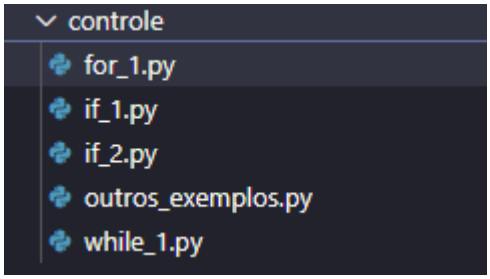
Relatório 2 - Prática: Linguagem de Programação Python (I)

Igor Carvalho Marchi

Descrição da atividade

Pacotes e módulos



O pacote é uma forma de agrupar os módulos relacionados em uma única estrutura de diretórios, de modo que possam ser facilmente gerenciados e reutilizados, já os módulos são arquivos em Python que podem conter definições e implementações de funções, classes e variáveis, além de código executável.

Import

É utilizado para importar módulos de outros pacotes, conseguindo trazer o código feito dentro de outro módulo para seu código principal, podendo ser de duas formas:

```
import controle.if_1
```
OU
```
from controle import if_1
```

Tipos básicos de variáveis

- **string**: Cadeias de texto.
- **integer**: Números inteiros.
- **float**: Números decimais.
- **booleana:** Valores booleanos (True ou False).

Conjuntos

```
conj = ({1, 2, 3, 3, 3, 3})
```

Os conjuntos são um tipo de estrutura que não permite valores duplicados, isto é mesmo que tenha 4 valores igual a 3, no painel será mostrado apenas {1, 2, 3}, ou seja, são utilizados para armazenar elementos únicos.

Lista

```
nums = [1, 2, 3]
```

Utilizam colchetes [], suportam valores duplicados e podem ser modificadas.

```
nums.append(3)  # adicione elementos na lista
```

append() serve para adicionar elementos na lista.

```
len(nums)
```

len() Serve para ler o tamanho da lista.

```
nums.insert(0, -200)
```

Insert() Serve para inserir um numero em uma posição da lista.

Tuplas

```
nomes = ('Ana', 'Bia', 'Gui', 'leonardo', 'Ana')
```

Não é possível mudar as tuplas, elas devem ser preenchidas dentro de parênteses (), além disso diferente dos conjuntos elas aceitam valores repetidos. Após as aspas ‘ ’ é necessário a presença de uma vírgula.

Dicionários

```
aluno = {
    'nome': 'Pedro Henrique',
    'nota': 9.2,
    'ativo': True
}
```

São estruturas de mapeamento com pares chave {}, com valores, acessados por meio de suas chaves, onde são declarados ex:

‘nome’ : ‘Pedro Henrique’

Para buscar o nome Pedro Henrique basta chamar por ‘nome’

```
print(aluno['nome'])  # Mostra valor que esta dentro da chave
```

Operadores

Unários: são operadores que aceitam apenas um operando, primeiro operador unário é o not ou acrescentar um sinal de menos fazendo mudar o sinal de operação, se já for menos virá mais, ser for um valor positivo virará negativo

```
print(not False)  # Inverte o valor, se for False vira True e vice-versa
print(not True)
print(-3)  # Acrescenta um sinal de menos
```

Ternário: O operador ternário é expressado através de um if e else em uma única linha, Ex:

```
# se cumprir todas primeira condições saida "Em casa", se não "Uhuuu"
status = 'Em casa' if lockdow or grana <= 100 else 'Uhuuu'

print(f'O status e: {status}')
```

Aritméticos: são operadores binários que utilizam de dois operandos:

```
print(x + y) # 10+5 retorna 15
print(x - y) # 10-5 retorna 5
print(x * y) # 10*5 retorna 50
print(x / y) # 10/5 retorna 2
print(x % y) # 10%5 retorna 0
print(x**y) # 10**5 retorrna 100000
```

Operadores Relacionais

Fala se a condição é verdadeira ou falsa, Ex:

```
# Fala se a condição é True ou false
print(x > y)  # Se x é maior que y
print(x >= y) # se x é maior ou igual a y
print(x < y) # Se x é menor que y
print(x <= y) # Se x é mneor ou igual a y
print(x == y) # Se o valor de x é igual ao do y
print(x != y) # Se o valor de x é diferente do de y
```

Operadores Atributivos

Atribui no resultado seja (atribuir valor, somar, subtrair, multiplicar, dividir, resto)

```
resultado += resultado # resultado = resultado + resultado
resultado += 3  # resultado = resultado + 3
resultado -= 1  # resultado = resultado - 1
resultado *= 4  # resultado = resultado * 4
resultado /= 2  # resultado = resultado / 2
resultado %= 6  # resultado = resultado % 6
print(resultado)
```

Operadores Lógicos

Quando se deseja utilizar expressões onde ocorre situações de “e”, “ou”, “Diferente”, “Negar” são utilizados as expressões and, or, xor, e not.

```
print(b1 and b2 and b3)  # Para ser True todos tem que ser True
print(b1 or b2 or b3)  # Precisa que apenas um seja True para ser True
print(b1 != b2)  # xor ^ para ser True os dois tem que ser diferente
print(not b1)  # inverte o valor
```

Estrutura de controles

If/elif/else: São estruturas que criam condições a serem determinadas.

```
if nota >= 9 and comportado:  # condição de "Se"
    print('Duas palavras: para bens! :P')
    print('Quadro de Honra')
elif nota >= 7:  # condição de "senão se"
    print("Aprovado")
elif nota >= 5.5:
    print('Recuperação')
elif nota >= 3.5:
    print('Recuperação + trabalho')
else:  # condção de "senão"
    print('Reprovado')
print(nota)
```

