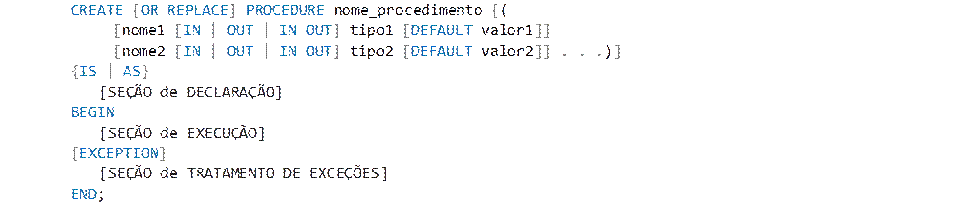
**Procedimentos, Funções e Packages**

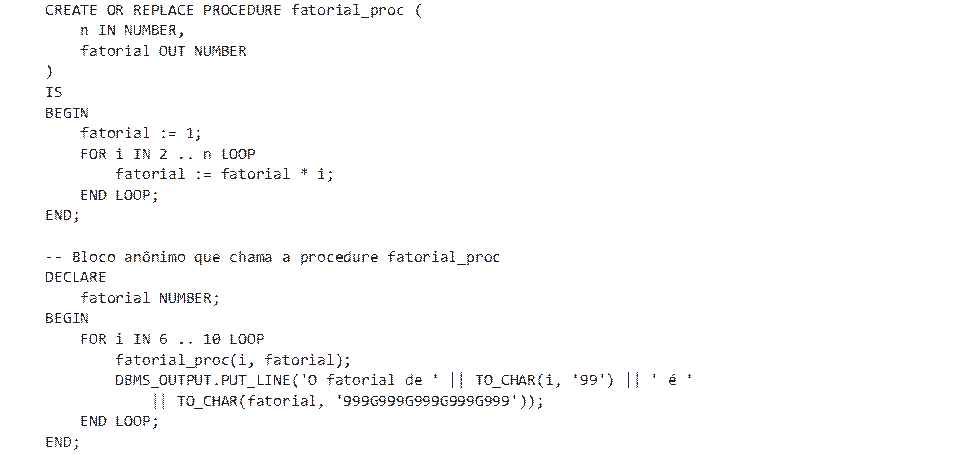
* **PROCEDIMENTOS ARMAZENADOS (STORED PROCEDURES)**

Um procedimento armazenado, também conhecido como stored procedure, é um bloco PL/SQL identificado, reutilizável, armazenado como um objeto no banco de dados. Diferentemente dos blocos anônimos, um procedimento pode ter parâmetros de entrada, saída ou entrada/saída. Sua forma geral é:

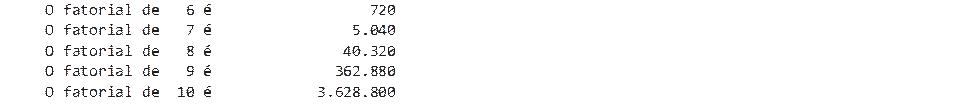


O comando CREATE PROCEDURE cria a stored procedure. Após o nome, há uma relação de parâmetros entre parênteses. Se não houver parâmetros declarados, os parênteses devem ser omitidos.

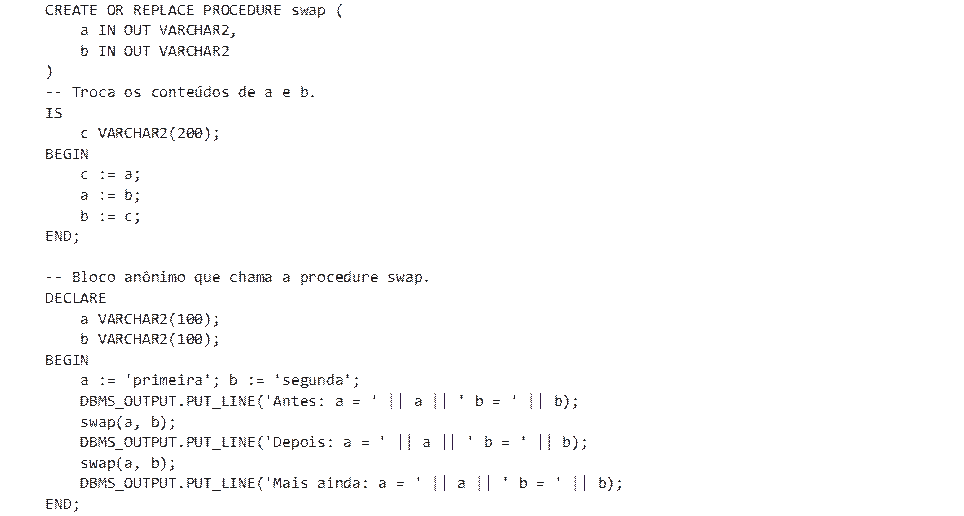
Cada parâmetro declarado pode ser de entrada (IN), de saída (OUT) ou de entrada e saída (IN OUT). Se não for especificado, o compilador assume como parâmetro de entrada (IN). A execução da procedure termina quando é encontrado o comando RETURN ou o END do bloco principal. O exemplo a seguir cria a stored procedure fatorial\_proc, que retorna o fatorial do número inteiro n.



Na procedure fatorial\_proc, foram declarados 2 parâmetros, um de entrada, n, e outro de saída, fatorial, que retorna o valor calculado para o bloco que chamou a procedure. O resultado, após a execução do bloco anônimo, é mostrado a seguir.



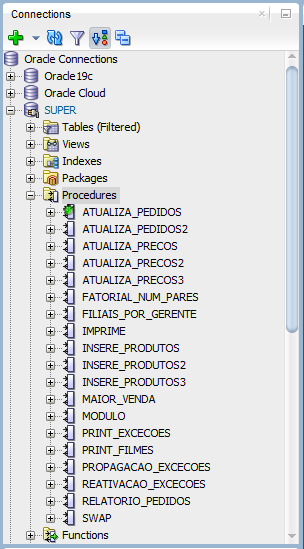
No exemplo a seguir, a procedure swap troca o conteúdo de duas variáveis do tipo VARCHAR2.  Ambos os parâmetros são do tipo IN OUT, fornecendo os valores para a procedure e retornado o resultado. Observe que não é necessário definir o tamanho do tipo VARCHAR2 para os parâmetros a e b (na realidade, não é permitido). Isto vale para todos os tipos em que é possível definir o tamanho.



