O que são as tipagens estática e dinâmica em programação

Juliana Amoasei
13/08/2021

Esse artigo faz parte da
Formação Python e orientação a objetos

Linguagens como JavaScript e Python são, para muitas pessoas, a porta de entrada na programação. São linguagens de sintaxe mais enxuta do que o Java, por exemplo, e permitem que alguém que está começando escreva seus primeiros programas de forma mais rápida.

Para declarar uma variável no Javascript ou Python basta escrever o nome e atribuir a ela algum dos valores possíveis, normalmente um texto entre aspas, um número (sem aspas) ou um dos valores booleanos: *true* ou *false*.

```
//JavaScript
const numero = 50

//Python
numero = 50
```

Porém, quando falamos nos **tipos de dados**, um conceito fundamental em programação, a questão vai além do valor que damos às variáveis. Além de sabermos o que são números, strings de caracteres, booleanos, etc, há também outros dois conceitos importantes que normalmente acompanham a ideia de tipo: **tipagem estática** e **tipagem dinâmica**, e quais linguagens utilizam cada um deles.

Neste artigo vamos revisar o que são estes conceitos, o porquê de serem importantes e como eles afetam as linguagens e a forma de escrever código.

O que são tipos?

Antes de começarmos, é importante relembrar o que são tipos. Um dos aspectos mais fundamentais em uma linguagem de programação é saber com quais tipos ela trabalha e como manipulá-los utilizando **variáveis**.

Podemos definir tipos como **valores que são manipulados por um programa**. São os blocos básicos e principais que usamos para programar. Afinal de contas, a programação é basicamente processar dados e, para fazer isso, o programa precisa

saber **qual é o tipo de dado que será processado**.
A maior parte das linguagens de programação trabalha com variações baseadas nos quatro **tipos primitivos** abaixo:

- **INT** ou número inteiro: valores numéricos inteiros (positivos ou negativos);
- **FLOAT** ou o chamado "ponto flutuante": valores numéricos com casas após a vírgula (positivos ou negativos);
- **BOOLEAN** ou booleanos: representado apenas por dois valores, "verdadeiro" e "falso". Também chamados de *operadores lógicos*;
- **TEXT**: sequências ou cadeias de caracteres, utilizados para manipular textos e/ou outros tipos de dados não numéricos ou booleanos, como hashes de criptografia.

O **JavaScript**, por exemplo, tem como tipos primitivos embutidos na estrutura básica da linguagem: *number*, *string*, *boolean* e *symbol* (de "nome simbólico", usado entre outras coisas para criar propriedades únicas em objetos). Já o **C#** (C Sharp) trabalha com uma quantidade maior de tipos primitivos, de acordo com o espaço de memória que será ocupado pela variável: *Boolean*, *Byte*, *SByte*, *Int16*, *UInt16*, *Int32*, *UInt32*, *Int64*, *UInt64*, *IntPtr*, *UIntPtr*, *Char*, *Double* e *Single*. Outras linguagens podem trabalhar com outras variações.

Além dos primitivos, existem outros tipos estruturados, como *funções*, *arrays*, *objetos* e suas variações também dependem da linguagem.

Agora que já sabemos o básico sobre o que são tipos, vamos entender o que são linguagens *estaticamente tipadas* e *dinamicamente tipadas*.

Linguagens estaticamente tipadas

Uma linguagem é definida como *estaticamente tipada* quando a pessoa que está programando precisa informar explicitamente o tipo de cada dado utilizado no sistema: variáveis, parâmetros de funções, valores de retorno, etc. Uma vez definido o tipo, estas variáveis estão restritas ao tipo declarado; a checagem (*type checking*) é feita na compilação do programa ou em tempo de execução (*runtime*), dependendo se a linguagem for compilada ou interpretada.

> *Linguagens compiladas são aquelas em que o código escrito pela pessoa é*

> *"traduzido" para linguagem de máquina, gerando outro código que aí sim pode ser executado - exemplos de linguagens compiladas são Java, C e Go. Nas linguagens interpretadas o código escrito é traduzido para linguagem de máquina durante a execução - por exemplo, Python e JavaScript.*
> *A checagem de tipos serve para minimizar a possibilidade de erros causados por tipos de dados errados. Por exemplo, no caso do programa receber dados no formato de string para fazer uma soma, ao invés de dados do tipo número.*

Assim, quem está programando estabelece e declara o tipo de dado que será associado à variável, como neste exemplo em C#:

```
String mensagem = "Alura";
Int32 numero = 100;
bool status = true;
```

No exemplo acima foram declaradas três variáveis, cada uma com um tipo de dado diferente: `String` para o texto, `Int32` para um número inteiro de 32 bits e `bool` para um tipo booleano (no caso, `true`). Uma vez definido, o tipo de dado na variável não pode mais ser modificado; variáveis declaradas como `string`, por exemplo, só poderão receber dados desse tipo.

Algumas linguagens estaticamente tipadas trabalham com a chamada *inferência de tipo*, técnica em que o programa consegue determinar sozinho o tipo da variável sem que seja necessário deixar essa informação explícita no código. Porém, a checagem de tipos continua ocorrendo normalmente.

