Unoeste
Campus I Rua José Bongiovani, 700
Cidade Universitária CEP 19050 900
Fone (18) 3229 2003

Faculdade de Informática de Presidente Prudente
Fone (18) 3229 1060
www.unoeste.br/fipp
e-mail: fipp@fipp.unoeste.br

FIPP FACULDADE DE INFORMÁTICA DE PRESIDENTE PRUDENTE CAMPUS I

Estruturas de Dados II

Prof. Francisco Assis da Silva

– Interpretador de programas escritos em Python –

Seu trabalho é fazer um interpretador (uma simulação de execução) de um pequeno programa na linguagem Python.

Exemplo de programa em Python:

```
def calcula(x, y):
    return x + y

a = 5
b = 10
c = 0
c = calcula(a, b)
print("%d %d %d" % (a, b, c))
a = calcula(7, a+b+c)
print("%s %d %d" % ("7", a+b+c, a))
c = calcula(a*b, a//b)
print("%d %d %d" % (a*b, a//b, c))
a = calcula(b, b+b)
msg = 'resultado ='
print("%s %d %d %d" % (msg, a, b, c))
```

Na Figura 1 é mostrada a tela do interpretador (simulador), com a execução do programa apresentado no exemplo acima.

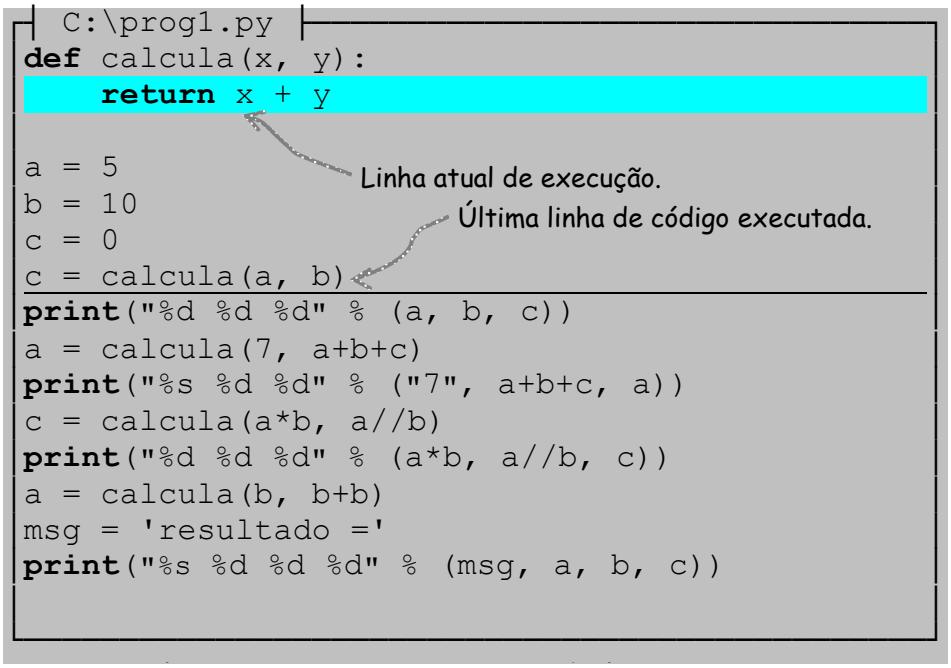


Figura 1. Simulação da execução de um programa na linguagem Python contendo a tela do interpretador (simulador).

A partir da abertura do código-fonte do programa (arquivo ".py"), você deverá percorrer linha por linha do arquivo e inserir cada termo (*token*) em uma estrutura de Lista de Listas, conforme pode ser observado no exemplo na Figura 2.

