Relatório 2 - Linguagem de Programação Python (I)

Lucas Scheffer Hundsdorfer

Descrição da atividade

São dois vídeos focados para aprendermos o básico de Python:

Import: Serve 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 pacote.modulo.teste
```

```
from pacote.modulo import teste
```

Comentário: Comentário de múltiplas linhas com “”” “”” também pode ser uma string

```
x = """print("Hello World")"""

print(x)
```

Variáveis: Para declaração de variáveis em Python, não é necessário declarar o seu tipo, apenas o nome e o seu valor, como no exemplo abaixo:

```
x = 24
nome = 'Lucas'
y = 43.29
```

Print: Para printar as variáveis não é possível concatenar com tipos diferentes, precisamos utilizar ou o cast, para trocar o tipo da variável ou usar o print format dessa forma:

```
print(f'{nome} tem {x} anos!')
```

```
print(nome + " tem " + str(x)  + " anos!")
```

Lista: Um tipo de variável que permite armazenar uma sequência de valores em forma organizada, conseguindo também adicionar, remover, ver a quantidade de elementos da lista, dentro da lista temos vários comandos:

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

nums.append(4)
```

utilizado para adicionar valores ao final da lista.

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

print(nums.lens())
```

utilizado para mostrar quantos elementos existem dentro da lista.

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

nums[2] = 40
```

utilizado para substituir o valor no índice específico, lembrando que o índice de uma lista começa em 0.

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

nums.insert(0, 50)
```

utilizado para adicionar um elemento na lista, porém realocando os que já estavam.

Tuplas: As tuplas diferentemente das listas não aceitam modificação, alguns comandos para listas também serve para as tuplas, é necessário em sua declaração ter pelo menos uma vírgula após as aspas, pois se não se obtém uma string se houver apenas um elemento.

Consegue se obter um valor booleano afirmando se existe certo elemento dentro da tupla assim:

```
nomes = ('Lucas', 'Gustavo', 'Yuri', 'Gabriel')
print('Lucas' in nomes)
```

Consegue imprimir os elementos de um índice especificado até outro índice:

```
nomes = ('Lucas', 'Gustavo', 'Yuri', 'Gabriel')
print(nomes[0:3])
```

Conjuntos/Set: Um tipo de estrutura que não permite valores duplicados e não possui uma ordem definida, não apresentando índice.

Dicionários: Um tipo de estrutura usada para armazenar pares de chave e valor, ao contrário da lista onde o valor é acessado pelo índice no dicionário ele é acessado através da chave, de forma mais intuitiva, como no próximo exemplo:

```
aluno = {
    'nome' : 'Lucas Scheffer',
    'nota' : 6,
    'periodo': 2,
    'curso': 'Ciências da Computação'
}

print(aluno['nome'])
```

Assim, conseguindo acessar o valor 'Lucas Scheffer', apenas com a chave nome'

Operadores Unários: são operadores que aceitam apenas um operando, primeiro operador unário é o not:

```
print(not False)

print(not True)
```

Ele nega o valor, saindo assim True e False respectivamente.

Os outros dois operadores unários são "-" e "+", sendo o '-' deixando o valor negativo, e se já for negativo deixando positivo, e o '+' já não faz diferença, apenas não altera os valores.

Operadores Aritméticos: são operadores binários que utilizam de dois operandos:

Adição '+', Retorna o resultado da soma de dois valores.

Subtração '-' Retorna o resultado da diferença entre o primeiro e o segundo.

Multiplicação '*' Retorna o resultado da multiplicação de dois valores.

Divisão '/' Retorna o resultado da divisão do primeiro valor pelo segundo.

Divisão Inteira '//' Retorna uma divisão com valores inteiros, arredondando para baixo.

Potenciação '**' Retorna o resultado do primeiro valor elevado ao segundo.

Módulo ‘%’ Retorna o resto da divisão do primeiro pelo segundo valor.

Operadores Relacionais: Operadores binários que sempre retornam um valor falso ou verdadeiro.

Igualdade ‘==’ Retorna verdadeiro se os dois valores forem iguais.

Desigualdade ‘!=’ Retorna verdadeiro se os dois valores forem diferentes.

Maior que ‘>’ Retorna verdadeiro se o primeiro valor for maior que o segundo.

Menor que ’<’ Retorna verdadeiro se o primeiro valor for menor que o segundo.

Maior ou igual ‘>=’ Retorna verdadeiro se o primeiro valor for maior ou igual que o segundo

Menor ou igual ‘<=’ Retorna verdadeiro se o primeiro valor for menor ou igual que o segundo

Operadores Atributivos: Operadores binários que atribuem valor a alguma variável, porém existem vários tipos de atribuição.

‘+=’ acrescenta ao valor atual pelo segundo valor informado.

‘-=’ decrementa o valor atual pelo segundo valor informado.

‘*=’ multiplica o valor atual pelo segundo valor informado.

‘/=’ divide o valor atual pelo segundo valor informado.

‘%=’ resto do valor atual pelo segundo valor informado.

Operadores Lógicos: Operadores binários que também são chamados de operadores booleanos.

E lógico ‘And’ Retorna verdadeiro apenas quando os dois são verdadeiros.

Ou lógico ‘Or’ Retorna verdadeiro quando qualquer um dos dois é verdadeiro.

Não Lógico ‘Not’ Retorna verdadeiro quando é falso e retorna falso quando é verdadeiro.

Operadores Ternários: São basicamente if em apenas uma linha, facilita a legibilidade do código e deixa ele mais otimizado, vou apresentar a mesma versão de código utilizando operador ternário e if else:

