# **Linguagem de Programação**

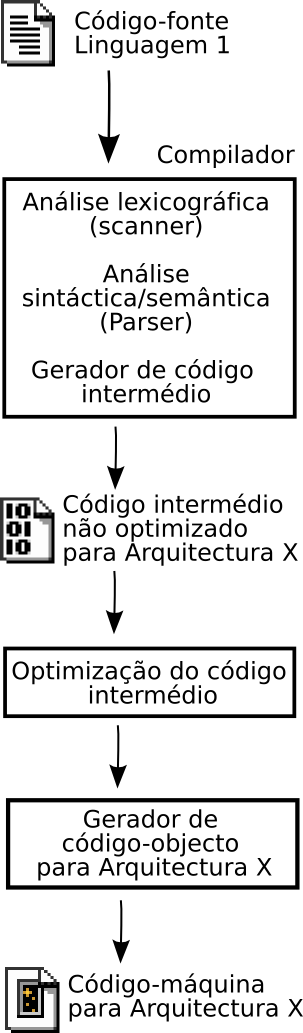
Todo programa é um conjunto de instruções, seja um programa que some dois números, seja um envio de solicitação pela internet. Compiladores e interpretadores recebem código legível por seres humanos e convertem-no para código de máquina, legível pelo computador.

Em uma linguagem compilada, a máquina de destino traduz o programa diretamente. Em uma linguagem interpretada, o código fonte não é traduzido diretamente pela máquina de destino. Em vez disso, um programa *diferente*, o interpretador, lê e executa o código.

# **Código-fonte**

(source code em inglês) é o conjunto de palavras ou símbolos escritos de forma ordenada, contendo instruções em uma das [linguagens de programação](https://pt.wikipedia.org/wiki/Linguagens_de_programa%C3%A7%C3%A3o" \o "Linguagens de programação) existentes, de maneira lógica. Existem linguagens que são compiladas e as que são interpretadas. As linguagens compiladas, após ser compilado o código-fonte, transformam-se em [software](https://pt.wikipedia.org/wiki/Software" \o "Software), ou seja, programas executáveis. Este conjunto de palavras que formam linhas de comandos deverá estar dentro da padronização da linguagem escolhida, obedecendo critérios de execução. Atualmente, com a diversificação de linguagens, o código pode ser escrito de forma totalmente modular, podendo um mesmo conjunto de códigos ser compartilhado por diversos programas e, até mesmo, linguagens.

# **Interpretação e compilação**



Uma linguagem de programação pode ser convertida, ou traduzida, em código de máquina por [compilação](https://pt.wikipedia.org/wiki/Compilador" \o "Compilador) ou interpretada por um processo denominado [interpretação](https://pt.wikipedia.org/wiki/Interpretadores" \o "Interpretadores). Em ambas ocorre a tradução do código fonte para código de máquina. Se o método utilizado traduz todo o texto do programa (também chamado de código), para só depois executar o programa, então diz-se que o programa foi compilado e que o mecanismo utilizado para a tradução é um compilador (que por sua vez nada mais é do que um programa). A versão compilada do programa tipicamente é armazenada, de forma que o programa pode ser executado um número indefinido de vezes sem que seja necessária nova compilação, o que compensa o tempo gasto na compilação. Isso acontece com linguagens como [Pascal](https://pt.wikipedia.org/wiki/Pascal_(linguagem_de_programa%C3%A7%C3%A3o)" \o "Pascal (linguagem de programação)) e [C](https://pt.wikipedia.org/wiki/C_(linguagem_de_programa%C3%A7%C3%A3o)" \o "C (linguagem de programação)). Se o texto do programa é executado à medida que vai sendo traduzido, como em [JavaScript](https://pt.wikipedia.org/wiki/JavaScript), [BASIC](https://pt.wikipedia.org/wiki/BASIC), [Python](https://pt.wikipedia.org/wiki/Python) ou [Perl](https://pt.wikipedia.org/wiki/Perl), num processo de tradução de trechos seguidos de sua execução imediata, então diz-se que o programa foi interpretado e que o mecanismo utilizado para a tradução é um interpretador. Programas interpretados são geralmente mais lentos do que os compilados, mas são também geralmente mais flexíveis, já que podem interagir com o ambiente mais facilmente.

