Programação de Computadores I – BCC701

Conteúdo Teórico em Linguagem Python

Módulo 8 – Estruturas Heterogêneas

2020/1

Elaborado pela Comissão de Unificação da Disciplina BCC701,
com a colaboração de professores e estagiários docentes
http://www.decom.ufop.br/bcc701/

DECOM – Departamento de Computação
ICEB – Instituto de Ciências Exatas e Biológicas
UFOP – Universidade Federal de Ouro Preto

Sumário

MÓDULO

Estruturas Heterogêneas

No módulo anterior trabalhamos com vetores e matrizes, que se caracterizam pela possibilidade de armazenar diversos valores de um mesmo tipo de dados (estrutura de dados *homogênea*). Quando uma determinada estrutura de dados pode ser composta por diversos valores, sendo estes de tipos de dados diferentes, temos uma **estrutura de dados heterogênea**.

Por exemplo, para representar o cadastro de empregados, poderia ser implementada uma estrutura de dados composta pelos seguintes campos: *nome*, *idade*, *endereço*, *telefone*, *salário* e *cargo*. Nesse caso os campos *idade*, *telefone* e *salário* podem ser do tipo **numérico** enquanto que os demais campos podem ser do tipo **textual**.

8.1 Definição

Basicamente, uma **estrutura de dados heterogênea** pode ser definida como uma coleção de uma ou mais **variáveis relacionadas** (**campos**), onde cada variável pode ser de um tipo distinto. Esta estrutura heterogênea também é conhecida como **registro**. Então, um **registro** contém **campos**, estes, por sua vez, possuem valores associados, conforme a representação do exemplo dado anteriormente:

$$empregado \begin{cases} nome & = & < valor\ textual > \\ idade & = & < valor\ numérico > \\ endereco & = & < valor\ textual > \\ telefone & = & < valor\ numérico > \\ salario & = & < valor\ numérico > \\ cargo & = & < valor\ textual > \end{cases}$$

8.2 Sintaxe básica

Assim como ocorreu com vetores e matrizes, escolheremos uma estrutura com sintaxe simples, mas que permita representar o conceito de estruturas heterogêneas através de **registros** na linguagem **Python** na forma como foi definida conceitualmente na seção anterior. Esta estrutura em **Python** se chama **dicionário**, e permite manipular pares *chave-valor*, conforme o exemplo a seguir.

Exemplo 1:

O código a seguir cria uma variável chamada `empregado` que será utilizada para armazenar os dados de um empregado com os **campos** listados anteriormente. Inicialmente, todos os campos do tipo textual foram definidos com valor de uma **string** vazia '' e os campos numéricos com o valor $zero$. Porém, no momento de criação os valores podem ser definidos conforme sua necessidade. Ou seja, se você já possui nome, idade, etc, pode fornecê-los no momento da criação da variável.

```
1 empregado = {
2     "nome": '',
3     "idade": 0,
4     "endereco": '',
5     "telefone": 0,
6     "salario": 0,
7     "cargo": ''
8 }
```

Pela representação anterior, o registro `empregado` é:

$$empregado \begin{cases} nome & = & '' \\ idade & = & 0 \\ endereco & = & '' \\ telefone & = & 0 \\ salario & = & 0 \\ cargo & = & '' \end{cases}$$

É importante destacar que, após a criação de um registro, `empregado` é uma variável que pode ser manipulada em qualquer parte do programa. Para fazer o acesso ao conteúdo de um campo do registro, deve-se usar o nome do registro seguido por colchetes "[]" e dentro dos colchetes o nome do campo. Perceba que há uma grande semelhança com vetores e matrizes, porém, ao invés de índices, temos a **chave**, que no caso corresponde ao nome do campo em questão. Em **Python**, a chave pode ser representada por outros tipos de dados, não apenas **string**, como faremos uso aqui. No exemplo a seguir mostramos como modificar o registro criado anteriormente, para definir todos os campos de forma individual.

```
1 empregado["nome"] = 'Joaquim'
2 empregado["idade"] = 25
3 empregado["endereco"] = 'Rua A'
4 empregado["telefone"] = 53535353
5 empregado["salario"] = 1212.12
6 empregado["cargo"] = 'Vendedor'
```

Python diferencia letras maiúsculas de minúsculas, se você tentar acessar `empregado["Nome"]` ou `empregado["NOME"]`, por exemplo, ocorrerá um erro, pois estas **chaves** não existem no registro.

