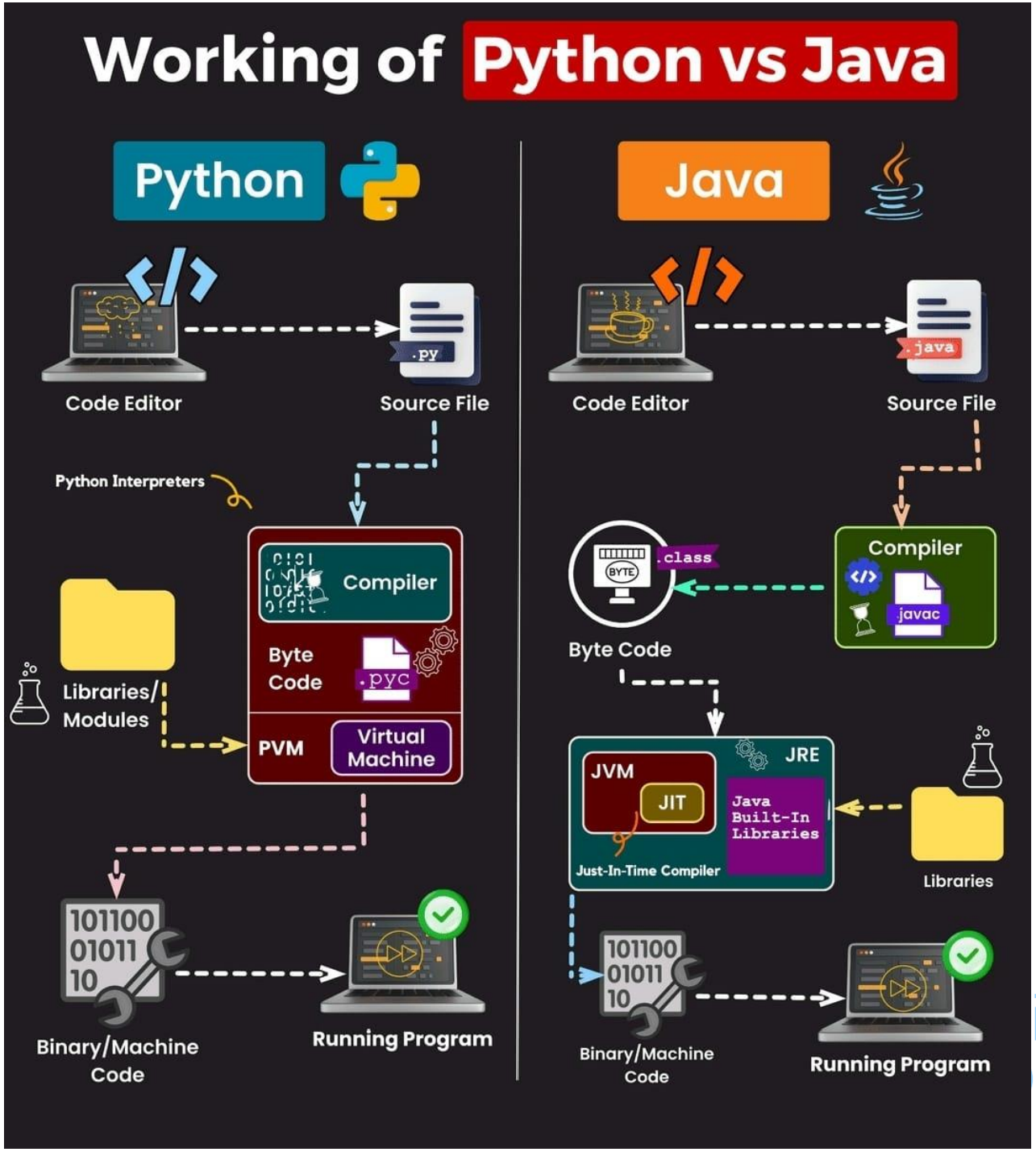
JRE

Executes Java programs.
Includes JVM.

