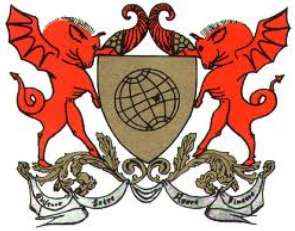
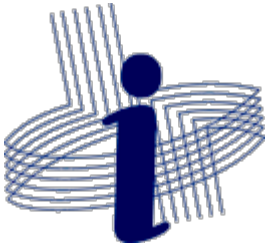


Universidade Federal de Viçosa
Departamento de Informática
Centro de Ciências Exatas e Tecnológicas



INF101 – Introdução à Programação II

Dicionários e Conjuntos

Definição

- Dicionários são estruturas de dados semelhantes às listas, mas com atributos de acesso diferentes
- Um *dicionário* é composto por um conjunto de pares de chaves e valores
- O dicionário consiste em relacionar uma chave a um valor específico
- Em outras linguagens, são também conhecidos por mapeamentos, listas associativas ou, até mesmo, por tabelas *hash*


Exemplo

Produto	Preço
Alface	R$0,45
Batata	R$1,20
Tomate	R$2,30
Feijão	R$5,50

- Dicionário correspondente

```
tabela = { "Alface": 0.45,
           "Batata": 1.20,
           "Tomate": 2.30,
           "Feijão": 5.50 }
```

Acesso

- Em Python, um dicionário é criado por meio de um par de chaves: `{ ... }`
- O acesso é feito pelo componente chave do par de valores
- Por exemplo, para obter o preço do feijão no dicionário do exemplo anterior, basta escrever `tabela["Feijão"]`, onde `tabela` é o nome da variável do tipo dicionário e `"Feijão"` é a chave
- O valor produzido pela notação acima é `5.50`


Acesso e Atribuição

- Diferentemente das listas, onde o índice (de acesso) é um número inteiro, os dicionários usam as chaves como índices
- Quando atribuímos um valor a uma chave, duas coisas podem ocorrer:
  1. Se a chave já existe, o valor associado é alterado para o novo valor
  2. Se a chave ainda não existe, o novo par, chave-valor, será inserido ao dicionário

Exemplos

```
>>> tabela = { "Alface": 0.45,
...            "Batata": 1.20,
...            "Tomate": 2.30,
...            "Feijão": 5.50 }
>>> print(tabela["Tomate"])
2.3
>>> print(tabela)
{'Batata': 1.2, 'Alface': 0.45, 'Tomate': 2.3, 'Feijão': 5.5}
>>> tabela["Tomate"] = 2.50
>>> print(tabela["Tomate"])
2.5
>>> tabela["Cebola"] = 1.35
>>> print(tabela)
{'Cebola': 1.35, 'Alface': 0.45, 'Tomate': 2.5, 'Batata': 1.2,
'Feijão': 5.5}
```


Acesso a Chave Inexistente

```
>>> tabela = { "Alface": 0.45,
...            "Batata": 1.20,
...            "Tomate": 2.30,
...            "Feijão": 5.50 }
>>> print(tabela["Manga"])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Manga'
```


Verificação da Existência de uma Chave

- O operador `in` é usado para verificar a existência de uma chave:

```
>>> tabela = { "Alface": 0.45,
...            "Batata": 1.20,
...            "Tomate": 2.30,
...            "Feijão": 5.50 }
>>> "Manga" in tabela
False
>>> "Batata" in tabela
True
```



Obtenção da Lista de Chaves e a de Valores

- Os métodos `keys()` e `values()` produzem geradores que podem ser usados eventualmente em comandos `for` e para produzir uma lista com a função `list`

```
>>> tabela = { "Alface": 0.45,
...            "Batata": 1.20,
...            "Tomate": 2.30,
...            "Feijão": 5.50 }
>>> tabela.keys()
dict_keys(['Alface', 'Tomate', 'Batata', 'Feijão'])
>>> tabela.values()
dict_values([0.45, 2.3, 1.2, 5.5])
```