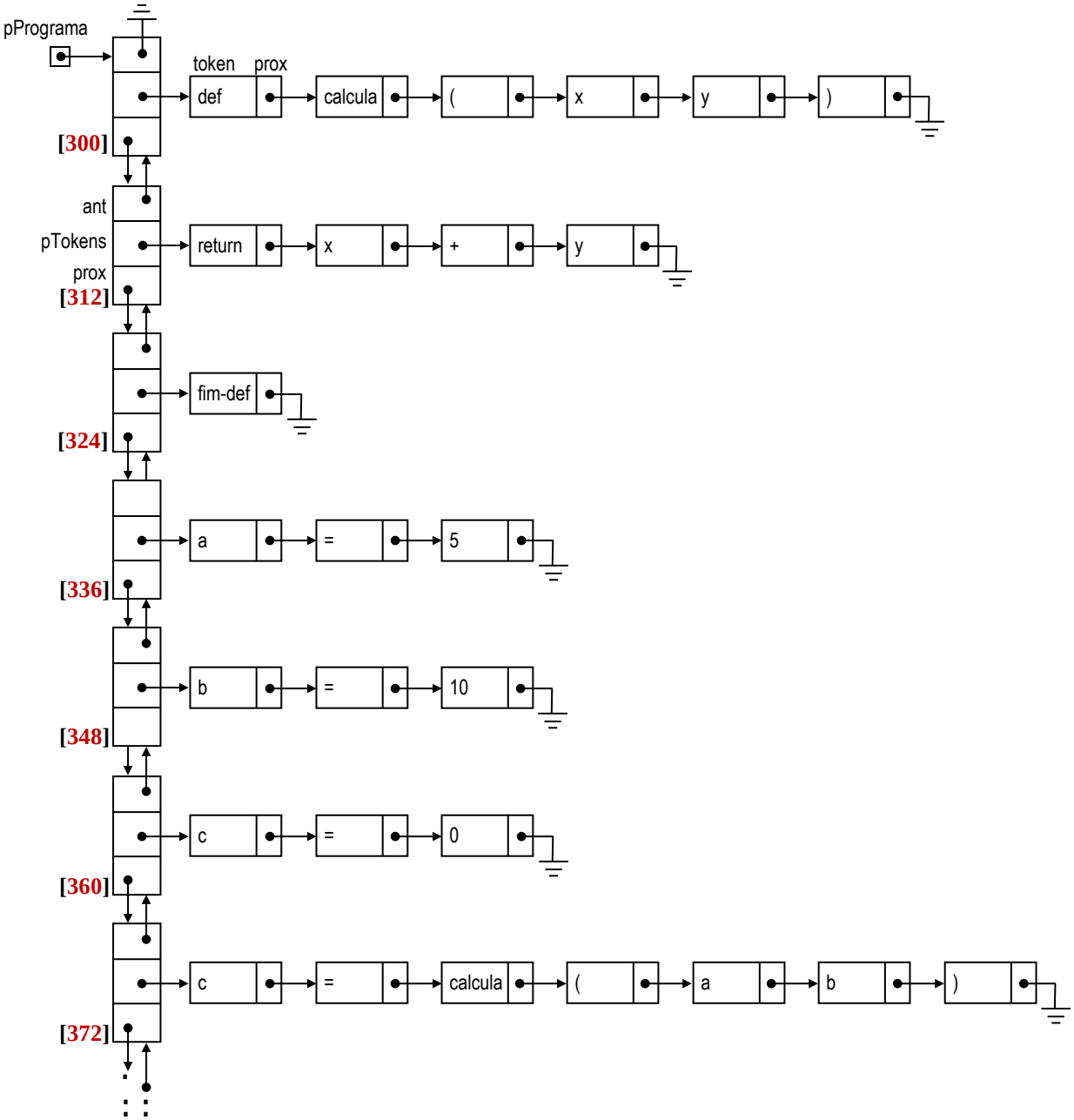


Figura 2. Parte inicial do programa de exemplo inserido na estrutura de Lista de Listas, contendo cada termo do programa em Python separado, linha por linha.

Durante a execução passo a passo do programa, você precisará gerenciar a memória RAM do programa Python utilizando uma pilha, contendo as variáveis do programa e também os endereços das chamadas de funções usadas para controlar escopo. Essa pilha deve ser implementada de forma dinâmica (caixinhas com ponteiros). A Figura 3 é apenas uma representação dessa pilha de variáveis em formato de tabela, contendo o estado atual da memória no momento da execução do programa de exemplo (Figura 1). Observe que no endereço &112 da pilha de variáveis contém um ponteiro (endereço representativo de memória [372]) para o *heap* em que a sétima caixa de linha (na lista vertical) foi alocada. Isso indica a linha de execução que fez a chamada da função "c = calcula(a, b)", ou seja, quando a função "calcula" terminar a sua execução, os argumentos de retorno serão atribuídos às variáveis (nesse caso apenas a variável "c") e o interpretador (simulador) passará a executar as demais linhas, abaixo dessa.

	&	identificador	valor	ponteiro
Topo →	120	y	10	NULL
	116	x	5	NULL
	112	[**endereço**]	-	**[372]**
	108	c	0	NULL
	104	b	10	NULL
	100	a	5	NULL

Figura 3. Estado da memória RAM do programa de exemplo em execução (Figura 1).