Embora haja essa distinção entre [linguagens interpretadas](https://pt.wikipedia.org/wiki/Linguagem_interpretada" \o "Linguagem interpretada) e [compiladas](https://pt.wikipedia.org/wiki/Linguagem_compilada), as coisas nem sempre são tão simples. Há linguagens compiladas para um código de máquina virtual (sendo esta [máquina virtual](https://pt.wikipedia.org/wiki/M%C3%A1quina_virtual" \o "Máquina virtual) apenas mais um software, que emula a máquina virtual sendo executado em uma máquina real), como [Java](https://pt.wikipedia.org/wiki/Java_(linguagem_de_programa%C3%A7%C3%A3o)" \o "Java (linguagem de programação)) (compila para a [plataforma Java](https://pt.wikipedia.org/wiki/Plataforma_Java" \o "Plataforma Java)[]](https://pt.wikipedia.org/wiki/Linguagem_de_programa%C3%A7%C3%A3o#cite_note-jit-38)) e [C#](https://pt.wikipedia.org/wiki/C_Sharp" \o "C Sharp) (compila para a [plataforma CLI](https://pt.wikipedia.org/wiki/Common_Language_Infrastructure" \o "Common Language Infrastructure)). E também há outras formas de interpretar em que os [códigos fontes](https://pt.wikipedia.org/wiki/C%C3%B3digo_fonte" \o "Código fonte), ao invés de serem interpretados linha-a-linha, têm blocos "compilados" para a memória, de acordo com as necessidades, o que aumenta a performance dos programas quando os mesmos módulos são chamados várias vezes, técnica esta conhecida como [JIT](https://pt.wikipedia.org/wiki/JIT" \o "JIT).

Em [Ciência da Computação](https://pt.wikipedia.org/wiki/Ci%C3%AAncia_da_Computa%C3%A7%C3%A3o" \o "Ciência da Computação), compilação just-in-time (JIT), também conhecida como tradução dinâmica, é a compilação de um programa em [tempo de execução](https://pt.wikipedia.org/wiki/Tempo_de_execu%C3%A7%C3%A3o" \o "Tempo de execução) , usando uma abordagem diferente da compilação anterior à execução. Geralmente, consiste em transformar o código em [código de máquina](https://pt.wikipedia.org/wiki/C%C3%B3digo_de_m%C3%A1quina" \o "Código de máquina), que é então executado diretamente, mas também pode se referir a tradução para outros formatos.

A compilação JIT é uma combinação de duas abordagens tradicionais de tradução para código de máquina: anterior à execução e durante a execução (interpretação), portanto combina algumas vantagens e desvantagens de ambas. A grosso modo, a compilação JIT combina a velocidade do código compilado à flexibilidade da interpretação, com o gargalo do interpretador e da compilação. A compilação dinâmica permite otimizações que não são possíveis em compilações estáticas, levando em alguns casos a códigos mais rápidos.

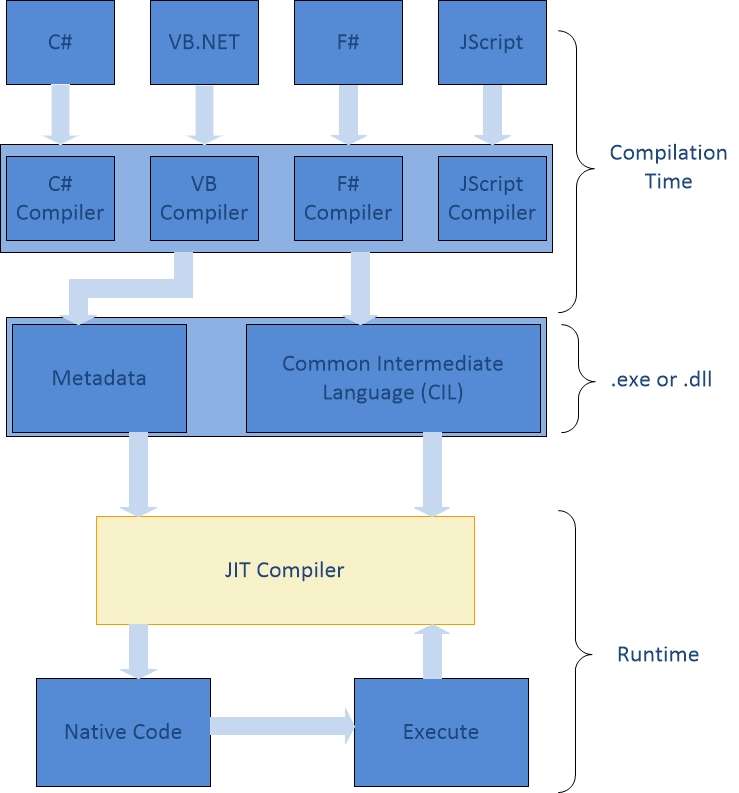
Como exemplo, podemos citar a linguagem Java. Nela, um compilador traduz o código java para o código intermediário (e portável) da JVM. As JVMs originais interpretavam esse código, de acordo com o código de máquina do computador hospedeiro, porém atualmente elas compilam, segundo a técnica JIT o código JVM para código hospedeiro.

