Algoritmos Computacionais

Conteúdo: Tuplas

Prof. Dsc. Giomar Sequeiros
giomar@eng.uerj.br

Tuplas

Tuplas

- Em Python, uma tupla é uma **sequência imutável** de objetos.
- As tuplas são definidas utilizando parênteses e os elementos são separados por vírgulas. Veja a sintaxe básica para criar uma tupla:
- Sintaxe básica:

```
minha_tupla = (elemento1, elemento2, elemento3, ...)
```

- Exemplo:

```
# Criando uma tupla com elementos
frutas = ('maçã', 'banana', 'laranja', 'uva')

# Acessando elementos da tupla
print(frutas[0])  # Saída: maçã
print(frutas[2])  # Saída: laranja
```

Tuplas

- Em Python, a sintaxe para criar uma tupla é bastante flexível. Existem várias formas de declarar uma tupla. A seguir são mostrados alguns exemplos:
  - Utilizando parênteses:

```
tupla1 = (1, 2, 3)
```

  - Utilizando vírgulas:

```
tupla2 = 4, 5, 6
```

  - Utilizando a função tuple():

```
lista = [7, 8, 9]
tupla3 = tuple(lista)
```

  - Utilizando a função zip() para combinar várias sequências em uma tupla:

```
numeros = (1, 2, 3)
letras = ('a', 'b', 'c')
tupla4 = tuple(zip(numeros, letras))
```

  - Utilizando a sintaxe de desempacotamento de tupla:

```
x, y, z = 10, 20, 30
tupla5 = x, y, z
```

Tupla vazia e unitária

- Uma tupla **vazia** é uma tupla que não contém nenhum elemento. É representada por parênteses vazios "()". Exemplo:

```
tupla_vazia = ()
print(tupla_vazia)   # Saída: ())
```

- Uma tupla **unitária** é uma tupla que contém um **único elemento**. Para criar uma tupla unitária é preciso incluir uma vírgula após o elemento. Exemplo:

```
tupla_unitaria = (42,)
print(tupla_unitaria)   # Saída: (42,)
```

  - A inclusão da vírgula é importante para diferenciar uma tupla unitária de um valor em parênteses, que seria interpretado apenas como um valor entre parênteses. Por exemplo:

```
valor = (42)
print(valor)   # Saída: 42 (não é uma tupla)
```

Concatenação de tuplas

- A concatenação de tuplas é a operação de combinar duas ou mais tuplas em uma única tupla. A concatenação de tuplas é realizada usando o operador de adição (+) ou o método extend().
- Utilizando o operador de adição (+):

```
tupla1 = (1, 2, 3)
tupla2 = (4, 5, 6)
tupla_concatenada = tupla1 + tupla2
print(tupla_concatenada)  # Saída: (1, 2, 3, 4, 5, 6)
```

- Utilizando o método extend():

```
tupla1 = (1, 2, 3)
tupla2 = (4, 5, 6)
tupla1_list = list(tupla1)  # Convertendo a tupla1 em lista
tupla1_list.extend(tupla2)  # Extendendo a lista com os elementos da tupla2
tupla_concatenada = tuple(tupla1_list)  # Convertendo a lista de volta para uma tupla
print(tupla_concatenada)  # Saída: (1, 2, 3, 4, 5, 6)
```

Concatenação de tuplas

- Também podemos realizar a concatenação de mais de duas tuplas de uma vez:

```
tupla1 = (1, 2)
tupla2 = (3, 4)
tupla3 = (5, 6)
tupla_concatenada = tupla1 + tupla2 + tupla3
print(tupla_concatenada)  # Saída: (1, 2, 3, 4, 5, 6)
```

Concatenação de tuplas

- Para realizar a autoconcatenação de uma tupla, ou seja, concatenar uma tupla consigo mesma múltiplas vezes, podemos utilizar o operador *. Exemplo:

```
tupla = (1, 2, 3)
n = 3
tupla_autoconcatenada = tupla * n
print(tupla_autoconcatenada)
```

Saída:

```
(1, 2, 3, 1, 2, 3, 1, 2, 3)
```

- Observe que o operador de multiplicação (*) atua multiplicando a tupla pelo número especificado (n) e cria uma nova tupla com os elementos repetidos. A tupla original não é modificada.
- Além disso, se n for igual a 0, a operação retornará uma tupla vazia, exemplo:

```
tupla = (1, 2, 3)
n = 0

tupla_autoconcatenada = tupla * n
print(tupla_autoconcatenada)  # Saída: ()
```

Fatiamento de tuplas

- O fatiamento de tuplas permite extrair porções específicas de uma tupla. Podemos selecionar um intervalo de elementos de uma tupla usando a sintaxe de fatiamento, que é semelhante à utilizada em listas e strings.