O simulador deve conseguir lidar com os seguintes comandos e estruturas:

a) Fazer chamadas de funções com passagem de parâmetro e retornar valores por `return`;
b) Fazer chamadas de funções dentro de outra função;
c) Resolver equações aritméticas e algumas funções matemáticas: raiz quadrada e módulo (valor absoluto);
d) Executar comparação usando `if`, `elif` e `else` e operadores lógicos `and`, `or` e `not`;
e) Manipular listas em Python, que será simulada com Lista Generalizada;
f) Executar a estrutura de repetição `while` e `for`;
g) Executar o comando de exibição `print`.

#Função print(...)

Exemplos:

```
calc = 2 + 2
operacao = 'soma'
print('Resultado: ', calc)
print('Resultado: ' + str(calc))
print('Resultado: %d' % calc)
print('O Resultado da %s eh %d' % (operacao, calc))
```

#Operadores

Aritméticos	Comparação	Lógicos
+	== (igual)	and
-	!= (diferente)	or
*	> (maior)	not
/	< (menor)	
// (parte inteira)	>= (maior ou igual)	
%	<= (menor ou igual)	
**	in not in	

Exemplos:

```
print("3 * 2 =", 3*2)          #6
print("7 / 2 =", 7/2)          #3.5
print("7 // 2 =", 7//2)        #3
print("4 - 2 =", 4-2)          #2
print("3 ** 2 =", 3**2)        #9
print("3 % 2 =", 3%2)          #1
print("p" in "python")         #True
print(1 in [1,2,3])            #True
print("d" in ["a","b"])        #False
print("d" not in ["a","b"]) #True
```

#Comando de Seleção (if)

```
if condição:
    comando(s)
elif condição:
    comando(s)
else:
    comando(s)
```

Exemplos:

```
var1 = 1
var2 = 2
if var1 == 1:
    print("var1 igual a 1")
elif var1 !=1 and var2 == 2:
    print("var1 diferente de 1 e var2 igual a 2")
elif var1 >= 1000 or var2 <= -1000:
    print("var1 maior que 1000 ou var2 menor que -1000")
else:
    print("nenhuma das alternativas anteriores")

i = 5
if i in [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]:
    print('esta contido')
```

Comando de repetição (while)

```
while condição:
    comando(s)
```

Exemplo:

```
i = 0
while i < 10:
    print("i = ", i) # 0 1 2 3 4 5 6 7 8 9
    i = i + 1
```

Comando de repetição (for)

```
for variável in faixa:
    comando(s)
```

Sintaxe: **range**(**start**, **stop**, **step**)
onde: **start**: partida
stop: parada (stop - 1)
step: passo (incremento [+] ou decremento [-])

Exemplos:

```
for i in range(10):
    print("i = ", i) # 0 1 2 3 4 5 6 7 8 9


for i in range(0, 11):
    print(i) # 0 1 2 3 4 5 6 7 8 9 10
```

```
for i in range(0, 10, 2):
    print(i) # 0 2 4 6 8

for i in range(0, 11, 2):
    print(i) # 0 2 4 6 8 10
```

Listas

Exemplos:

```
lista = ['oi', 2.0, 5, [10, 20]]
print(lista[0])    # oi
print(lista[1])    # 2.0
print(lista[2])    # 5
print(lista[3])    # [10, 20]
print(lista[3][1]) # 20
```

Podemos acessar pedaços (slices) da lista. Exemplos:

```
lista = [1, 2, 3, 4, 5, 6]
print(lista[2:4]) # [3, 4]
print(lista[:2])  # [1, 2]
print(lista[2:])  # [3, 4, 5, 6]
```

Podemos ordenar uma lista. Exemplo:

```
frutas = ['laranja', 'banana', 'abacaxi']
frutas.sort()
print(frutas)      # ['abacaxi', 'banana', 'laranja']
print(len(frutas)) # 3
```

Podemos inserir ao final de uma lista. Exemplo:

```
frutas.append('ameixa')
print(frutas)      # ['abacaxi', 'banana', 'laranja', 'ameixa']
print(len(frutas)) # 4
```

Podemos inserir em uma posição da lista. Exemplo:

```
frutas.insert(2, 'abacaxi')
print(frutas)    # ['abacaxi', 'banana', 'abacaxi', 'laranja', 'ameixa']
```

Podemos remover o primeiro elemento encontrado na lista. Exemplo:

```
frutas.remove('abacaxi')
print(frutas)    # ['banana', 'abacaxi', 'laranja', 'ameixa']
```

