

Universidade Federal de Pernambuco  
Centro de Informática

# Tuplas (e Registros) e dicionários (e Tabelas)

Prof. Roberto Souto Maior de Barros  
[roberto@cin.ufpe.br](mailto:roberto@cin.ufpe.br)

# Registros



- São variáveis compostas **heterogêneas**.
- Visam representar um conjunto de variáveis (potencialmente) de tipos diferentes e que estão relacionados.
  - **Exemplo**: endereço, formado por sub-componentes rua, número, bairro, cidade, etc.
- Em muitas linguagens, servem também como *artifício* para “burlar” a limitação de só retornar um *único* resultado em funções (ou métodos).



# Registros



- Python *não* possui um tipo básico para registro!
- Uma opção para implementar a funcionalidade de registros e seus componentes em Python seria utilizar *classes* e *atributos* (que são as estruturas básicas da *orientação a objetos*), como em geral se faz em Java...
  - A classe (tipo) faria o papel do tipo registro.
  - Os atributos (dados) da classe fariam o papel dos seus componentes.
- Esta *não* será a opção que adotaremos...



# Registros e Tuplas



- A estrutura de Python que parece mais indicada para representar registros é a tupla – tipo *tuple*.
    - A principal diferença é que os registros suportados em outras linguagens utilizam um **nome** para cada um dos *componentes*, enquanto que no tipo *tuple* de Python o agrupamento é feito somente pela **ordem**...
      - Neste sentido, Python permite a manipulação de tuplas usando **índices** e **slices**, com exatamente as mesmas regras existentes para strings e listas.
- OBS:** Um artifício que pode ser adotado para ter *nomes* e ficar mais parecido com as outras LP é usar *variáveis* auxiliares na manipulação/modificação de tuplas...



# Registros e Tuplas



- Mais sobre registros e tuplas em Python...
  - Outra diferença é que, como no caso dos *strings*, o tipo *tuple* de *Python* lida com objetos **imutáveis**.
    - Porém, de forma semelhante à manipulação de strings, isto *não impede* que seja criado *outro objeto* com um conteúdo diferente (caso seja necessário) e que este novo objeto seja atribuído à *mesma variável*...
  - Tuplas são delimitadas por *parênteses* e os elementos são separados por *vírgulas*.
  - Assim como o tipo *list*, *tuple* também é um tipo como outro qualquer e pode ser usado sem definição prévia.
  - Tuplas também são implementadas como *objetos*.



# Algumas observações...

---



- Como no caso das listas, os elementos de uma tupla podem ser atribuídos a variáveis separadas, usando a atribuição múltipla.
  - Da mesma forma, o número de variáveis utilizado deve ser exatamente igual ao tamanho da tupla.
- Existem *vários* outros operadores e funções para manipular *tuplas*, geralmente com finalidades ou funcionalidades já cobertas por outros tipos.
  - Porque a maioria delas *não está* relacionada a *registros* e/ou *não estão* disponíveis em outras linguagens, *não* vamos cobri-las neste curso.



# Registros e Tuplas



- Exemplos:

rua = 'Rua da Hora' # Variáveis independentes...

numero = 230

cidade = 'Recife'

uf = 'PE'

cep = '52020-000'

phones = [12345678, 987654321]

end = (rua, numero, cidade, uf, cep, phones) # Cria a tupla...

endReduz = end [0:2] # Resultado é ('Rua da Hora', 230)

endR2 = end[0] + ', ' + str(end[1]) # Idem, em formato *string*...

r, n, ci, u, ce, f = end # Recuperando todos os itens da tupla...

end [1] = 200 # Causa erro fatal!



# Tabelas



- Uma aplicação bastante comum de *registros* são as chamadas *tabelas*:
  - Normalmente, tabelas são implementadas como um *array* de *registros*.
  - Geralmente, um dos componentes do registro serve como **identificador único**, o que é chamado de **chave** e normalmente tem tipo *numérico* (por eficiência).
- **Exemplos:**
  - Tabela de cursos de uma universidade.
    - A chave normalmente será um *código* de curso.
  - Tabela de departamentos de uma empresa.



# O tipo *dictionary* de Python



- Uma opção para implementar tabelas em Python é usar o tipo nativo *dictionary* (dicionário).
  - A *tabela* será um *dicionário* e os seus registros serão os elementos (itens) do dicionário:
    - A chave do dicionário será a chave da tabela.
    - O conteúdo associado à chave do dicionário (em geral) será uma *tupla* contendo os outros dados do registro.
  - Esta representação é boa quando:
    - a tabela é grande e/ou
    - pode ser alterada durante a execução do programa e/ou
    - não existe uma preocupação com a ordem de impressão dos elementos da tabela.



# Dicionários - sintaxe



- Dicionários são delimitadas por *chaves* (`{` e `}`) com seus elementos (itens) separados por *vírgulas*.
  - Cada elemento tem uma *chave* e um *conteúdo*, que são separados por *dois pontos* (`:`).
  - Apesar das *chaves* de um dicionário poderem ter tipo *string* (e outros), o ideal é que tenham tipo *inteiro*.
  - Não pode haver duas chaves iguais em um dicionário.
- O acesso a um item específico de um dicionário é feito usando a *chave* escrita entre *colchetes* (`[` e `]`).
- O tipo *dictionary* é um tipo como outro qualquer e pode ser usado sem definição prévia.



