# Aula 11

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Estrutura de Dados
  - Dicionário (dict)

# Dicionário (*dict*)

- O tipo de dado dicionário, é uma estrutura de dados eficiente e mutável, acessada por uma chave não repetida, que pode ser de qualquer tipo imutável, tal como número, String ou Tupla.

# Dicionário (*dict*)

- O tipo de dado dicionário, é uma estrutura de dados eficiente e mutável, acessada por uma chave não repetida, que pode ser de qualquer tipo imutável, tal como número, String ou Tupla.

- O uso básico deste tipo de dado se dá quando se necessita de uma estrutura eficiente para armazenamento de pares **chave:valor**.

# Dicionário (*dict*)

- O tipo de dado dicionário, é uma estrutura de dados eficiente e mutável, acessada por uma chave não repetida, que pode ser de qualquer tipo imutável, tal como número, String ou Tupla.

- O uso básico deste tipo de dado se dá quando se necessita de uma estrutura eficiente para armazenamento de pares **chave:valor**.

- Para se recuperar um **valor**, basta informar sua **chave**.

# Dicionário (*dict*)

- Uma variável pode ser um dicionário contendo chave de tipo imutável, tal como números inteiros e de ponto flutuante, Strings e Tuplas, e valor de qualquer tipo.

# Dicionário (*dict*)

- Uma variável pode ser um dicionário contendo chave de tipo imutável, tal como números inteiros e de ponto flutuante, Strings e Tuplas, e valor de qualquer tipo.

- Dicionários tem pares, compostos de **chave:valor**, iteráveis sobre a **chave**, podendo seus elementos serem acessados por uma estrutura **for**.

# Dicionário (*dict*)

- Uma variável pode ser um dicionário contendo chave de tipo imutável, tal como números inteiros e de ponto flutuante, Strings e Tuplas, e valor de qualquer tipo.

- Dicionários tem pares, compostos de **chave:valor**, iteráveis sobre a **chave**, podendo seus elementos serem acessados por uma estrutura **for**.

- Além disso, um dicionário pode ser escrito diretamente no vídeo via comando **print**.

# As funções *dict()*, *del()* e *len()*

A função **dict()** associa um dicionário vazio a uma variável.

```
pares = dict()                    # ou pares = { }
print(pares)
```

# As funções *dict()*, *del()* e *len()*

A função **dict()** associa um dicionário vazio a uma variável.

```
pares = dict()                    # ou pares = { }
print(pares)
```

Para adicionar um novo par ao dicionário, por exemplo um par número:string, basta atribuir o valor ao nome do dicionário seguido pela chave entre colchetes. Caso já exista esta chave, o valor é atualizado.

```
pares = dict()
pares[13] = “Valor da Sorte”
print(pares)
```

# As funções *dict()*, *del()* e *len()*

A função **dict()** associa um dicionário vazio a uma variável.

```
pares = dict()                    # ou pares = { }
print(pares)
```

Para adicionar um novo par ao dicionário, por exemplo um par número:string, basta atribuir o valor ao nome do dicionário seguido pela chave entre colchetes. Caso já exista esta chave, o valor é atualizado.

```
pares = dict()
pares[13] = “Valor da Sorte”
print(pares)
```

A função **del** nomeDict[**chave**] retira o par **chave:valor** do dicionário. Caso a **chave** não esteja no dicionário um erro ocorre - KeyError.

```
pares = {51: “Boa ideia”, 13: “Valor da Sorte”, 31: “Dia do Azar”}
del pares[13]                     # ou pares.pop(13)
print(pares)
```

# As funções *dict()*, *del()* e *len()*

A função **dict()** associa um dicionário vazio a uma variável.

```
pares = dict()                    # ou pares = { }
print(pares)
```

Para adicionar um novo par ao dicionário, por exemplo um par número:string, basta atribuir o valor ao nome do dicionário seguido pela chave entre colchetes. Caso já exista esta chave, o valor é atualizado.

```
pares = dict()
pares[13] = “Valor da Sorte”
print(pares)
```

A função **del** nomeDict[**chave**] retira o par **chave:valor** do dicionário. Caso a **chave** não esteja no dicionário um erro ocorre - KeyError.

```
pares = {51: “Boa ideia”, 13: “Valor da Sorte”, 31: “Dia do Azar”}
del pares[13]                     # ou pares.pop(13)
print(pares)
```

A função **len()** retorna o tamanho do dicionário.

```
pares = {51: “Boa ideia”, 13: “Valor da Sorte”,31:”Dia do Azar}
print(len(pares))
```

