escola britânica de artes criativas & tecnologia

Desenvolvedor Full Stack Python

Estruturas de Dados

Módulo 02 | Python: Estruturas de Dados

Caderno de Aula

Professor André Perez

Tópicos

1. Listas;
2. Conjuntos;
3. Dicionários.

Aulas

1. Listas

1.1. Motivação

O aplicativo do seu banco registra toda a sua movimentação financeira. O final do dia, o app consolida o saldo final para que você possa controlar sua vida financeira.

In [ ]:
```
dia_11_saldo_inicial = 1000
```

In [ ]:
```
dia_11_transacao_1 = 243
dia_11_transacao_2 = -798.58
dia_11_transacao_3 = 427.12
dia_11_transacao_4 = -10.91
```

In [ ]:
```
dia_11_saldo_final = dia_11_saldo_inicial + \
                     dia_11_transacao_1 + \
                     dia_11_transacao_2 + \
                     dia_11_transacao_3 + \
                     dia_11_transacao_4
```

In [ ]:
```
print(dia_11_saldo_final)
```

Será que exista uma forma melhor de armazenar as transações diárias?

1.2. Definição

Armazenam sequências mutáveis e ordenadas de valores. São do tipo `list` :

In [ ]:
```
usuario_web = [
    'André Perez',
    'andre.perez',
    'andre123',
    'andre.perez@gmail.com'
]

print(usuario_web)
print(type(usuario_web))
```

In [ ]:
```
idade = 20
saldo_em_conta = 723.15
usuario_loggedin = True

usuario_web = [
    'André Perez',
    idade,
    'andre.perez',
    'andre123',
    'andre.perez@gmail.com',
    saldo_em_conta,
    usuario_loggedin
]

print(usuario_web)
print(type(usuario_web))
```

1.3. Operações

As operações da estrutura do tipo *list* são:

- `+` (concatenação).

Exemplo: Fabricantes de *hardware* mobile

In [ ]:
```
fabricantes_mobile_china = ['xiaomi', 'huawei']
fabricantes_mobile_eua = ['apple', 'motorola']
fabricantes_mobile = fabricantes_mobile_china + fabricantes_mobile_eua

print(fabricantes_mobile_china)
print(fabricantes_mobile_eua)
print(fabricantes_mobile)
```

Outra operação muito utilizada é a de fatiamento (*slicing*), semelhante ao de *strings*:

Fatiamento fixo:

In [ ]:
```
print(f'0: {fabricantes_mobile[0]}')
print(f'-1: {fabricantes_mobile[-1]}')
```

Fatiamento por intervalo:

In [ ]:
```

```

In [ ]:
```
fabricantes_mobile_china = fabricantes_mobile[0:2]
fabricantes_mobile_eua = fabricantes_mobile[2:len(fabricantes_mobile)]

print('china: ' + str(fabricantes_mobile_china))
print('eua: ' + str(fabricantes_mobile_eua))
```

Podemos adicionar elementos a uma posição específica da lista:

In [ ]:
```
print(fabricantes_mobile)
```

In [ ]:
```
fabricantes_mobile[2] = 'nokia'
print(fabricantes_mobile)
```

1.4. Métodos

São métodos nativos do Python que nos ajudam a trabalhar no dia a dia com listas.

In [ ]:
```
juros = [0.05, 0.07, 0.02, 0.04, 0.08]
print(juros)
```

In [ ]:
```
# inserir um elemento sem substituir: list.insert(index, val)
juros.insert(0, 0.10)
print(juros)
```

In [ ]:
```
# inserir um elemento no fim da lista: list.append(val)
juros.append(0.09)
print(juros)
```

In [ ]:
```
# remover um elemento pelo valor: list.remove(val)
juros.remove(0.1)
print(juros)
```

In [ ]:
```
# remover um elemento pelo índice: list.pop(val)
terceiro_juros = juros.pop(2)
print(terceiro_juros)
```

In [ ]:
```
print(juros)
```

1.5. Conversão

Podemos converter alguns tipos de variáveis em listas, como *strings*.