# Dicionários - sintaxe



- **OBS:** Os dicionários também são implementados como *objetos* e são **mutáveis**, como as listas.
- **Exemplos:**
  - `d1 = { }`    **# Cria um dicionário vazio – mais simples...**
  - `d2 = dict ( )`    **# Função que faz a mesma coisa...**
  - `d1 = {10 : 'Dez', 20 : 'Vinte'}`    **# Criação já com itens...**
  - `d1[30] = 'Trinta'`    **# Insere novo item com chave 30...**
  - `d1[10] = 'Dez.'`    **# Se a chave já existir, muda o valor...**
  - `valor = d1[10]`    **# Recupera o conteúdo pela chave...**
  - `valor = d1[50]`    **# Causa erro fatal!**
  - `existe = 20 in d1`    **# Retorna *True* se chave existir.**



# Dicionários - comando `for`



- O comando `for` pode ser usado para percorrer as *chaves* (e conseqüentemente os *elementos*) de um dicionário “uma a uma”...

# Sintaxe do comando `for` para dicionários...

```
for ch in qualquerDicionario :  
    comandoUsandoChave
```

- **Exemplo:**

```
dicion = {1:10, 2:20, 3:30, 4:40}
```

```
for ch in dicion : # ch recebe cada chave de dicion.
```

```
    print (ch, dicion [ch]) # mas ordem não é garantida.
```



# Tabelas sem usar dicionários...

---



- Outra opção para implementar tabelas em Python seria usar as representações recomendadas para *arrays* e *registros*. Especificamente:
  - A *tabela* será uma *lista* e seus elementos serão todos de um mesmo tipo de *tupla* previamente escolhido para representar o registro.
  - Um dos componentes desta tupla será a chave.
- Esta representação é boa quando:
  - a tabela é pequena e/ou
  - não é alterada durante a execução do programa e/ou
  - queremos manter a tabela ordenada pelas chaves.



# Exemplo completo

---



- Fazer um programa em Python para:
  - Ler uma tabela com N profissões, onde
    - O valor de N é informado antes pelo usuário.
    - Cada profissão é formada por um código (*número positivo*) e um nome e uma área (ambos *String*).
  - Depois o usuário fornecerá uma lista de códigos para que o programa informe o nome/área das profissões.
  - Se o código da profissão não existir na tabela, mostrar a mensagem “Profissão ... não existe na tabela.” e continuar.
  - O programa deve parar com a digitação de um código inválido (negativo ou zero).



# Resolução 1 – usando dicionário



# Profissões - V1 - Tabela = Dicionário com chaves e resto dos registros.

```
n = int(input ('Digite o tamanho da tabela de profissões: '))
```

```
while (n < 1) :
```

```
    n = int(input ('Tamanho deve ser inteiro e positivo. Tente novamente: '))
```

```
tab = { }      # Criação do dicionário...
```

```
for i in range (n) :
```

```
    codP = int(input ('Digite o código de uma profissão: '))
```

```
    while (codP < 1) :
```

```
        codP = int(input ('Código deve ser inteiro e positivo. Tente novamente: '))
```

```
    nomeP = input ('Digite o nome da profissão %d:\n' % (codP))
```

```
    areaP = input ('Digite a área da profissão %d:\n' % (codP))
```

```
    tab [codP] = (nomeP, areaP)      # Inserção no dicionário...
```

# Resolução 1 – usando dicionário



...

```
print ('Tabela com %d profissões foi lida corretamente.' % (n))
```

```
print ('Tabela ->', tab)
```

```
codP = int(input ('Digite um código de profissão para busca (<=0 para parar): '))
```

```
while codP > 0 :
```

```
    if codP in tab :      # Verifica se a profissão existe na tabela...
```

```
        nomeP, areaP = tab[codP]      # Recupera os outros dados...
```

```
        print ('Profissão %d é %s e sua área é %s.' % (codP, nomeP, areaP))
```

```
    else:
```

```
        print ('Profissão %d não existe na tabela.' % (codP))
```

```
    codP = int(input ('Digite outro código para busca (<=0 para parar): '))
```

```
print ('Fim de Programa')
```





# Resolução 2 – usando lista



# Profissões – V2 - Tabela = Lista de registros com chave inclusa no registro.

```
n = int(input ('Digite o tamanho da tabela de profissões: '))
```

```
while (n < 1) :
```

```
    n = int(input ('Tamanho deve ser inteiro e positivo. Tente novamente: '))
```

```
tab = [None]*n    # Criação da lista com tamanho correto (mais eficiente)...
```

```
for i in range (n) :
```

```
    codP = int(input ('Digite o código de uma profissão: '))
```

```
    while (codP < 1) :
```

```
        codP = int(input ('Código deve ser inteiro e positivo. Tente novamente: '))
```

```
    nomeP = input ('Digite o nome da profissão %d:\n' % (codP))
```

```
    areaP = input ('Digite a área da profissão %d:\n' % (codP))
```

```
    tab[i] = (codP, nomeP, areaP)    # Inserção na lista...
```

# Resolução 2 – usando lista



... # Omiti a impressão da tabela para caber no *slide*...

```
codP = int(input ('Digite um código de profissão para busca (<=0 para parar): '))

while codP > 0 :
    i = 0      # É preciso percorrer a lista como é feito em outras linguagens...
    while (i < n) and (codP != tab [i] [0]) :
        i = i + 1
    if i < n :    # Verifica se a profissão existe na tabela...
        nomeP, areaP = tab [i] [1:]    # Recupera os outros dados...
        print ('Profissão %d é %s e sua área é %s.' % (codP, nomeP, areaP))
    else:
        print ('Profissão %d não existe na tabela.' % (codP))

    codP = int(input ('Digite outro código para busca (<=0 para parar): '))

print ('Fim de Programa')
```

