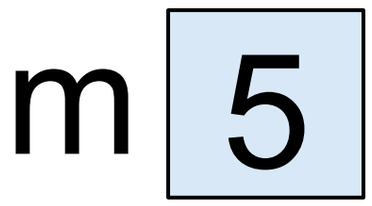
INVOCAÇÃO E RECURSIVIDADE

INVOCAÇÃO

- Uma função pode ser utilizada numa instrução através de um **invocação**

- Uma invocação é composta pelo **nome** da função que se pretende invocar seguido dos **argumentos** a passar a essa função

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



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

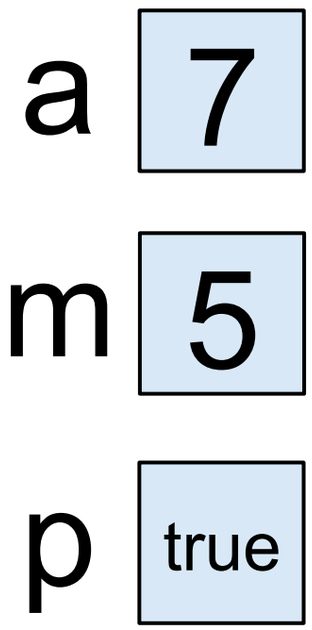
VARIÁVEIS COMO ARGUMENTOS

- 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;

int m = min(a, 5);

boolean p = isPrime(a);
```

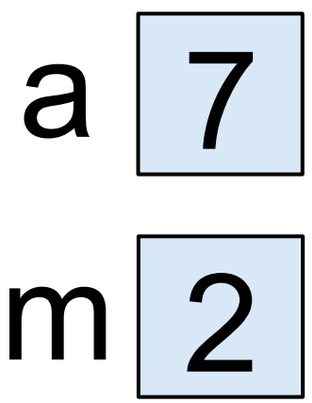


EXPRESSÕES COMO ARGUMENTOS

- 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;

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

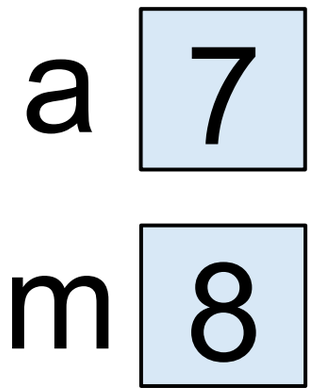


RESULTADOS DE FUNÇÕES COMO ARGUMENTOS

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

```
int a = 7;

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



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

INVOCAÇÕES NO CONTEXTO DE ESTRUTURAS DE CONTROLO

- Uma função booleana (i.e., cujo tipo de devolução seja `boolean`) pode ser usada nas condições das estruturas de controlo

```
static int numberOfPrimesUpTo(int n) {
    int numberOfPrimes = 0;
    int i = 1;
    while(i != n + 1) {
        if(isPrime(i)) {
            numberOfPrimes = numberOfPrimes + 1;
        }
        i = i + 1;
    }
    return numberOfPrimes;
}
```

RECURSIVIDADE

- Quando uma função contém **invocações a si própria**, esta é considerada uma função recursiva

- Recursividade é um conceito fundamental em computação. Nalguns paradigmas de programação (por exemplo, no paradigma 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: A SUCESSÃO DE FIBONACCI

- Definição matemática da função para obter o *n*-ésimo número da sucessão de Fibonacci

$$f(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ f(n-1) + f(n-2) & \text{outros casos} \end{cases}$$

- Função recursiva em Java

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

RECURSIVIDADE E INVOCAÇÕES INFINITAS

- Invocações recursivas podem causar um comportamento similar a um ciclo infinito. Exemplo (cálculo de fatorial):

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

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

PAPÉIS DAS VARIÁVEIS: ITERAÇÃO

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

```
static int numberOfDivisorsOf(int n) {
    int numberOfDivisors = 0;
    int i = 1;
    while(i != n + 1) {
        if(n % i == 0) {
            numberOfDivisors = numberOfDivisors + 1;
        }
        i = i + 1;
    }
    return numberOfDivisors;
}
```

O papel da variável i é tomar **iterativamente** os valores inteiros do intervalo [1, n].

A RETER

- Invocação
  - Variáveis como argumentos
  - Expressões como argumentos
  - Invocação de funções como argumentos
- Recursividade
  - Funções que se invocam a si próprias
  - Invocações infinitas
- Papeis das variáveis
  - Iteração