A tradução é tipicamente feita em várias fases, sendo as mais comuns a [análise léxica](https://pt.wikipedia.org/wiki/An%C3%A1lise_l%C3%A9xica" \o "Análise léxica), a [análise sintática](https://pt.wikipedia.org/wiki/An%C3%A1lise_sint%C3%A1tica" \o "Análise sintática) (ou [parsing](https://pt.wikipedia.org/wiki/Parser)), a [geração de código](https://pt.wikipedia.org/wiki/Gera%C3%A7%C3%A3o_de_c%C3%B3digo" \o "Geração de código) e a [otimização](https://pt.wikipedia.org/wiki/Otimiza%C3%A7%C3%A3o" \o "Otimização). Em compiladores também é comum a geração de código intermediário.

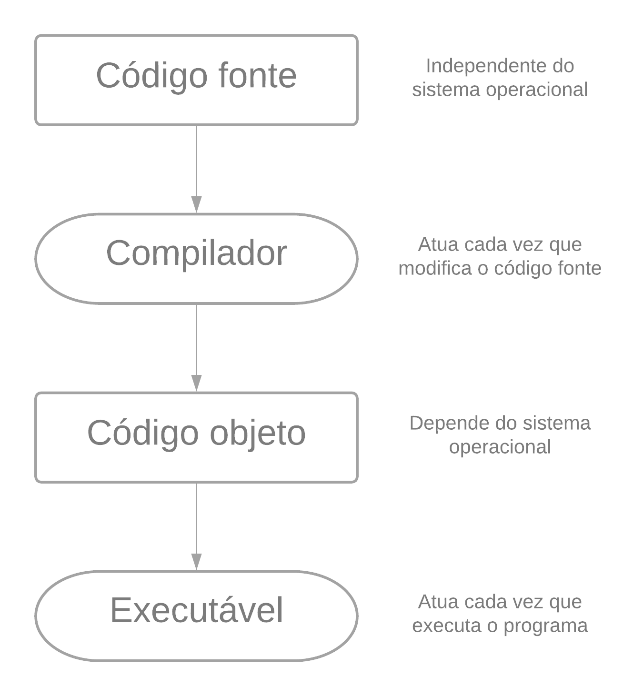
# **linguagem de programação**

é um método padronizado, formado por um conjunto de [regras sintáticas](https://pt.wikipedia.org/wiki/Sintaxe" \o "Sintaxe) e semânticas (A sintaxe é a parte da gramática que se preocupa com a relação entre as palavras. A semântica forma o sentido), de implementação de um [código fonte](https://pt.wikipedia.org/wiki/C%C3%B3digo_fonte" \o "Código fonte) - que pode ser [compilado](https://pt.wikipedia.org/wiki/Compilador" \o "Compilador) e transformado em um [programa de computador](https://pt.wikipedia.org/wiki/Programa_de_computador" \o "Programa de computador), ou usado como [script](https://pt.wikipedia.org/wiki/Linguagem_de_script" \o "Linguagem de script) [interpretado](https://pt.wikipedia.org/wiki/Linguagem_interpretada) - que informará [instruções](https://pt.wikipedia.org/wiki/C%C3%B3digo_de_m%C3%A1quina" \o "Código de máquina) de processamento ao [computador](https://pt.wikipedia.org/wiki/Computador" \o "Computador). Permite que um programador especifique precisamente quais os dados que o computador irá atuar, como estes dados serão armazenados ou transmitidos e, quais ações devem ser tomadas de acordo com as circunstâncias. Linguagens de programação podem ser usadas para expressar [algoritmos](https://pt.wikipedia.org/wiki/Algoritmo" \o "Algoritmo) com precisão. O conjunto de palavras (lexemas classificados em tokens), compostos de acordo com essas regras, constituem o código fonte de um [software](https://pt.wikipedia.org/wiki/Software" \o "Software). Esse código fonte é depois traduzido para [código de máquina](https://pt.wikipedia.org/wiki/C%C3%B3digo_de_m%C3%A1quina" \o "Código de máquina), que é executado pelo [microprocessador](https://pt.wikipedia.org/wiki/Microprocessador" \o "Microprocessador). Uma das principais metas das linguagens de programação é que programadores tenham uma maior produtividade, permitindo expressar suas intenções mais facilmente do que quando comparado com a linguagem que um computador entende nativamente ([código de máquina](https://pt.wikipedia.org/wiki/C%C3%B3digo_de_m%C3%A1quina" \o "Código de máquina)). Assim, linguagens de programação são projetadas para adotar uma sintaxe de nível mais alto, que pode ser mais facilmente entendida por programadores humanos. Linguagens de programação são ferramentas importantes para que [programadores](https://pt.wikipedia.org/wiki/Programa%C3%A7%C3%A3o_de_computadores" \o "Programação de computadores) e [engenheiros de software](https://pt.wikipedia.org/wiki/Engenharia_de_software) possam escrever programas mais organizados e com maior rapidez. Linguagens de programação também tornam os programas menos dependentes de computadores ou ambientes computacionais específicos (propriedade chamada de portabilidade). Isto acontece porque programas escritos em linguagens de programação são traduzidos para o código de máquina do computador no qual será executado em vez de ser diretamente executado. Uma meta ambiciosa do [Fortran](https://pt.wikipedia.org/wiki/Fortran" \o "Fortran), uma das primeiras linguagens de programação, era esta independência da máquina onde seria executada