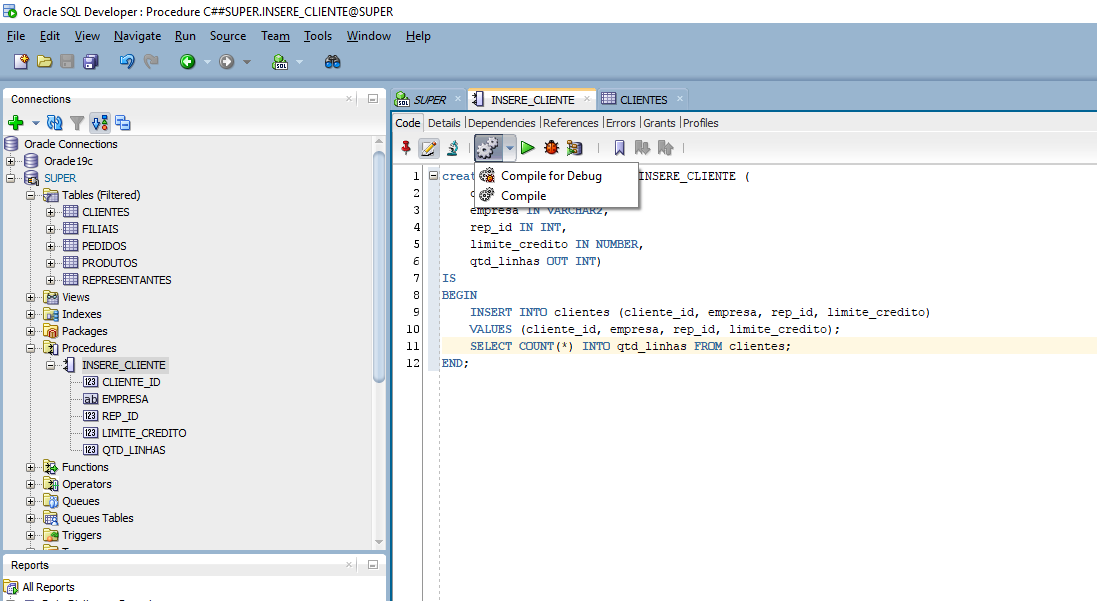
O resultado, após a execução do bloco anônimo, é mostrado a seguir.​

https://paperx-dex-assets.s3.sa-east-1.amazonaws.com/images/1679501928551-mVzCcmZo8k.png

​Deve-se executar o bloco de criação da procedure antes de qualquer referência a ela para que seja armazenada no banco de dados e passe a “existir”. Após sua criação, a procedure pode ser visualizada no explorador de objetos do SQL Developer:​​



A alteração de uma procedure é simples. Ao clicar sobre o seu nome, no explorador de objetos, uma janela de edição é aberta. Basta fazer as alterações e clicar em Compile que as alterações são verificadas e, não havendo erros, a nova versão é armazenada. Na figura a seguir é mostrada a janela de edição do SQL Developer com a indicação do comando Compile.



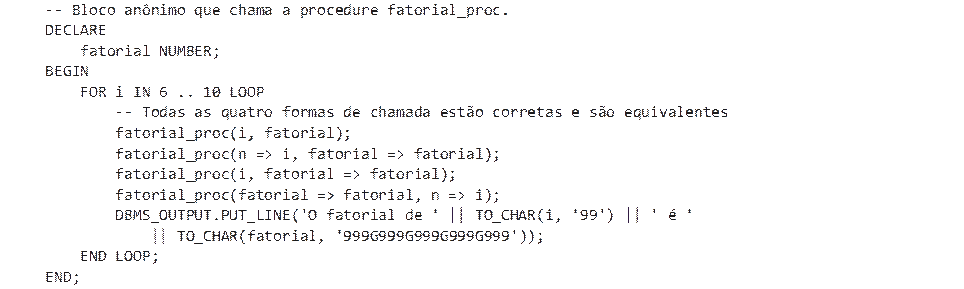
Pode-se remover uma procedure de duas formas: através do comando DROP PROCEDURE nome ou no explorador de objetos, clicando com o botão direito do mouse sobre a procedure e selecionando DROP.

            Os parâmetros declarados na definição da procedure são chamados de **parâmetros formais**, enquanto aqueles utilizados na chamada da procedure são chamados de **parâmetros atuais**. No momento de chamada de uma procedure e antes de sua execução, os parâmetros atuais devem ser mapeados nos parâmetros formais.

            Na chamada de uma procedure, os parâmetros atuais podem ser associados aos formais pela posição (referência posicional) ou associando-os explicitamente (referência por nome)

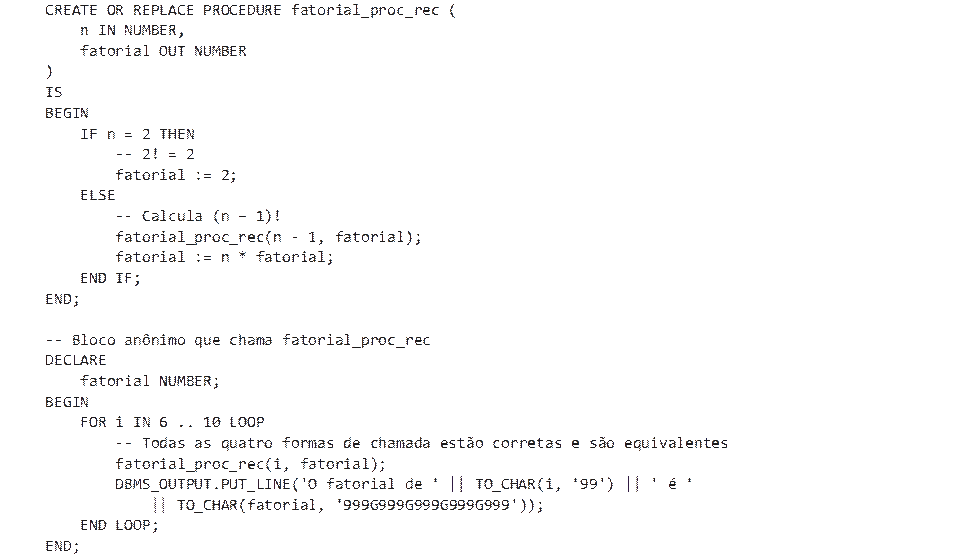
            Na referência posicional, a associação é feita pela ordem dos parâmetros atuais na chamada. Nesta forma, a quantidade e os tipos dos parâmetros atuais devem ser iguais aos dos parâmetros formais. Esta é a forma utilizada nos exemplos anteriores.

            Na referência por nome, a associação é feita utilizando-se a notação parâmetro\_formal => parâmetro\_atual. A ordem e quantidade de parâmetros atuais não precisam ser iguais às dos formais. Deve-se tomar cuidado, porém, para que sejam atribuídos valores padrão aos parâmetros não obrigatórios. Além disto, não é permitido referências por nome antes de referências posicionais. No exemplo a seguir, o bloco anônimo chama a procedure fatorial utilizando as duas formas de passagem de parâmetros.

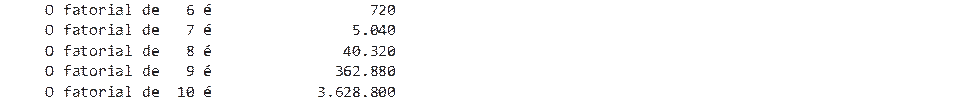


Uma procedure pode chamar outras procedures e até a si própria. Nesta última situação, a procedure é dita recursiva. Deve-se tomar muito cuidado com procedures recursivas para que elas não fiquem se chamando indefinidamente. Toda procedure recursiva deve ter uma condição de saída.