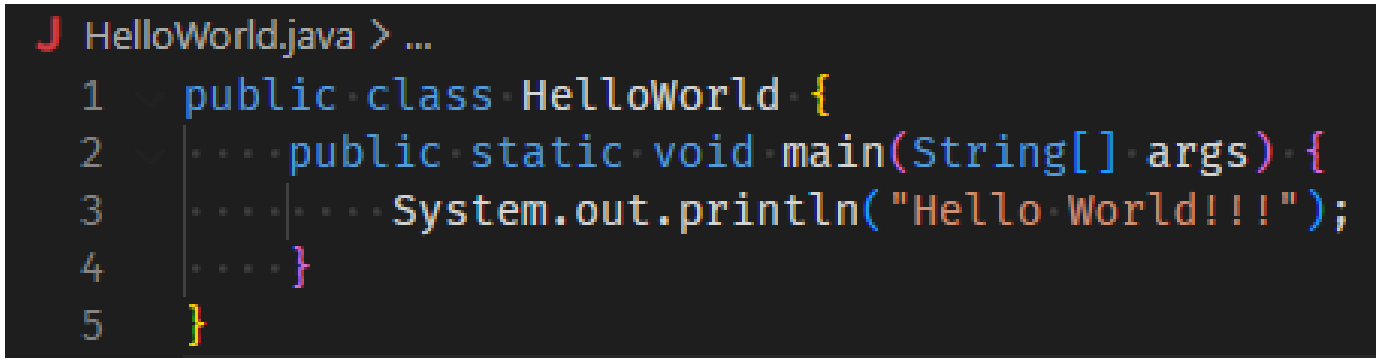
JDK

The development kit that includes JRE, a Java compiler, a debugger and more.

Python x Java

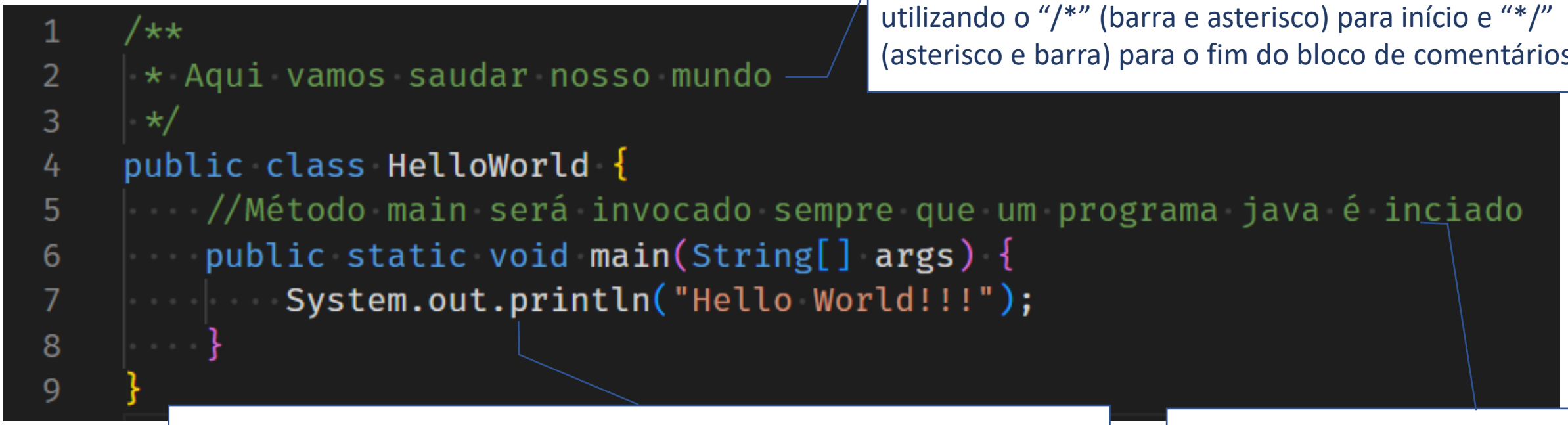


O Velho e bom Hello World!



Ciência da Computação

O Velho e bom Hello World!



Comentários com mais de uma linha podem ser feitos utilizando o "/*" (barra e asterisco) para início e "*/" (asterisco e barra) para o fim do bloco de comentários

Método **main** é aquele que será invocado assim que o programa iniciar. Neste nosso exemplo, este método apenas está escrevendo na tela a expressão "Hello World!!!".
Mais adiante teremos explicação melhor da biblioteca System

Comentários de única linha podem ser escritos com "//" (duas barras) no início

Ciência da Computação

O Velho e bom Hello World!

- No Java é utilizado as “chaves” => { } para definir o início e o fim de um bloco de instruções.
- Já o “ponto e vírgula” => ; é utilizado para definir o final de uma instrução.
- Dessa forma se você quiser você pode escrever todo seu código em uma linha só, porém, é claro que não é uma boa ideia.

```
1  /**
2   * Aqui vamos saudar nosso mundo
3   */public class HelloWorld {public static void main(String[] args) {System.out.println("Hello World!!!");}}
```

Ciência da Computação

Declaração de duas variáveis com os nomes “var1” e “var2” ambas do tipo Inteiro.
Tudo aquilo que vem após “//” (duas barras) é considerado comentário e não é interpretado pela JVM