## **Linguagem compiladas**



Linguagem compilada é uma [linguagem de programação](https://pt.wikipedia.org/wiki/Linguagem_de_programa%C3%A7%C3%A3o" \o "Linguagem de programação) em que o [código fonte](https://pt.wikipedia.org/wiki/C%C3%B3digo_fonte" \o "Código fonte), nessa linguagem, é executado diretamente pelo [sistema operacional](https://pt.wikipedia.org/wiki/Sistema_operativo" \o "Sistema operativo) ou pelo [processador](https://pt.wikipedia.org/wiki/Processador" \o "Processador), após ser traduzido por meio de um processo chamado compilação, usando um [programa de computador](https://pt.wikipedia.org/wiki/Programa_de_computador" \o "Programa de computador) chamado [compilador](https://pt.wikipedia.org/wiki/Compilador), para uma [linguagem de baixo nível](https://pt.wikipedia.org/wiki/Linguagem_de_programa%C3%A7%C3%A3o_de_baixo_n%C3%ADvel" \o "Linguagem de programação de baixo nível), como [linguagem de montagem](https://pt.wikipedia.org/wiki/Linguagem_de_montagem" \o "Linguagem de montagem) ou [código de máquina](https://pt.wikipedia.org/wiki/C%C3%B3digo_de_m%C3%A1quina).

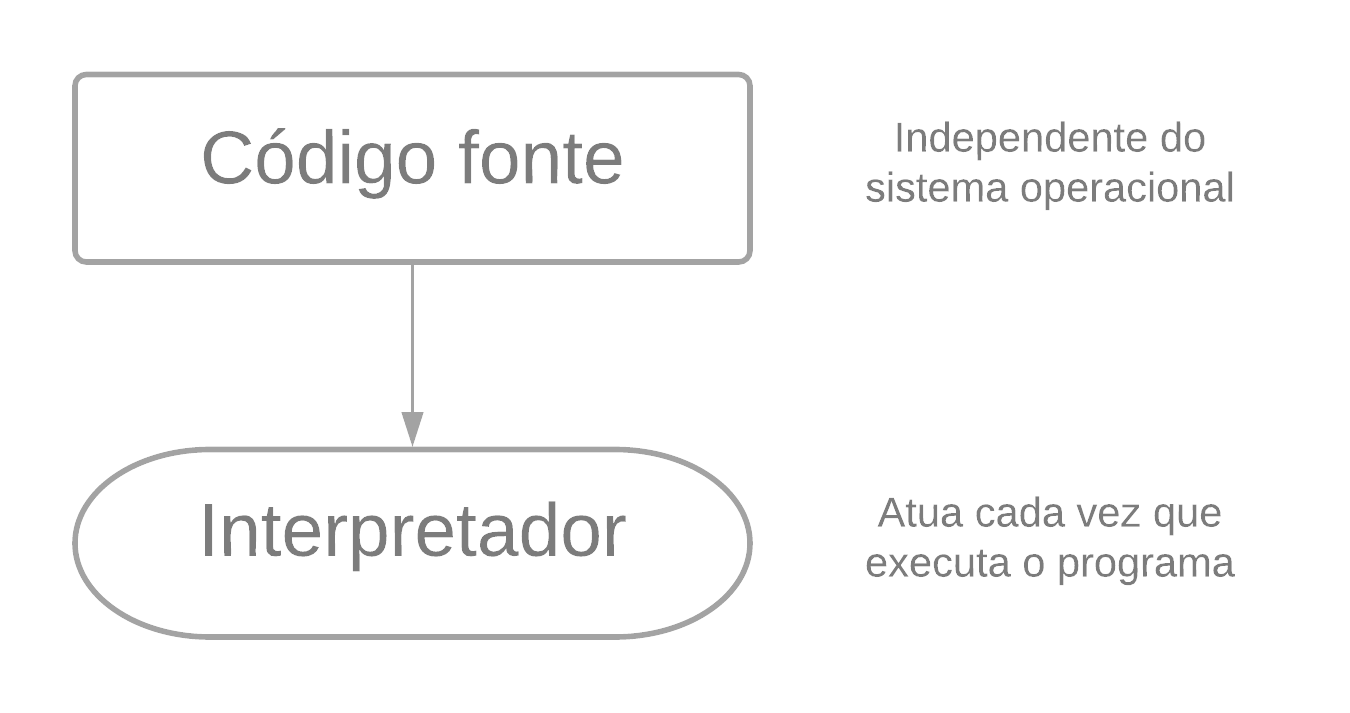
Teoricamente, qualquer linguagem pode ser compilada ou Linguagem interpretada e, por causa disso, há algumas linguagens que possuem ambas implementações