            Considere o fatorial de um número n.  Pela definição “tradicional”, n! = n ∙ (n – 1) ∙ (n – 2) ∙ … ∙ 2 ∙ 1. Pode-se definir o fatorial de forma recursiva: n! = n ∙ (n – 1)!. O exemplo a seguir implementa recursivamente a função fatorial\_proc\_rec.

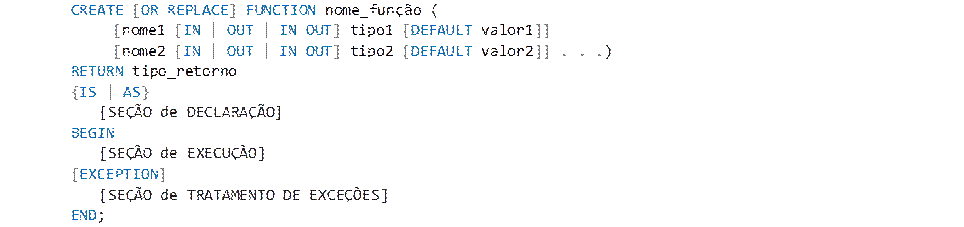


O resultado, após a execução do bloco anônimo, é mostrado a seguir.​



* **FUNÇÕES ARMAZENADAS OU DEFINIDAS PELO USUÁRIO (STORED FUNCTIONS)**

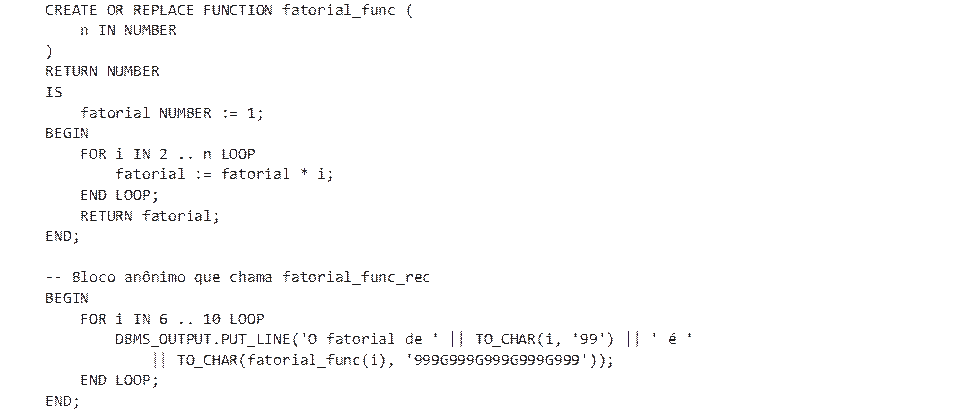
Uma função definida pelo usuário é muito similar a uma procedure exceto pelo fato de que a função retorna um valor. Sua forma geral é:



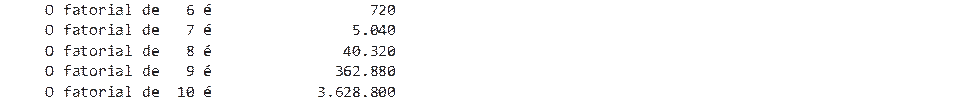
A declaração de parâmetros e as seções obrigatórias e opcionais são idênticas às de uma procedure. O tipo\_retorno define o tipo do valor retornado e pode ser qualquer tipo válido em PL/SQL. O comando RETURN deve ser utilizado para encerrar uma função, retornando o resultado ao chamador. Um erro ocorre se a execução chegar ao END do bloco principal.

            O valor retornado pela função deve ser compatível com o tipo declarado. O compilador PL/SQL faz automaticamente a conversão de tipos, quando possível. Por exemplo, um valor FLOAT retornado por uma função com tipo declarado de retorno NUMBER é automaticamente convertido. Já o retorno de um valor VARCHAR2 por esta mesma função causará erro.

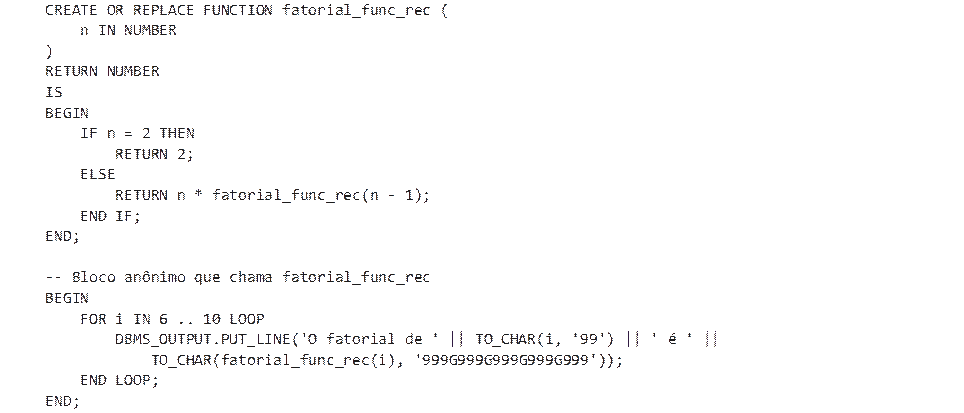
            Como nas procedures, os parâmetros podem ser passados por posição, por nome ou combinando as duas formas. O exemplo a seguir implementa a procedure fatorial\_func através de uma função.



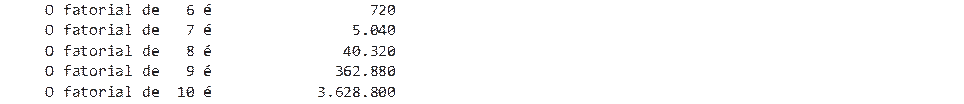
O resultado, após a execução do bloco anônimo, é mostrado a seguir.​



​Assim como ocorre com as procedures, as funções também podem ser recursivas. O exemplo a seguir apresenta a função fatorial\_func\_rec, que calcula o fatorial de um número inteiro, implementada através de uma função recursiva.​​



O resultado, após a execução do bloco anônimo, é mostrado a seguir.​



Como procedures e funções são objetos armazenados no banco de dados, o seu escopo é o banco de dados onde foram definidas. Deve-se tomar cuidado para não utilizar identificadores repetidos (nomes) para procedures e funções. Alteração e exclusão de funções são feitas da mesma forma que em procedures, através do explorador de objetos ou através do comando DROP FUNCTION.

