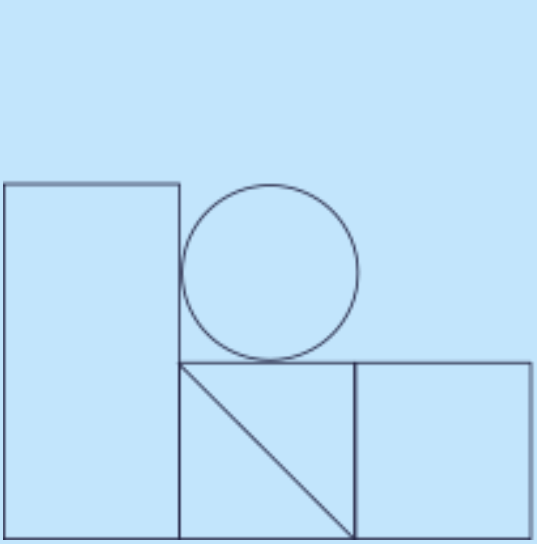
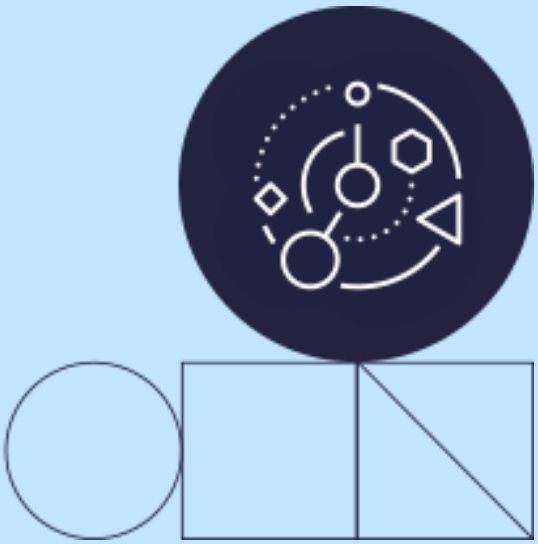
IBM SkillsBuild

IBM

Noções básicas de programação

Quais são os tipos de dados que existem na programação?





Índice

Introdução

Os dados são qualquer objeto que possa ser manipulado pelo computador. Um dado pode ser um caracter lido a partir de um teclado, informações armazenadas em um disco, um número na memória central, etc. Diferentes tipos de dados são representados de diferentes maneiras no computador: por exemplo, um número inteiro não é armazenado internamente da mesma forma que um personagem. Embora idiomas de alto nível permitam, até certo ponto, ignorar a representação interna dos dados, alguns conceitos mínimos devem ser conhecidos.

No nível da máquina, todos os dados são representados usando uma seqüência finita de bits. Já se deduz deste fato que nem todos os dados são representáveis em um computador. A definição de um tipo de dado inclui a definição do conjunto de valores permitidos e as operações que podem ser realizadas sobre esses valores. Quando um dado é usado em um programa, seu tipo deve ser determinado para que o tradutor saiba como manuseá-lo e armazená-lo.

Dependendo do idioma, pode ou não ser necessário declarar expressamente o tipo de cada dado no programa.

Nem todos os tipos de dados existem em todas as linguagens de programação. Alguns idiomas são mais ricos do que outros a este respeito.

Os tipos de dados em uma linguagem de programação podem ser muito variados, por isso é difícil perguntar quantos tipos de dados existem em uma linguagem, já que podemos até criar os nossos próprios, por meio de enumerações ou estruturas.

Em geral, todas as linguagens de programação de alto nível trabalham com os mesmos tipos de dados básicos. Embora seja verdade que a sintaxe e a maneira de criar variáveis muda dependendo se elas são fraca ou fortemente tipadas.

A digitação refere-se à forma como declaramos os tipos de variáveis. Por exemplo, alguns declaramos como inteiros, outros como cordas, flutuadores, e assim por diante. E em alguns idiomas, não precisamos declarar o tipo, porque o tipo é adivinhado.

Por outro lado, a digitação forte não permite operações entre objetos de diferentes tipos. Não podemos acrescentar um fio e um número inteiro. No entanto, em línguas pouco dactilografadas, nós podemos.