As linguagens compiladas são convertidas diretamente na máquina em um código de máquina que o processador pode executar. Como resultado, elas tendem a ser mais rápidas e mais eficientes em sua execução do que as linguagens interpretadas. Elas também dão ao desenvolvedor mais controle sobre alguns aspectos do hardware, como o gerenciamento da memória e o uso da CPU.

As linguagens compiladas necessitam de uma etapa de "build" (montagem) elas precisam, primeiramente, ser compiladas manualmente. Você precisa "remontar" o programa sempre que precisar fazer uma alteração. Em nosso exemplo do molho, toda a tradução já está escrita antes de chegar até você. Se o autor original decidir usar um tipo diferente de óleo de oliva, a receita inteira precisaria ser traduzida novamente e reenviada a você.

Exemplos de linguagens compiladas puras são o C, o C++, o Erlang, o Haskell, o Rust e o Go.

## **Linguagem interpretada**

Os interpretadores passam por um programa linha por linha e executam cada comando. Aqui, se o autor decidir que quer usar um tipo diferente de óleo de oliva, só precisaria remover o antigo e adicionar o novo. Seu amigo tradutor poderia informar isso a você quando a mudança acontecesse.

Linguagens interpretadas, antigamente, eram significativamente mais lentas do que as linguagens compiladas. Porém, com o desenvolvimento da [compilação just-in-time](https://guide.freecodecamp.org/computer-science/just-in-time-compilation), essa distância vem diminuindo.

Exemplos de linguagens interpretadas comuns são o PHP, o Ruby, o Python e o JavaScript.

A maioria das linguagens de programação pode ter implementações compiladas e interpretadas – a linguagem em si não é necessariamente compilada ou interpretada. Porém, para fins de simplicidade, elas são normalmente referidas deste modo.

Python, por exemplo, pode ser executado como um programa compilado ou como uma linguagem interpretada em modo interativo. Por outro lado, a maioria das ferramentas de linha de comando, ou CLIs, e shells podem, em teoria, ser classificadas como linguagens interpretadas.

## **Linguagem dinamica**

Linguagem dinâmica, também conhecida como linguagem de scripting, ainda é algo obscuro para a grande maioria dos desenvolvedores. Entender e aplicar seus conceitos é algo que pode causar espanto e muitas dúvidas. Porém, é importante conhecer as vantagens e destantagens deste tipo de linguagem, saber quando é válido ou não a sua utilização no desenvolvimento de sistemas. Estas linguagens são linguagens de alto nível, a grande maioria com [tipagem](https://celodemelo.wordpress.com/2008/02/28/linguagem-fortemente-tipada-strong-typing/" \t "_blank) dinâmica e com um Protocolo de Meta-Objeto ([Meta-Object Protocol](http://en.wikipedia.org/wiki/Metaobject" \t "_blank)), ou MOP. Estas duas características proporcionam muitas facilidades e um enorme poder no desenvolvimento. Porém, quando não usadas com cuidado, podem causar grandes problemas.

A tipagem dinâmica está ligado a habilidade da linguagem de programação em escolher o tipo de dado de acordo com o valor atribuído à variável em tempo de execução dinamicamente. Veja o exemplo abaixo na linguagem PHP:

$variavel = "Elton Fonseca";

echo gettype($variavel); //string

$variavel = 340;

echo gettype($variavel); //integer

$variavel = 340.89;

echo gettype($variavel); //double

$variavel = true;

echo gettype($variavel); //Boolean

Muita gente confunde, acha que linguagem de tipagem dinâmica não possui tipos. Na verdade, ela possui tipos normalmente, a diferença está apenas na capacidade da linguagem em escolher o tipo automaticamente.