Podemos remover de uma posição da lista. Exemplo:

```
frutas.pop(3)
print(frutas)    # ['banana', 'abacaxi', 'laranja']
```

Mais operações com listas: Exemplo:

```
frutas = ['laranja', 'banana', 'abacaxi']
frutas.append([3, 2, 1])
frutas.append([[9.9, ['a', 'b']]])
print(frutas) # ['laranja', 'banana', 'abacaxi', [3, 2, 1], [[9.9, ['a', 'b']]]]

frutas[3].sort()
print(frutas) # ['laranja', 'banana', 'abacaxi', [3, 2, 1], [[9.9, ['a', 'b']]]]
```

Funções

Exemplos:

```
def quadrado_e_cubo(x):
    quadrado = x ** 2
    cubo = x ** 3
return quadrado, cubo

print(quadrado_e_cubo(5)) # 25, 125

#---------------------------
def imprime_nome(nome):
    print("Nome: " + nome)

imprime_nome("Joao")   # Nome: Joao
imprime_nome("Jose")   # Nome: Jose
imprime_nome("Maria") # Nome: Maria
```

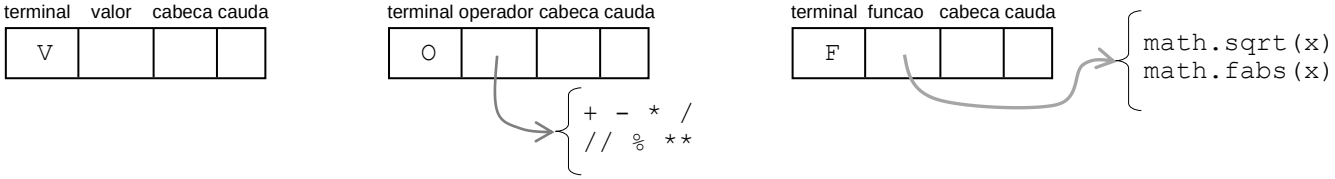
Seu programa deve permitir ao usuário abrir um arquivo fonte na linguagem Python (F7), executar o programa passo a passo (F8), mostrar o conteúdo da memória RAM no momento atual da execução (F9) e mostrar a tela (`print` dos resultados) (F10). Para a execução passo a passo, deve-se apertar ENTER para executar a linha atual (linha com fundo demarcado, observe a Figura 1) e passar para a próxima linha.

Você deve utilizar apenas estruturas de listas para construir seu programa interpretador da linguagem Python (simulador):

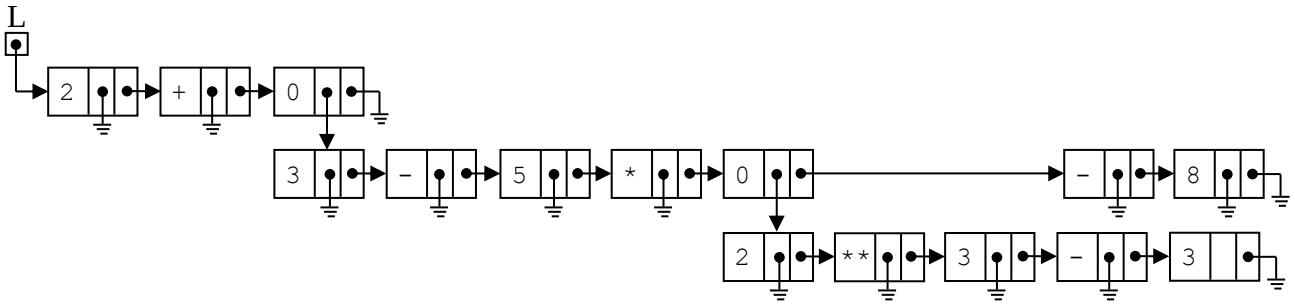
a) Deverá ter uma lista para representar a pilha de variáveis;
b) Deverá ter uma lista de listas para guardar as linhas de comandos do programa em Python que abriu. Cada nodo da lista principal (considere uma lista vertical) terá uma lista vinculada a ela para guardar em cada nodo os termos (*tokens*) da linha de comandos (considere uma lista horizontal vinculada ao nodo vertical).
   Exemplo: Se a linha de comandos fosse "`def calcula(x, y):`", você teria, na lista horizontal, um nodo para "`def`", outro para "`calcula`", outro para "`(`", outro para "`x`", outro para "`y`" e outro para "`)`";

Para resolver as expressões matemáticas, você deverá utilizar uma lista generalizada, diferente do modelo conceitual de aula. A partir de uma **string** (char [ ]) passada por parâmetro para uma de suas funções, crie um algoritmo para construir uma lista generalizada segundo as prioridades e resolver a expressão "podando" os nodos da lista generalizada. A medida que cada linha de operação for resolvida, os nodos da lista generalizada devem ser removidos e sobrar apenas um nodo com o resultado, cujo valor deve ser retornado. O nodo da lista deverá ser construído com o uso de **union**, uma vez que o nodo poderá ser um valor (**float**), um operador (**char[2]**) ou uma função (**sqrt** ou **abs**).