In [ ]:
```
email = 'andre.perez@gmail.com'
caracteres_email = list(email)

print(email)
print(caracteres_email)
```

1.6. Revisitando a motivação

In [ ]:
```

```

```
dia_11_saldo_inicial = 1000
```

In [ ]:
```
dia_11_transacoes = []

dia_11_transacoes.append(243)
dia_11_transacoes.append(-798.58)
dia_11_transacoes.append(427.12)
dia_11_transacoes.append(-10.91)

print(dia_11_transacoes)
```

In [ ]:
```
dia_11_saldo_final = dia_11_saldo_inicial + \
                     dia_11_transacoes[0] + \
                     dia_11_transacoes[1] + \
                     dia_11_transacoes[2] + \
                     dia_11_transacoes[3]

print(dia_11_saldo_final)
```

2. Conjuntos

2.1. Motivação

Você trabalha como analista de dados de mídias sociais e precisa descobrir todas as *hashtags* que alcançaram o *top trending* do Twitter durante uma semana. Você já conseguiu as *hashtags* por dia da semana:

In [ ]:
```
hashtags_seg = ['#tiago', '#joao', '#bbb']
hashtags_ter = ['#sarah', '#bbb', '#fiuk']
hashtags_qua = ['#gil', '#thelma', '#lourdes']
hashtags_qui = ['#rafa', '#fora', '#danilo']
hashtags_sex = ['#juliete', '#arthur', '#bbb']
```

Um simples concateção de listas fará com que a *hashtag* #bbb, entre outras, apareça mais de uma vez.

In [ ]:
```
hashtags_semana = hashtags_seg + \
                  hashtags_ter + \
                  hashtags_qua + \
                  hashtags_qui + \
                  hashtags_sex

print(hashtags_semana)
```

2.2. Definição

Armazenam sequências imutáveis e desordenadas valores, sem repetição. São do tipo `set`:

In [ ]:
```
frutas = {'banana', 'maca', 'uva', 'uva'}

print(frutas)
print(type(frutas))
```

2.3. Operações

As operações da estrutura do tipo *set* são:

- `-` (diferença).

Exemplo: Países da europa.

In [ ]:
```
norte_europa = {'reino unido', 'suecia', 'russia', 'noruega', 'dinamarca'}
escandinavia = {'noruega', 'dinamarca', 'suecia'}
```

In [ ]:
```
norte_europa_nao_escandivano = norte_europa - escandinavia
print(norte_europa_nao_escandivano)
```

In [ ]:
```
escandivano_nao_norte_europa = escandinavia - norte_europa
print(escandivano_nao_norte_europa)
```

2.4. Métodos

São métodos nativos do Python que nos ajudam a trabalhar no dia a dia com conjuntos.

In [ ]:
```
cursos = {'Exatas', 'Humanas', 'Biológicas'}
print(cursos)
```

In [ ]:
```
# inserir um elemento no conjunto: set.add(val)
cursos.add('Saúde')
print(cursos)
```

In [ ]:
```
# remover um elemento no conjunto: set.remove(val)
cursos.remove('Saúde')
print(cursos)
```

2.5. Conversão

Podemos converter conjuntos para lista e vice e versa.

In [ ]:
```
times_paulistas = {'São Paulo', 'Palmeiras', 'Corinthians', 'Santos'}

print(times_paulistas)
print(type(times_paulistas))
```

In [ ]:
```
print(list(times_paulistas))
print(type(list(times_paulistas)))
```

2.6. Revisitando a motivação

In [ ]:
```
print(hashtags_semana)
print(len(hashtags_semana))
```

In [ ]:
```
hashtags_semana = list(
    set(
        hashtags_seg + \
```

```
        hashtags_ter + \
        hashtags_qua + \
        hashtags_qui + \
        hashtags_sex
    )
)

print(hashtags_semana)
print(len(hashtags_semana))
```