O tipo da variável assume o tipo do valor. Assim como a tipagem estática também existe a verificação, mas ela é feita em cima do dado já que a variável pode ter qualquer dado, essa verificação é feita em momento de execução, então é comum que o desenvolvedor se preocupe mais com os tipos e, em certos momentos, precisará verificar o tipo do dado que está sendo trabalhado. Compilação e execução ocorrem ao mesmo momento, então podemos dizer que é interpretada, compila e já executa. Costuma-se utilizar a [notação húngara](https://en.wikipedia.org/wiki/Hungarian_notation" \t "_blank), em algumas linguagens, para denotar o qual é o tipo da variável

## **Linguagem estatica**

Linguagens com tipagem estática não permitem ao desenvolvedor alterar o tipo da variável depois de declarada. Geralmente a verificação de tipo é feita em tempo de compilação. Podemos ver o exemplo abaixo na linguagem Java:

public class MyClass {

public static void main(String args[]) {

int variavel = 10;

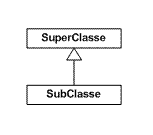
variavel = "Elton Fonseca"; //error: incompatible types: String cannot be converted to int

}

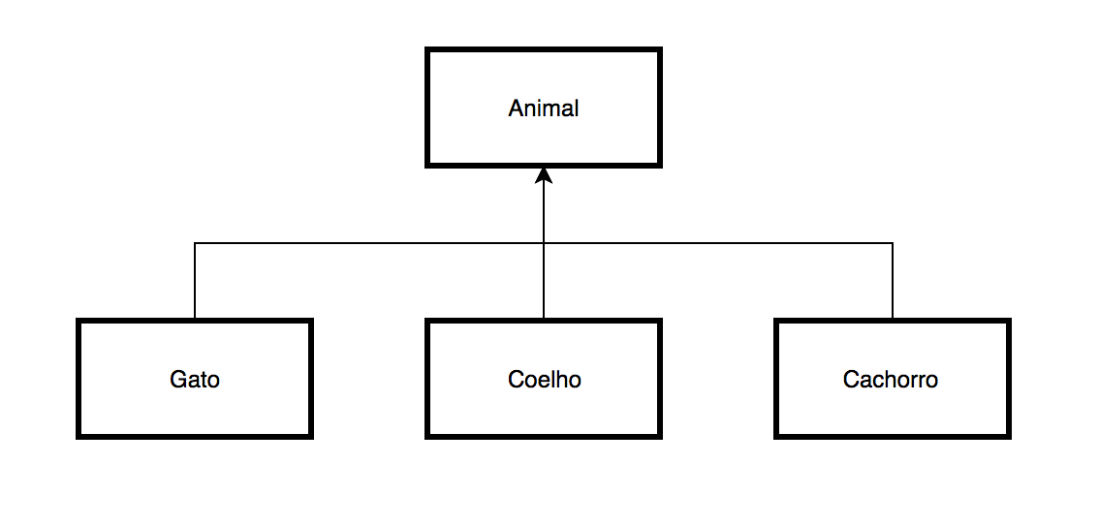
}

Quando tentamos atribuir um valor de um tipo diferente do que foi declarado na variável temos um erro.

# **Herança**



Herança é um princípio de [orientação a objetos](https://pt.wikipedia.org/wiki/Orienta%C3%A7%C3%A3o_a_objetos" \o "Orientação a objetos), que permite que  [classes](https://pt.wikipedia.org/wiki/Classe_(programa%C3%A7%C3%A3o))  compartilhem  [atributos](https://pt.wikipedia.org/wiki/Atributo_(programa%C3%A7%C3%A3o))  e [métodos](https://pt.wikipedia.org/wiki/M%C3%A9todo_(programa%C3%A7%C3%A3o)" \o "Método (programação)), através de "heranças". Ela é usada na intenção de reaproveitar código ou comportamento generalizado ou especializar operações ou atributos. O conceito de herança de várias classes é conhecido como [herança múltipla](https://pt.wikipedia.org/wiki/Heran%C3%A7a_m%C3%BAltipla" \o "Herança múltipla). Como exemplo pode-se observar as classes 'aluno' e 'professor', onde ambas possuem atributos como nome, endereço e telefone. Nesse caso pode-se criar uma nova classe chamada por exemplo, 'pessoa', que contenha as semelhanças entre as duas classes, fazendo com que aluno e professor herdem as características de pessoa, desta maneira pode- se dizer que aluno e professor são subclasses de pessoa. Também podemos dizer que uma classe pode ser abstrata(abstract) ou seja ela não pode ter uma instância, ela apenas "empresta" seus atributos e metódos como molde para novas classes.



