Prof. Me. Alexandre Henrick

Análise e Desenvolvimento de Sistemas - 6º período Sistemas de Informação

Revisão Python

Sintaxe

Comentários:

- Comentários de linha única começam com #.
- Comentários de várias linhas podem ser feitos usando três aspas simples (''') ou três aspas duplas (""").

```
# Este é um comentário de linha única

'''
Este é um comentário
de várias linhas.
'''
```

Indentação:

- Python utiliza a indentação para delimitar blocos de código. Não há uso de chaves ou palavras-chave como begin e end.
- Geralmente, é usado um espaço em branco de quatro espaços para cada nível de indentação.

```
if True:
    print("Indentação correta")
```

Variáveis:

- As variáveis não precisam ser explicitamente declaradas. Elas são definidas quando atribuídas.
- A convenção para nomear variáveis é usar letras minúsculas e underscores para separar palavras (snake_case).

```
nome = "Alice"
idade = 30
saldo_conta = 1000.50
```

Tipos de Dados:

- Python é dinamicamente tipado, ou seja, o tipo da variável é inferido automaticamente.
- Tipos de dados incluem inteiros (int), números de ponto flutuante (float), strings (str), booleanos (bool), listas (list), tuplas (tuple), dicionários (dict), conjuntos (set) etc.

```
inteiro = 42
decimal = 3.14
texto = "Olá, mundo!"
verdadeiro = True
lista = [1, 2, 3]
tupla = (4, 5, 6)
dicionario = {'chave': 'valor'}
conjunto = {7, 8, 9}
```

Operadores Aritméticos:

- +, -, *, /, // (divisão inteira), % (resto da divisão), ** (potenciação).

```
soma = 5 + 3
subtracao = 10 - 2
multiplicacao = 4 * 2
divisao = 10 / 2
divisao_inteira = 9 // 2
resto_divisao = 9 % 2
potenciacao = 2 ** 3
```

Estruturas de Controle:

- if, elif, else para condicionais.
- for para loops com iteração.
- while para loops com condição.
- break para sair de um loop.
- continue para continuar para a próxima iteração de um loop.

```
idade = 18
if idade >= 18:
    print("Maior de idade")
else:
    print("Menor de idade")

for numero in range(5):
    print(numero)

contador = 0
```

```
while contador < 5:
    print(contador)
    contador += 1
```

Funções:

- Definidas usando a palavra-chave **def**.
- Podem ter argumentos e valores de retorno.
- **return** para retornar valores.

```
def saudacao(nome):
    return f"Olá, {nome}!"

mensagem = saudacao("Alice")
print(mensagem)
```

Listas e Dicionários:

- Listas são coleções ordenadas mutáveis.
- Dicionários são coleções de pares chave-valor.
- Indexação começa em 0.

```
numeros = [1, 2, 3, 4, 5] # Lista
pessoa = {'nome': 'Alice', 'idade': 30} # Dicionário

# ====== OPERAÇÕES COM LISTAS ======

print(numeros[2])  # Acessando elementos em lista. Saída: 3

# Método append adiciona um elemento a lista. No caso abaixo, adiciona ao
final da lista
numeros.append(6)

# Método remove exclui um elemento da lista. No exemplor abaixo, exclui o
elemento 3
numeros.remove(3)

# Podemos usar o "in" para verificar se um elemento existe na lista. A
operação com "in" nos retorna um valor booleano (True ou False)
existe_na_lista = 4 in numeros

# Método sort() ordena a lista em ordem crescente
numeros.sort()

# O método reverse() inverte a ordem de uma lista
numeros.reverse()
```

```
# Podemos "fatiar" uma lista da maneira abaixo. Aqui estamos dizendo que queremos retornar os elementos do índice 1 ao 3
nova_lista = numeros[1:4]

## Feature mais interessante do python: List Comprehension
quadrados = [X ** 2 for x in range(10)]
print(quadrados)

# ====== OPERAÇÕES COM DICIONÁRIOS ======

print(pessoa['nome'])  # Acessando elementos em dicionários Saída: Alice

## Adicionando mais uma chave e valor
dicionario['email'] = 'exemplo@gmail.com'

## Usamos o operador "del" para um par de chave-valor
del dicionario['email']

## Assim como nas listas, também podemos usar o "in" para verificar se uma chave existe no dicionário
existe_chave = 'nome' in dicionario

## Os métodos keys() e values() retornam as chaves e valores de um dicionário
dicionario.keys()
dicionario.values()

## Os exemplos acima não retornam em um formato de lista. Para isso, podemos usar o método list() como no exemplo abaixo
chaves = list(dicionario.keys())
valores = list(dicionario.values())

## Podemos iterar sobre um dicionário usando o método items()
for chave, valor in dicionario.items():
    print(f'{chave}: {valor}')
```

Strings:

- Delimitadas por aspas simples (') ou duplas (").
- Podem ser concatenadas usando o operador +.
- Fstrings

```
frase = "Python é incrível!"
print(frase)

concatenacao = "Olá, " + "mundo!"
print(concatenacao)
```

```
nome = "Alexandre"
print(f'O nome do usuário é {nome}')
```

Módulos e Bibliotecas:

- Módulos são arquivos Python contendo funções e variáveis.
- Bibliotecas são conjuntos de módulos que oferecem funcionalidades específicas.
- import para usar módulos e bibliotecas externas.

```
import math

raiz_quadrada = math.sqrt(25)
print(raiz_quadrada)
```

Tratamento de Exceções:

- try, except para capturar exceções.
- finally para código que deve ser executado independentemente da exceção.

```
try:
    resultado = 10 / 0
except ZeroDivisionError:
    print("Erro: divisão por zero")
finally:
    print("Finalizado")
```

Classes e Orientação a Objetos:

- Definidas usando a palavra-chave class.
- Possuem métodos (funções) e atributos (variáveis).
- Herança, encapsulamento e polimorfismo são suportados.
- **init** é como chamamos os métodos construtores em python
- O 'self' usamos para referir ao próprio objeto de uma classe dentro dos métodos dessa classe.

```
class Animal:
    def __init__(self, nome): # Método construtor de 'Animal' sempre vai
receber como parâmetro 'nome', que será um atributo da classe
        self.nome = nome

    def falar(self):
        pass

class Gato(Animal):
    def falar(self):
```

```
        return "Long Johnson!"

cat = Gato("Whiskas")
print(cat.falar())
```

Usando classes e métodos abstratos:

```
'''
Abstract Base Class (ABC). Módulo do python que nos ajuda a contruir
classes
e métodos abstratos.
'''
from abc import ABC, abstractmethod

class Animal(ABC):
    '''
    As classes que implementarem "Animal" precisam
    implementar o método "alimentar()"
    '''
    @abstractmethod # Usamos esse decorador para definir um método abstrato
    def alimentar(self):
        pass
```

Arquivos:

- open() para abrir arquivos.
- read(), write(), close() para operações em arquivos.
- O gerenciamento de contexto com with é recomendado.

```
with open("arquivo.txt", "w") as arquivo:
    arquivo.write("Olá, mundo!")

with open("arquivo.txt", "r") as arquivo:
    conteudo = arquivo.read()
    print(conteudo)
```