Após as alterações feitas anteriormente, o registro pode ser representado desta maneira:

$$empregado \begin{cases} \textit{nome} & = & `Joaquim' \\ \textit{idade} & = & 25 \\ \textit{endereco} & = & `RuaA' \\ \textit{telefone} & = & 53535353 \\ \textit{salario} & = & 1212.12 \\ \textit{cargo} & = & `Vendedor' \end{cases}$$

Exemplo 2:

Neste exemplo criamos um registro para representar um livro definindo os dados na atribuição. Depois fazemos acesso de **leitura**, para imprimir seus valores na tela. Como o acesso ao registro estará dentro do **print**, para diferenciar as aspas da mensagem das aspas da chave do registro, usamos aspas simples para delimitar a mensagem e as aspas duplas para delimitar a chave do registro.

```
1  livro = {
2      "titulo": 'Algoritmos e lógica de programação',
3      "autor": 'Marco Antonio Furlan de Souza',
4      "ano": 2005,
5      "editora": 'Cenage Learning',
6  }
7
8  print('Dados do livro: ')
9  print(f'- Título: {livro["titulo"]}')
10 print(f'- Autor: {livro["autor"]}')
11 print(f'- Ano: {livro["ano"]}')
12 print(f'- Editora: {livro["editora"]}')
```

Exemplo 3:

Agora nós apresentamos um programa que modifica o registro após a sua criação, mas agora inserindo um novo campo. Uma vez que o dicionário existe, podemos inserir novos campos simplesmente com uma atribuição usando a **chave** do novo campo.

```
1  livro = {
2      "titulo": 'Algoritmos e lógica de programação',
3      "autor": 'Marco Antonio Furlan de Souza',
4      "ano": 2005,
5      "editora": 'Cenage Learning',
6  }
7
8  livro["isbn"] = '978-8575227183'
9
10 print('Dados do livro: ')
11 print(f'- Título: {livro["titulo"]}')
12 print(f'- Autor: {livro["autor"]}')
13 print(f'- Ano: {livro["ano"]}')
14 print(f'- Editora: {livro["editora"]}')
15 print(f'- ISBN: {livro["isbn"]}')
```

8.3 Vetores e matrizes de registros

Em programação de computadores, podemos criar uma estrutura de dados que combina registros e vetores ou registros e matrizes a fim de resolver problemas mais complexos. Essa situação acontece, quando temos que armazenar, em nosso programa, vários registros que possuem a mesma estrutura. Por exemplo, armazenar dados de diversos alunos. Nesse caso, os dados de cada aluno será um registro, assim, temos vários registros diferentes com a mesma estrutura.

Exemplo 4:

Neste exemplo criaremos um programa que armazena o nome, a nota e a frequência dos alunos de uma turma em um único vetor através do registro aluno. As informações são fornecidas pelo usuário. Em seguida, ele imprime os alunos aprovados, que possuem nota maior ou igual a 6 e frequência maior ou igual a 75.

```
1  N = int(input('Quantidade de alunos: '))
2  alunos = [ ]
3  for a in range(N):
4      aluno = { }
5      print(f'Dados do aluno {a+1}:')
6      aluno["nome"] = input('- Nome: ')
7      aluno["nota"] = float(input('- Nota: '))
8      aluno["freq"] = int(input('- Frequência: '))
9      alunos.append(aluno)
10 print('\nAlunos aprovados:')
11 for a in range(N):
12     if alunos[a]['nota'] >= 6 and alunos[a]['freq'] >= 75:
13         print(f'- {alunos[a]["nome"]}')
```

Observe que inicialmente criamos um vetor vazio chamado `alunos`. Depois temos um laço **for** para fazer a entrada de dados de cada aluno, inserindo-os individualmente no vetor. Para cada aluno, a primeira ação é criar um registro vazio, sem nenhum campo, de forma similar à criação do vetor vazio, a diferença é o uso das chaves: `aluno = { }`. Estabelecemos aqui um padrão, o vetor, que representa vários alunos, está no plural `alunos`, e o registro que representa um único aluno, está no singular `aluno`. em seguida fazemos três entradas de dados, uma para cada campo, criando novos campos para o registro vazio criado no início do laço. Por fim, o registro `aluno` é adicionado ao vetor `alunos`. Com o término da execução do laço temos todos os dados de alunos armazenados no vetor, passamos então a imprimir os alunos aprovados. Um novo laço é iniciado para avaliar todos os alunos, imprimindo apenas os alunos aprovados. Para isso utilizamos uma decisão, verificando se o aluno `a` (variável contadora do laço, que determina os índices válidos do vetor), possui nota maior do que 6 e frequência maior do que 75. Observe que são duas condições ligadas pelo operador lógico `and` e que, para obter a nota do aluno precisamos acessar o índice do vetor e o nome do campo (`alunos[a]["nota"]`) e para obter a frequência do aluno precisamos acessar o índice do vetor e o nome do campo (`alunos[a]["freq"]`). Construindo a expressão lógica do **if** com estes componentes e as operações relacionais necessárias para atender à condição de aprovação. Caso a condição resulte em **verdadeiro**, deve-se imprimir o nome do aluno, caso contrário, não tem nada a ser feito. De forma similar, para obter o nome do aluno precisamos acessar o índice do vetor e o nome do campo (`alunos[a]["nome"]`).