Digitação fraca

Linguagens de programação mal digitadas são aquelas em que não indicamos o tipo de variável ao declará-la. A própria linguagem se encarrega de "adivinhar" que tipo é nossa variável.

A verdadeira diferença é que podemos atribuir, por exemplo, um valor inteiro a uma variável que antes tinha uma cadeia. Veremos mais adiante em detalhes os tipos de dados que existem.

Também podemos operar aritmeticamente sobre variáveis de diferentes tipos. Por exemplo, acrescente "x" + 5.

Vantagens

- Esquecemos de declarar o tipo
- Podemos mudar o tipo da variável na mosca. Por exemplo, atribuir uma cadeia a uma int, ou seja, alternar entre letras e números
- Nós escrevemos menos código

Desvantagens

- Ao fazer operações, às vezes elas dão errado. Por exemplo, podemos tentar adicionar 500 + "400,00" + 10, o que será errado porque "400,00" é definido em vírgulas invertidas, portanto é considerado uma cadeia, ou seja, uma cadeia de texto, não uma cadeia numérica. Veremos este conceito mais tarde.
- Temos que fazer cascata muitas vezes. Às vezes, teremos que forçar a cascata das variáveis para que elas se comportem como desejamos e não gerem erros como os mencionados acima.

  Nota: A fundição consiste na mudança forçada do tipo de uma variável. Por exemplo, dizendo ao nosso programa que uma variável cujo valor é numérico é tratada como alfanumérica, ou seja, como uma cadeia.
- Código menos expressivo. Ao declarar os argumentos de uma função, não sabemos se ela espera um flutuador (um número decimal), um número inteiro, um fio... etc. Temos que ir à função, ver o que ela faz e inferir o tipo de variável que ela espera.
- Insegurança: há a possibilidade de um atacante descobrir uma vulnerabilidade onde esperamos uma variável de um certo tipo, mas outra é recebida.

Idiomas que o utilizam

- PHP
- Javascript

Exemplo:

Vamos ver o que acontece no JavaScript quando fazemos a seguinte operação:

```
let resultado = "x" + 5;
```

Em uma linguagem fortemente dactilografada daria um erro, mas em JavaScript nada acontece:

```
> let resultado = "x" + 5;
<· undefined
> resultado
<· "x5"
```

O que fizemos neste exemplo é criar uma variável que chamamos de "resultado" e lhe demos o valor da soma de "x+5". Em idiomas fortemente digitados isto teria dado um erro, uma vez que letras e números não podem ser adicionados juntos. No caso do Javascript, por ser uma linguagem pouco dactilografada, ele nos permite fazer isso.

Digitação forte

É aqui que indicamos o tipo de dados ao declarar a variável. Este tipo nunca pode ser alterado. E não podemos operar entre tipos diferentes. Ou seja, se declararmos, por exemplo, uma variável como numérica, nunca poderemos colocar dentro dela um dado que não seja numérico.

Vantagens

- Código expressivo: agora saberemos que tipo de argumento uma função espera.
- Menos erros: Esqueceremos de ver o tipo da variável antes de fazer operações com ela.

Desvantagens

- Escreva mais código: temos que especificar o tipo de variável ao declará-la.

Idiomas que a utilizam

Para citar apenas alguns...

- C
- C#
- Java
- Ruby
- Python

Exemplo

Vamos tentar a operação "x" + 5 em Python, e ver o que acontece:

```
>>> resultado = "x" + 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: must be str, not int
>>>
```

É um erro que especifica que caracteres alfanuméricos e números não podem ser adicionados juntos.

Em geral, os idiomas fortemente digitados são mais seguros, pois não permitem operações com variáveis de diferentes tipos.

Aqui estão **três exemplos** de tipos de dados básicos que são os mais usados: **números, texto e datas**.

Se estamos trabalhando com **números**, podemos adicionar, subtrair, multiplicar, dividir... e muitas outras operações, tais como a comparação.

Se estivermos trabalhando com **textos**, podemos comparar (se forem os mesmos ou não), podemos substituir parte do texto, adicionar texto a um texto existente, etc.

Com **datas** também podemos adicionar ou subtrair datas (ou dias, meses, anos), compará-las, etc.

