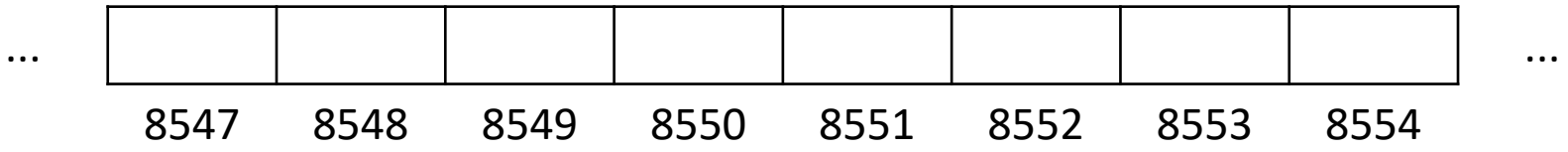
Uso da memória do computador em funções

Sumário

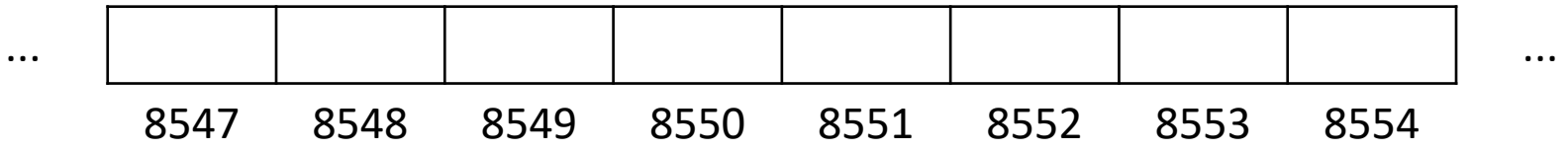
Memória de um Computador

- A memória de um computador pode ser entendida como um conjunto de posições de armazenamento, cada uma possuindo um endereço

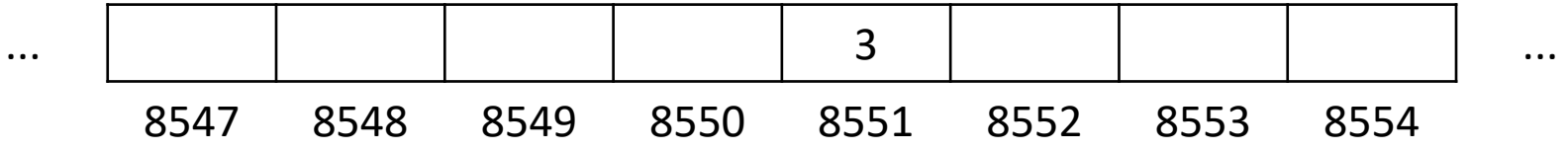


Memória de um Computador

- A memória de um computador pode ser entendida como um conjunto de posições de armazenamento, cada uma possuindo um endereço

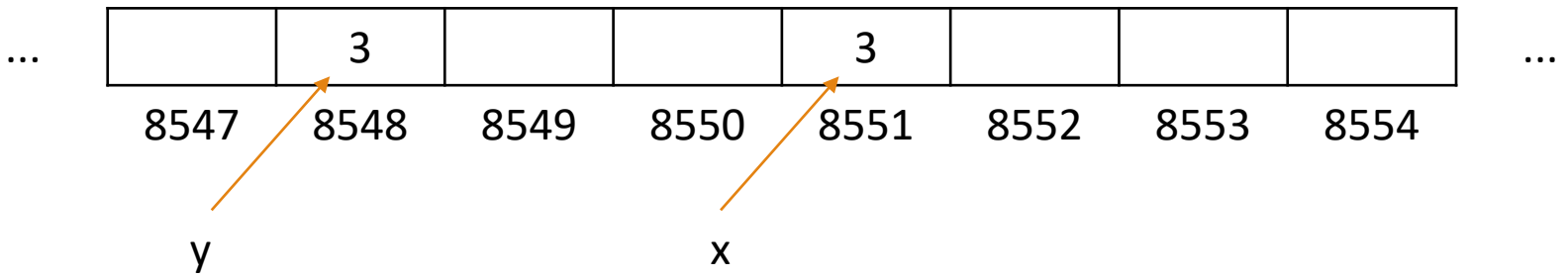


- Na definição de uma nova variável, por exemplo, x = 3, o valor 3 é alocado em uma posição disponível na memória, e o nome x é utilizado para referenciar esse valor



