



Tipos de Dados

Apontamentos sobre os tipos de dados na programação, int, float, double, char, boolean, string...

Page

- Para armazenar um qualquer dado precisamos de uma "variável"
- A cada variável vamos:
 - Atribuir um tipo de dados
 - Identificá-la por um nome
 - Atribuir um dado (valor)
 - Poder alterar o valor armazenado
- Antes de usar uma qualquer variável temos de a declarar
- Para isso é necessário saber qual vai ser o seu uso para assim saber qual o tipo que a variável deve ter
- Cada variável vai ser armazenada em memória e o espaço que ocupa depende do seu tipo
- Para definir uma variável usamos a sintaxe `tipo nomeVariavel;`
- Podemos definir mais que uma variável do mesmo tipo de as separarmos por ,

JavaScript ▾

```
int i; // declaração de uma variável do tipo inteiro

char ch, opcao; // declaração de 2 variaveis do tipo char

float f, raizquad, pi // 3 variaveis do tipo float

double constanteKepler, numeroNeper; // 2 doubles
```

- **Regras do Java para atribuir nomes:**
 - Só pode conter caracteres de a...z, A...Z, 1...9 e _
 - Não pode começar por um número
 - Não podem ser iguais às palavras reservadas do Java
 - Não se podem definir 2 variáveis com o mesmo nome dentro do mesmo bloco de código
 - Não podem ter espaços
- **Diretrizes/Sugestões da escrita na linguagem Java:**
 - Deve começar com letra minúscula
 - Deve ser suficientemente esclarecedor do seu significado, sem ser demasiado extenso
 - Evitar abreviaturas que podem ser enganosas
 - Não usar acentos nem caracteres regionais
 - Usar o tipo de escrita CamelCase (começar com letra minúscula e cada nova palavra começar com letra maiúscula, de forma a fazer o efeito das bossas do camelo)

- Não esquecer que o compilador faz distinção entre minúsculas e maiúsculas

JavaScript ▾

```
int estaVariavelServeParaCalcularODiaDeAnos;    // mau exemplo, nome muito extenso
int diaDeAnos                                   // este nome é melhor

int x,y,z                                       // mau exemplo, nomes ambíguos, exceto
                                                // exceto se representarem coordenadas

double e;                                       // mau exemplo, nome ambíguo
double numero Neper                            // ERRO, não pode levar espaços
double numeroNeper                             // boa solução

char existeVidaET?                             // ERRO, não se pode usar o ?
```

- Depois de declarada a variável pode-se armazenar lá um valor:
 - Em linguagem de programação chama-se atribuição
 - Quando se coloca um valor numa variável `var` diz-se que se está a atribuir esse valor a `var`
- Uma variável só pode armazenar um único valor
 - Se se atribuir outro valor à mesma variável, o anterior perde-se
- A atribuição é feita usando a sintaxe: `variável = expressão`
- A variável a usar é sempre colocada do lado esquerdo do sinal `=` e o valor a atribuir do lado direito

JavaScript ▾

```
umaVar = 12;

uma Var = -500;

umaVar = 2 * 50;
```

- Quando se declara uma variável ela é logo inicializada com um valor
 - Há compiladores que inicializam o valor a zero
 - Em Java é o que acontece
- Deve-se sempre que possível, inicializar a variável na sua declaração

Java ▾

```
int umaVar = 12; // declaração e incialização da variável
```

- Em Java há vários tipos de dados predefinidos, entre eles:
 - `int`
 - `float`
 - `double`
 - `char`
 - `boolean`
- Existe ainda outro tipo de dados que se pode considerar:
 - `String`

Números Inteiros

- O tipo `int` representa um número inteiro, ou seja, um número do conjunto de números naturais
- Pode representar números positivos ou negativos

Java ▾

```
int umaVar;           // variável do tipo int

umaVar = -12;         // pode assumir número positivo

umaVar = -500;        // ou negativo
```


