

Relatório sobre a atividade “2 - Prática: Linguagem de Programação Python (I)”

Lucas Gabriel Arenhardt

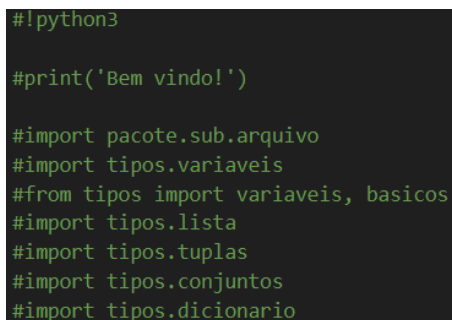
1. Introdução

Nesta atividade, foram apresentados dois vídeos sobre Python 3. O conteúdo abordado em ambos foi um geral acerca da linguagem, contendo desde a criação de variáveis, até criação de funções e coisas um pouco mais complexas.

2. Conteúdo

2.1. Funcionalidades básicas

Na primeira parte da aula, foi demonstrada a criação de pastas e arquivos, importação de pacotes, a criação de comentários e a função print().



```
#!python3

#print('Bem vindo!')

#import pacote.sub.arquivo
#import tipos.variaveis
#from tipos import variaveis, basicos
#import tipos.lista
#import tipos.tuplas
#import tipos.conjuntos
#import tipos.dicionario
```

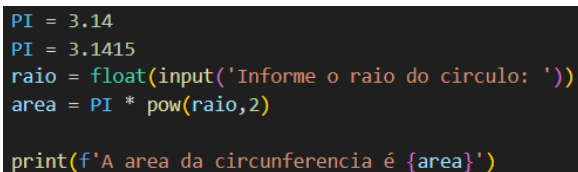
Imagem 1 - Exemplo de importações, comentários e funções print().

2.2. Tipos

Neste módulo, foram apresentados diferentes tipos de dados em python, sendo eles: int, float, string, bool, listas, tuplas, conjuntos e dicionários.

Int e float são tipos numéricos, que possuem valores, amplamente utilizados para fazer operações aritméticas.

O tipo string é relacionado a caracteres. Mesmo que receba um número como entrada, ele não interpreta o valor numérico, apenas o caractere. Então, caso tente somar, por exemplo, 2 e 5, sendo ambos do tipo string, a saída será 25 e não 7.



```
PI = 3.14
PI = 3.1415
raio = float(input('Informe o raio do circulo: '))
area = PI * pow(raio,2)

print(f'A area da circunferencia é {area}')
```

Imagem 2 - Exemplo de operações com int, float e string.

O valor booleano nada mais é do que True e False. Bastante utilizados para implementar problemas de lógica, que usam tabelas-verdade.

A lista permite armazenar uma coleção de itens, que podem ser repetidos, em uma variável. Tais itens são indexados e é possível acessá-los individualmente através do seu respectivo índice. Permite modificações. A única diferença entre a lista e a tupla, é que a tupla é imutável.

Os conjuntos são semelhantes aos conjuntos da matemática. Também armazenam uma coleção de itens, porém tais itens não são repetidos. Mesmo que tente incluir algum item mais de uma vez, o conjunto simplesmente o ignora.

Dicionários são uma coleção com elementos chave-valor. É realmente como um dicionário, onde cada termo possui sua tradução. Assim, você encontra um elemento pela sua chave correspondente, diferentemente das listas e tuplas onde o identificador é apenas o índice.

```
carro = {  
    'marca': 'Honda',  
    'ano': 2004,  
    'modelo': 'Civic',  
    'revisado': True  
}
```

Imagem 3 - Exemplo de um dicionário.

2.3. Operadores

Os operadores são símbolos ou palavras reservadas que permitem realizar operações, algo essencial na programação. Foram apresentados operadores unários, ternários, aritméticos, lógicos, relacionais, e de atribuição.

Operadores unários desempenham um papel sobre apenas um operando. Como exemplo temos: +, -, not.

Operadores ternários são um recurso de tomada de decisão. Como exemplo, temos “if” e “else”.

Operadores aritméticos são os responsáveis pelos cálculos matemáticos. ‘+’, ‘-’, ‘/’, ‘*’ e ‘%’ são alguns deles.

Operadores lógicos estão relacionados aos valores booleanos. Bastante estudados em lógica de programação. Como exemplo: “and”, “or”.

Operadores relacionais exprimem uma relação entre dois valores, uma comparação. Podem ser representados por: ‘>’, ‘<=’, ‘==’, ‘!=’.

Por fim, os operadores de atribuição tem como função atribuir um valor a alguma variável. Não se limita apenas a '=', pois também estão inclusos como operadores de atribuição os seguinte elementos: '+=', '-=', '/=', '*=', entre outros.

2.4. Estruturas de controle

As estruturas de controle permitem guiar o fluxo de execução do programa, abrindo possibilidades de escolha.

A primeira apresentada na aula foi a estrutura condicional (if-elif-else). Ela permite executar diferentes blocos de comando com base nas condições impostas.

```
ano = int(input('Informe o ano do carro: '))

if(ano <= 2004):
    print('Seu carro ja possui 20 anos ou mais')
elif(2004 < ano < 2014):
    print('Seu carro possui entre 10 a 20 anos')
else:
    print('Seu carro possui 10 anos ou menos')
```

Imagem 4 - Exemplo de estrutura condicional.