Para utilizar a herança no Python é bem simples. Assim com vimos no diagrama acima, vamos criar quatro classes para representar as entidades Animal, Gato, Cachorro e Coelho.

class Animal():

def \_\_init\_\_(self, nome, cor):

self.\_\_nome = nome

self.\_\_cor = cor

def comer(self):

print(f"O {self.\_\_nome} está comendo")

Após isso, criamos as três classes “filhas” da classe Animal. Para definir que estas classes são herdeiras da classe Animal, declaramos o nome da classe pai nos parenteses logo após definir o nome da classe, como podemos ver abaixo:

import animal

class Gato(animal.Animal):

def \_\_init\_\_(self, nome, cor):

super().\_\_init\_\_(nome, cor)

import animal

class Cachorro(animal.Animal):

def \_\_init\_\_(self, nome, cor):

super().\_\_init\_\_(nome, cor)

import animal

class Coelho(animal.Animal):

def \_\_init\_\_(self, nome, cor):

super().\_\_init\_\_(nome, cor)

Note que as classes filhas só estão repassando seus dados de nome e cor para a classe Pai através do super() e que nenhum método foi implementado dentro dessas classes.

Agora, por herdar da classe Animal, as classes Gato, Cachorro e Coelho podem, sem nenhuma alteração, utilizar o método comer(), definido na classe Animal pois elas herdam dessa classe, logo elas possuem a permissão de invocar este método:

import gato, cachorro, coelho

gato = gato.Gato("Bichano", "Branco")

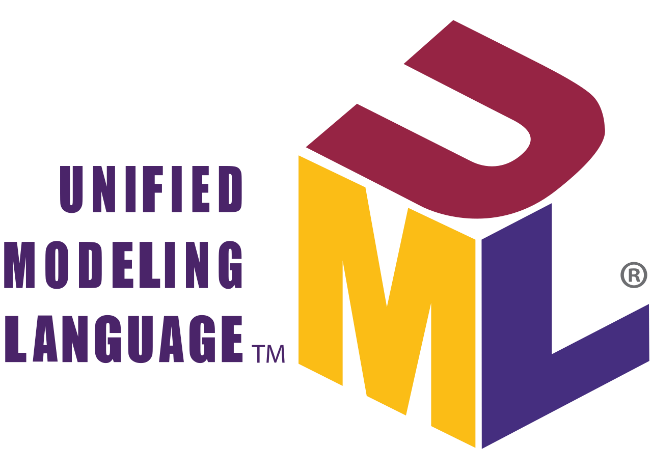
cachorro = cachorro.Cachorro("Totó", "Preto")

coelho = coelho.Coelho("Pernalonga", "Cinza")

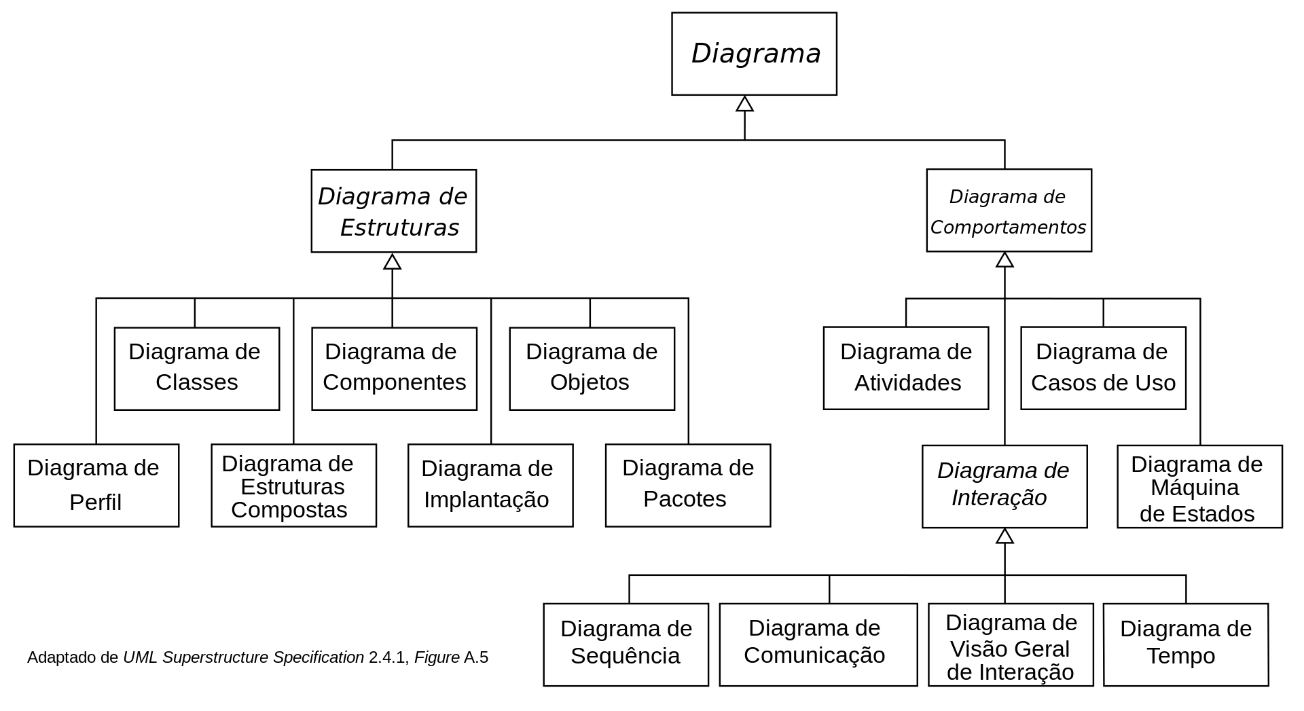
gato.comer()