- Operações que se podem efetuar com inteiros:

Java ▾

```
int num1 = 10, num2 = 3
int res;
```

Operação	Descrição	Exemplo	Resultado
+	Soma	<code>res = num1 + num2;</code>	13
-	Subtração	<code>res = num1 - num2;</code>	7
*	Multiplicação	<code>res = num1 * num2</code>	30
/	Divisão inteira	<code>res = num1 / num2</code>	3
%	Resto da divisão	<code>res = num1 % num2</code>	1

- O resultado da divisão de 2 números inteiros é SEMPRE um número inteiro
 - Por isso `10 / 3` não dá 3,333(3) mas sim 3

 Para saber se um número é divisível por outro basta calcular o resto da divisão. Se for zero é porque é divisível, senão não é.

- Além das operações normais ainda existem os operadores: `+=`, `-=`, `*=` e `/=`
 - A operação é a mesma mas o resultado é armazenado na própria variável
 - é apenas uma maneira mais eficiente

Java ▾

```
num1 += num2;
```

- Para somas e subtrações existem ainda os operadores `++` e `--`
 - Estes operadores equivalem a somar ou subtrair o valor da variável de 1
 - O resultado da soma fica na variável
 - O operador pode ficar no lado esquerdo (prefixo, `++var`) ou direito (sufixo, `var++`)
 - Se for sufixo primeiro é usado o valor para outro cálculo qualquer e só depois incrementado
 - Se for prefixo primeiro é incrementado e só depois é que é usado

Java ▾

```
int num1 = 10, int num2 = 3;
```

```
num1++      // num1 fica com o valor 11, equivale a fazer num1 = num1 + 1
++num2;     // num2 fica com o valor 4, equivale a fazer num2 = num2 + 1
```

Java ▾

```
int num1 = 10, int num2 = 3;
int res;

res = num1 * (num2++);    // res fica com o valor 10 * 3 = 30
                          // num2 fica com o valor 4
                          // equivalente a res = num1 * num2
                          // em que o num2 = num2 + 1
```

Java ▾

```
int num1 = 10, int num2 = 3;
int res;

res = num1 * (++num2);    // res fica com o valor 10 * 4 = 40
                          // num2 fica com o valor 4
                          // equivalente a num2 = num2 + 1
                          // em que o res = num1 * num2
```

⚠ O uso de operadores `++` e `--` em instruções compostas é desaconselhado pois o código pode ficar ininteligível

- Os números inteiros podem ser vistos como números binários, por isso existem os operadores: `<<`, `>>`, `<=<` e `>=>`
 - Cada um destes operadores desloca para a esquerda (`<<`) ou direita (`>>`) um determinado número de bits

Java ▾

```
int num1 = 2              // 2 em binário é 00000010
int num2;

num2 = num1 << 2;         // deslocar os bits de num1 2 posições para a esquerda, o resultado em num2 é
00001000 = 8

num2 >>= 2;               // deslocar os bits de num2 2 posições para a direita, o resultado em num2 é
00000010 = 2
```

ℹ Uma maneira fácil de multiplicar um número por 2 é deslocar esse número 1 bit para a esquerda

- O método `print` permite a escrita de valores inteiros

Java ▾

```
int num1 = 10;

System.out.println("O valor de num1 é " + num1);
```

Java ▾

```
int num1 = 10, num2 = 3;

System.out.println("O valor de num1 é " + num1 + " e num2 é " + num2);
```

- O método `printf` também permite a escrita de valores inteiros
 - Além disso permite a impressão formatada da saída
 - Para isso recorre-se a uma string de formatação
 - Cada formatação é precedida de `%`
 - `%d`, por exemplo, significa que o valor deve ser impresso como inteiro

Java ▾

```
int num1 = 10;

System.out.print("O valor de num1 é %d", num1);
```

Java ▾

```
int num1 = 10, num2 = 3;

System.out.print("O valor de num1 é %d" e num2 é %d, num1, num2);
```