Obtenção da Lista de Itens

- O método `items()` produz um conjunto com os itens de um dicionário e seu resultado também pode ser usado como iterador em comandos for

```
>>> tabela = { "Alface": 0.45,
...            "Batata": 1.20,
...            "Tomate": 2.30,
...            "Feijão": 5.50 }
>>> tabela.items()
dict_items([('Feijão', 5.5), ('Batata', 1.2),
('Tomate', 2.3), ('Alface', 0.45)])
```

Remoção de uma Chave

- Para apagar uma chave juntamente com o respectivo valor, usamos o comando `del`

```
>>> tabela = { "Alface": 0.45,
...            "Batata": 1.20,
...            "Tomate": 2.30,
...            "Feijão": 5.50 }
>>> del tabela["Tomate"]
>>> print(tabela)
{'Batata': 1.2, 'Alface': 0.45, 'Feijão': 5.5}
```

Listas × Dicionários

- Uma lista e um dicionário têm semelhanças
- Ambos estruturas de dados armazenam uma coleção de valores
- A grande diferença entre eles está no modo de indexação para acessá-los
- Uma lista usa números inteiros para os índices
- Um dicionário usa uma forma de índice chamada chave e o tipo da chave pode ser quase tudo, com a condição de que as chaves têm que ser únicas


Listas × Dicionários

- Usam-se dicionários quando os dados devem ser acessados por suas chaves e raramente haverá necessidade de acessá-los todos de uma vez só
- Além disso, com dicionários pode-se acessar o valor associado a uma chave rapidamente com baixo custo de pesquisa
- Porém dicionários não ordenam suas chaves em ordem nenhuma nem na ordem de inserção das chaves, como nas pilhas ou filas


Exemplo

- Aqui está algum código que cria uma lista e um dicionário e ilustra operações semelhantes em ambos tipos de dados:

```
lst = []  # uma lista vazia
dct = {}  # um dicionário vazio

# O método append adiciona itens (no fim) a uma lista
lst.append('Bicicleta')
lst.append('Corrida')
lst.append('Pilates')
```

Exemplo (Cont.)

```
# As próximas linhas adicionam ao dicionário alguns dados
dct['54328'] = 'Jasão Silva'
dct['55131'] = 'Paulo Gomes'
dct['57963'] = 'Ana Lobo'

# Podemos iterar sobre uma lista usando um laço for
for i in range(len(lst)):
    print(i, lst[i])

# Podemos iterar sobre um dicionário usando um laço for
for chave in dct.keys():
    print(chave, dct[chave])
```


Dicionários de Listas

- Em Python, podemos ter dicionários em que as chaves estejam associadas a listas ou, na realidade, a qualquer outra estrutura de dados, inclusive, a outros dicionários; isto é, podemos ter dicionário de dicionários
- Podemos ter, por exemplo, em um banco de dados de empréstimos de uma biblioteca, o registro dos livros que um dado leitor tomou emprestados


Dicionários de Listas

```
livros = { 'Alice': ['Tintin', 'Asterix', 'Mulherzinhas'],
           'Luiz':  ['Asterix'],
           'Ana':   ['Asterix', 'Mulherzinhas'],
           'Rui':   [],
           'João':  ['Tintin', 'Asterix', 'Dune'] }
```

- Neste caso, o nome do leitor é a chave e o valor associado é a lista dos livros tomados emprestados pelo leitor de uma biblioteca
- Por exemplo, se quisermos imprimir a lista dos livros que estão com Ana, podemos escrever o seguinte código:

```
for lst in livros['Ana']:
    print(lst)
```

Dicionários de Listas

- Considere agora o dicionário contendo o estoque de uma quitanda:

```
estoque = { 'tomate': [1000, 2.30],
            'alface': [ 500, 0.45],
            'batata': [2150, 1.20],
            'feijão': [ 100, 5.50] }
```

- Neste caso, o nome do produto é a chave e a tupla associada contém a quantidade disponível e o preço do respectivo produto