Memória de um Computador

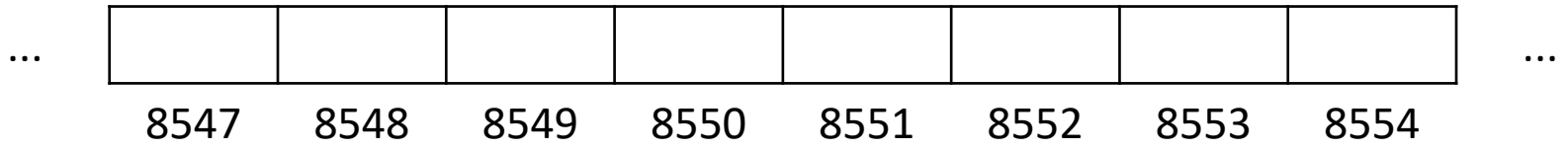
- O comando **y = x** faz com que uma cópia do valor de x seja inserida em uma nova posição de memória. O nome y é utilizado para referenciar esse valor.



Memória de um Computador

- Na chamada de uma função ocorre a mesma situação: cópias dos valores são alocadas em novas posições de memória

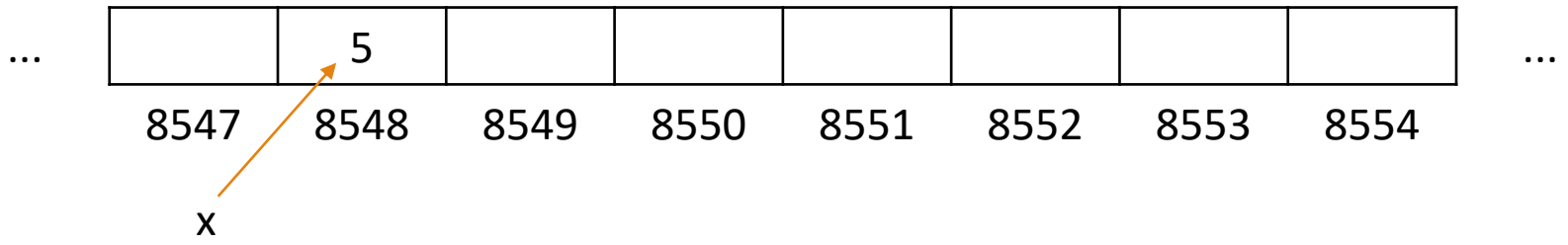
```
def soma(val1, val2):
    res = val1 + val2
    return res

x = 5
y = 8
val = soma(x, y)
```



Memória de um Computador

- Na chamada de uma função ocorre a mesma situação: cópias dos valores são alocadas em novas posições de memória

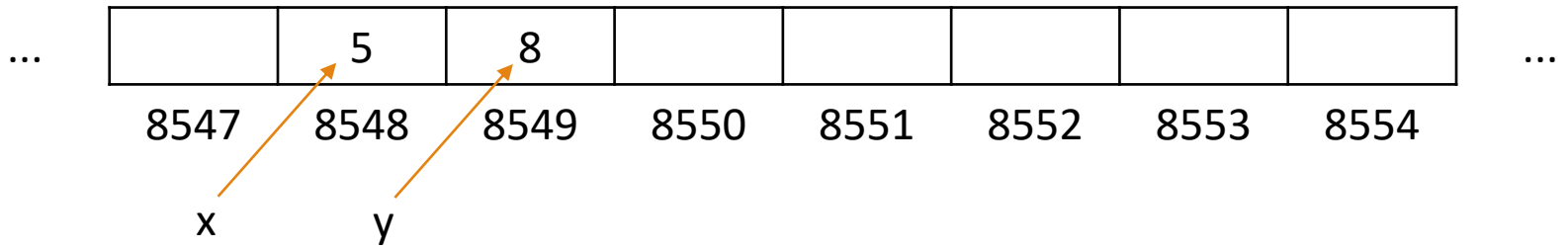
```
def soma(val1, val2):
    res = val1 + val2
    return res

x = 5
y = 8
val = soma(x, y)
```



Memória de um Computador

- Na chamada de uma função ocorre a mesma situação: cópias dos valores são alocadas em novas posições de memória