Atribui o valor 512 à variável “var1” e imprime na tela o texto entre aspas concatenando o valor da variável

```java
public class Calcula {
    public static void main(String[] args) {
        int var1; // esta linha declara uma variável, porém não a inicializa.
        int var2 = 2; //esta linha declara e já inicializa atribuindo um valor.

        var1 = 512; // atribui um valor à variável (deve ser observado o tipo)
        System.out.println("var1 possui o valor: " + var1);

        var2 = var1 / var2;
        System.out.println("var2 possui o valor: " + var2);
        System.out.println("O calculo foi: " + var1 + "/" + var2);
    }
}
```

Realiza a divisão do valor da variável “var1” por 2 através do operador “/” (barra) e atribui a variável “var2”. Após imprime na tela os textos com os valores das variáveis

Ciência da Computação

Aqui temos o exemplo de declaração de uma variável com o tipo **double** que aceita números racionais. No caso da atribuição de valores deve-se atentar a utilizar o padrão americano, ou seja, “.” (ponto) ao no lugar de “,” (vírgula)

```java
public class MyExample {
    public static void main(String[] args) {
        int var1 = 10;
        double var2 = 10.0; // aqui pode ser atribuído com casas decimais usando o ponto (.)

        System.out.println("var1 / 4 = " + var1 / 4);
        System.out.println("var2 / 4 = " + var2 / 4);
    }
}
```

A primeira linha irá retornar o cálculo utilizando a variável do tipo inteiro, sendo assim seu resultado será somente a parte inteira da divisão, ou seja, **2** (dois).
A segunda linha irá retornar a divisão em formato decimal, ou seja, **2,5** (dois e meio)

```java
public class MyExample {
    public static void main(String[] args) {
int var1 = 10;    double var2 = 10.0; // aqui pode ser atribuído com casas decimais
        System.out.println("var1 / 4 = " + var1 / 4);
System.out.println("var2 / 4 = " + var2 / 4);
    }}
```

Na linguagem Java o que determina o início e fim dos comandos são as **chaves** "{}" e o **ponto e vírgula** ";".
Desta forma, não importa se escrevemos o código de forma organizada ou tudo em uma única linha.
Mas é claro que para dar manutenção e entender o código é bom manter o mínimo de organização.

Dica: No VSCode utilize o atalho ALT + SHIFT + F para organizar o código automaticamente

Palavras Chave em Java

Tabela 1-1 **As palavras-chave Java**

abstract	assert	boolean	break	byte	case
catch	char	class	const	continue	default
do	double	else	enum	extends	final
finally	float	for	goto	if	implements
import	instanceof	int	interface	long	native
new	package	private	protected	public	return
short	static	strictfp	super	switch	synchronized
this	throw	throws	transient	try	void
volatile	while				

Além das palavras-chave, Java reserva as palavras a seguir: **true**, **false** e **null**. São valores definidos pela linguagem. Você não pode usar essas palavras em nomes de variáveis, classes e assim por diante.

Identificadores em Java

Em Java, um identificador é o nome dado a um método, uma variável ou qualquer outro item definido pelo usuário. Os identificadores podem ter de um a vários caracteres. Os nomes de variável podem começar com qualquer letra do alfabeto, um sublinhado ou um cifrão. Em seguida pode haver uma letra, um dígito, um cifrão ou um sublinhado. O sublinhado pode ser usado para melhorar a legibilidade do nome da variável, como em **line_count**. As letras maiúsculas e minúsculas são diferentes, ou seja, para Java, **myvar** e **MyVar** são nomes diferentes. Aqui estão alguns exemplos de identificadores aceitáveis:

Test	x	y2	MaxLoad
$up	_top	my_var	sample23

Lembre-se, você não pode iniciar um identificador com um dígito. Logo, **12x** é um identificador inválido, por exemplo.

Tipo de Dados no Java

- Java é uma Linguagem de Programação **Fortemente Tipada**.
- Todas as operações têm a compatibilidade de seus tipos verificadas pelo compilador;
- Evita a ocorrência de muitos erros em tempo de execução e aumenta a confiabilidade;
- **NÃO** existe o conceito de variável sem tipo;
- O tipo determina quais operações podem ou não podem ser aplicadas a uma variável;
- Existem os tipos Orientados a Objetos que são definidos por classe, e os não Orientados a Objetos, também conhecidos como Tipos Primitivos. Nessa aula vamos estudar estes últimos;

Ciência da Computação

Tipo de Dados **Primitivos** no Java

- São valores binários comuns. Não são objetos no sentido de OO;
- Todos os outros tipos de dados são construídos a partir dos tipos primitivos;
- Java é inflexível no sentido das operações suportadas por cada um dos tipos, isso garante a interoperabilidade;

Tabela 2-1 Tipos de dados primitivos internos de Java

Tipo	Significado
boolean	Representa os valores verdadeiro/falso
byte	Inteiro de 8 bits
char	Caractere
double	Ponto flutuante de precisão dupla
float	Ponto flutuante de precisão simples
int	Inteiro
long	Inteiro longo
short	Inteiro curto

Ciência da Computação

Tipo de Dados **INTEIROS** no Java

- Os mais usado é o **int**, pois atende na maioria dos casos;
  - Mesmo em situações onde o intervalo não precisa ser muito como no caso de índices de arrays, controle de laços, etc.