Dicionário de Tuplas

- O código a seguir processa uma lista de operações de venda calculando o preço total da venda e atualizando também a quantidade em estoque:

```
venda = [ ('tomate', 5), ('batata', 10), ('alface', 5) ]
total = 0
print('Vendas:\n')
for operacao in venda:
    produto, quantidade = operacao
    preco = estoque[produto][1]
    custo = preco * quantidade
    print('%12s: %3d x %6.2f = %6.2f' %
          (produto, quantidade, preco, custo))
    estoque[produto][0] -= quantidade
    total += custo
```


Dicionário de Tuplas

```
print('\n      Custo total: %15.2f\n' % total)
print('Estoque:\n')
for chave, dados in estoque.items():
    print('Descrição: ', chave)
    print('Quantidade: ', dados[0])
    print('Preço: %6.2f\n' % dados[1])
```


Conjuntos

- Conjuntos, como na matemática, são coleções de elementos
- Diferentemente dos dicionários, os elementos de um conjunto consistem apenas em valores—não há chaves
- Semelhantemente aos dicionários, os elementos não são ordenados e cada elemento é único


Conjuntos

- Semelhantemente às listas, os elementos de um conjunto podem ser de tipos diferentes
- Diferentemente das listas, os elementos de um conjunto não são acessados por meio de índices, pois não há ordem entre os elementos
- Em certas circunstâncias, usar conjunto é mais eficiente que usar listas
- O conjunto é mutável, portanto os elementos podem ser modificados a qualquer momento


Conjuntos

- Porém existe, em Python, um outro tipo de conjunto que é imutável; neste caso, ele é denominado *frozenset*
- Portanto, em um *frozenset*, os elementos não podem ser modificados

Criação de Conjuntos

- Para criar um conjunto vazio em Python, usamos a função *builtin* denominada `set()`:

```
>>> alunos_de_inf101 = set()
>>> type(alunos_de_inf101)
<class 'set'>
```

- Para popular um conjunto, usamos o método `add()`:

```
>>> alunos_de_inf101.add('Jasão Freitas')
>>> alunos_de_inf101.add('Ana Lopes')
>>> alunos_de_inf101.add('João Marques')
```


Criação de Conjuntos

- Para exibir os elementos correntes de um conjunto, fazemos assim:

```
>>> alunos_de_inf101
{'Ana Lopes', 'João Marques', 'Jasão Freitas'}
```

- Podemos usar também uma maneira menos maçante (para o ser humano):

```
alunos_de_inf101 = set(['Jasão Freitas', 'Ana Lopes', 'João Marques'])
```

- Nestes exemplos, os elementos são *strings*, mas poderiam ser números inteiros, números de ponto-flutuante, listas, tuplas etc.

Operações com Conjuntos

- Pertinência
  - Use o operador `in`:

```
>>> aluno = 'João Marques'
>>> aluno in alunos_de_inf101
True
>>> aluna = 'Teresa Oliveira'
>>> aluna in alunos_de_inf101
False
```

Operações com Conjuntos

- União
  - Use o método `union()`:

```
>>> alunos_de_inf103 = set(['Aline Souza',
'João Marques', 'Antônio Soares'])
>>> alunos_de_inf =
alunos_de_inf101.union(alunos_de_inf103)
>>> alunos_de_inf
{'Ana Lopes', 'João Marques', 'Aline Souza',
'Antônio Soares', 'Jasão Freitas'}
```


Operações com Conjuntos

- Interseção
  - Use o método `intersection()`:

```
>>> alunos_de_ambas =
alunos_de_inf103.intersection(alunos_de_inf101)
>>> alunos_de_ambas
{'João Marques'}
```

Operações com Conjuntos

- Diferença
  - Use o método `difference()`:

```
>>> alunos_so_de_inf101 =
alunos_de_inf101.difference(alunos_de_inf103)
>>> alunos_de_inf101
{'Jasão Freitas', 'João Marques', 'Ana Lopes'}
>>> alunos_de_inf103
{'Antônio Soares', 'João Marques', 'Aline Souza'}
>>> alunos_so_de_inf101
{'Ana Lopes', 'Jasão Freitas'}
```

