Relatório 2 - Prática: Linguagem de Programação Python

Guilherme Loan Schneider

Descrição da atividade

Foi possível compreender, por meio dos vídeos disponibilizados, várias noções básicas de Python que serão utilizadas mais à frente. A principal noção é que o Python possui tipagem dinâmica, ou seja, o tipo de variável é definido no momento da atribuição, permitindo uma mesma variável assumir vários tipos de dados diferentes durante a execução do código. Além disso, como outras linguagens, Pyhton é Case-sensitive, ou seja, deve-se levar em conta a utilização de caracteres maiúsculo ou minúsculo ("Variavel1" é diferente de "variavel1").

É importante salientar também que a indentação do código é extremamente crucial nessa linguagem, visto que, por exemplo, caso uma linha esteja mal indentada dentro de um laço "For", o seu funcionamento é interrompido.

Os conteúdos abordados nas aulas foram os seguintes:

- A aula começa elucidando como é feito os "imports" de arquivos diferentes, utilizando, por exemplo, "import operadores.logicos" ou também "from operadores import logicos".
- Utilização de F-strings, que permitem a interpolação de valores dentro de strings, por meio de chaves ({'…'}).
- Os tipos de variáveis presentes na linguagem, sendo eles:
    - Tipo Básico, como valores inteiros, strings, float, boolean, double, entre outros. Para ser possível essa visualização, utilizou-se o print(type(var));
    - As Tuplas, que são similares a vetores, no entanto, são imutáveis;
    - O tipo Lista, que armazena dados em sequência, similar a outras linguagens, permite alteração e manipulação de valores;
    - Tipo Dicionário, que se assemelha as estruturas de dados em C. Possui também manipulação e alteração de valores;
    - Tipo Conjunto, é similar a um vetor também, no entanto não aceita conjuntos repetidos e não é possível acessar os valores de forma independente (Exemplo: conj[1]);
- Os operadores da linguagem, sendo eles:
    - Operadores de atribuição, "+=", "*=", "-=", "/=".
    - Operadores lógicos, como "and", "or" e "not". Vale ressaltar que estes devem ser escritos e não por meio de símbolos como "&" ou "|".
    - Operadores ternários, que são basicamente um if simplificado, podendo ser definido em apenas uma linha, como por exemplo "status = 'Dirigindo' if b1 and combustivel >= 10 else 'A pé'".
- As funções de controle, sendo elas:
    - A função For e If, que funcionam exatamente como em outras linguagens. Vale ressaltar que o Pyhton permite mais manipulações de forma fácil, como a utilização do "end" no print, que adiciona o que o usuário deseja no final de cada string.

- A função While, que também funciona exatamente como em outras linguagens, no entanto, aqui não se tem uma representação exata de como seria o "Do-While".

- As funções definidas pelo usuário, sendo elas:
  - Funções básicas, que nortearam a estrutura de uma função, com e sem return, em Python. É importante salientar que, diferentemente de Java, a linguagem não faz sobrecarga de métodos, ou seja, não podemos ter dois métodos com o mesmo nome.
  - Funções Lambda, que são funções anônimas e normalmente servem de argumento para outras funções. São normalmente escritas em uma única linha, como por exemplo "obter_nomes = lambda aluno: aluno['nome']", que para cada aluno, o conteúdo da variável "nome" é resgatado pela função lambda e atribuído para a variável "obter_nomes".
  - Função Filter, que como o próprio nome sugere, ele retorna elementos de uma lista que satisfazem uma condição, como por exemplo:

```
verifica_aluno_aprovado = lambda aluno: aluno['nota'] >= 7
aprovados = list(filter(verifica_aluno_aprovado, alunos))
```

  - Função Map, que percorre um vetor, de forma a efetuar alguma manipulação com os valores desse vetor, sem alterá-los. Consiste em dois argumentos, o primeiro é a função a ser aplicada nos valores, e o segundo o vetor a ser analisado. Segue um exemplo abaixo:

```
notas = [6.4, 7.2, 5.8, 8.4, 6.5, 7.0, 5.5]

def somar_ponto (nota):

return nota + 1.5

notas_finais = map(somar_nota, notas)
```

  - A função Reduce, que permite reduzir um conjunto de elementos a um único valor, ou seja, ele funcionará como um acumulador. A função reduce recebe dois argumentos, o primeiro é a função que será aplicada (normalmente consiste em uma variavel que acumula o valor e outra que acessa os valores na lista) e o segundo é a lista que será percorrida. Segue um exemplo abaixo:

```
print(reduce(total_nota, notas, 0)) # 0 é o valor inicial do acumulador
```

  - Função Funcional, que permite que uma função seja atribuída a uma variável, ou uma função que retorna uma outra função, ou também uma função que recebe uma outra função como parâmetro.