- O tamanho de um inteiro (número de *bytes* que ocupa em memória) varia de processador e até de sistema operativo:
 - Num sistema de 32 *bits* ocupa 4 *bytes*
 - Num sistema de 16 *bits* ocupa 2 *bytes*
- No "sistema" Java um inteiro é de 4 *bytes* - 32 *bits*
- O número de *bytes* que uma variável ocupa influencia a gama de valores que pode representar
- Para permitir outras gamas o Java tem os seguintes inteiros
 - *byte* (8 *bits*)
 - *short* (16 *bits*)
 - *int* (32 *bits*)
 - *long* (64 *bits*)
- A tabela representa os vários tipos de inteiros, a sua ocupação de memória e a gama de valores

Tipo de inteiro	<i>bytes</i>	Mínimo	Máximo
<i>byte</i>	1	-128	127
<i>short</i>	2	-32 768	32 767
<i>int</i>	4	- 2 147 483 648	2 147 483 647
<i>long</i>	8	-1E20	1E20

Números Reais

- O tipo `float` representa um número do conjunto de números reais
- O tipo `double` também representa um número do conjunto de números reais, mas com maior precisão (o dobro)
- Ambos representam números com casas decimais
- Podem representar números positivos ou negativos
- Um número com casa decimal é, por defeito, um `double`
 - Para indicar que se trata de um `float` deve-se colocar `f` ou `F` no final do número

Java ▾

```
float umValor = 10.5f; // variável do tipo float
double PI = 3.1415927; // valor de PI

umValor = 12.5E14F; // corresponde a 12.5x10^14, note-se o uso do F

umValor = -50.0F // também pode ter valores negativos
```

- Operações que se podem efetuar com números reais:
 - Todas as que se podem usar com inteiros

- exceções: %, %=, <<, >>, <= e >=
- Os significados das operações mantêm-se os mesmos
 - exceto que agora as divisões retornam um número real (não há resto)
- Escrever reais:
 - O método `print` também permite a escrita de valores reais

Java ▾

```
double PI = 3.1415927, numeroNeper = 2.71828183;

System.out.print("O valor de PI é " + PI + " e o '\n' é " + numeroNeper);
```

- O método `printf` também permite a escrita de valores reais
 - %f apresentar como número decimal
 - %e ou %E apresentar em notação científica
 - %g ou %G apresentar em notação ou decimal consoante o número concreto

Java ▾

```
double PI = 3.1415927, numeroNeper = 2.71828183;

System.out.printf("O valor de PI é %f e o '\n' é %E, PI, numeroNeper");
```

- O espaço em memória ocupado por um real é:
 - Nos `floats` de 4 bytes
 - Nos `doubles` de 8 bytes

Caracteres

- O tipo `char` representa um, e um só, caractere
- Para representar caracteres em máquinas que só sabem trabalhar com números é usada uma codificação
 - Isto significa que a cada caractere corresponde um número
 - A tabela ASCII é a codificação mais conhecida
 - Por exemplo o caractere 'a' é representado pelo número 97
 - No caso do Java usa-se a codificação UTF que usa 2 bytes
- Assim para atribuir o valor a um caractere pode-se optar por colocar o seu código ou o próprio caractere entre plicas "

Java ▾

```
char umaLetra;           // umaLetra é uma variável do tipo char

umaLetra = 'a';          // umaLetra assume o código de a, que é 97

umaLetra = 97;           // umaLetra assume o código 97 que corresponde a 'a'

// as 2 expressões são equivalentes, mas a primeira é mais compreensível
```



Um erro comum com caracteres é assumir que uma variável do tipo `char` pode assumir vários caracteres, mas como todas as variáveis, só pode assumir um e um só caractere

- Operações qe se podem efetuar com caracteres:
 - Todas as que se podem usar com inteiros