Operações com Conjuntos

- Diferença simétrica
  - Use o método `symmetric_difference()`:

```
>>> alunos_so_de_uma_inf = alunos_de_inf101.symmetric_difference(alunos_de_inf103)
>>> alunos_de_inf101
{'Jasão Freitas', 'João Marques', 'Ana Lopes'}
>>> alunos_de_inf103
{'Antônio Soares', 'João Marques', 'Aline Souza'}
>>> alunos_so_de_uma_inf
{'Jasão Freitas', 'Ana Lopes', 'Antônio Soares', 'Aline Souza'}
```


Operações com Conjuntos

- Iteração sobre conjunto
  - Um conjunto pode ser iterado em um comando `for`:

```
>>> for a in alunos_de_inf101:
...     print(a)
João Marques
Jasão Freitas
Ana Lopes
```

Operações com Conjuntos

- Modificação de conjunto
  - Atualização de um conjunto
    - Use o método `update()`; o método `update()` suporta adição em massa:

```
>>> alunos_de_inf101
{'Jasão Freitas', 'João Marques', 'Ana Lopes'}
>>>
>>> alunos_de_inf101.update(['Ciro Silva', 'Dario Alencar'])
>>> alunos_de_inf101
{'Dario Alencar', 'Ana Lopes', 'Jasão Freitas', 'Ciro Silva', 'João Marques'}
```

Operações com Conjuntos

- Modificação de conjunto
  - Remoção de elementos
    - Há dois métodos, `remove()` e `discard()`
    - A diferença entre eles está na questão de que o método `remove()` exige que o elemento a ser removido tem que estar no conjunto; o método `discard()` não tem esta exigência


Operação com Conjuntos

- Remoção de elementos
  - Usando o método `remove()`:

```
>>> alunos_de_inf101
{'João Marques', 'Jasão Freitas', 'Ciro Silva',
'Ana Lopes', 'Dario Alencar'}
>>>
>>> alunos_de_inf101.remove('Ana Lopes')
>>> alunos_de_inf101.remove('Ciro Silva')
>>>
>>> alunos_de_inf101
{'Dario Alencar', 'Jasão Freitas', 'João
Marques'}
```


Operações com Conjuntos

- Remoção de elementos
  - A diferença de comportamento dos métodos `remove()` e `discard()`:

```
>>> alunos_de_inf101
{'Dario Alencar', 'Jasão Freitas', 'João Marques'}
>>> alunos_de_inf101.discard('Ana Lopes')
>>>
>>> alunos_de_inf101.remove('Ciro Silva')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Ciro Silva'
```

Operações com Conjuntos

- Observação
  - Podemos fazer adição em massa com o método `update()`, mas não podemos remover ou descartar em massa
  - Com os métodos `remove()` e `discard()` só podemos remover ou descartar só um elemento de cada vez


Exercício

- Para finalizar esta aula, vamos considerar o problema de construir um horário escolar
- Vamos resolver este problema de maneira bem simples, sem pretensão de tentar resolver o problema em uma situação realística
- A construção de um horário é um problema combinatório de difícil solução
- Nas disciplinas de Pesquisa Operacional, talvez vocês vejam algum método de solução deste problema de modo que possa ser aplicado em situações reais


Horário Escolar

- Vamos supor inicialmente que haja uma lista com as disciplinas oferecidas em um semestre
- Também vamos supor que haja um dicionário contendo o conjunto das disciplinas matriculadas por cada aluno no dado semestre
- O nome do aluno será a chave e o conjunto de disciplinas em que o aluno se matriculou será o respectivo valor