Tipos de dados para variáveis

Cada linguagem de programação pode trabalhar com muitos tipos de dados.

Mas de todos eles, teremos sempre os **tipos de dados primitivos**.

Estes são incorporados na linguagem de programação e nos ajudam a fazer coisas mais complicadas.

Vamos falar sobre os tipos de dados mais comuns, que podemos encontrar na maioria das linguagens de programação, tais como Java, C ou C++.

Número do tipo de dados: int

Os números são frequentemente representados em uma linguagem de programação de diferentes maneiras, pois é importante decidir uma série de questões:

- Qual o tamanho que vamos usar.
- Se vai ou não ter números decimais.
- Se for negativo.

Vamos começar com o tipo primitivo `int`. Este tipo de dado representa qualquer número sem casas decimais, seja positivo ou negativo.

Embora seja comum encontrá-lo escrito no código fonte como `int`, existem outros idiomas, como o visual básico, onde é escrito `integer`.

Tipos de dados número real: double ou float

Se estivermos interessados em utilizar um número com casas decimais, normalmente encontramos o tipo de dado `double` ou `float`.

Chamamos isto de número de ponto flutuante. Declará-lo como um ou outro tipo dependerá da precisão decimal que você deseja ter. Os `float` são utilizados para números não muito grandes com decimais e o tipo `double` para números muito grandes com decimais.

Como podemos distinguir uma variável se ela foi declarada como `int` ou como `float`? Por exemplo, se encontrarmos um número com um ponto decimal (**3,14**), deduz-se que ele é do tipo `float`. Entretanto, podemos definir um número sem casas decimais como um flutuador, portanto o programa adicionará as casas decimais. Por exemplo, se definirmos uma variável como um `float` e lhe dermos o valor de 456, nosso programa a lerá como 456,00.

Você também pode ver um número com a letra F ou com a letra D, para distinguir se é um `float` ou um `double` . Por exemplo, **3,56F**.

Tipos de dados de corda: char ou string

Geralmente é um valor alfanumérico. Se for um único personagem individual, temos o tipo `char`.

Um `char` é um personagem Unicode, e geralmente está entre aspas simples (' ').

Mas se for uma cadeia de caracteres, ou seja, caracteres um após o outro formando uma seqüência (uma palavra ou uma frase), normalmente a encontramos como `string`.

O tipo de `string` deve ser escrito entre aspas duplas (" ") para diferenciá-lo do `char`, embora possa ser diferente, dependendo da linguagem de programação.

No código a seguir, veremos o uso do tipo de dados da `string`:

```
miVariable = "Olá Mundo";
```

Declaramos uma variável na qual introduzimos como valor a frase "Olá Mundo", portanto, esta variável é do tipo `string`.

Tipo	Tamanho em bytes	Exemplo
Int	2	1, 55, 73, 1500
Float	4	4.33, 5.92, 75.22, $5e^{-2}$
Double	8	7.518, 9.513, $7e^{-5}$
Char	1	T, Y, %, ¡, #, 52
Void	0	No hay valor

Tipo de datos booleano: boolean

Os valores lógicos são representados pelo tipo primitivo `boolean`. Ela representa se uma condição é ou não cumprida.

Eles geralmente têm dois valores identificados, `true` ou `false` . Em alguns idiomas, pode ser equivalente aos números **0 e 1**.

Uma variável pode ser usada para representar qualquer um de dois **valores**.

Por exemplo, poderíamos falar de verdadeiro ou falso, ligado ou desligado, ativo ou não ativo, etc.

Tipos de dados abstratos

Agora que sabemos o que são os tipos de dados primitivos, podemos saber o **que significam os tipos de dados abstratos**.

Os **tipos de dados na programação orientada a objetos** são geralmente os básicos que já vimos antes, mas podemos encontrar tipos de dados que são classes ou objetos. Isto será discutido com mais profundidade quando analisarmos a programação orientada a objetos.

Os **tipos de dados abstratos** aumentam e ampliam a complexidade, uma vez que têm um *conjunto de valores* e *operações* associadas a eles.

Então entraríamos no *encapsulamento* destes dados, herança para aproveitar melhor as operações que codificamos e *polimorfismo*, mas estes são conceitos que serão explicados mais tarde.