cachorro.comer()

coelho.comer()



# **UML**

A UML (***do***[***inglês***](https://pt.wikipedia.org/wiki/L%C3%ADngua_inglesa)***Unified Modeling Language, em [português](https://pt.wikipedia.org/wiki/L%C3%ADngua_portuguesa" \o "Língua portuguesa) Linguagem de Modelagem Unificada***) é uma linguagem-padrão para a elaboração da estrutura de projetos de software. Ela poderá ser empregada para a visualização, a especificação, a construção e a documentação de artefatos que façam uso de sistemas complexos de software. Em outras palavras, na área de [Engenharia de Software](https://pt.wikipedia.org/wiki/Engenharia_de_software" \o "Engenharia de software), a UML é uma [linguagem de modelagem](https://pt.wikipedia.org/wiki/Linguagem_de_modelagem" \o "Linguagem de modelagem) que permite representar um sistema de forma padronizada (com intuito de facilitar a compreensão pré-implementação). A UML é adequada para a modelagem de sistemas, cuja abrangência poderá incluir desde sistemas de informação corporativos a serem distribuídos a aplicações baseadas na Web e até sistemas complexos embutidos de tempo real. É uma linguagem muito expressiva, abrangendo todas as visões necessárias ao desenvolvimento e implantação desses sistemas. 

A UML (Unified Modeling Language) não é uma [metodologia](https://pt.wikipedia.org/w/index.php?title=Metodologia_(engenharia_de_software)&action=edit&redlink=1" \o "Metodologia (engenharia de software) (página não existe)) de desenvolvimento, o que significa que ela não diz para você o que fazer primeiro e em seguida ou como projetar seu sistema, mas ela lhe auxilia a visualizar seu desenho e a comunicação entre os objetos (e em certos casos a identificação dos processos).

Basicamente, a UML permite que desenvolvedores visualizem os produtos de seus trabalhos em diagramas padronizados. Junto com uma notação gráfica, a UML também especifica significados, isto é, [semântica](https://pt.wikipedia.org/wiki/Sem%C3%A2ntica" \o "Semântica). É uma notação independente de [processos](https://pt.wikipedia.org/wiki/Processo" \o "Processo), embora o [RUP](https://pt.wikipedia.org/wiki/IBM_Rational_Unified_Process" \o "IBM Rational Unified Process) (Rational Unified Process) tenha sido especificamente desenvolvido utilizando a UML.

É importante distinguir entre um modelo UML e um [diagrama](https://pt.wikipedia.org/wiki/Diagrama" \o "Diagrama) (ou conjunto de diagramas) de UML. O último é uma representação gráfica da informação do primeiro, mas o primeiro pode existir independentemente. O XMI ([XML](https://pt.wikipedia.org/wiki/XML" \o "XML) Metadata Interchange) na sua versão corrente disponibiliza troca de modelos mas não de diagramas.

Os objetivos da UML são: especificação, documentação, estruturação para sub-visualização e maior visualização lógica do desenvolvimento completo de um sistema de informação.

# **Depuração**

Depuração corresponde ao processo de localizar e corrigir erros ou bugs no código-fonte de qualquer software. Quando o software não funciona conforme o esperado, os programadores estudam o código para determinar as causas dos erros. Eles usam ferramentas de depuração para executar o software em um ambiente controlado, verificar o código detalhado, e analisar e corrigir o problema.

# **Pytest**

# Polimorfismo

Classe abstrata

Interface

Tratamento de exceções

I/Ostream

Netwok stream

Fluxo de E/S (I/O Streams)

Padrão adaptador

Padrão singleton

Padrão metodo fabrica

Padrao fabrica abstrata

Padrao prototipo

Padrao estado

Padrao MVC

Padrao Decorador

Padrao template method

Padrao fachada

Padrao mediador

Principio solid

Template method no serpro

Padrao observador

Padrao flyweight

Refatoração

Injeção de dependencias e spring

Frameworks = components + partters