3. Dicionários

3.1. Motivação

Para se conectar a uma rede wi-fi, você precisa de duas informações: o nome da rede e a senha de acesso. Quando você vai acessar uma nova rede, você encontra uma lista de redes disponíveis:

In [ ]:
```
wifi_disponiveis = ['rede1', 'cnx_cnx', 'uai-fi', 'r3d3']
print(wifi_disponiveis)
```

Você consegue identificar quais são os nome de redes e suas respectivas senhas? Talvez uma `list` não seja a melhor opção para armazenar esse tipo de dado.

3.2. Definição

Armazenam sequências no formato chave-valor. São do tipo`dict`.

In [ ]:
```
brasil = {'capital': 'Brasília', 'idioma': 'Português', 'populacao': 210}

print(brasil)
print(type(brasil))
```

Não é permite chaves duplicadas.

In [ ]:
```
carro = {
    'marca': 'Volkswagen',
    'modelo': 'Polo',
    'ano': 2021,
    'ano': 2004
}

print(carro)
```

Podemos criar dicionários compostos:

In [ ]:
```
cadastro = {
    'andre': {
        'nome': 'Andre Perez',
        'ano_nascimento': 1992,
        'pais': {
            'pai': {
              'nome': '<nome-do-pai> Perez',
              'ano_nascimento': 1971
            },
            'mae': {
              'nome': '<nome-da-mae> Perez',
```

```
                'ano_nascimento': 1973
            },
        }
    }
}

print(cadastro)
```

In [ ]:
```
cadastro['andre']['pais']['mae']['ano_nascimento']
```

3.3. Operações

In [ ]:
```
credito = {'123': 750, '789': 980}
```

Elementos são acessados pela sua chave.

In [ ]:
```
score_123 = credito['123']
score_789 = credito['789']

print(score_123)
print(score_789)
```

Elementos são atualizados pela sua chave.

In [ ]:
```
credito['123'] = 435
print(credito)
```

Para adicionar um novo elemento, basta criar um novo elemento chave-valor:

In [ ]:
```
credito['456'] = 1000
print(credito)
```

3.4. Métodos

São métodos nativos do Python que nos ajudam a trabalhar no dia a dia com dicionários.

In [ ]:
```
artigo = dict(
    titulo='Modulo 02 | Python: Estruturas de Dados',
    corpo='Topicos, Aulas, Listas, Conjuntos, Dicionários, ...',
    total_caracteres=1530
)
```

In [ ]:
```
# adicionar/atualizar um elemento pelo chave-valor: dict.update(dict)
print(artigo)
artigo.update({'total_caracteres': 7850})
print(artigo)

artigo['total_caracteres'] = 7850
```

In [ ]:
```
# remover um elemento pelo chave: dict.pop(key)
print(artigo)
total_caracteres = artigo.pop('total_caracteres')
print(artigo)
```

3.5. Conversão

Podemos converter as chaves e os items de um dicionário em uma lista.

In [ ]:
```
artigo = dict(
    titulo='Modulo 02 | Python: Estruturas de Dados',
    corpo='Topicos, Aulas, Listas, Conjuntos, Dicionários, ...',
    total_caracteres=1530
)
```

In [ ]:
```
chaves = list(artigo.keys())

print(chaves)
print(type(chaves))
```

In [ ]:
```
valores = list(artigo.values())

print(valores)
print(type(valores))
```

3.6. Revisitando a motivação

In [ ]:
```
wifi_disponiveis = []
```

In [ ]:
```
rede = {'nome': 'rede1', 'senha': 'cnx_cnx'}
wifi_disponiveis.append(rede)
```

In [ ]:
```
rede = {'nome': 'uai-fi', 'senha': 'r3d3'}
wifi_disponiveis.append(rede)
```

In [ ]:
```
print(wifi_disponiveis)
```