Tipo	Tamanho em bits	Intervalo
byte	8	–128 a 127
short	16	–32.768 a 32.767
int	32	–2.147.483.648 a 2.147.483.647
long	64	–9.223.372.036.854.775.808 a 9.223.372.036.854.775.807

Tipo de Dados de **Ponto Flutuante** no Java

- No Java temos dois tipos de dados de ponto flutuante:

- **Float:** possui 32 bits de tamanho e é considerado de precisão simples.
- **Double:** possui 64 bits de tamanho e é considerado de precisão dupla.

- O **double** é o mais usado por oferecer uma precisão melhor e também porque todas as funções matemáticas da biblioteca de classes Java usam valores double.
  - Por exemplo o método **sqrt()** da classe **Math**. Ele retorna um valor do tipo **double** que é a raiz quadrada de seu argumento também **double**.

Tipo de Dados de **Caracteres** no Java

- O Java usa Unicode para definir o conjunto de caracteres possíveis. Diferente de muitas outras linguagens que utilizam caracteres de 8 bits.
- O Unicode define um conjunto que pode representar todos os caracteres encontrados em todos os idiomas.
- Em Java o tipo de dados **char** é de 16 bits o que permite um amplitude de 0 até 65.536, diferente de 8 bits.

- **Char** é um tipo de 16 bits sem sinal, ou seja, podemos trata-lo aritmeticamente.

Tipo de Dados **de Caracteres** no Java

```
class ChartAritimetico {
    public static void main(String args[]){
        char ch;
        ch = 'X';
        System.out.println("ch = " + ch);
        ch++;
        System.out.println("ch = " + ch);
        ch = 90;
        System.out.println("ch = " + ch);
    }
}
```

A saída deste algoritmo seria:

ch = X
ch = Y
ch = Z

Ciência da Computação

Tipo de Dados **booleano** no Java

- O Java utiliza o tipo **boolean** para representar os valores verdadeiro/falso.
- Para definir os valores o Java utiliza as palavras chaves **true** e **false.**

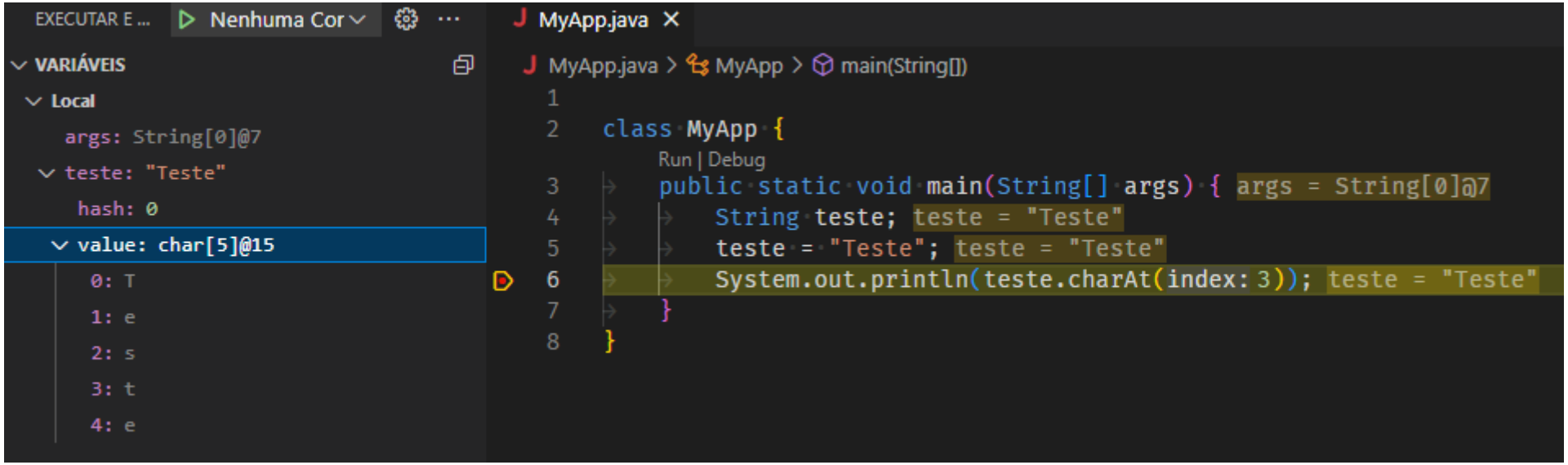
```
// Demonstra valores booleanos.
class BoolDemo {
  public static void main(String args[]) {
    boolean b;

    b = false;
    System.out.println("b is " + b);
    b = true;
    System.out.println("b is " + b);

    // um valor booleano pode controlar a instrução if
    if(b) System.out.println("This is executed.");

    b = false;
    if(b) System.out.println("This is not executed.");

    // o resultado de um operador relacional é um valor booleano
    System.out.println("10 > 9 is " + (10 > 9));
  }
}
```