A seguir um exemplo de execução do programa:

```
Quantidade de alunos: 3
Dados do aluno 1:
- Nome: João
- Nota: 6.5
- Frequência: 65
Dados do aluno 2:
- Nome: Maria
- Nota: 10
- Frequência: 95
Dados do aluno 3:
- Nome: José
- Nota: 3.8
- Frequência: 100

Alunos aprovados:
- Maria
```

O uso de registro em matrizes acontece de forma bastante similar, a diferença é que para matriz precisamos acessar dois índices antes de acessar o nome do campo. Como exemplo, imagine que você agora esteja representando o desempenho dos alunos em variadas disciplinas, para cada disciplina uma nota e uma frequência. Você pode representar os alunos nas linhas e as disciplinas nas colunas. Para obter a nota do aluno de índice de linha `lin` para a disciplina de índice de coluna `col`, você poderia acessar: `alunos[lin][col]["nota"]`. O termo `alunos[lin][col]` representa o acesso a um elemento específico da matriz, e o termo `["nota"]` representa o acesso ao campo nota deste elemento da matriz.

8.4 Registros como argumento ou retorno de função

Como os registros são armazenados em variáveis, assim como vetores e matrizes, eles podem ser utilizados como argumentos ou valor de retorno de uma função.

Exemplo 5:

Neste exemplo criamos uma função que recebe um registro que representa um **retângulo** com os campos **base** e **altura** e retorna um registro de medidas contendo o **perímetro** e a **área** do retângulo.

```
1  def calcRetangulo(retangulo):
2      resultado = { }
3      resultado["perímetro"] = 2 * (retangulo["base"] + retangulo["
           altura"])
4      resultado["área"] = retangulo["base"] * retangulo["altura"]
5      return resultado
6
7  ret = { "base": 10, "altura": 10 }
8  med = calcRetangulo(ret)
9  print(f'O perímetro do retângulo é {med["perímetro"]}')
10 print(f'A área do retângulo é {med["área"]}')
```

Observe que, como a chave do registro é do tipo textual, pode-se utilizar caracteres acentuados sem nenhum problema. Porém, é aconselhável que se estabeleça padrões nas definições de nomes de campos, uma boa prática é optar por não usar caracteres acentuados ou especiais. **Mais importante aqui é observar que os nomes dos campos usados dentro e fora da função devem ser exatamente iguais, diferentemente dos nomes das variáveis**.

Veja o resultado da execução do programa anterior:

```
O perímetro do retângulo é 40
A área do retângulo é 100
```

8.5 Exercícios propostos

Exercício 1

Implemente uma versão do Exemplo 4 para armazenar os dados de alunos em diferentes disciplinas. Sendo assim, deve-se utilizar uma matriz para armazenar os dados, em cada elemento da matriz será armazenada a nota e a frequência. As linhas representam alunos e as colunas representam disciplinas, que serão referenciadas pelos seus respectivos índices. Primeiramente o programa pergunta a quantidade de alunos e disciplinas, em seguida pergunta nota e frequência para cada elemento da matriz. Depois disso, para cada aluno, o programa deve identificar a disciplina com a maior nota, imprimindo a nota com precisão de uma casa decimal e a frequência nesta disciplina. Notas são valores reais e frequência inteiros.