Em seguida, foi apresentado o laço de repetição (for). Esse é utilizado para iterar sobre uma sequência ou para repetir um bloco de código um número específico de vezes.

```
texto = 'Civic é muito é massa'

for letra in texto:
    print(letra, end = ' ')

print('')
```

Imagem 5 - Exemplo de laço de repetição (for).

Por fim, foi mostrado o laço de repetição (while). Ele executa um bloco de código enquanto uma condição especificada for verdadeira.

```
total = 0
nota = 0
qtde = 0

while nota != -1:
    nota = float(input('Informe a nota ou -1 pra sair: '))
    if nota != -1:
        qtde += 1
        total += nota

print(f'A media da turma é {total/qtde}')
```

Imagem 6 - Exemplo de laço de repetição (while).

2.5. Funções

As funções são blocos de código reutilizáveis que realizam uma tarefa específica. São definidas usando a palavra-chave 'def' e podem ou não retornar um valor através da palavra-chave 'return'. São chamadas usando o nome da função seguido de parênteses contendo, se necessário, os argumentos.

Os argumentos são valores de entrada na função. Eles permitem que informações específicas sejam fornecidas à função para que ela realize as operações com tais dados.

```
def soma(*nums):  
    total = 0  
    for n in nums:  
        total += n  
    return total
```

Imagem 7 - Função definida com 'def'

Neste módulo de aula, também foram apresentadas as funções 'lambda', 'map' e 'reduce'. As funções 'lambda' são anônimas e úteis para criar funções simples e pequenas sem a necessidade de defini-las formalmente com 'def'. Com 'map' é possível aplicar uma função a cada item de um conjunto de dados iterável, a fim de realizar uma transformação em todos os itens do conjunto. Por fim, 'reduce' aplica uma função de dois argumentos cumulativamente a todos os itens de um iterável. É útil para realizar somas ou outras operações cumulativas.

```
from functools import reduce  
  
carros = [  
    {'nome': 'Civic', 'ano': 2004},  
    {'nome': 'Cruze', 'ano': 2020},  
    {'nome': 'Ranger', 'ano': 2018},  
    {'nome': 'Uno', 'ano': 2000},  
    {'nome': 'Sander', 'ano': 2016},  
]  
  
carro_atual = lambda carro: carro['ano'] > 2015  
obter_ano = lambda carro: carro['ano']  
somar = lambda a, b: a+b  
  
carros_atuais = [carro for carro in carros if carro['ano'] > 2015]  
ano_carros_atuais = [carro['ano'] for carro in carros_atuais]  
  
print(ano_carros_atuais )
```

Imagem 8 - Função 'lambda'.

```

from functools import reduce

def somar_nota(delta):
    def calc(nota):
        return nota+delta
    return calc

notas = [5.5, 7.9, 8.2, 3.1]
notas_finais_1 = list(map(somar_nota(1), notas))
notas_finais_2 = list(map(somar_nota(1.5), notas))

print(notas_finais_1)
print(notas_finais_2)

def somar(a, b):
    return a+b

total = reduce(somar, notas, 0)
print(total)

```

Imagem 9 - Funções 'map' e 'reduce'.

2.6. Classes

Uma classe é uma estrutura que contém atributos e métodos compartilhados por objetos que são instância dessa classe. É definida utilizando a palavra-chave 'class' seguida pelo nome desejado.

```

class Carro:
    def __init__(self) -> None:
        self.__velocidade = 0

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

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

```

Imagem 10 - Exemplo de classe.

Uma característica das classes é a herança. Ela permite que uma classe filha herde os atributos e métodos de uma classe pai. A classe filha pode adicionar novos atributos e métodos ou sobrescrever os existentes.

```

class Ferrari(Carro):
    def acelerar(self):
        super().acelerar()
        return super().acelerar()

```

Imagem 11 - Classe Ferrari que tem Carro como classe Pai.

Por fim, temos os decoradores. Eles permitem adicionar funcionalidades extras a uma função existente sem modificar seu código-fonte. Como exemplo, temos: '@property', '@classmethod', '@staticmethod' e '@.setter'.

```
class Produto:
    def __init__(self, nome, preco, desc=0):
        self.nome = nome
        self.__preco = preco
        self.desc = desc

    @property
    def preco(self):
        return self.__preco

    @preco.setter
    def preco(self, novo_preco):
        if novo_preco > 0:
            self.__preco = novo_preco

    @property
    def preco_final(self):
        return (1- self.desc) * self.__preco
```

Imagem 12 - Classe com decoradores '@property' e '@.setter@'.

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

    def inst(self):
        return "Teste"

    @classmethod
    def inc(cls):
        cls.contador += 1
        return cls.contador

    @classmethod
    def dec(cls):
        cls.contador -= 1
        return cls.contador

    @staticmethod
    def mais_um(n):
        return n+1
```

Imagem 13 - Classe com decoradores '@classmethod' e '@staticmethod'.

3. Conclusão

Após assistir às aulas e realizar as práticas, tive um maior conhecimento acerca do python. Eu já utilizava o python antes, mas aprendi algumas coisas novas ao assistir aos vídeos.