A saída gerada por esse programa é mostrada aqui:

```
b is false
b is true
This is executed.
10 > 9 is true
```

Tipo de Dados **String** no Java

- **String** no Java nada mais é do que um array de **char**.
- Quando declaramos um atributo e atribuímos um valor, internamente é criado um array de char.
- Desta forma um String com valor “Teste” é um array:
  - [‘T’,’e’,’s’,’t’,’e’].



Sequências de escape de caracteres

Tabela 2-2 Sequências de escape de caracteres

Sequência de escape	Descrição
\'	Aspas simples
\"	Aspas duplas
\\	Barra invertida
\r	Retorno de carro
\n	Nova linha
\f	Avanço de página
\t	Tabulação horizontal
\b	Retrocesso
ddd	Constante octal (onde *ddd* é uma constante octal)
\u*xxxx*	Constante hexadecimal (onde *xxxx* é uma constante hexadecimal)

Operadores aritméticos

Operador	Significado
+	Adição (também mais unário)
–	Subtração (também menos unário)
*	Multiplicação
/	Divisão
%	Módulo
++	Incremento
– –	Decremento

Operadores relacionais

- O resultado destes operadores sempre será um valor **boolean**.
- Em Java todos os objetos podem ser comparados com os operadores **igual (==)** e **diferente (!=)**.
- Todos os operadores relacionais podem ser aplicados aos tipos numéricos e char.
- **Boolean** só pode ser comparado quanto a igualdade ou diferença.
  - **true > false // isso não tem significado em Java**

Operador	Significado
= =	Igual a
!=	Diferente de
>	Maior que
<	Menor que
>=	Maior ou igual a
<=	Menor ou igual a

Ciência da Computação

Operadores lógicos

- O resultado destes operadores sempre será um valor **boolean** e seus operandos também deverão ser do tipo **boolean**.

Operador	Significado
&	AND
|	OR
^	XOR (exclusive OR)
||	OR de curto-circuito
&&	AND de curto-circuito
!	NOT

p	q	p & q	p | q	p ^ q	!p
Falso	Falso	Falso	Falso	Falso	Verdadeiro
Verdadeiro	Falso	Falso	Verdadeiro	Verdadeiro	Falso
Falso	Verdadeiro	Falso	Verdadeiro	Verdadeiro	Verdadeiro
Verdadeiro	Verdadeiro	Verdadeiro	Verdadeiro	Falso	Falso

Operadores lógicos

Operador	Significado
&	AND
|	OR
^	XOR (exclusive OR)
||	OR de curto-circuito
&&	AND de curto-circuito
!	NOT