Exemplos de execução:

Exemplo 1:

```
Quantidade de alunos: 2
Quantidade de disciplinas: 2

Dados do aluno 1:
 - Dados da disciplina 1:
  - Nota: 4
  - Frequência: 4
 - Dados da disciplina 2:
  - Nota: 3
  - Frequência: 3

Dados do aluno 2:
 - Dados da disciplina 1:
  - Nota: 2
  - Frequência: 2
 - Dados da disciplina 2:
  - Nota: 1
  - Frequência: 1

Maiores notas e frequências:
 - Aluno 1: nota = 4.0; frequência = 4
 - Aluno 2: nota = 2.0; frequência = 2
```

Exemplo 2:

```
Quantidade de alunos: 3
Quantidade de disciplinas: 2

Dados do aluno 1:
 - Dados da disciplina 1:
  - Nota: 1
  - Frequência: 10
 - Dados da disciplina 2:
  - Nota: 2
  - Frequência: 20

Dados do aluno 2:
 - Dados da disciplina 1:
  - Nota: 3
  - Frequência: 30
 - Dados da disciplina 2:
  - Nota: 4
  - Frequência: 40

Dados do aluno 3:
 - Dados da disciplina 1:
  - Nota: 5
  - Frequência: 50
 - Dados da disciplina 2:
  - Nota: 6
  - Frequência: 60

Maiores notas:
 - Aluno 1: nota = 2.0; frequência = 20
 - Aluno 2: nota = 4.0; frequência = 40
 - Aluno 3: nota = 6.0; frequência = 60
```

Exercício 2

Implemente um programa que receba informações sobre retas, armazenando-as em um vetor. Primeiramente pergunta-se quantas retas serão informadas, e em seguida pede-se as coordenadas dos dois pontos (**A** e **B**) que definem cada reta. Uma reta deve ser representada por um registro que contenha dois campos, representando os pontos. Cada ponto é um registro que possui as coordenadas **X** e **Y** como campos. Depois que o vetor estiver totalmente preenchido, você deverá calcular e imprimir o Comprimento de cada reta armazenada no vetor. O Comprimento é obtido através de uma função, definida em seu programa, que recebe a reta como argumento e retorna o comprimento: $C = \sqrt{(B_X - A_X)^2 + (B_Y - A_Y)^2}$. As coordenadas **X** e **Y** são números reais, e a saída das medidas deve possuir precisão de 2 casas decimais. **Dicas:** (1) para obter o ponto **B** de uma **reta**, você deve usar: `reta["B"]`; (2) para obter a coordenada **X** do ponto **B** desta mesma **reta**, você deve usar: `reta["B"]["X"]`, uma vez que temos aqui um registro "dentro" de outro registro.

Exemplos de execução:

Exemplo 1:

```
Informe a quantidade de retas: 1

Reta 1:
 - Coordenada X de A: 1
 - Coordenada Y de A: 4
 - Coordenada X de B: 1
 - Coordenada Y de B: 1

Medidas das retas:
 - Reta 1: 3.00
```

Exemplo 2:

```
Informe a quantidade de retas: 2

Reta 1:
 - Coordenada X de A: 3.5
 - Coordenada Y de A: 2.1
 - Coordenada X de B: 6
 - Coordenada Y de B: 8.6

Reta 2:
 - Coordenada X de A: -3
 - Coordenada Y de A: 0.9
 - Coordenada X de B: 5.8
 - Coordenada Y de B: -9.1

Medidas das retas:
 - Reta 1: 6.96
 - Reta 2: 13.32
```

8.6 Solução dos Exercícios Propostos

Exercício 1

Implemente uma versão do Exemplo 4 para armazenar os dados de alunos em diferentes disciplinas. Sendo assim, deve-se utilizar uma matriz para armazenar os dados, em cada elemento da matriz será armazenada a nota e a frequência. As linhas representam alunos e as colunas representam disciplinas, que serão referenciadas pelos seus respectivos índices. Primeiramente o programa pergunta a quantidade de alunos e disciplinas, em seguida pergunta nota e frequência para cada elemento da matriz. Depois disso, para cada aluno, o programa deve identificar a disciplina com a maior nota, imprimindo a nota com precisão de uma casa decimal e a frequência nesta disciplina. Notas são valores reais e frequência inteiros.

Exemplos de execução:

Exemplo 1:

```
Quantidade de alunos: 2
Quantidade de disciplinas: 2

Dados do aluno 1:
 - Dados da disciplina 1:
  - Nota: 4
  - Frequência: 4
 - Dados da disciplina 2:
  - Nota: 3
  - Frequência: 3

Dados do aluno 2:
 - Dados da disciplina 1:
  - Nota: 2
  - Frequência: 2
 - Dados da disciplina 2:
  - Nota: 1
  - Frequência: 1

Maiores notas e frequências:
 - Aluno 1: nota = 4.0; frequência = 4
 - Aluno 2: nota = 2.0; frequência = 2
```