# Criando um Dicionário de nomes:fones

O programa abaixo faz a leitura de cinco nomes e telefones e cria um dicionário com até cinco nomes distintos digitados pelo usuário. Ocorre uma escrita do conteúdo do dicionário a cada tentativa de inclusão de **nome:fone**.

```
pares = dict()                              # ou pares = { }
for i in range(5):
    nome = input("Digite nome: ")
    fone = input("Digite o telefone de "+nome+ ": ")
    pares[nome] = fone
    print(pares)
```

# Criando um Dicionário de nomes:fones

O programa abaixo faz a leitura de cinco nomes e telefones e cria um dicionário com até cinco nomes distintos digitados pelo usuário. Ocorre uma escrita do conteúdo do dicionário a cada tentativa de inclusão de **nome:fone**.

```
pares = dict()                          # ou pares = { }
for i in range(5):
    nome = input("Digite nome: ")
    fone = input("Digite o telefone de "+nome+ ": ")
    pares[nome] = fone
    print(pares)
```

# Criando um Dicionário Diretamente

```
pares = {"Maria":"3456-7922", "Ana":"3214-2211", "Giovanna":"4564-1234"}
print(pares)
```

# Visualizações sobre Dicionário

**d.items():** retorna uma visualização de todos os pares (chave, valor) de um dicionário **d**.

**d.keys():** retorna uma visualização de todas as chaves de um dicionário **d**.

**d.values():** retorna uma visualização de todos os valores de um dicionário **d**.

# Iterando sobre Chaves ou Valores

```
for chave in pares:
    print(chave, ":", pares[chave])
```

# Iterando sobre Chaves ou Valores

```
for chave in pares:
    print(chave, ":", pares[chave])
```

```
for chave, valor in pares.items():
    print(chave, ":", valor)
```

# Iterando sobre Chaves ou Valores

```
for chave in pares:
    print(chave, ":", pares[chave])
```

```
for chave, valor in pares.items():
    print(chave, ":", valor)
```

```
for chave in sorted(pares):
    print(chave, ":", pares[chave])
```

# Iterando sobre Chaves ou Valores

```
for chave in pares:
    print(chave, ":", pares[chave])
```

```
for chave, valor in pares.items():
    print(chave, ":", valor)
```

```
for chave in sorted(pares):
    print(chave, ":", pares[chave])
```

```
for chave in sorted(pares.keys()):
    print(chave, ":", pares[chave])
```

# Iterando sobre Chaves ou Valores

```
for chave in pares:
    print(chave, ":", pares[chave])
```

```
for chave, valor in pares.items():
    print(chave, ":", valor)
```

```
for chave in sorted(pares):
    print(chave, ":", pares[chave])
```

```
for chave in sorted(pares.keys()):
    print(chave, ":", pares[chave])
```

```
for valor in pares.values():
    print(valor)
```

# Exemplo de Aplicação de Dicionário

Supermercado de Produtos, onde cada Produto possui:

(a) Código não repetido (a chave) – representado por String;
(b) Descrição – representada por String;
(c) Quantidade – representada por inteiro;
(d) Preço – representado por número de ponto flutuante; e
(e) Data de Validade – representada por Tupla (dia, mês, ano).

# Exemplo de Aplicação de Dicionário

Supermercado de Produtos, onde cada Produto possui:

(a) Código não repetido (a chave) – representado por String;
(b) Descrição – representada por String;
(c) Quantidade – representada por inteiro;
(d) Preço – representado por número de ponto flutuante; e
(e) Data de Validade – representada por Tupla (dia, mês, ano).

```
produtos = { }
preencher(produtos, 5)
mostrar(produtos)
vender(produtos, {“xkk”:3, “yzb”:2})
mostrar(produtos)
```

```
# Subprogramas
def mostrar(prods):
    for chave, valor in sorted(prods.items()):
        print(chave, " – ", valor)
    return None
def preencher(prods, entradas):
    for i in range(entradas):
        cod = input("Código: ")
        desc = input("Descrição: ")
        qtd = int(input("Quantidade: "))
        preco = float(input("Valor: "))
        data = input("Limite de Validade - dd/mm/aa: ")
        partes = data.split("/")
        prods[cod] = [desc, qtd, preco, (int(partes[0]),int(partes[1]), int(partes[2]))]
    return None
def vender(prods, codsQtds):
    return None
# Programa Principal
produtos = { }
preencher(produtos, 5)
mostrar(produtos)
vender(produtos, {"xkk":3, "yzb":2})
mostrar(produtos)
```

```
# Subprogramas
def mostrar(prods):
    for chave, valor in sorted(prods.items()):
        print(chave, " – ", valor)
    return None
def preencher(prods, entradas):
    for i in range(entradas):
        cod = input("Código: ")
        desc = input("Descrição: ")
        qtd = int(input("Quantidade: "))
        preco = float(input("Valor: "))
        data = input("Limite de Validade - dd/mm/aa: ")
        partes = data.split("/")
        prods[cod] = [desc, qtd, preco, (int(partes[0]),int(partes[1]), int(partes[2]))]
    return None
def vender(prods, codsQtds):
    return None
# Programa Principal
produtos = { }
preencher(produtos, 5)
mostrar(produtos)
vender(produtos, {"xkk":3, "yzb":2})
mostrar(produtos)
```

```
# Subprogramas
def mostrar(prods):
    for chave, valor in sorted(prods.items()):
        print(chave, " – ", valor)
    return None
def preencher(prods, entradas):
    for i in range(entradas):
        cod = input("Código: ")
        desc = input("Descrição: ")
        qtd = int(input("Quantidade: "))
        preco = float(input("Valor: "))
        data = input("Limite de Validade - dd/mm/aa: ")
        partes = data.split("/")
        prods[cod] = [desc, qtd, preco, (int(partes[0]),int(partes[1]), int(partes[2]))]
    return None
def vender(prods, codsQtds):
    return None
# Programa Principal
produtos = { }
preencher(produtos, 5)
mostrar(produtos)
vender(produtos, {"xkk":3, "yzb":2})
mostrar(produtos)
```

```
# Subprogramas
def mostrar(prods):
    for chave, valor in sorted(prods.items()):
        print(chave, " – ", valor)
    return None
def preencher(prods, entradas):
    for i in range(entradas):
        cod = input("Código: ")
        desc = input("Descrição: ")
        qtd = int(input("Quantidade: "))
        preco = float(input("Valor: "))
        data = input("Limite de Validade - dd/mm/aa: ")
        partes = data.split("/")
        prods[cod] = [desc, qtd, preco, (int(partes[0]),int(partes[1]), int(partes[2]))]
    return None
def vender(prods, codsQtds):
    return None
# Programa Principal
produtos = { }
preencher(produtos, 5)
mostrar(produtos)
vender(produtos, {"xkk":3, "yzb":2})
mostrar(produtos)
```

```
# Subprogramas
def mostrar(prods):
    for chave, valor in sorted(prods.items()):
        print(chave, " – ", valor)
    return None
def preencher(prods, entradas):
    for i in range(entradas):
        cod = input("Código: ")
        desc = input("Descrição: ")
        qtd = int(input("Quantidade: "))
        preco = float(input("Valor: "))
        data = input("Limite de Validade - dd/mm/aa: ")
        partes = data.split("/")
        prods[cod] = [desc, qtd, preco, (int(partes[0]),int(partes[1]), int(partes[2]))]
    return None
def vender(prods, codsQtds):
    return None
# Programa Principal
produtos = { }
preencher(produtos, 5)
mostrar(produtos)
vender(produtos, {"xkk":3, "yzb":2})
mostrar(produtos)
```

```
# Subprogramas
...
def vender(prods, codsQtds):
    totalGasto = 0
    for chave in codsQtds:
        if chave not in prods:
            print(chave, “- Código Inexistente”)
        else:
            item = prods[chave]
            if item[1]<codsQtds[chave]:
                print(chave, “- Quantidade Insuficiente”)
            else:
                item[1]-=codsQtds[chave]
                prods[chave]=item
                print(item[0], “x”, codsQtds[chave], “- Subtotal:”, item[2]*codsQtds[chave])
                totalGasto += item[2]*codsQtds[chave]
    print(“Total da Nota Fiscal:”, totalGasto)
    return None
# Programa Principal
produtos = { }
preencher(produtos, 5)
mostrar(produtos)
vender(produtos, {“xkk”:3, “yzb”:2})
mostrar(produtos)
```

# Consultando Dicionários

Se consultarmos diretamente uma chave inexistente em um dicionário, obteremos uma exceção e uma respectiva mensagem de erro: **KeyError**. Veja o exemplo abaixo:

# Consultando Dicionários

Se consultarmos diretamente uma chave inexistente em um dicionário, obteremos uma exceção e uma respectiva mensagem de erro: **KeyError**. Veja o exemplo abaixo:

```
pares = {51: "Boa ideia", 13: "Valor da Sorte", 31: "Dia do Azar"}
print(pares[51])          # escreve Boa Ideia
print(pares[52])          # aborta a execução com KeyError: 52
```

# Consultando Dicionários

Se consultarmos diretamente uma chave inexistente em um dicionário, obteremos uma exceção e uma respectiva mensagem de erro: **KeyError**. Veja o exemplo abaixo:

```
pares = {51: "Boa ideia", 13: "Valor da Sorte", 31: "Dia do Azar"}
print(pares[51])          # escreve Boa Ideia
print(pares[52])          # aborta a execução com KeyError: 52
```

Para se evitar este tipo de erro durante a consulta podemos utilizar a função **get**, descrita a seguir.