### *O QUE SÃO PACOTES (*PACKAGES*)?*

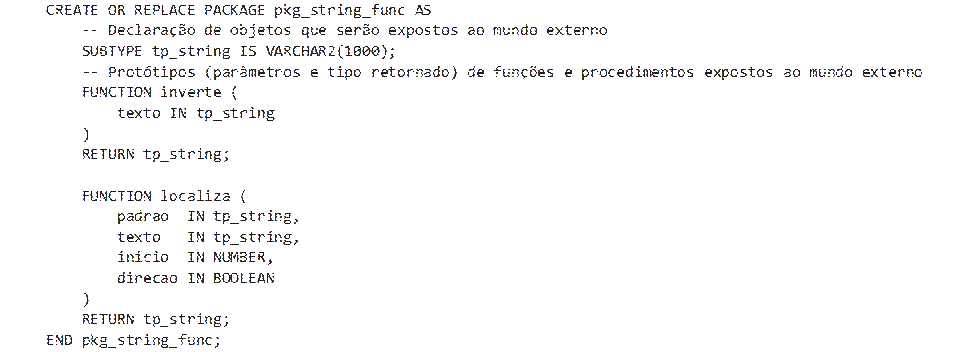
Packages são objetos que agrupam elementos da linguagem PL/SQL, tais como variáveis, constantes, tipos, cursores e subprogramas, que guardam relação entre si. Packages se assemelham a módulos de algumas linguagens de programação, como Python. O uso de packages oferece ao desenvolvedor inúmeras vantagens:

1.     **Modularidade.** Packages “empacotam” itens que guardam relação lógica entre si, além de oferecerem uma interface simples ao desenvolvedor. O entendimento da função de cada item de uma package é mais simples, agilizando o desenvolvimento de aplicações;

**2.     Isolamento de detalhes.** Uma package possui duas partes: especificação (interface) e corpo (implementação da lógica). Apenas a sua especificação é exposta aos desenvolvedores, escondendo detalhes de implementação desnecessários ou mesmo confidenciais;

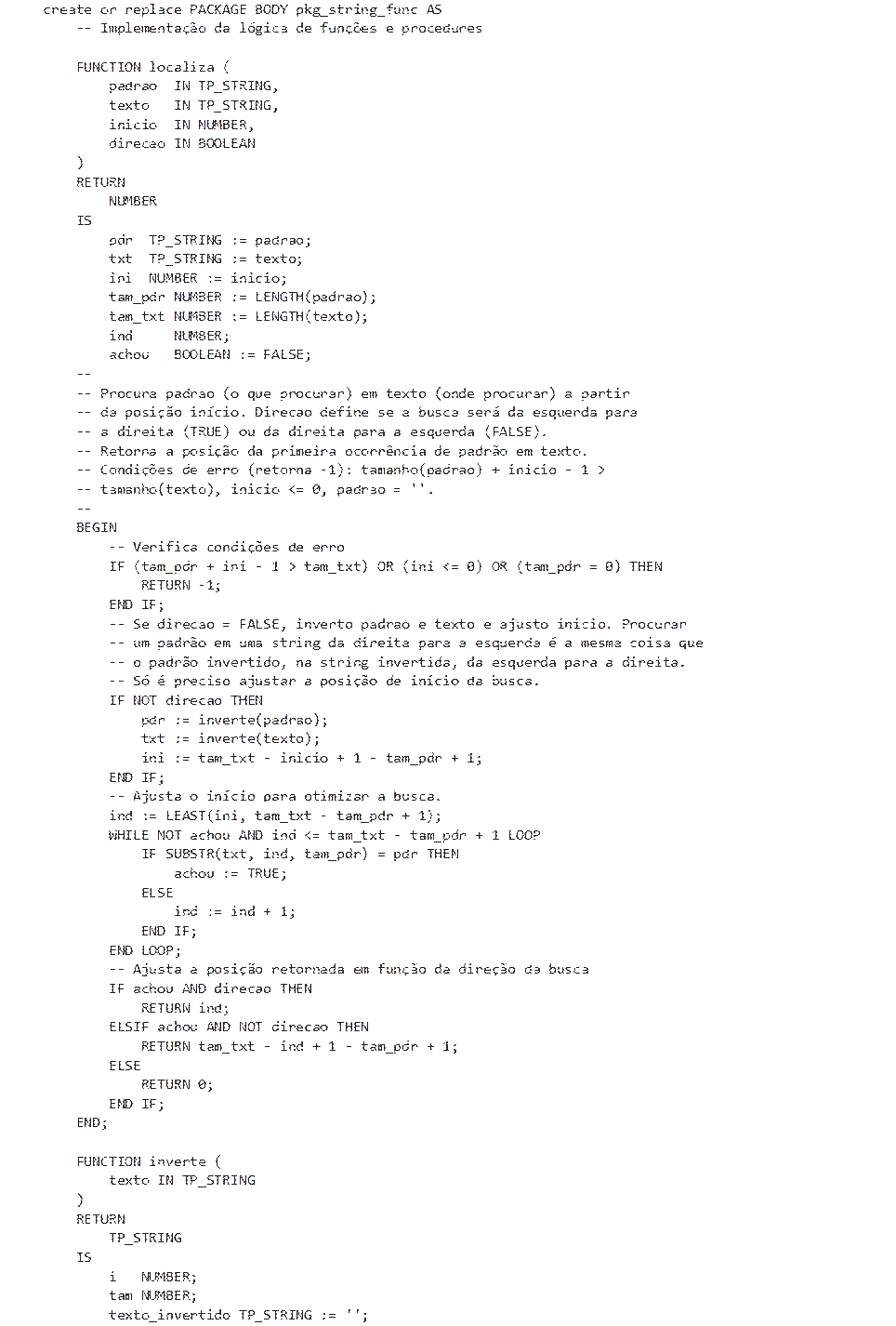
**3.     Objetos globais e persistentes.** Todos os objetos declarados na especificação de uma package são globais ao esquema ao qual pertencem. Todos os objetos do esquema podem referencia-los. Além disto, as variáveis e cursores globais de uma package têm seus valores preservados ao longo da sessão.

A especificação de uma package envolve a declaração de objetos que serão expostos aos desenvolvedores. Nela, apenas aquilo que é necessário saber a respeito dos objetos é declarado. A declaração de procedures em uma package, por exemplo, só contém a relação de parâmetros (e seus tipos). No exemplo a seguir, é mostrada a declaração de uma package que implementa duas funções úteis: inverte e localiza. A primeira, inverte(texto), recebe como parâmetro de entrada texto do tipo TP\_STRING, declarado dentro da própria package, retornando o seu conteúdo invertido. A diretiva SUBTYPE define um novo tipo a partir de um tipo atômico (no caso VARCHAR2). A segunda função, localiza(padrão, texto, inicio, direção) retorna a posição da primeira ocorrência de padrão em texto, a partir da posição inicio. Até aí nenhuma novidade. A função pré-definida INSTR() faz exatamente isto. A diferença está no parâmetro direção, que define se a busca será feita da esquerda para a direita (padrão nas funções de busca) ou na direção oposta. [1] No próprio corpo da função há vários comentários que descrevem a sua implementação.