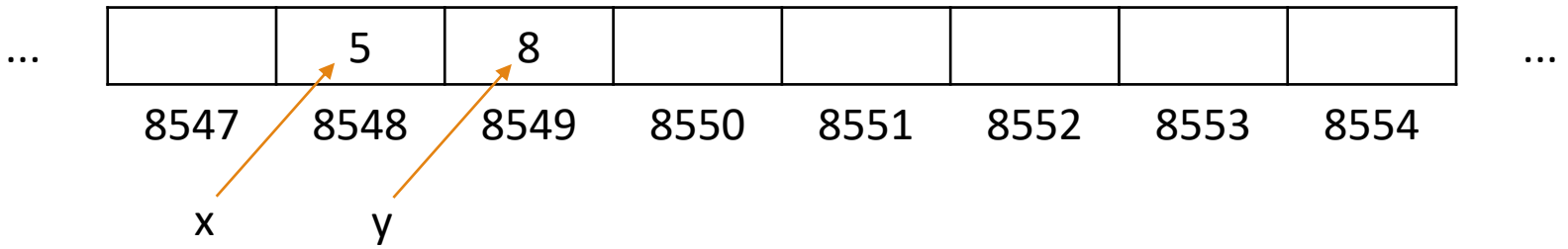
```
def soma(val1, val2):
    res = val1 + val2
    return res

x = 5
y = 8
val = soma(x, y)
```



Memória de um Computador

- Na chamada de uma função ocorre a mesma situação: cópias dos valores são alocadas em novas posições de memória

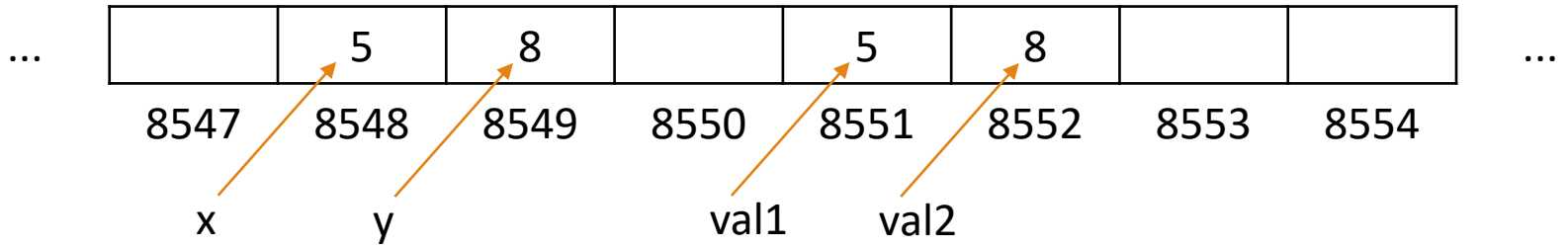
```
def soma(val1, val2):
    res = val1 + val2
    return res

x = 5
y = 8
val = soma(x, y)
```



Memória de um Computador

- Na chamada de uma função ocorre a mesma situação: cópias dos valores são alocadas em novas posições de memória

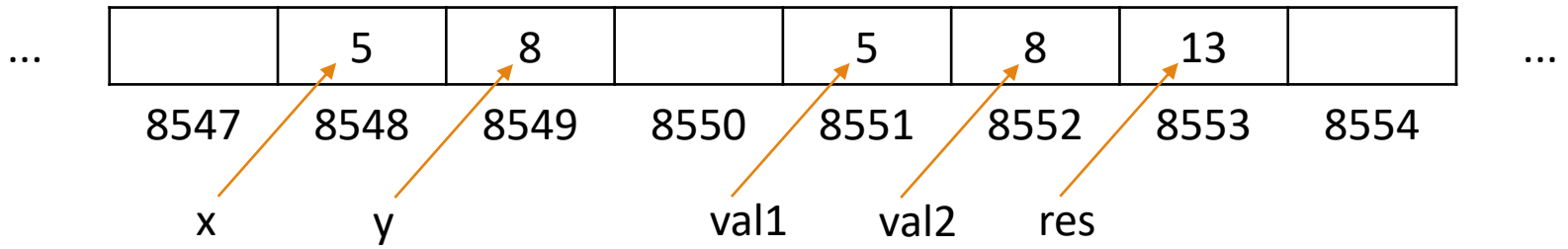
```
def soma(val1, val2):
    res = val1 + val2
    return res

x = 5
y = 8
val = soma(x, y)
```



