Call Stack e Recursão

Fundamentos de Programação em Python

Pontifícia Universidade Católica de Campinas

Prof. Dr. Denis M. L. Martins

Objetivos de Aprendizagem

- Compreender o funcionamento da Call Stack na execução de funções em Python.
- Identificar como a Call Stack é utilizada durante chamadas recursivas.
- Desenvolver funções recursivas simples e analisar seu comportamento.

Parte 1: Introdução à Call Stack

O que é a Call Stack?

- **Definição:** A *Call Stack* (pilha de chamadas) é uma estrutura de dados utilizada para armazenar as informações de chamadas de funções durante a execução de um programa.
- **Analogia:** Pense numa pilha de pratos: você só pode adicionar ou remover pratos do topo. A Call Stack funciona da mesma forma, mas com informações sobre as funções.
- **Propósito:** Gerenciar o fluxo de execução do código, garantindo que cada função retorne corretamente ao seu ponto de origem.

Como Funciona?

1. Quando uma função é chamada:
    - Uma nova "frame" (ou entrada) é adicionada (emplilhada) à Call Stack. Essa frame contém informações sobre a função em execução (parâmetros, variáveis locais, etc.).
    - A função que está no topo da pilha é a que está em execução no momento.
2. A função executa seu código.
3. Quando a função termina (retorna), sua frame é removida (desempilhada) da Call Stack.
4. O controle retorna para a função que a chamou (a "caller").

Exemplo 1

```
def funcao_A():
    print("Função A sendo chamada")
    funcao_B()
    print("Função A retornando")

def funcao_B():
    print("Função B sendo chamada")
    print("Função B retornando")

funcao_A()
```

Exercício 1: Simulação de Call Stack

Desenhe a *Call Stack* passo a passo, mostrando as frames que são adicionadas e removidas

```
def inicio():
    a()
    b()

def a():
    c()

def b():
    d()

def c():
    print("C")

def d():
    print("D")

inicio()
```

Resolução: Ordem das chamadas na pilha:

1. `inicio()` é chamado.
2. Dentro de `inicio()`, chama `a()` → empilha `a()`.
3. `a()` chama `c()` → empilha `c()` → imprime "C" → `c()` termina → desempilha `c()`.
4. `a()` termina → desempilha `a()`.
5. De volta em `inicio()`, chama `b()` → empilha `b()`.
6. `b()` chama `d()` → empilha `d()` → imprime "D" → `d()` termina → desempilha `d()`.
7. `b()` termina → desempilha `b()`.
8. `inicio()` termina → desempilha `inicio()`.

Exercício 2: Simulação de Call Stack

Desenhe a *Call Stack* passo a passo, mostrando as frames que são adicionadas e removidas. Anote os valores das variáveis em cada frame.

```
def funcao_X():
    y = 10
    funcao_Y(y)
    print("Função X retornando")

def funcao_Y(z):
    print("Função Y recebendo:", z)
    print("Função Y retornando")

funcao_X()
```

Parte 2: Introdução à Recursão

O que é Recursão?

- **Definição**: Recursão é quando uma função **chama a si mesma** para resolver um problema.
- É essencial que exista uma **condição de parada**, senão ocorre um loop infinito.
- Cada chamada recursiva cria um novo frame na **Call Stack**.

Por que usar Recursão?

- Útil para problemas que podem ser divididos em subproblemas menores.
- Elegante para percorrer estruturas como árvores, listas e resolver algoritmos clássicos (fatorial, Fibonacci, torres de Hanoi).

Exemplo Clássico: Fatorial Recursivo

```python
def fatorial(n):
    if n == 0:
        return 1
    else:
        return n * fatorial(n - 1)

print(fatorial(5))  # Resultado: 120
```

Análise:

- `fatorial(5)` depende de `fatorial(4)`, que depende de `fatorial(3)` ...
- A *Call Stack* cresce até o caso base `(n == 0)` e começa a retornar.

Exercício: Trace a Call Stack da função abaixo

```
def conta(n):
    if n == 0:
        return
    print(n)
    conta(n - 1)
```

Perguntas:

- Quantos frames são empilhados?
- Qual é o valor da pilha no momento da chamada `conta(3)`?

Exercício de Implementação

1. Implemente uma função recursiva que calcule o n-ésimo número de Fibonacci.
2. Implemente uma função recursiva para inverter uma string.

Conclusão

- A Call Stack controla a ordem de execução das funções.
- Recursão depende da Call Stack para funcionar corretamente.
- A má gestão da condição de parada pode levar ao erro `RecursionError`.

Dúvidas e Discussão