Observe que, na especificação da package, apenas as informações para chamada de funções e os tipos declarados estão disponíveis. Com estas informações, é possível aos desenvolvedores escreverem programas que utilizam estes objetos.

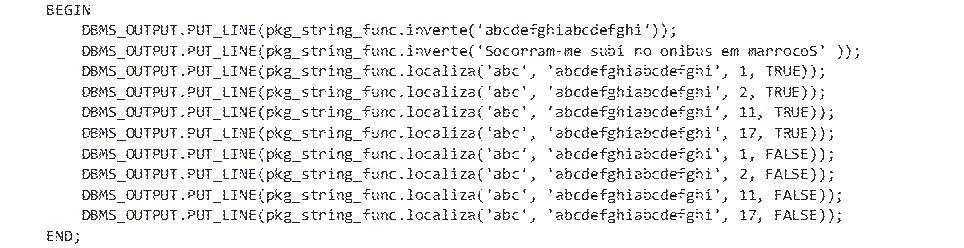
O corpo da package é declarado através do comando CREATE PACKAGE BODY, utilizando-se o mesmo nome utilizado em sua especificação. Nele, a lógica das funções e procedures são implementados. No exemplo a seguir, é mostrada a declaração do corpo da package PKG\_STRING\_FUNC.



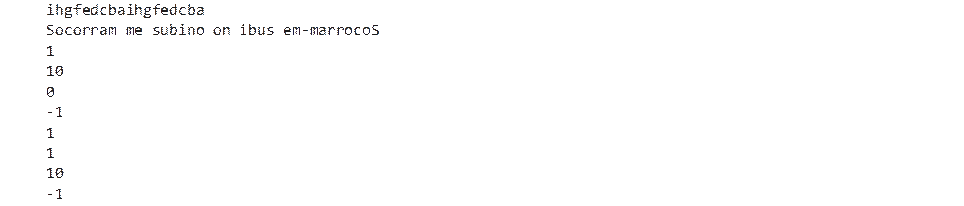
Observe que não é necessário repetir as declarações de variáveis e tipos, já que estes foram completamente definidos na especificação da package. Tampouco as funções e procedures devem ser colocadas na mesma ordem em que aparecem na especificação.

Referências a objetos especificados em uma package utilizam a notação de ponto: nome\_package.objeto.

No bloco anônimo a seguir são utilizados os objetos definidos em PKG\_STRING\_FUNC.

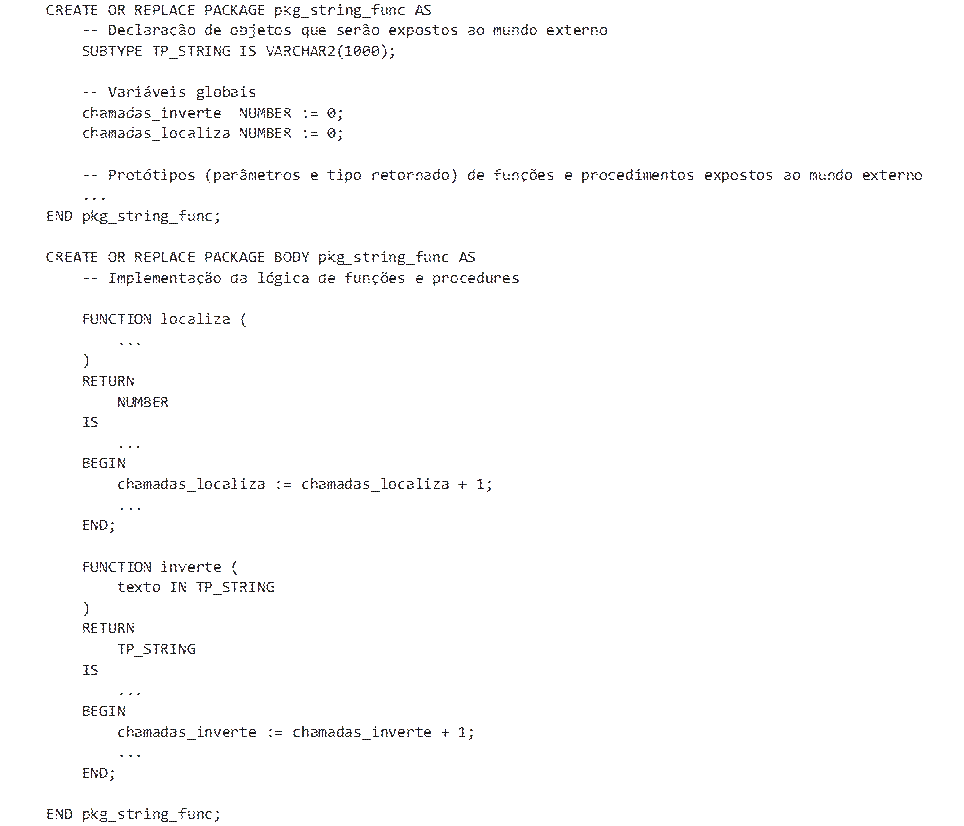


O resultado, após a execução é mostrado a seguir.​

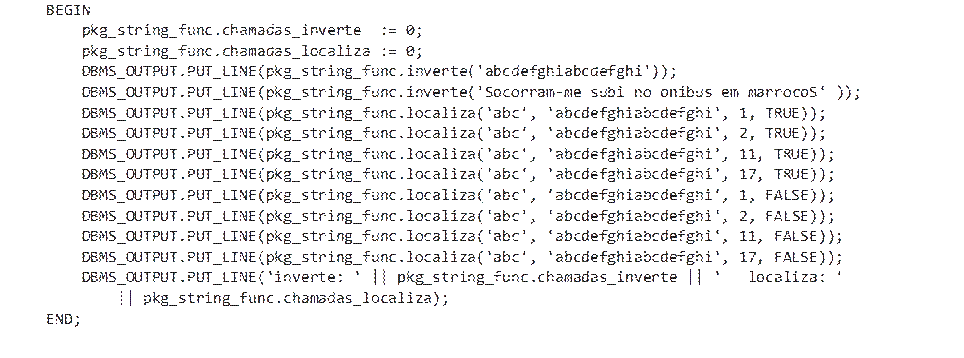


### *PERSISTÊNCIA DE VARIÁVEIS EM PACKAGES*

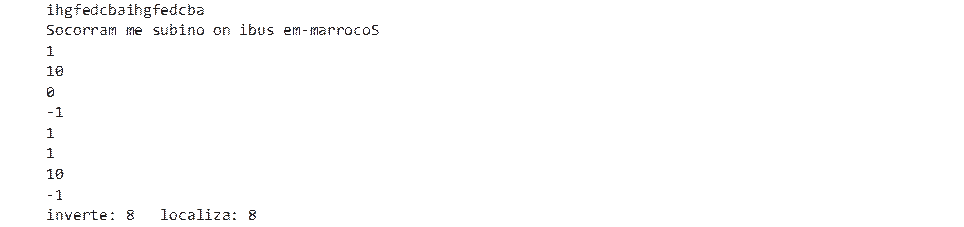
Variáveis podem ser declaradas e inicializadas na especificação de packages. Diferentemente do que ocorre com variáveis de procedures e funções, que são destruídas quando a execução do respectivo programa armazenado termina, variáveis em packages são persistentes, ou seja, continuam a existir e mantém seus valores durante toda a sessão corrente. Esta é uma forma de se declararem variáveis verdadeiramente globais: qualquer programa executado na sessão pode referenciar estas variáveis. A seguir, são declaradas duas variáveis na especificação da package PKG\_STRING\_FUNC para contabilizar o número de vezes que cada função é executada (são mostrados apenas os trechos da especificação e do corpo da package que foram alterados).