Memória de um Computador

- Na chamada de uma função ocorre a mesma situação: cópias dos valores são alocadas em novas posições de memória

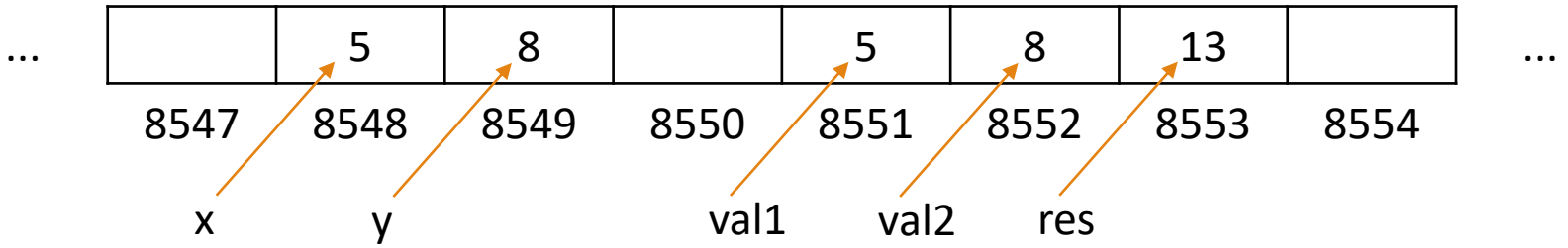
```
def soma(val1, val2):
    res = val1 + val2
    return res

x = 5
y = 8
val = soma(x, y)
```



Memória de um Computador

- Na chamada de uma função ocorre a mesma situação: cópias dos valores são alocadas em novas posições de memória

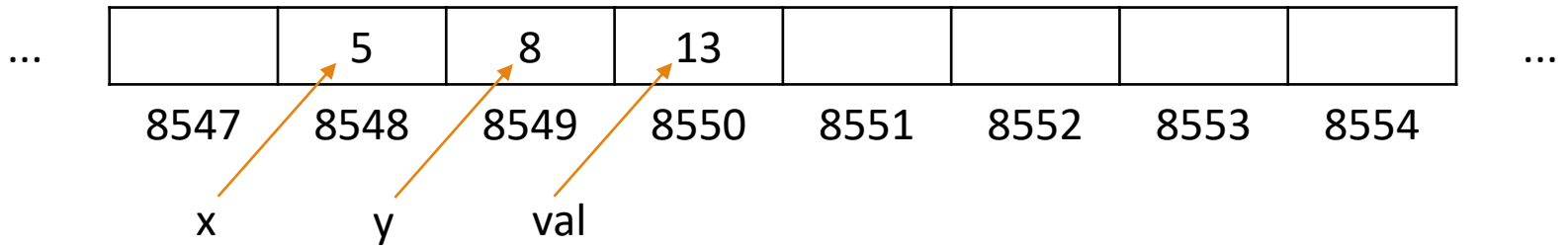
```
def soma(val1, val2):
    res = val1 + val2
    return res

x = 5
y = 8
val = soma(x, y)
```



Memória de um Computador

- Na chamada de uma função ocorre a mesma situação: cópias dos valores são alocadas em novas posições de memória

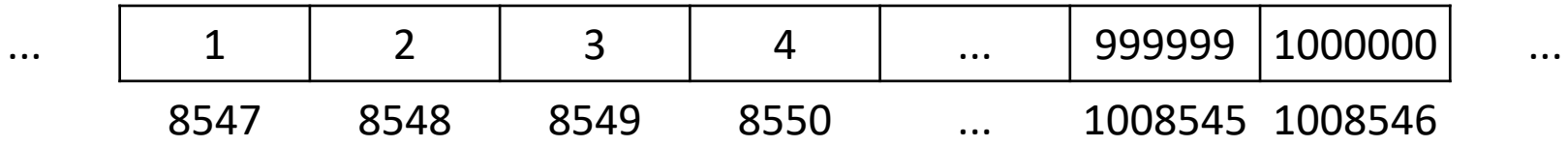
```
def soma(val1, val2):
    res = val1 + val2
    return res

x = 5
y = 8
val = soma(x, y)
```



Memória de um Computador

- A cópia de valores na memória possui custo computacional relevante
- E se tivermos uma lista com diversos valores?

```
valores = [1, 2, 3, 4, ..., 999999, 1000000]
```



