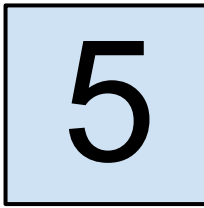


Invocação e recursividade

Invocação

Uma função pode ser utilizada numa instrução mediante uma **invocação**, composta pelo nome da função e argumentos.

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

m 

Os argumentos têm que ser compatíveis com a assinatura. Por exemplo, se a função recebe dois inteiros, os argumentos terão que ser dois inteiros.

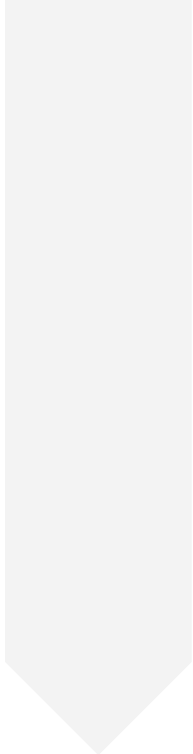
Variáveis em argumentos

Os argumentos não têm que ser um valor directo. Um argumento pode ser dado através de uma variável. Desta forma, o argumento será o valor guardado na variável no momento em que a função é invocada.

```
int a = 7;
```

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

```
boolean p = isPrime(a);
```



a

7

m

5

p

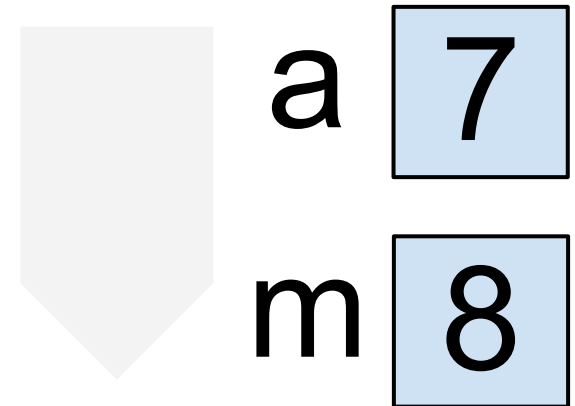
true

Resultado de funções em argumentos

Um argumento pode também ser dado em termos do resultado da invocação de uma função.

```
int a = 7;
```

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



As invocações dadas como argumento para uma função são executadas primeiro, de modo a que o seu resultado possa ser utilizado como argumento dessa mesma função.

Condições booleanas e funções

Uma função booleana (i.e. cujo tipo de retorno seja *boolean*) pode ser utilizada nas condições das estruturas de controlo. Exemplo:

```
static int countPrimesUpTo(int n) {  
    int nPrimes = 0;  
    int i = 1;  
    while(i <= n) {  
        if(isPrime(i)) {  
            nPrimes = nPrimes + 1;  
        }  
        i = i + 1;  
    }  
    return nPrimes;  
}
```

Recursividade

Quando uma função contém invocações a si própria a função é caracterizada como sendo **recursiva**.

Recursividade é um conceito fundamental em computação. Nalguns paradigmas de programação (p.e. funcional), a recursividade assume uma importância fulcral.

Dada a sua proximidade com as definições matemáticas, a "elegância" das definições de funções recursivas tornam-as atrativas em certos contextos.

Recursividade (exemplo Fibonacci)

Definição matemática da função para obter o n-ésimo número da sequência de Fibonacci:

$$F(n) = \begin{cases} 0, & \text{se } n = 0; \\ 1, & \text{se } n = 1; \\ F(n-1) + F(n-2) & \text{outros casos.} \end{cases}$$

Função recursiva em Java:

```
static int fib(int n) {  
    if(n <= 1) {  
        return n;  
    }  
    else {  
        return fib(n - 1) + fib(n - 2);  
    }  
}
```

Ciclos infinitos e recursividade

Invocações recursivas podem causar ciclos infinitos. Exemplo (cálculo de fatorial):

```
static int factorial(int n) {  
    return n * factorial(n - 1);  
}
```

Neste exemplo, embora o programa seja válido, resulta em ciclo infinito porque não há uma instrução na função que condicione a invocação recursiva. Desta forma, a função invoca-se a si mesma infinitamente.

Iteração

Um padrão comum na forma de utilizar variáveis consiste em efectuar **iterações**. Por exemplo:

```
static int divisors(int n) {  
    int c = 0;  
    int i = 1;  
    while(i <= n) {  
        if(n % i == 0) {  
            c = c + 1;  
        }  
        i = i + 1;  
    }  
    return c;  
}
```

O papel da variável **i** é tomar iterativamente os valores compreendidos no intervalo $[1, n]$.