Exemplo de função atribuída para uma variável:

```
def soma(a, b):
    return a + b
somar = soma
print(somar(2,3))
```

Exemplo de uma função que retorna outra função:

```
def soma_parcial(a):
    def concluir_soma(b):
        return a + b
    return concluir_soma
```

Exemplo de uma função que recebe outra função como parâmetro:

```
def operacao_aritmetica(fn, op1, op2):
    return fn(op1, op2)
```

- Função List Comprehension, é uma construção sintática que permite criar uma lista a partir de outra lista ou sequência iterável. Como por exemplo:

```
alunos = [ {'nome': 'Benio', 'nota': 8.5}, {'nome': 'João', 'nota': 7.2}, {'nome': 'Maria', 'nota': 6.5}, {'nome': 'José', 'nota': 9.0}, {'nome': 'Ana', 'nota': 5.5}, {'nome': 'Paulo', 'nota': 4.7}, {'nome': 'Pedro', 'nota': 6.3}, {'nome': 'Carla', 'nota': 8.1}, {'nome': 'Marta', 'nota': 7.7}, {'nome': 'Luiz', 'nota': 6.9} ]
aprovados = [aluno for aluno in alunos if aluno['nota'] >= 7]
```

Aqui, o primeiro “aluno” indica o que será retornado da lista, nesse caso, são todas as informações de aluno. Em seguida, há um laço “For" que fará a iteração entre cada elemento da lista, seguido de uma condição, que retornará apenas os aprovados de determinada matéria.

- Funções que utilizam *args e **kwargs, onde ambos consistem em argumentos que podem ser passados para uma função sem a necessidade de definir quantos são. O “*” é um operador de desempacotamento, que permite que uma função receba um número variável de argumentos. Já o seu uso duplo (**), permite que uma função receba um número variável de argumentos nomeados.

  Exemplo de uso do *args:

```
def soma(*args):
    total = 0
    for i in args:
        total += i
    return total
```

Exemplo de uso do **kwargs:

```
def resultado_final(**kwargs):
    status = 'aprovado' if kwargs['nota'] >= 6 else 'reprovado'
    return f'{kwargs["nome"]} está {status}'
```

- Orientação a Objetos;
    - Herança – Assim como em Java, consiste em herdar atributos e métodos de outra classe, reutilizando e especializando o comportamento da classe base, ou seja, seria criar uma classe genérica "Carro" e ao especificar o modelo de carro, como "Fusion" ou "Ferrari", tem-se a opção de criar atributos especiais para essa classe. Abaixo segue o exemplo citado anteriormente:

```
class Carro:
    def __init__(self):
        self.__velocidade = 0

    @property
    def get_velocidade(self):
        return self.__velocidade

    def acelerar(self):
        self.__velocidade += 5
        return self.__velocidade

    def frear(self):
        self.__velocidade -= 5
        return self.__velocidade

class Fusion(Carro):
    pass # Indica que essa classe não possui atributos específicos

class Ferrari(Carro):
    def acelerar(self): #Aqui é alterado a função acelerar para a
classe específica Ferrari
        super().acelerar()
        return super().acelerar()
```

    - Vale ressaltar a utilização da função @property, que é uma forma de definir métodos em uma classe que podem ser acessados como atributos (sem a utilização dos parênteses).
    - Membros de uma classe, que significam os elementos básicos de uma classe, que podem ser divididos em duas categorias: as variáveis e os métodos. Abaixo segue o exemplo utilizado em aula:

```
class Contador:
    contador = 0 #Atributo de classe

    # Esse método não pode ser acessado sem criar uma instância da
classe
    def inst(self):
        return 'Objeto criado'
```

```
    @classmethod
    def incrementar(self):
        self.contador += 1
        return self.contador

    @classmethod
    def decrementar(self):
        self.contador -= 1
        return self.contador

    # É utilizado para criar métodos que não acessam os atributos
da classe
    @staticmethod
    def dividir_por_2(valor):
        return valor / 2
```

- É possível perceber a utilização de diferentes nomenclaturas, como o @classmethod e o @staticmethod, onde o primeiro é utilizado para fazer com que métodos sejam utilizados sem criar uma instância da classe, e o outro é utilizado para criar métodos que não possuem relação com os atributos da classe.
- Por fim, um exemplo de classe "Produto", que possui uma especificidade, o preço dos produtos é um atributo privado, ou seja, ele não pode ser acessado diretamente, é necessário a utilização de métodos para acessá-lo. O código acaba sendo mais um exemplo da utilização dos conhecimentos supracitados, no entanto, é importante salientar a utilização da nomenclatura @get_preco.setter, que transforma o método "set_preco" em um setter para o atributo privado "__preco". Abaixo segue o caso anteriormente citado:

```
    @property
    def get_preco(self):
        return self.__preco

    @get_preco.setter
    def set_preco(self, valor):
        if valor > 0:
            self.__preco = valor
```

Aqui, chega-se ao final dos conteúdos passados na aula.

Conclusões

A partir das aulas, foi possível aprender e compreender os conceitos fundamentais de Python, abordando desde noções básicas, como tipagem dinâmica e case sensitivity, até tópicos mais avançados, como orientação a objetos e manipulação de funções. Destaca-se o uso de

propriedades, funções lambda, filtros, e comprehensions, além da importância da indentação e da encapsulação de dados. Os exemplos práticos, explicações detalhadas e comentários facilitam a compreensão das principais características da linguagem, proporcionando uma base sólida para o uso da linguagem Python.

Referencias

PYTHON 3 Curso Rápido Parte #1 2020 - 100% Prático!

PYTHON 3 Curso Rápido Parte #2 2020 - 100% Prático!