- Os significados das operações mantêm-se os mesmos

```
Java ▾  
  
char umaLetra = 'a'      // umaLetra assume o código de a, que é 97  
  
umaLetra++;              // incrementa-se o código da letra + 1  
                          // umaLetra passa para o código 98 que representa o caractere 'b'  
  
umaLetra -= 32;           // 98 - 32 = 66  
                          // 66 é o código ASCII do caractere 'B'
```

- Escrever caracteres:

- O `print` permite a escrita de caracteres

```
Java ▾  
  
char umaLetra = 'a';  
  
System.out.print("umaLetra representa o caractere: \"'\" + umaLetra + "\"'\");
```

- o `printf` também
 - `%c` significa que é um caractere

```
Java ▾  
  
char umaLetra = 'a';  
  
System.out.printf("umaLetra representa o caractere: \"'%c'\"", umaLetra);
```

- O espaço em memória ocupado por um caractere é 2 bytes
 - Não esquecer um caractere é, na realidade, um número

Booleanos

- O tipo `boolean` representa um valor lógico
 - `true` - para valor lógico verdadeiro
 - `false` - para valor lógico falso
- Ocupa apenas um *bit*

```
Java ▾  
  
boolean estaAberto = false;    // está aberto ou fechado (false = fechado)  
  
boolean estaLigado;            // estaLigado é uma variável do tipo booleano  
  
estaLigado = true;              // assume o valor verdade  
  
estaLigado = false;            // assume o valor falso
```

- Operações com booleanos
 - Os booleanos não aceitam as operações tradicionais
 - As que podem ser usadas são
 - `!` Negação do valor
 - `&&` e (*and*)
 - `||` ou (*or*)

Java ▾

```
boolean estaAberto = false;

estaAberto = !estaAberto;    // muda o valor de estaAberto (passa a true)
```

- Escrever booleanos

- Usando `print`

Java ▾

```
boolean estaAberto = false;    // está aberto ou fechado (false = fechado)

System.out.print("O interruptor está aberto? " + estaAberto);
```

- Usando `printf`

- Usar o `%b`

Java ▾

```
boolean estaAberto = false;    // está aberto ou fechado (false = fechado)

System.out.print("O interruptor está aberto? %b", estaAberto);
```

Conversões

- Por vezes é necessário fazer conversões entre tipos:
 - Somar um inteiro a um real
 - Usar um caracter como se fosse um inteiro
 - ...
- Essa conversão pode ser automática, ou expressa pelo programador
- Só se podem fazer conversões entre tipos numéricos
 - O tipo `boolean` não pode ser convertido para nenhum dos outros
- Exemplos de conversão automática
 - Aquando da atribuição
 - Quando se pretende atribuir um valor de um tipo a outro
 - O compilador só o permite se o tipo final for "superior"

Java ▾

```
char umChar;
int umInt;
float umFloat;

umInt = umChar;    // conversão automática (char é inferior a int)

umFloat = umInt;    // conversão automática (int é inferior a float)

umInt = umFloat;    // não há conversão automática (float é superior a int), origina erro
```

- Em operações aritméticas
 - Se os operandos de uma operação de tipos diferentes todos são promovidos ao maior tipo

Java ▾

```
char umChar;
int umInt;
float umFloat;
```



```
umInt = umChar + 10;           // conversão automática (umChar é promovido a int)

umFloat = umInt * 2.3f;        // conversão automática (umInt é promovido a float)
```

- Para forçar conversões de tipos usa-se o **casting**
 - Coloca-se entre parênteses o tipo para o qual queremos converter
 - O compilador faz as conversões, mesmo quando se perde resolução

Java ▾


```
char umChar;
int umInt;
float umFloat;

umInt = (int)umChar;           // conversão explícita

umFloat = (float)umInt;        // conversão explícita

umInt = (int)umFloat;          // conversão explícita (ATENÇÃO: perda de resolução)

umInt = (int)(umFloat / 230);  // conversão explícita do resultado da operação
```

 Input Simples