A seguir, o mesmo bloco anônimo utilizado no exemplo anterior.​



​A primeira vez em que qualquer objeto de uma package é referenciado em uma sessão provoca a sua inicialização. Isto faz com que a package seja carregada na memória, suas variáveis sejam inicializadas e seu código de inicialização seja executado (será visto na sequência). Como as duas variáveis globais são inicializadas com 0, os dois primeiros comandos não seriam necessários se a package  ainda não tivesse sido referenciada na seção. O resultado, após a execução é mostrado a seguir.​​

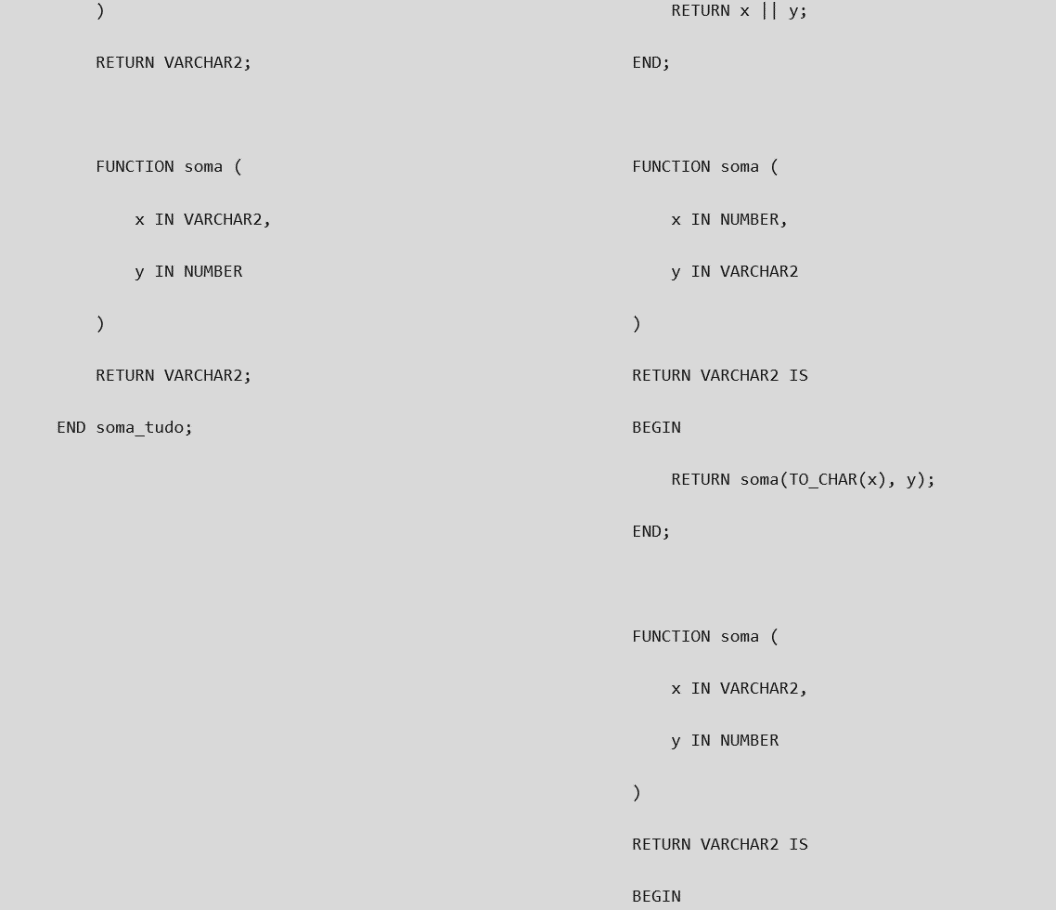
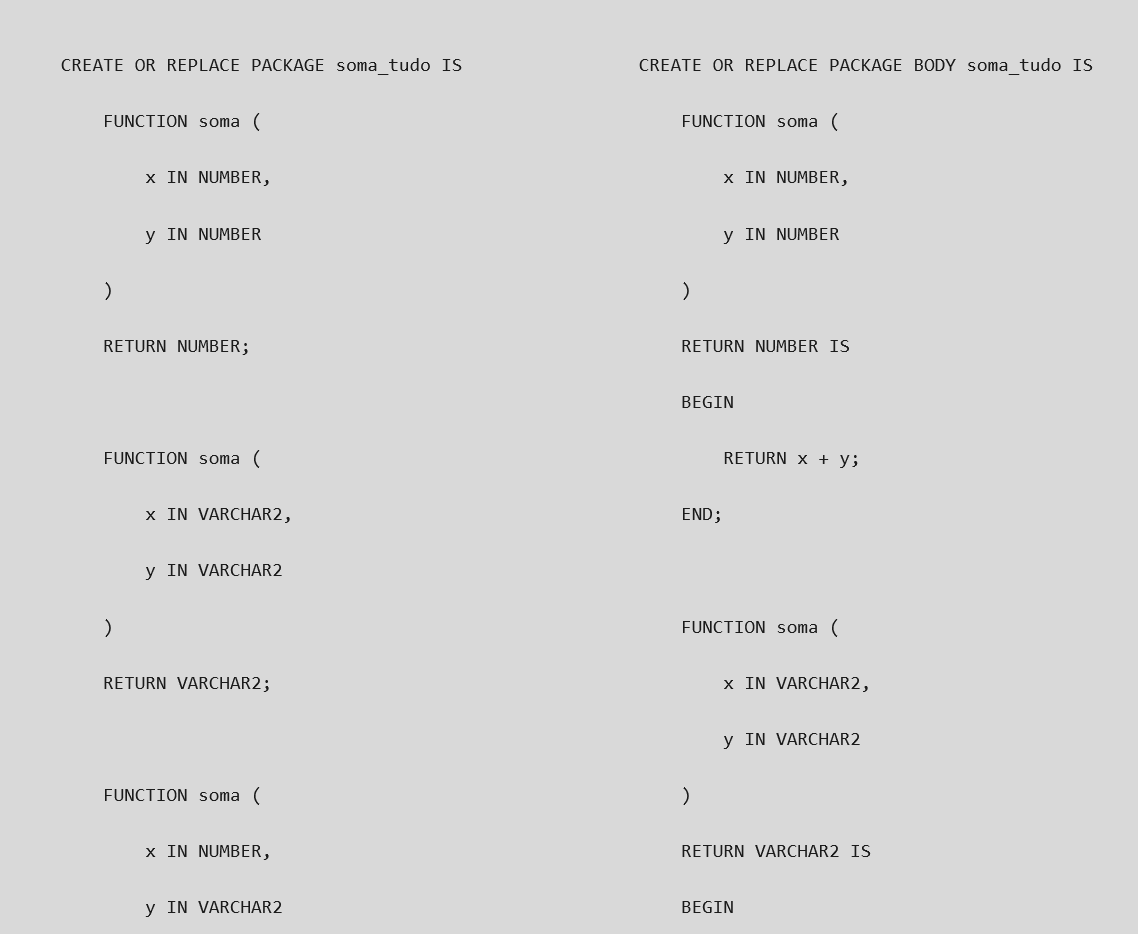


