



Aula 2

Variáveis, Funções, Tipos de Dados e Operadores

Iniciativa Conjunta:



Com o apoio de:





Tipos de Dados

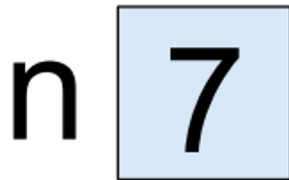
- Restringem os valores que uma expressão, como uma variável ou função, podem assumir.
- Definem as operações que podem ser feitas nos dados, o significado dos dados e a maneira como os valores desse tipo podem ser armazenados.

int	<i>inteiros</i>	1, 2, -1, 0, 80000
double	<i>decimais</i>	3.1416, 0.01, -1.2
char	<i>caracteres</i>	a, '8', '?'
boolean	<i>booleanos</i>	true, false
...		

O que são variáveis?

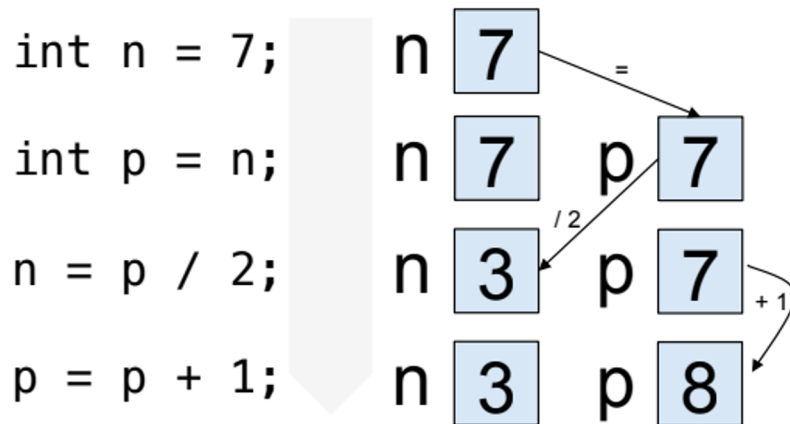
- Uma variável pode ser vista como um espaço em memória onde um valor de determinado tipo pode ser guardado.
- As variáveis têm três características
 - **Tipo**
 - **Nome**
 - **Valor**
- Ao definir uma variável estamos a indicar o seu **tipo**, o seu **nome** e o seu **valor** inicial.

```
int n = 7;
```



Variáveis - Atribuição

- O valor das variáveis é definido usando o operador de atribuição (=)





Instruções e Blocos de Instrução

- Uma instrução é uma ação na execução do programa que pode mudar o seu estado (variáveis). Em Java, uma instrução termina sempre com um ponto-e-vírgula (;)

```
int n = 0;  
int p = 0;
```

- Um bloco de instruções é um conjunto de instruções entre chavetas que será executado sequencialmente

```
{  
    n = n + 1;  
    p = n % 2;  
}
```

Funções

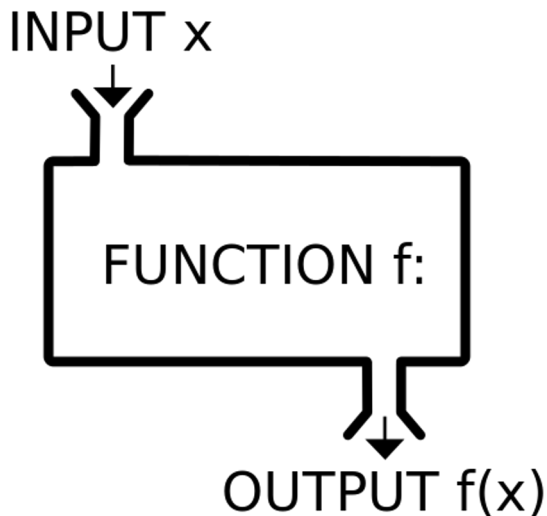
- Em Matemática, uma função $f()$ associa a um argumento x , um valor y . Ou seja, $y = f(x)$.

Se $f(x) = x^2$
então $f(3) = 9$

- Em Programação, o conceito de função é similar:

```
static int square (int x) {  
    return x * x;  
}
```

- As funções são **implementações** de métodos de resolução de problemas.





Funções - Assinatura

- Para que seja possível utilizar uma função, tal como na Matemática, esta tem de ter um nome. O nome da função deve indicar aquilo que a função calcula.
- Uma função tem uma assinatura, onde se define o **nome**, o **tipo do valor** devolvido (o que sai), e os **parâmetros** da função (o que entra)

```
static int square (int x) {  
    ...  
}
```



Funções - Corpo

- O corpo de uma função é a explicação ao computador do método de resolução de um problema. Esta definição aparece a seguir à assinatura da função, entre chavetas.

```
static int square (int x) {  
    return x * x;  
}
```




Funções - Parâmetros e Argumentos

- Os parâmetros indicam a que informação é que a função se aplica.

x é um parâmetro do tipo *int*

- Em termos matemáticos, diríamos que o domínio da função é o conjunto dos inteiros
- Os argumentos são os valores que se dão aos parâmetros

$0, 2, 3, -1$ são possíveis argumentos



Funções - Parâmetros e Argumentos

- Uma função pode ter vários parâmetros, por exemplo:

```
static int sum (int a, int b) {  
    ...  
}
```

- (2, 3), (-1, 5), (10, 20) são pares de possíveis argumentos



Funções - Retorno

- A assinatura da função define que o tipo de devolução é um inteiro (**int**)

```
static int sum (int a, int b) {  
    return a + b;  
}
```
- Em termos matemáticos, diríamos que o contradomínio da função é o conjunto dos inteiros.
- A instrução **return** indica qual o valor devolvido pela função.
 - O valor devolvido tem de ser compatível com o que é indicado na assinatura da função (neste caso, um número inteiro).
 - A expressão **x*x** calcula o quadrado de **x** e a instrução **return** devolve o valor calculado.