# A Função *get()*

A função **get(chave)** retorna **None** se não existir aquela **chave** no dicionário, caso contrário retorna o respectivo **valor**.

```
pares = { }
print(pares.get(44))                    # escreve None
```

```
pares = {51: "Boa ideia", 13: "Valor da Sorte", 31: "Dia do Azar"}
print(pares.get(13))                    # escreve Valor da Sorte
```

# A Função *get()*

A função **get(chave)** retorna **None** se não existir aquela **chave** no dicionário, caso contrário retorna o respectivo **valor**.

```
pares = { }
print(pares.get(44))                    # escreve None
```

```
pares = {51: "Boa ideia", 13: "Valor da Sorte", 31: "Dia do Azar"}
print(pares.get(13))                    # escreve Valor da Sorte
```

A função **get(chave,default)** retorna **default** se não existir aquela **chave** no dicionário, caso contrário retorna o respectivo **valor**.

```
pares = {51: "Boa ideia", 13: "Valor da Sorte", 31: "Dia do Azar"}
print(pares.get(22, "Vazia"))           # escreve Vazia
print(pares.get(51, "Vazia"))           # escreve Boa Ideia
```

# Uso de Dicionário para Matriz Esparsa

```
# Subprograma
def mostra(vs):                          # exibe a matriz organizadamente
    qtdLinhas = vs["Número de Linhas"]
    qtdColunas = vs["Número de Colunas"]
    for linha in range(qtdLinhas):
        for coluna in range(qtdColunas):
            valor = vs.get((linha,coluna),0)
            print("%4d"%valor, end=' ')      # saída formatada - 4 espaços para cada valor
        print()                              # pula para a próxima linha
    return
# Programa Principal
# valores mantém apenas as dimensões e os poucos números diferentes de zero
valores = {}
valores["Número de Linhas"]= 5           # chave string mantém a quantidade de linhas
valores["Número de Colunas"] = 10        # chave string mantém a quantidade de colunas
valores[(1,1)]= 13                       # chave tupla (1,1) com um valor não zero
valores[(0,8)] = -4                      # chave tupla (0,8) com um valor não zero
valores[(4,7)] = 75                      # chave tupla (4,7) com um valor não zero
print(valores)
mostra(valores)
```

# Uso de Dicionário para Matriz Esparsa

```
# Subprograma
def mostra(vs):                          # exibe a matriz organizadamente
    qtdLinhas = vs["Número de Linhas"]
    qtdColunas = vs["Número de Colunas"]
    for linha in range(qtdLinhas):
        for coluna in range(qtdColunas):
            valor = vs.get((linha,coluna),0)
            print("%4d"%valor, end=' ')      # saída formatada - 4 espaços para cada valor
        print()                              # pula para a próxima linha
    return
# Programa Principal
# valores mantém apenas as dimensões e os poucos números diferentes de zero
valores = {}
valores["Número de Linhas"]= 5           # chave string mantém a quantidade de linhas
valores["Número de Colunas"] = 10        # chave string mantém a quantidade de colunas
valores[(1,1)]= 13                       # chave tupla (1,1) com um valor não zero
valores[(0,8)] = -4                      # chave tupla (0,8) com um valor não zero
valores[(4,7)] = 75                      # chave tupla (4,7) com um valor não zero
print(valores)
mostra(valores)
```

# Uso de Dicionário para Matriz Esparsa

```
# Subprograma
def mostra(vs):                          # exibe a matriz organizadamente
    qtdLinhas = vs["Número de Linhas"]
    qtdColunas = vs["Número de Colunas"]
    for linha in range(qtdLinhas):
        for coluna in range(qtdColunas):
            valor = vs.get((linha,coluna),0)
            print("%4d"%valor, end=' ')      # saída formatada - 4 espaços para cada valor
        print()                              # pula para a próxima linha
    return
# Programa Principal
# valores mantém apenas as dimensões e os poucos números diferentes de zero
valores = {}
valores["Número de Linhas"]= 5           # chave string mantém a quantidade de linhas
valores["Número de Colunas"] = 10        # chave string mantém a quantidade de colunas
valores[(1,1)]= 13                       # chave tupla (1,1) com um valor não zero
valores[(0,8)] = -4                      # chave tupla (0,8) com um valor não zero
valores[(4,7)] = 75                      # chave tupla (4,7) com um valor não zero
print(valores)
mostra(valores)
```

# Faça os Exercícios Relacionados a essa Aula

Clique no botão para visualizar os enunciados:

# Aula 11

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Estrutura de Dados
  - Dicionário (dict)