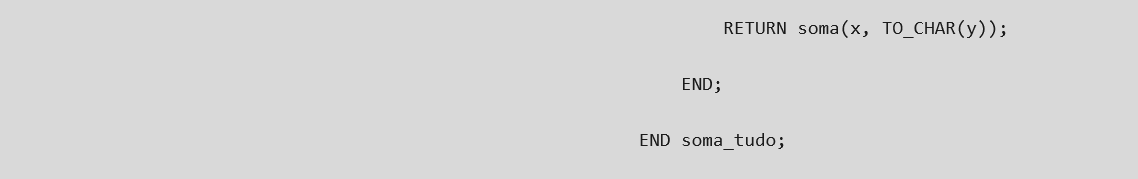
De fato, a procedure localiza foi chamada 8 vezes. Mas, no bloco anônimo há apenas duas chamadas procedure inverte. Por que o contador apresenta o valor 8? Não se esqueça que inverte é chamada dentro de localiza quando direção = FALSE. E como o contador é incrementado dentro de inverte, são contabilizadas todas as chamadas a ela.

### OVERLOADING*DE*PROCEDURES *E FUNÇÕES*

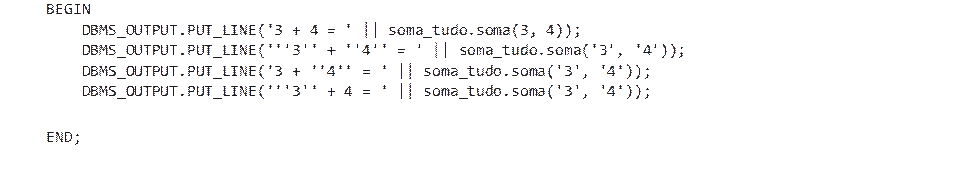
Imagine que se deseje criar uma procedure ou função que aceite diferentes quantidades ou tipos de argumentos, mas que implemente basicamente a mesma lógica. A solução imediata seria a criação de diferentes versões, uma para cada tipo/quantidade de parâmetros, porém todas com a basicamente a mesma lógica. O problema surge quando forem necessárias alterações na lógica das várias versões. Simplesmente **todas**elas deverão ser alteradas. A solução este problema é utilizar overloading.

Overloading é a possibilidade de se declarar procedures ou funções com o mesmo nome, porém com quantidades e/ou tipos de parâmetros diferentes. Overloading é permitido dentro de packages, mas não para procedures ou funções individuais. O compilador escolhe que versão será utilizada em função da quantidade/tipo dos parâmetros atuais. No exemplo a seguir, são definidas 4 funções soma, que retornam a soma dos dois parâmetros de entrada, se ambos forem de tipos numéricos, e a concatenação de ambos, caso contrário.

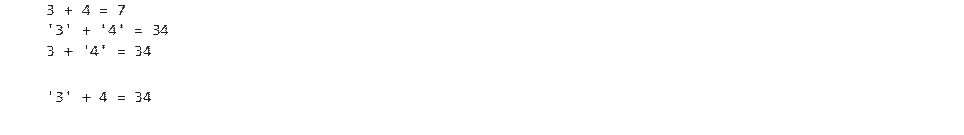




Observe que as quatro versões possuem parâmetros com tipos diferentes. Observe também que é possível chamar soma de dentro de soma, sem que isto signifique que a função seja recursiva. O bloco anônimo a seguir exemplifica o uso das quatro versões.​



O resultado, após a execução do bloco anônimo, é mostrado a seguir.​

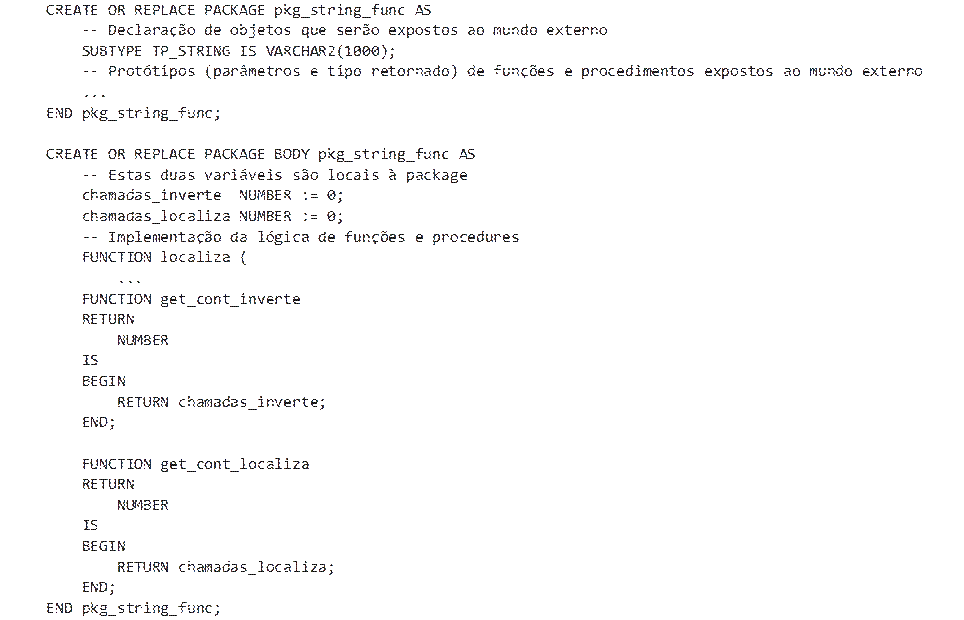


### *OBJETOS PRIVADOS E INICIALIZAÇÃO*

Packages permitem a criação de objetos privados, visíveis apenas a procedures e funções em seu interior. Estes objetos devem ser declarados apenas no corpo da package, já que tudo que é declarado em sua especificação é visível externamente.