Operadores lógicos de curto-circuito

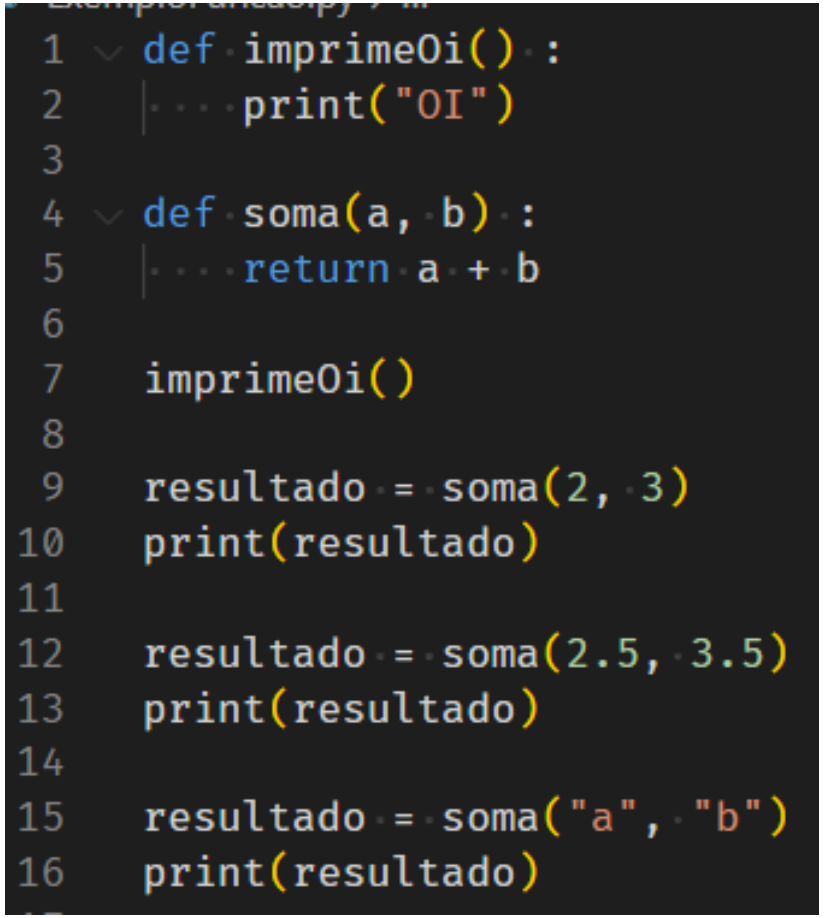
Java fornece versões especiais de *curto-circuito* de seus operadores lógicos AND e OR que podem ser usadas para produzir código mais eficiente. Para entender o porquê, considere o seguinte. Em uma operação AND, se o primeiro operando for falso, o resultado será falso não importando o valor do segundo operando. Em uma operação OR, se o primeiro operando for verdadeiro, o resultado da operação será verdadeiro não importando o valor do segundo operando. Logo, nesses dois casos, não há necessidade de avaliar o segundo operando. Quando não avaliamos o segundo operando, economizamos tempo e um código mais eficiente é produzido.

O operador AND de curto-circuito é **&&** e o operador OR de curto-circuito é **||**. Seus equivalentes comuns são **&** e **|**. A única diferença entre as versões comum e de curto-circuito é que a versão comum sempre avalia cada operando e a versão de curto-circuito só avalia o segundo operando quando necessário.

```java
public class CurtoCircuito {
    public static void main(String[] args) {
        int num1 = 10;
        int num2 = 2;

        if (num2 > 0 && num1 % num2 == 0) {
            System.out.println(num2 + " é fator de " + num1);
        }

        num2 = 0;
        /*Neste exemplo não ocorrerá erro, pois com o operador
        * de curto-circuito impedirá que a segunda condição,
        * que envole uma divisão, seja verificada.
        */
        if (num2 > 0 && num1 % num2 == 0) {
            System.out.println(num2 + " é fator de " + num1);
        }
        /*Neste exemplo teremos um erro em tempo de execução
        * Pois a segunda condição será verificada independente
        * do resultado da primeira.
        */
        if (num2 > 0 && num1 % num2 == 0) {
            System.out.println(num2 + " é fator de " + num1);
        }
    }
}
```

Ciência da
Computação

Funões em Python - Relembrando



```python
def imprimeOi() :
    print("OI")

def soma(a, b) :
    return a + b

imprimeOi()

resultado = soma(2, 3)
print(resultado)

resultado = soma(2.5, 3.5)
print(resultado)

resultado = soma("a", "b")
print(resultado)
```

Ciência da Computação

Métodos em Java (Funções)

```java
public class ExemploMetodo {
    /*a palavra chave "static" serve para podermos
     * invocar (chamar) o método sem uma instância
     * da classe
     */
    static void imprimeOi() { //"void" significa que o método NÃO terá retorno
        System.out.println("OI");
    }
    static int soma(int a, int b) {//os parâmetros de entrada devem ser tipados
        return a + b;
    }
    /*Se os parâmetros de entrada são diferentes
     * os métodos podem ter o mesmo nome
     * Isso se chama de SOBRECARGA DE MÉTODOS
     */
    static double soma(double a, double b) {
        return a + b;
    }
    /*Como a linguagem é compilada, é possível invocar (chamar)
     * os métodos em qualquer parte do código
     */
    public static void main(String[] args) {
        imprimeOi();

        int resultadoInt = soma(2, 3);
        System.out.println(resultadoInt);

        double resultadoDouble = soma(2.5, 3.5);
        System.out.println(resultadoDouble);
    }
}
```

Biblioteca System (java.lang.System)

Esta classe é mais conhecida pelos seus famosos métodos de entrada e saída de dados, como é o caso do tão famoso "System.out.println()", recurso tão utilizado que o próprio Eclipse possui um atalho para sua criação, basta digitar: **syso + CTRL ESPAÇO,** automaticamente o Eclipse cria o comando "System.out.println()".