Memória de um Computador

- A cópia de valores na memória possui custo computacional relevante
- E se tivermos uma lista com diversos valores?

```
valores = [1, 2, 3, 4, ..., 999999, 1000000]
```

- Criar uma nova variável com o conteúdo de valores possuiria alto custo computacional
- Por isso, o comando

```
dados = valores
```

não copia os valores na memória

Ponteiros

- Sabemos que em programação variáveis são utilizadas para armazenar valores
- Um ponteiro é uma variável que armazena um **endereço de memória**
- Variáveis que armazenam listas de valores na verdade são ponteiros para endereços de memória

Ponteiros

- Primeiramente, vamos considerar o seguinte código. O que ele imprime na tela?

```
a = 5
b = a
b = 10
print(a)
```

Ponteiros

- Primeiramente, vamos considerar o seguinte código. O que ele imprime na tela?

```
a = 5
b = a
b = 10
print(a)
```

- Ele imprime o valor 5

Ponteiros

- Vamos agora analisar o código abaixo, o que ele imprime na tela?

```
l1 = [1, 2, 3, 4]
l2 = l1
l2[1] = 50
print(l1)
```

Ponteiros

- Vamos agora analisar o código abaixo, o que ele imprime na tela?

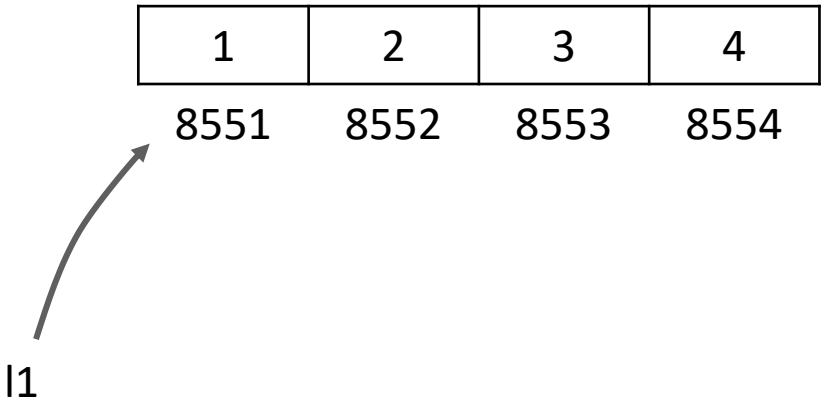
```
l1 = [1, 2, 3, 4]
l2 = l1
l2[1] = 50
print(l1)
```

- Ele imprime a lista [1, 50, 3, 4]!
- Porque isso aconteceu?

Ponteiros

O comando `l1=[1, 2, 3, 4]` executa as seguintes operações:

1. Os valores 1, 2, 3 e 4 são armazenados na memória
2. O nome l1 é associado ao endereço de memória do primeiro valor

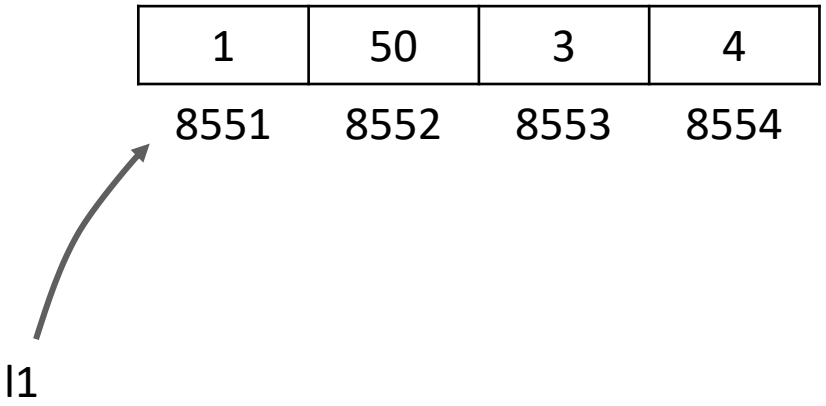


Quando executamos o comando l1[1]=50, ele na verdade significa: coloque o valor 50 no endereço de memória 8551+1.