- Sintaxe:

```
tupla[início:fim:passo]
```

- Onde:
  - O **início** é o índice do elemento onde o fatiamento começará (inclusive).
  - O **fim** é o índice do elemento onde o fatiamento terminará (exclusivo).
  - O **passo** é um valor opcional que define o intervalo entre os elementos a serem selecionados.

Fatiamento de tuplas: Exemplos

```
tupla = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)

# Fatiamento simples
sub_tupla = tupla[2:6]
print(sub_tupla)  # Saída: (3, 4, 5, 6)

# Fatiamento com passo
sub_tupla_passo = tupla[1:9:2]
print(sub_tupla_passo)  # Saída: (2, 4, 6, 8)

# Fatiamento do início até um índice específico
sub_tupla_inicio = tupla[:5]
print(sub_tupla_inicio)  # Saída: (1, 2, 3, 4, 5)

# Fatiamento de um índice específico até o final
sub_tupla_fim = tupla[7:]
print(sub_tupla_fim)  # Saída: (8, 9, 10)

# Fatiamento reverso
sub_tupla_reversa = tupla[::-1]
print(sub_tupla_reversa)  # Saída: (10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
```

Exemplo 1

- Iterando sobre uma tupla utilizando while:

```
frutas = ('maçã', 'banana', 'laranja', 'uva')
i = 0
while i < len(frutas):
    print(frutas[i])
    i += 1
```

Exemplo 2

- Função que retorna uma tupla contendo apenas os elementos pares de uma tupla de inteiros

```
def retorna_pares(t):
    """Retorna uma tupla com os elementos pares de uma tupla de inteiros"""
    t_aux = ()
    i = 0
    while i < len(t):
        if t[i] % 2 ==0:
            t_aux += (t[i],) #concatena tupla unitária
        i+=1

    return t_aux
```

Teste

```
>>> retorna_pares((12,3,4,9))
(12, 4)
```

Exemplo 3

- Função que retorna a soma dos elementos de uma tupla de números

```
def soma(t):
    """Retorna a soma dos elementos de uma tupla de números"""
    resultado = 0 #acumulador
    i = 0 # contador
    while i < len(t):
        resultado += t[i]
        i+=1

    return resultado
```

Teste

```
>>> soma((1,3,4,5))
13
```

Exemplo 4

- Função que retorna o elemento mínimo de uma tupla e a sua posição

```
def minimo(t):
    """Retorna o menor elemento e a posição dele
em uma tupla de números"""
    min = t[0]
    i = 1 # contador
    pos_min = 0
    while i < len(t):
        if t[i] < min:
            min = t[i]
            pos_min = i
        i+=1

    return min, pos_min
```

Teste

```
>>> minimo((1,3,4,-8,5))
(-8, 3)
>>> minimo((1,3,4,-8,5))[0]
-8
>>> menor, pos = minimo((1,3,4,-8,5))
>>> menor
-8
>>> pos
3
```

Exemplo 5

- Função para retornar o mínimo e o máximo de uma tupla de números:

```
def min_max(tupla):
    """
    Retorna o mínimo e o máximo de uma tupla de números.
    tuple->tuple
    """
    return min(tupla), max(tupla)
```

Teste

```
numeros = (5, 2, 8, 3, 1)
minimo, maximo = min_max(numeros)
print("Mínimo:", minimo)   # Saída: 1
print("Máximo:", maximo)   # Saída: 8
```

Exemplo 6

- Função para retornar o mínimo e o máximo de uma tupla de números:

```
def calcular_media(tupla):
    """
    Calcula a média dos elementos em uma tupla de números.
    tuple -> float
    """
    return sum(tupla) / len(tupla)
```

Teste

```
notas = (8.5, 9.0, 7.5, 9.5)
media = calcular_media(notas)
print("Média:", media)  # Saída: 8.875
```

Exercício 1

- Escreva uma função chamada **media_pares_impares(tupla)** que recebe uma tupla de números como entrada e retorna uma tupla contendo a média dos números pares e a média dos números ímpares da tupla. Por exemplo:

```
media_pares_impares((1, 2, 3, 4, 5, 6))   # Saída: (4, 3)
```

Exercício 2

- Escreva uma função chamada dividir_tupla(tupla) que recebe uma tupla como entrada e retorna duas tuplas, uma contendo os elementos de índice par e outra contendo os elementos de índice ímpar da tupla original. Por exemplo:

```
dividir_tupla((1, 2, 3, 4, 5, 6))  # Saída: ((1, 3, 5), (2, 4, 6))
```