A Classe System

A classe System não pode ser estendida pois seus métodos são estáticos, sem a necessidade de criação de uma instância em memória. Além disso, esta classe tem 3 variáveis que são:

- err: Sendo este do tipo PrintStream, armazena a saída padrão para mensagens de erro do sistema.
- In: Sendo este do tipo InputStream, armazena a entrada padrão do sistema, que pode ser, por exemplo, um teclado ou um arquivo de texto (mostraremos mais a frente como funciona).
- out: Sendo do mesmo tipo da variável "err" (PrintStream), esta variável armazena a saída padrão de mensagens da aplicação.

https://www.devmedia.com.br/destrinchando-a-classe-java-lang-system/29808

Classe Scanner (Package java.util)

A Classe Scanner é tem como objetivo separar a entrada dos textos em blocos e entrega-los em formatos de tipos primitivos.
Para podermos utilizá-la precisamos importar em nosso projeto.

```java
import java.util.Scanner;

class MyApp {
    public static void main(String args[]) {
        String nome = null;
        int idade;
        Scanner ler = new Scanner(System.in);
        System.out.println("Digite seu nome");
        nome = ler.nextLine();
        System.out.println("Digite sua idade");
        idade = ler.nextInt();
        ler.nextLine();
        System.out.println("O nome digitado foi: " + nome + ". Você possui " + idade + " anos.");
        ler.close();
    }

}
```

O primeiro passo é instanciarmos um objeto da classe (**new**) passando como parâmetro a fonte do texto. No nosso caso será a linha de comando (**System.in**)

Aqui o sistema aguarda a digitação e recebe a linha inteira digitada

Aqui um número inteiro é aguardado.

https://https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Ciência da Computação

Classe Scanner (Package java.util)

O objeto *System.in* é o que faz a leitura do que se escreve no teclado. Veja abaixo como são invocados alguns dos métodos principais que correspondem com a assinatura que retorna um valor do tipo que foi invocado. Ou seja, para cada um dos primitivos existe uma chamada do método para retornar o valor especificado na entrada de dados, sempre seguindo o formato `nextTipoDado()`.

```
1  Scanner sc = new Scanner(System.in);
2
3  float numF = sc.nextFloat();
4  int num1 = sc.nextInt();
5  byte byte1 = sc.nextByte();
6  long lg1 = sc.nextLong();
7  boolean b1 = sc.nextBoolean();
8  double num2 = sc.nextDouble();
9  String nome = sc.nextLine();
```

https://www.devmedia.com.br/como-funciona-a-classe-scanner-do-java/28448

Métodos da Classe

Na tabela abaixo são apresentados os principais métodos da classe Scanner.

Método	Descrição
close()	Fecha o escaneamento de leitura.
findInLine()	Encontra a próxima ocorrência de um padrão ignorando máscaras ou strings ignorando delimitadores.
hasNext()	Retorna um valor booleano verdadeiro (true) se o objeto Scanner tem mais dados de entrada.
hasNextXyz()	Retorna um valor booleano como verdadeiro (true) se a próxima entrada a qual Xyz pode ser interceptada como Boolean, Byte, Short, Int, Long, Float ou Double.
match()	Retorna o resultado da pesquisa do último objeto Scanner atual.
next()	Procura e retorna a próxima informação do objeto Scanner que satisfazer uma condição.
nextBigDecimal(), nextBigInteger()	Varre a próxima entrada como BigDecimal ou BigInteger.
nextXyz()	Varre a próxima entrada a qual Xyz pode ser interceptado como boolean, byte, short, int, long, float ou double.
nextLine()	Mostra a linha atual do objeto Scanner e avança para a próxima linha.
radix()	Retorna o índice atual do objeto Scanner.
remove()	Essa operação não é suportada pela implementação de um Iterator.
skip()	Salta para a próxima pesquisa de um padrão especificado ignorando delimitadores.
string()	Retorna uma string que é uma representação do objeto Scanner.

Classe Math (Package java.lang.Math)

- A classe **Math** em Java é uma classe utilitária que faz parte do pacote **java.lang**. Ela fornece um conjunto de métodos estáticos para executar operações matemáticas comuns, como funções trigonométricas, operações exponenciais, cálculos de raiz quadrada, entre outros. Essa classe é amplamente usada para realizar cálculos matemáticos em programas Java.
- **Funções trigonométricas**

```
double seno = Math.sin(0.5);
double cosseno = Math.cos(0.5);
double tangente = Math.tan(0.5);
```

- **Funções exponenciais e logarítmicas**:

```
double exponencial = Math.exp(2.0);
double logaritmoNatural = Math.log(10.0);
double logaritmoBase10 = Math.log10(100.0);
```