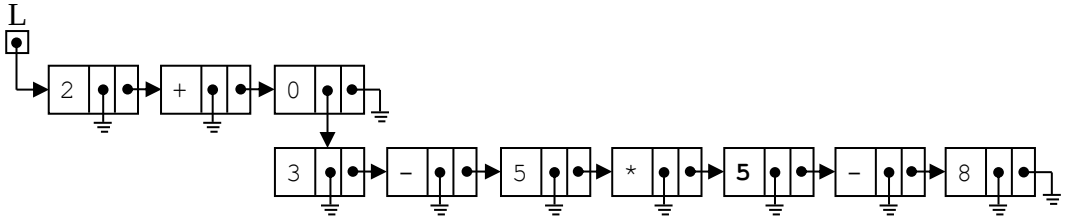
Tipos de nodos:



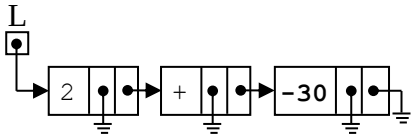
Exemplo de expressão aritmética a ser resolvido: "2+(3-5*(2**3-3)-8)"
A lista construída será:



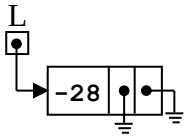
Após resolver os parênteses mais internos:



Após resolver o segundo nível de parênteses:



Após resolver toda a expressão:



Observações importantes:
Não utilize funções prontas da linguagem C, como por exemplo, strtok para separar string, filas, pilhas etc.
Para resolver cada lista de expressões (cada nível da lista generalizada), você pode utilizar duas pilhas (criadas por você), uma de valor e outra de operador.

Exemplos de códigos em Python contendo alguns comandos e estruturas que seu programa deverá ser capaz interpretar (simular a execução):

Exemplo 1:

```
def calcula1(I, J, K):
    I=I+3
    J=I+K
    K=K+1
    return I, J

def calcula2(I, J, K):
    I=I+2
    J=J+K
    K=K+1
    return K
```

```
def executa(p1, p2, p3):
    p3, p1 = calcula1(p3, p1, p2)
    p2 = calcula2(p2, p2, p2)
    return p1, p2, p3

A = 3
B = 5
C = 7
A, B, C = executa(A, B, C)
print("Primeiro valor modificado = ", A)
print("Segundo valor modificado = ", B)
print("Terceiro valor modificado = ", C)
```

Exemplo 2:

```
def faz_uma_repeticao_while(X, n):
    i=0
    result=1
    while(i<n):
        result=result * X
        i=i+1
    return result

s = faz_uma_repeticao_while(2,4)
print("Resultado = %d" % s)
```

Exemplo 3:

```
def faz_uma_repeticao_for(X, n):
    i=0
    result=1
    for i in range(n):
        result=result * X
    return result

s = faz_uma_repeticao_for(2,4)
print("Resultado = %d" % s)
```

Exemplo 4:

```
import math

def juros_composto(P, i, n):
    M = P*(1 + i/100)**n
    return M

res = juros_composto(2000, 1.5, 10)
print("Resultado = %.2f" % res)
```

Exemplo 5:

```
def funcao(chave):
    nomes = ["Joao", "Jose", "Maria"]
    numeros = [17, 123, 33]
    print(nomes)
    print(numeros)
    lista_mista = ["ola", 2.0, 5*2, [10, 20]]
    if chave in lista_mista:
        return "S"
    return "N"

ret = funcao("ola")
if ret == "S":
    print("Econtrou!!!")
```

Importante!!!
O trabalho é em dupla e deverá ser apresentado pelos dois alunos!

O que deve enviar no Aprender antes da apresentação:

a) Todos os códigos-fonte;
b) Os programas em Python usados para testar o simulador. Tem que ser diferentes dos exemplos escritos aqui nesse documento;
c) Um .docx com o desenho das estruturas (caixinhas das listas) usadas.