Ponteiros

O comando `l1=[1, 2, 3, 4]` executa as seguintes operações:

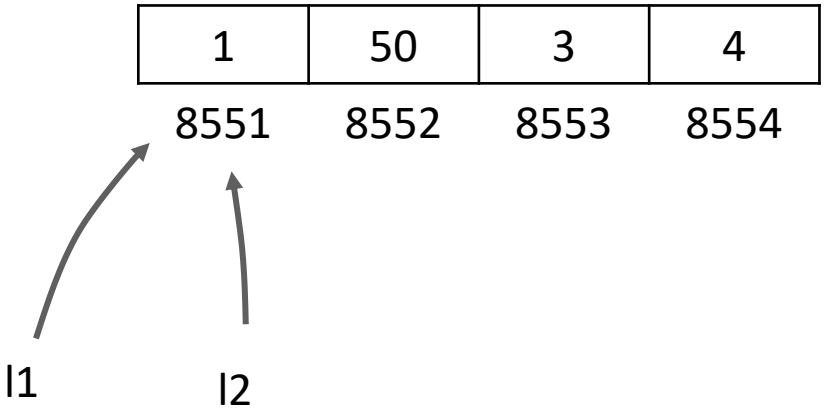
1. Os valores 1, 2, 3 e 4 são armazenados na memória
2. O nome l1 é associado ao endereço de memória do primeiro valor



Quando executamos o comando l1[1]=50, ele na verdade significa: coloque o valor 50 no endereço de memória 8551+1.

Ponteiros

O comando l2=l1 significa: “associe o nome l2 ao mesmo endereço que o nome l1”



Dessa forma, o comando l2[1]=50 tem exatamente o mesmo significado que l1[1]=50
Ambos os comandos significam: coloque o valor 50 no endereço de memória 8551+1.

Ponteiros

Mas como saber se estamos trabalhando com valores ou com ponteiros?

Em algumas linguagens, o programador indica se a variável é um ponteiro ou se ela representa o valor em si.

Em Python, certos tipos de variáveis se comportam como ponteiros, outros se comportam como valores.

Ponteiros

Comportamento dos tipos de variáveis que já vimos:

Tipo	Valor ou Ponteiro
Inteiro	Valor
Ponto flutuante	Valor
String	Valor
Booleano	Valor
Lista	Ponteiro

Em Python, listas são ponteiros

Listas se referem a posições de memória:

```
def alguma_funcao(valores):
    # A variável valores receberá uma cópia do conteúdo da
    # variável numeros. Mas o conteúdo da variável números
    # é o endereço de memória do primeiro elemento da lista
    # de valores. Portanto, tanto a variável numeros quanto
    # a variável valores se referem à mesma posição de memória.

    # Modifica o valor na posição de memória valores+2
    valores[2] = 25

numeros = [1, 2, 3, 4]
alguma_funcao(numeros)
print(numeros) # Imprime [1, 2, 25, 4]
```

Passagem de valor e de referência (ponteiro)

- A principal vantagem de passar o endereço dos valores ao invés dos valores em si é tornar o programa mais eficiente

Exercício resolvido - Soma acumulada

- Faça uma função que receba como entrada uma lista e calcule a soma acumulada dos valores da lista.
- Soma acumulada: cada elemento i do resultado é dado pela soma dos elementos de 0 a i (inclusive) da lista de entrada. Exemplo:
  - A entrada [3, 2, 1, 4, 2] dará o resultado [3, 5, 6, 10, 12]
- Importante! A função deve receber como entrada, além da lista de valores, uma lista na qual o resultado será armazenado. A função não retorna nenhum valor.

- Exemplo de uso:

```
valores = [3, 2, 1, 4, 2]
resultado = [0, 0, 0, 0, 0]
soma_acumulada(valores, resultado)
print(resultado)
```

Solução

```
def acumula(valores, resultado):
    """Calcula a soma acumulada da lista valores, inserindo o resultado
    na lista resultado. Cada elemento i de resultado será a soma dos
    elementos de valores entre 0 e i (inclusive).
    Parâmetros
        valores: Lista de valores
        resultado: Lista pré alocada na qual o resultado será inserido.
        Precisa ter o mesmo tamanho que valores
    Retorna
        None
    """

    n = len(valores)
    soma = 0
    for i in range(0, n):
        soma += valores[i]
        resultado[i] = soma

def testes():

    valores = [3, 1, 2, 3, 4]
    resultado = [0, 0, 0, 0, 0]
    acumula(valores, resultado)
    print(resultado)

testes()
```