- **Raiz quadrada e potenciação**:

```
double raizQuadrada = Math.sqrt(25.0);
double potencia = Math.pow(2.0, 3.0);
```

- **Funções de arredondamento**:

```
double paraCima = Math.ceil(7.3);
double paraBaixo = Math.floor(7.8);
double arredondado = Math.round(7.5);
```

https://https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Bloco try-cath – Tratamento de exceções

- O bloco try-catch é uma construção em Java que permite lidar com exceções (erros) que podem ocorrer durante a execução do código. Ele é usado para capturar e tratar exceções, evitando que o programa termine abruptamente devido a erros. Isso é particularmente útil quando você está lidando com operações que podem causar erros, como acesso a arquivos, conexões de rede, divisões por zero, entre outras situações.

```
try {
    // Código que pode gerar uma exceção
} catch (TipoDeExcecao excecao) {
    // Tratamento da exceção
}
```

https://https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Ciência da Computação

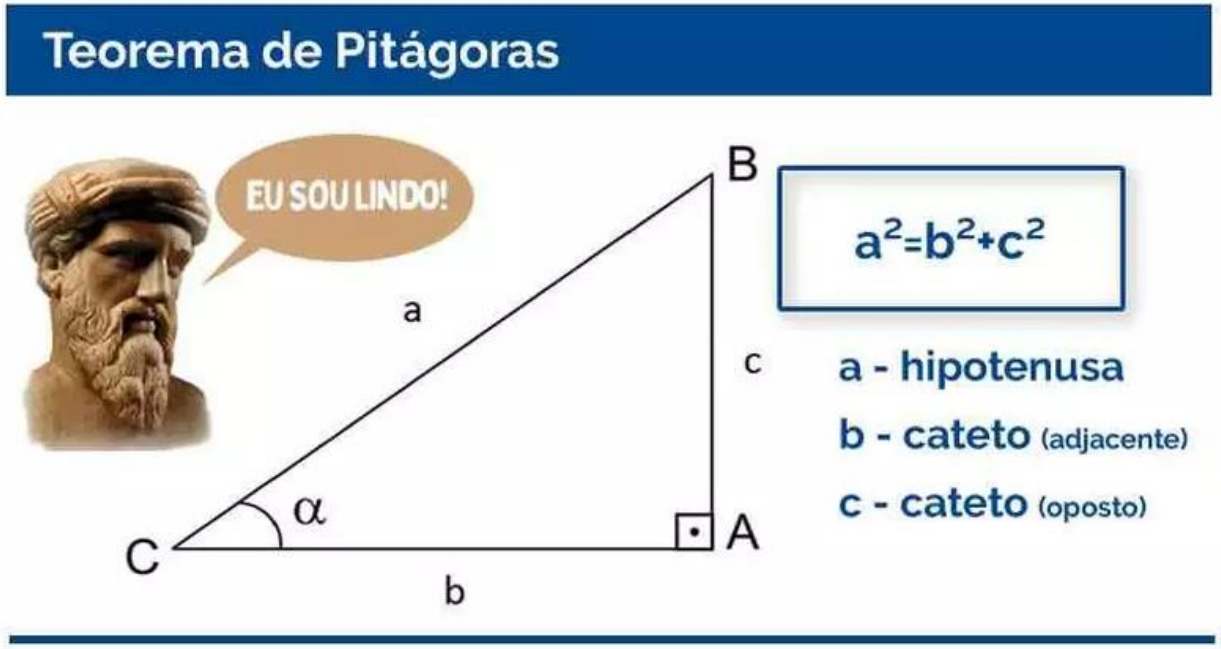
Bloco try-cath – Tratamento de exceções

- O código que pode gerar uma exceção é colocado dentro do bloco try.
- Se uma exceção ocorrer durante a execução do código no bloco try, o fluxo de controle é transferido para o bloco catch.
- O bloco catch contém o tratamento da exceção, onde você pode escrever código para lidar com o erro de forma apropriada. Isso pode incluir mensagens de erro, registros, recuperação, etc.

```
try {
    int resultado = 10 / 0; // Isso causará uma ArithmeticException
} catch (ArithmeticException excecao) {
    System.out.println("Erro de divisão por zero: " + excecao.getMessage());
}
```

https://https://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Agora é com Vocês ...



Vamos relembrar o Ensino Médio !!!

Escreva uma algoritmo que receba os valores dos dois catetos de um triângulo retângulo e retorne o valor da hipotenusa.

Para calcular a raiz quadrada de um valor deve-se utilizar o método **sqrt** da classe **Math**.

Exemplo: Math.sqrt(9); ➔ retorna o valor 3

Ciência da Computação


SEE YOU SPACE COWBOY ...