Conversões entre Tipos Numéricos

- int para double

```
(double) 8 -> 8.0
```

- double para int

```
(int) 7.3 -> 7
```

```
(int) 7.3 -> 7
```

- Esta conversão não corresponde a um arredondamento.
O número decimal é **truncado!**



Operadores Aritméticos

+	<i>adição</i>
-	<i>subtração</i>
*	<i>multiplicação</i>
/	<i>divisão</i>
%	<i>resto da divisão inteira</i>

O operador da divisão, ao ser utilizado com dois valores inteiros, efectua uma divisão inteira ($7/2 \rightarrow 3$).




Operadores Relacionais

==	<i>igual</i>
!=	<i>diferente</i>
<	<i>menor</i>
>	<i>maior</i>
>=	<i>maior ou igual</i>



Operadores Lógicos

&&	<i>conjunção (e / and)</i>
	<i>disjunção (ou / or)</i>
!	<i>negação</i>



Funções - Exemplo

```
static boolean isEven (int n) {  
    return n % 2 == 0;  
}
```

- Esta função verifica se um dado número **n** é par.
- O tipo de retorno é booleano (**boolean**).
 - A função devolve verdadeiro (**true**) se **n** é par, e falso (**false**) caso contrário.
- O resultado da expressão **n % 2** irá ser **0** ou **1**. Caso seja igual a zero (**== 0**), então a expressão **n % 2 == 0** é verdadeira, caso contrário é falsa. Esta avaliação determinará o valor a devolver.



Funções - Invocação

- As funções na programação podem ser utilizadas numa determinada instrução pelo computador através de uma **invocação**.
- Uma invocação é composta pelo **nome** da função que se pretende invocar, seguida dos **argumentos** a passar nessa função:

```
int m = min(5, 8); m 5
```

- Os argumentos das funções têm de ser compatíveis com a assinatura da mesma. Por exemplo, se uma função tem como parâmetros dois inteiros, os argumentos terão de ser dois inteiros.



Variáveis como Argumentos

- O valor de um argumento pode ser dado usando uma variável
- O argumento será o **valor guardado na variável** no momento em que a função é invocada.

```
int a = 7;           a 7
```

```
int m = min(a, 12);  m 5
```



Expressões Matemáticas

- Na programação, tal como na matemática, podemos declarar expressões que representam algoritmos e contas:

`int a = 12 - 5;` a 7

`int m = 60 / 10 - 1;` m 5



Expressões como Argumentos

- O valor de um argumento pode ser dado usando uma expressão
- O argumento será o **valor da expressão** no momento em que a função é invocada.

```
int a = 7;           a 7
int m = min(a - 5, 12); m 2
```

Resultados de Funções como Argumentos

- O valor de um argumento pode ser dado usando o valor devolvido por uma função

```
int a = 7;           a 7
int m = min(max(a, 9), 8) m 8
```

- As invocações usadas como argumento são executadas primeiro, de modo a que o valor devolvido possa ser utilizado como argumento



Incrementar e Decrementar

- Algumas expressões podem alterar imediatamente o valor da variável em que são implementadas, como é o caso dos operadores de incrementar e decrementar

```
int n = 6;  
n++;
```

n 7

```
int p = 9;  
p--;
```

p 8



Funções Úteis

- Java, tal como quase todas as outras linguagens de programação, inclui por defeito diversos pacotes de funções de utilidade
- Estes contêm instruções para os processadores efetuarem determinadas contas ou retornarem valores que podem ser utilizados como versatilidade no código.

```
int n = (int) Math.random() * 10;
```

- Cada vez que é avaliada esta expressão pelo código, a função invocada vai retornar um valor diferente entre **0** e **1** que de seguida é multiplicado por **10** e convertido em inteiro. A variável **n** vai conter um valor diferente todas as vezes.



Exercício 1

- Criar uma função que tenha 3 argumentos e retorne a soma dos primeiros dois, multiplicando pelo terceiro argumento

```
static int sumMult (int a, int b, int c) {  
    ...  
}
```

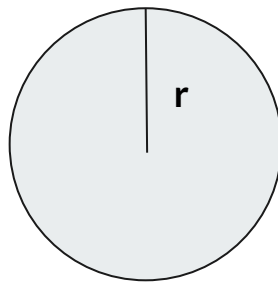
- E se quisermos multiplicar os primeiros dois e depois somar o terceiro?

Exercício 2

- Criar uma função que retorne a área de um círculo com raio r

```
static int circleArea(int r) {  
    ...  
}
```

$$\text{Área} = 2\pi r$$



- Dica: a expressão **Math.PI** devolve o valor de pi

Exercício 3

- Criar uma função que permita converter uma temperatura em graus Celsius para Fahrenheit
- Dica: quando em dúvida, o Google é vosso amigo 😊