Horário Escolar

- A solução final do problema será impressa de uma lista contendo as sessões de disciplinas não conflitantes que poderão ser ministradas concomitantemente
- Observe que o conjunto das disciplinas matriculadas por um aluno contém disciplinas que não podem ser dadas simultaneamente
- Portanto vamos codificar primeiro um procedimento de redução baseado na lista de disciplinas oferecidas e no dicionário de matrículas dos alunos

Horário Escolar

```
disciplinas = ['INF100', 'INF101', 'INF103', 'MAT140',
               'MAT141', 'MAT144', 'LET100', 'FIS203']
matriculas =
   { "Ana Lobo": {'INF101', 'MAT140', 'LET100'},
     "João Marques": {'INF100', 'INF103', 'MAT141'},
     "Jasão Silva": {'INF100', 'MAT144', 'INF103', 'LET100'},
     "Paulo Gomes": {'INF101', 'LET100'},
     "Aline Souza": {'INF100', 'MAT141', 'LET100', 'INF103'},
     "Antônio Soares": {'INF103', 'MAT144','LET100'},
     "Teresa Oliveira": {'INF101', 'MAT141', 'LET100'},
     "José Farias": {'MAT144', 'LET100', 'FIS203'},
     "Ivo Lopes": {'INF101', 'FIS203', 'MAT144', 'LET100'},
     "Luíza Xisto": {'INF101', 'MAT141'} }

emptySet = set()  # conjunto vazio
```

Horário Escolar

```
# Determina os conjuntos de disciplinas conflitantes a partir
# das matrículas individuais dos alunos nas disciplinas.

conflito = [emptySet for d in disciplinas]
for a in matriculas.keys():
    for d in range(len(disciplinas)):
        if disciplinas[d] in matriculas[a]:
            conflito[d] = conflito[d].union(matriculas[a])
```


Horário Escolar

- A tarefa principal consiste na construção do horário
- O horário é uma lista de sessões em que cada sessão é uma seleção de disciplinas que não conflitam
- A partir do conjunto completo de disciplinas, selecionamos os subconjuntos de disciplinas adequadas não conflitantes subtraindo-os de uma variável denominada `restante`, até que o conjunto de disciplinas restantes fique vazio


Horário Escolar

- Como fazemos uma “próxima seleção adequada” de disciplinas?
- De início, podemos pegar qualquer disciplina simples do conjunto de disciplinas restantes
- Em seguida, a escolha de outras candidatas pode ficar restrita às disciplinas do conjunto de disciplinas restantes que não conflitam com a que foi selecionada inicialmente; chamamos tal conjunto de `tentativa`

Horário Escolar

- Quando buscamos uma candidata do conjunto `tentativa`, vemos que sua escolha depende de se a interseção das disciplinas já selecionadas com o conjunto de `conflito` da candidata seja vazia ou não
- O resultado desta seleção é colocado na variável `sessao`

Horário Escolar

```
# Constrói o horário que é uma lista de sessões em que cada
# sessão é uma seleção de disciplinas que não conflitam.

restante = set(disciplinas)
horario = []
while restante != emptySet:
    i = 0
    d = disciplinas[i]
    while d not in restante:
        i = i + 1
        d = disciplinas[i]
    sessao = {d}
    tentativa = restante.difference(conflito[i])
    for s in range(len(disciplinas)):
        if disciplinas[s] in tentativa:
            if conflito[s].intersection(sessao) == emptySet:
                sessao = sessao.union({disciplinas[s]})
    restante = restante.difference(sessao)
    horario.append(sessao)
```


Horário Escolar

- Para conhecermos as sessões de disciplinas não conflitantes, imprimiremos a lista `horario` da seguinte forma:

```
# Imprime as sessões não conflitantes do horário.
for i in range(len(horario)):
    print(horario[i])
```

- No caso dos dados que foram atribuídos inicialmente às variáveis `disciplinas` e `matriculas`, obtemos o seguinte resultado:

```
{'INF101', 'INF100'}
{'FIS203', 'MAT140', 'INF103'}
{'MAT144', 'MAT141'}
{'LET100'}
```