Exemplo 2:

```
Quantidade de alunos: 3
Quantidade de disciplinas: 2

Dados do aluno 1:
 - Dados da disciplina 1:
  - Nota: 1
  - Frequência: 10
 - Dados da disciplina 2:
```

```
    - Nota: 2
    - Frequência: 20

  Dados do aluno 2:
   - Dados da disciplina 1:
    - Nota: 3
    - Frequência: 30
   - Dados da disciplina 2:
    - Nota: 4
    - Frequência: 40

  Dados do aluno 3:
   - Dados da disciplina 1:
    - Nota: 5
    - Frequência: 50
   - Dados da disciplina 2:
    - Nota: 6
    - Frequência: 60

  Maiores notas:
   - Aluno 1: nota = 2.0; frequência = 20
   - Aluno 2: nota = 4.0; frequência = 40
   - Aluno 3: nota = 6.0; frequência = 60
```

Solução:

```
nA = int(input('Quantidade de alunos: '))
nD = int(input('Quantidade de disciplinas: '))
alunos = [ ]
for a in range(nA):
    linha = [ ]
    print(f'\nDados do aluno {a+1}:')
    for d in range(nD):
        aluno = { }
        print(f' - Dados da disciplina {d+1}:')
        aluno["nota"] = float(input('  - Nota: '))
        aluno["freq"] = int(input('  - Frequência: '))
        linha.append(aluno)
    alunos.append(linha)
print('\nMaiores notas:')
for a in range(nA):
    maior = 0
    for d in range(nD):
        if alunos[a][d]["nota"] > alunos[a][maior]["nota"]:
            maior = d
    print(f' - Aluno {a+1}: nota = {alunos[a][maior]["nota"]:.1f};
       frequência = {alunos[a][maior]["freq"]}')
```

Exercício 2

Implemente um programa que receba informações sobre retas, armazenando-as em um vetor. Primeiramente pergunta-se quantas retas serão informadas, e em seguida pede-se as coordenadas dos dois pontos (**A** e **B**) que definem cada reta. Uma reta deve ser representada por um registro que contenha dois campos, representando os pontos. Cada ponto é um registro que possui as coordenadas **X** e **Y** como campos. Depois que o vetor estiver totalmente preenchido, você deverá calcular e imprimir o Comprimento de cada reta armazenada no vetor. O Comprimento é obtido através de uma função, definida em seu programa, que recebe a reta como argumento e retorna o comprimento: $C = \sqrt{(B_X - A_X)^2 + (B_Y - A_Y)^2}$. As coordenadas **X** e **Y** são números reais, e a saída das medidas deve possuir precisão de 2 casas decimais. **Dicas:** (1) para obter o ponto **B** de uma **reta**, você deve usar: `reta["B"]`; (2) para obter a coordenada **X** do ponto **B** desta mesma **reta**, você deve usar: `reta["B"]["X"]`, uma vez que temos aqui um registro "dentro" de outro registro.

Exemplos de execução:

Exemplo 1:

```
Informe a quantidade de retas: 1

Reta 1:
 - Coordenada X de A: 1
 - Coordenada Y de A: 4
 - Coordenada X de B: 1
 - Coordenada Y de B: 1

Medidas das retas:
 - Reta 1: 3.00
```

Exemplo 2:

```
Informe a quantidade de retas: 2

Reta 1:
 - Coordenada X de A: 3.5
 - Coordenada Y de A: 2.1
 - Coordenada X de B: 6
 - Coordenada Y de B: 8.6

Reta 2:
 - Coordenada X de A: -3
 - Coordenada Y de A: 0.9
 - Coordenada X de B: 5.8
 - Coordenada Y de B: -9.1

Medidas das retas:
 - Reta 1: 6.96
 - Reta 2: 13.32
```

Solução:

```
1  import math
2
3  def calcReta(reta):
4      d1 = reta["B"]["X"] - reta["A"]["X"]
5      d2 = reta["B"]["Y"] - reta["A"]["Y"]
6      C = math.sqrt(d1 ** 2 + d2 ** 2)
7      return C
8
9  N = int(input('Informe a quantidade de retas: '))
10 retas = [ ]
11 for i in range(N):
12     print(f'\nReta {i+1}:')
13     A = { }
14     A["X"] = float(input(' - Coordenada X de A: '))
15     A["Y"] = float(input(' - Coordenada Y de A: '))
16     B = { }
17     B["X"] = float(input(' - Coordenada X de B: '))
18     B["Y"] = float(input(' - Coordenada Y de B: '))
19     reta = { "A": A, "B": B }
20     retas.append(reta)
21 print('\nMedidas das retas:')
22 for i in range(N):
23     print(f' - Reta {i+1}: {calcReta(retas[i]):.2f}')
```