While: É uma estrutura que repetirá até sair da condição proposta.

```
while nota != -1:  # Só sai quando nota for igual a -1
    nota = float(input('Informe o numero ou -1 para sair: '))
    if nota != -1:  # Só soma se receber um numero diferente de -1
        total += nota
        qtd += 1
print(f'A media da turma é {total / qtd}')
```

For: É quase igual ao while, porém nele você pode determinar a quantidade de repetições de forma precisa.

```python
for i in range(10):  # de 0 a 9
    print(i)

for i in range(1, 11):  # de 1 a 10
    print(i)

for i in range(1, 100, 7):  # de 1 a 99 aumentando em 7 em 7
    print(i)
```

Funções

As funções são formas de realizar tarefas especificas em Python, de forma que possam ser reutilizadas em diversas partes do código, elas servem para organizar o código de forma que não precise fazer o processo diversas vezes, contornando isso através de uma linha com o nome da função, deixando mais eficiente.

```python
def soma(a, b):
    return a + b

print(soma(3, 4))
```

Pode se usar apenas chamando a função: ex soma (), atribuindo os valores dentro dela como a e b, soma(3,4) onde será executado o comando a + b resultando no valor 7.

Além disso, você pode guardar essa função dentro de uma variável:

```python
def sub(a, b):
    return a - b

subr = sub
print(subr(4,3))
```

Também é possível colocar uma função dentro de outra função, passando assim dois parâmetros diferentes, assim melhorando a velocidade do processamento:

```python
def soma_parcial(a):

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

resultado_final = soma_parcial(10)(12)
```

***args:** Utilizado quando não se tem a quantidade especifica de valores que serão passados, EX:

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

****kwargs:** permite passar variáveis com chave-valor, EX:

```python
def resultado_final(**kwargs):
    status = 'aprovado(a)' if (kwargs['nota'] >= 7) else 'reprovado(a)'
    return f'{kwargs["nome"]} foi {status}'
```

map() aplica uma função a cada elemento de uma lista, criando uma nova lista com os resultados. Ex:

```python
def somar_nota(delta):
    def somar(nota):
        return nota + delta
    return somar


notas = [6.4, 7.2, 5.4, 8.4]
notas_finais_1 = list(map(somar_nota(1.5), notas))
```

No código acima o map() está acrescentando a cada valor da lista + 1,5.

reduce() reduz uma lista a um único valor ao aplicar uma função cumulativa aos elementos. Ex:

```python
notas = [6.4, 7.2, 5.4, 8.4]

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

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

No código acima ele está somando todos os valores da lista e transformando em um só.

filter() filtra elementos de uma lista com base em uma condição booleana, retornando apenas os que atendem ao critério especificado.

```python
alunos = [
    {'nome': 'Ana', 'nota': '7.2'},
    {'nome': 'Breno', 'nota': '8.1'},
    {'nome': 'Claudia', 'nota': '8.7'},
    {'nome': 'Pedro', 'nota': '6.4'},
    {'nome': 'Rafael', 'nota': '6.7'},
]

def aluno_aprovado(aluno): lambda aluno : float(aluno['nota']) >= 7
alunos_aprovados = list(filter(aluno_aprovado, alunos))
```

No código acima ele está filtrando de acordo com os alunos que estão com notas igual a 7 ou maiores seguindo a função que está usando o lambda que é o responsável para dar a condição para filtração.

Classes

As classes definem atributos e métodos que determinam o comportamento e as características dos objetos criados a partir dela.

```python
class Produto:
    def __init__(self, nome, preco=1.99, desc=0):
        self.nome = nome
        self.__preco = preco
        self.desc = desc

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

__init__: Método especial usado para inicializar os atributos do objeto.

self: Representa a instância atual da classe e permite acessar atributos e métodos.

Getter e Setter

São métodos usados para acessar e modificar os atributos de uma classe. Ex:

```python
@preco.getter
def get_preco(self):
    return self.__preco

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

No getter o código está acessando o valor do preço, já no setter está modificando para um novo valor caso seja diferente do valor antigo.

Property: Permite que se defina métodos especiais para obter, definir ou deletar um atributo. Ex:

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

Classmethod: Podem ser chamados na classe em si e são frequentemente usados para métodos de fábrica ou operações que envolvem a classe como um todo. Ex:

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

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

Staticmethod: O método estático não recebe um primeiro argumento implícito e se comporta como uma função regular. Ex:

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

print(Contador.mais_um(99))
```

Herança: Herança é um dos pilares de POO que funciona da seguinte forma, uma classe herda os métodos e atributos de uma outra classe. Ex:

```python
class Carro:
    def __init__(self):
        self.__velocidade = 0
    @property
    def Velocidade(self):
        return self.__velocidade
    def acelerar(self):
        self.__velocidade += 5
        return self.__velocidade
    def frear(self):
        self.__velocidade -= 5
        return self.__velocidade

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

c1 = Ferrari()
print(c1.acelerar())
print(c1.acelerar())
print(c1.acelerar())
print(c1.frear())
```

Nesse código é possível ver que a Ferrari herda as funções de Carro, podendo assim utilizar as funções de acelerar e frear no código

Conclusões

Python é uma linguagem versátil com suporte a POO e ferramentas para organizar e reutilizar código, como pacotes, módulos e funções. Possui estruturas de dados básicas e operadores para controle lógico e aritmético.

Referencias

https://www.youtube.com/watch?v=oUrBHiT-lzo

https://www.youtube.com/watch?v=iq7JLIH-sV0

https://didatica.tech/o-que-sao-modulos-e-pacotes-em-python-e-como-usar/#:~:text=Um%20m%C3%B3dulo%20%C3%A9%20basicamente%20um,ser%20facilmente%20gerenciados%20e%20reutilizados.

https://www.hashtagtreinamentos.com/estruturas-de-dados-em-python?