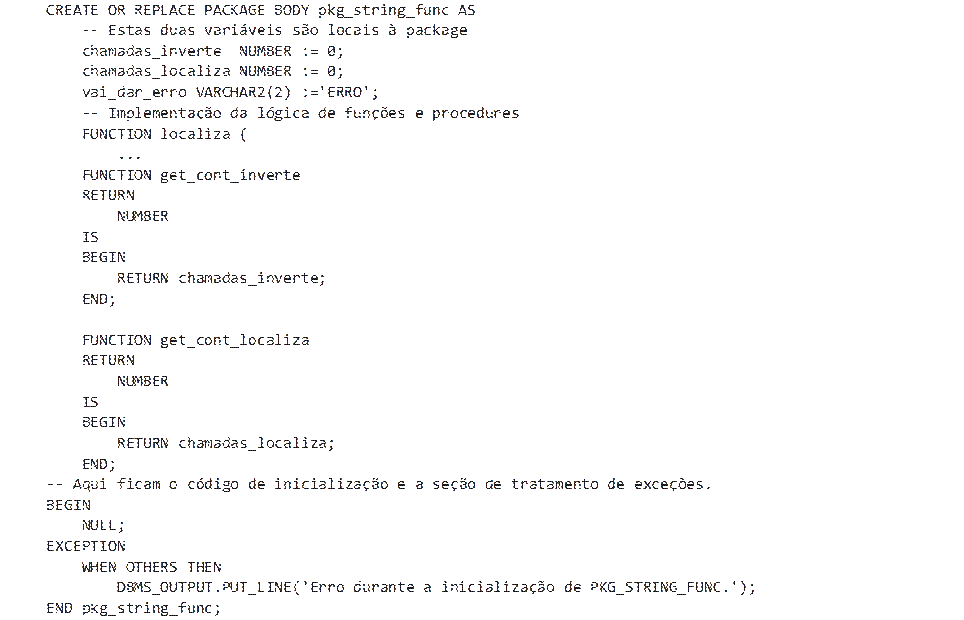
É possível também criar um trecho de código no corpo da package que seja executado uma única vez no momento em que a package é referenciada pela primeira vez.

No exemplo a seguir, os dois contadores de chamada serão declarados como privados e seus valores iniciais serão atribuídos pelo código de inicialização. E para que o mundo externo tenha acesso a eles, serão criadas duas funções, get\_cont\_inverte e get\_cont\_localiza, que retornam os valores dos respectivos contadores (no corpo da package, são mostradas apenas a declaração das variáveis privadas e a implementação das duas novas funções).

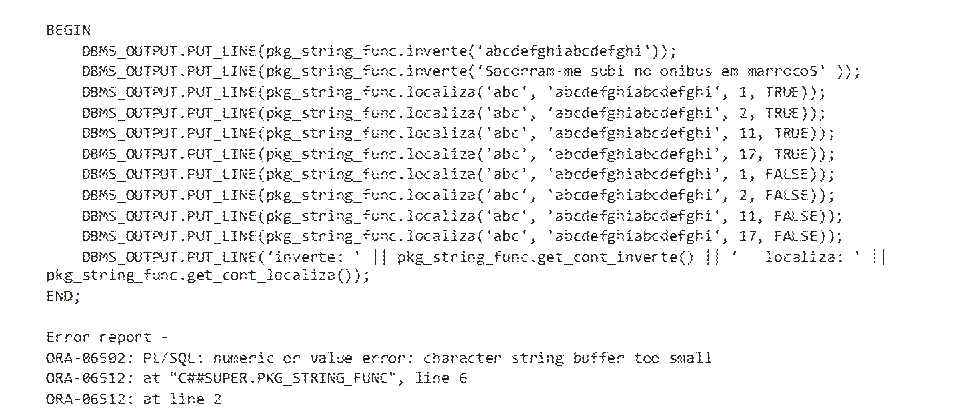


### *TRATAMENTO DE EXCEÇÕES EM*PACKAGES

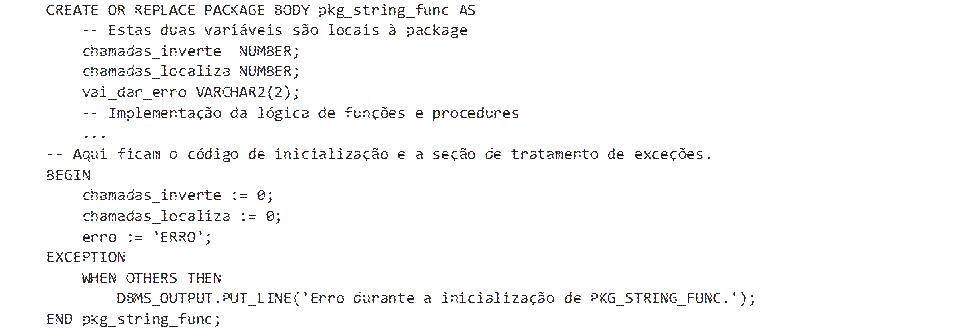
            Erros de execução podem ocorrer em qualquer parte de um programa, inclusive durante a inicialização de uma package. Para isto, é possível incluir uma seção de tratamento de exceções em packages, após o código de inicialização. Esta seção capturará todos os erros **exceto**aqueles gerados durante a atribuição de valores padrão a variáveis e constantes. Por isto, é recomendável que toda atribuição de valores iniciais seja feita pelo código de inicialização. Por exemplo, no lugar de chamadas\_inverte NUMBER := 0, deve-se declarar a variável sem atribuição de valor padrão (chamadas\_inverte NUMBER) e atribuir o valor padrão depois (NUMBER := 0), no código de inicialização. No exemplo a seguir é feita a atribuição de um literal com 3 caracteres a uma variável VARCHAR2(2) em sua declaração. Isto causará um erro não tratado (no corpo da package, é mostrada apenas a declaração desta variável e o código de inicialização com a seção EXCEPTION).



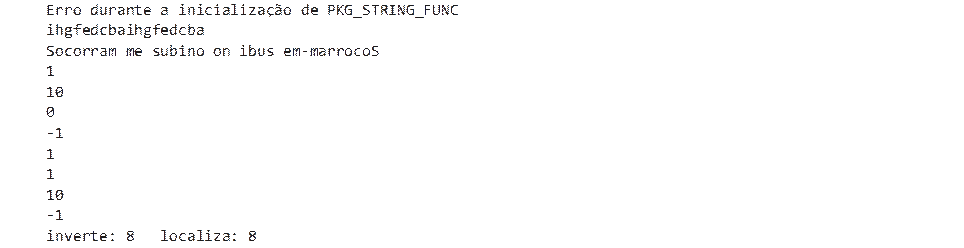
O bloco anônimo a seguir provocará erro (ORA-06502 Value Error) na primeira referência à PKG\_STRING\_FUNC.



Já no exemplo a seguir, o erro é capturado.​



O resultado exibido, após a execução, é mostrado a seguir.​



Pelo fato de o erro ter sido tratado, ele não é propagado e a execução do bloco anônimo pode continuar.