Exemplos de linguagens estaticamente tipadas são **C**, **C++**, **Java**. **Go** e **Rust**. Linguagens como Java, C e C++ também são **explicitamente tipadas**, ou seja, é necessário declarar o tipo ao criar a variável.

Linguagens dinamicamente tipadas

> *O tipo é determinado (inferido) em runtime (ou tempo de execução) de acordo com o valor do dado, e não a partir da variável.*

Nestes casos, o programa observa qual é o tipo de cada dado que está sendo declarado do código e, a partir disso, determina a tipagem. A sintaxe não exige que se informe explicitamente o tipo quando definimos variáveis; em algumas linguagens é possível informar explicitamente o tipo de dado, mas não é obrigatório.

Vamos ver um exemplo da diferença entre tipagem estática e dinâmica:

```
public int add(int a, int b) {
 return a + b;
}
```

Na função acima, os tipos dos parâmetros e do retorno estão explícitos, e a função só irá aceitar dados do tipo `int` (números inteiros), e também só vai devolver (ou retornar) dados do tipo `int`.

```
function soma(a, b) {
 return a + b;
}
```

Na função acima não há nenhuma indicação de tipo. O tipo, que será recebido pelos parâmetros da função, será definido durante a execução do programa. Além disso, a mesma função pode, pelo exemplo acima, receber tanto dados do tipo `int` (números inteiros) como `float` (números de ponto flutuante).

Alguns exemplos de linguagens com tipagem dinâmica são: **JavaScript**, **Python**, **PHP**, **Ruby**, entre outras.

Tipagem “forte” vs tipagem “fraca”

Além de utilizarem abordagens diferentes para a checagem de tipos, as linguagens também podem ter comportamentos diferentes quando o programa executa operações com tipos de dados não-esperados. Veja este exemplo em Python:

```
num = "10"
operacao = num + 10; //TypeError: cannot concatenate 'str' and 'int' ob
```

Vimos que uma tentativa de operação de soma entre string e número resulta justamente em um *type error*, ou seja, um erro de tipo: "não é possível concatenar objetos `str` e `int` ".

Agora vejamos este exemplo em JavaScript:

```
const num = "10";
const operacao = num + 10; //1010
```

O programa não gerou erros, pois o JavaScript fez uma **conversão implícita** do valor `10` de número para string, para então concatenar com a string `"10"` . Essa é uma característica de linguagens com tipagem "fraca", como o JavaScript e o PHP.

É interessante notar, também, que linguagens com tipagem dinâmica não têm necessariamente tipagem fraca, como vimos acima com Python.

Exemplos de linguagens com tipagem dinâmica e forte: Python, Ruby, Erlang, Clojure

Exemplos de linguagens com tipagem dinâmica e fraca: JavaScript, PHP, Perl

Exemplos de linguagens com tipagem estática e forte: C#, Java, Scala

Exemplos de linguagens com tipagem estática e fraca: C, C++

Conclusão

- Na **tipagem estática**, o tipo é inferido pela variável e a checagem (*type checking*) é feita durante a compilação;
- Na **tipagem dinâmica**, o tipo é inferido pelo valor do dado e a checagem (*type checking*) é feita em tempo de execução (*runtime*);

- Linguagens estaticamente tipadas têm uma performance de execução melhor, pois durante a execução o código já foi "traduzido" para linguagem máquina e a checagem/tipagem das variáveis já foi feito;
- Por outro lado, linguagens dinamicamente tipadas costumam ser mais ágeis durante o desenvolvimento e também são mais flexíveis.

Se você gostou desse conteúdo e quer saber mais, aqui na Alura temos formações para as linguagens mais utilizadas no mercado como **Java**, **Python** e **PHP**.

Juliana Amoasei

Desenvolvedora JavaScript com background multidisciplinar, sempre aprendendo para ensinar e vice-versa. Atuo em diversas iniciativas de inclusão em tecnologia desde 2018 e acredito no potencial do conhecimento como agente de mudança pessoal e social. Atualmente trabalho como instrutora na Escola de Programação da Alura e dou mentoria técnica a iniciantes na área de desenvolvimento web frontend e backend; fora da tela preta, me dedico ao Kung Fu e à nerdices em geral.

Veja outros artigos sobre
Programação

WHATSAPP
EMAIL E TELEFONE

BLOG

PROGRAMAÇÃO
FRONT-END
DATA SCIENCE
DEVOPS
UX & DESIGN
MOBILE
INOVAÇÃO & GESTÃO

AOVS Sistemas de Informática S.A
CNPJ 05.555.382/0001-33

NOSSAS REDES E APPS

ALURA NA SUA EMPRESA

Desenvolva seu time com apoio da nossa solução

CASES
COMUNIDADE
ARTIGOS EDUCAÇÃO CORPORATIVA
GUIA DE CARREIRA
NA MÍDIA
CONHEÇA MAIS

COMUNIDADE

SCUBA DEV
IMERSÕES
DEPOIMENTOS
NOSSOS INSTRUTORES

PARCEIROS

Métodos Ágeis | Softskills | Liderança e Gestão | Startups | Vendas