```
vendas = 1000

bonus = 100 if vendas > 500 else bonus = 0
```

```
if vendas > 500:
    bonus = 100
else:
    bonus = 0
```

Os dois apresentam o mesmo resultado, porém vai de programador qual ele irá utilizar.

IF/ELSE É uma estrutura condicional que permite executar blocos de comando dependendo se a condição for verdadeira ou falsa.

```
numero = float(input('Insira um numero'))

if numero % 2 == 0:
    print("O numero é par")
else:
    print("O numero é impar")
```

No código acima mostra que se o resultado da divisão da variável por 2 for igual a 0, o terminal printa que o número é par, caso o resultado seja diferente de 0 é printado que o número é ímpar, também existe o comando ‘elif’ que é para quando precisa de mais de 2 condições.

For: Uma estrutura de repetição que repete o bloco de comandos dentro até quando a condição imposta for verdadeira.

```
n = int(input("Digite um numero"))

for i in range(n):
    print(i)
```

No caso da imagem acima, o print vai percorrer de 0 até o número imposto - 1, com o for é possível percorrer todos os tipos de variáveis já citadas.

While: Outra estrutura de repetição porém com uma forma de escrita e outras utilidades.

```
total = 0
quantidade = 0
nota = 0

while nota != -1:
    nota = float(input("Insira a nota ou -1 para sair "))
    if nota != -1:
        total += nota
        quantidade += 1

media = str(total/quantidade)
print(media)
```

Função: Função é um bloco de código independente que pode ser chamado várias vezes durante o programa, com o objetivo de facilitar o código não exigindo que digite o mesmo bloco mais de uma vez.

```
def calculadora(a, b):
    print(f"A soma de {a, b} é :{a + b}")
    print(f"A diferença de {a, b} é :{a - b}")
    print(f"A multiplicação de {a, b} é :{a * b}")
    print(f"A divisão de {a, b} é :{a / b}")


calculadora(15, 4)
```

No exemplo acima, toda vez que quiser calcular com dois números diferentes, basta apenas passar parâmetros diferentes, não necessitando escrever tudo isso, só puxando a função ‘calculadora’

Você consegue tanto receber o valor que a função retorna dentro de uma variável, quanto guardar uma função dentro da variável e chamar usando a variável que foi salva.

```python
def calculadora(a, b):
    print(f"A soma de {a, b} é :{a + b}")
    print(f"A diferença de {a, b} é :{a - b}")
    print(f"A multiplicação de {a, b} é :{a * b}")
    print(f"A divisão de {a, b} é :{a / b}")


somas = calculadora
somas(15, 4)
```

Existe a possibilidade também de passar uma função como parâmetro de outra função, como no exemplo:

```python
def multiplicaco(a, b):
   return a * b

def divisao(a, b):
    return a / b

def calcular(funcao, a, b):
    return funcao(a, b)

resultado = calcular(multiplicaco, 40, 3)
print(resultado)

resultado = calcular(divisao, 40, 3)
print(resultado)
```

Podemos também passar uma função dentro de outra, passando assim dois parâmetros diferentes, com um objetivo de melhorar o processamento do programa:

```python
def multiplicaco(a):
    def multi(b):
        return a * b
    return multi

resultado = multiplicaco(15)(4)
print(resultado)
```

```python
def multiplicaco(a):
    def multi(b):
        return a * b
    return multi

resultado = multiplicaco(15)
resultados = resultado(4)
```

Utilizando a variável para guardar a função multiplicação, conseguimos multiplicar ‘15’ por vários outros números sem necessariamente passar a função toda apenas chamando a variável que já havia sido guardada.

Map/Reduce: A função Map, recebe uma lista como argumento e aplica uma função em cima e retorna a lista com a função aplicada.

```python
from functools import reduce

def mais(n):
    return n + 2

lista = [3, 5, 7, 9]

resultado = map(mais, lista)
print(list(resultado))
```

Aqui, a função ‘mais’ foi chamada e aplicada a lista, onde cada elemento recebeu mais 2.

Já o Reduce tem a função de agregar todos os elementos de um conjunto e aplicar a função

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

lista = [3, 5, 7, 9]

resultado = reduce(mais, lista)
print(resultado)
```

Aqui o resultado foi a soma de todos os elementos da lista.

Lambda: A expressão Lambda nos permite criar uma função anônima em apenas uma linha, podendo ser usada dentro de uma variável ou dentro de uma função, não necessitando de 'return'

```python
from functools import reduce

alunos = [
    {'nome' : 'Ana', 'nota': 6.2},
    {'nome': 'Lucas', 'nota': 7.5},
    {'nome': 'Jose', 'nota': 4.5},
    {'nome': 'Scheffer', 'nota': 8.6},
    {'nome': 'Gustavo', 'nota': 3.7},
]

alunoAprovado = lambda aluno: aluno['nota'] >= 7

alunoAprovados = filter(alunoAprovado, alunos)
print(list(alunoAprovados))
```

Aqui a expressão Lambda foi utilizada para retornar se era verdadeiro ou falso que o aluno tirou maior igual a 7, e após isso a função filter, filtrou quem tirou abaixo de 7, deixando apenas os alunos aprovados

Classe: Classe vem diretamente da programação orientada a objetos, classe é algo que funciona como uma base para criar instâncias, chamadas de objetos. Cada objeto criado a partir de uma classe possui os atributos e métodos definidos pela classe, mas com valores e estados específicos.

```python
class Produto:
    def __init__(self, nome, preco, quantidade):
        self.nome = nome                    #atributos da classe
        self.preco = preco
        self.quantidade = quantidade

    #metodo que retorna o valor em estoque
    def valorestoque (self):
        return self.preco * self.quantidade

#objeto instanciado da classe Produto
p1 = Produto('caneta', 1.99, 15)
valorTotal = p1.valorestoque()
print(valorTotal)
```

Getter e Setter:

Getter é um método que permite acessar um atributo em uma determinada classe.

Setter é um método que permite alterar um atributo em uma determinada classe.

```python
def get_quantidade(self):
    return self._quantidade

def set_quantidade(self, quantidade):
    self._quantidade = quantidade
```

Usado quando os atributos são privados, e não permitindo acesso assim.

Herança: Herança é um dos pilares de orientação a objetos que funciona da seguinte forma, uma classe herda os métodos e atributos de uma outra classe, facilitando a escrita e reutilizando o código já escrito.

```python
class Veiculo:
    def __init__(self, marca, modelo):
        self.marca = marca
        self.modelo = modelo

    def acelerar(self):
        print("Acelerando o veículo...")

class Carro(Veiculo):
    def __init__(self, marca, modelo, cor):
        super().__init__(marca, modelo)
        self.cor = cor

    def ligar_radio(self):
        print("Ligando o rádio do carro...")


ferrari = Carro('ferrari', 2024, 'vermelha')
ferrari.acelerar()
ferrari.ligar_radio()
```

No código acima, existe a classe Veiculo com seus atributos e métodos, e existe a classe Carro que herda esses atributos e métodos de veículo, e recebe mais um parâmetro chamado de ‘cor’, após a instanciação do objeto ‘ferrari’ é possível ver que o objeto recebe os argumentos da classe veiculo e consegue utilizar os métodos.

Conclusões

Com a finalização desse card consigo concluir que python é uma linguagem muito mais simples do que comparada a linguagem de baixo nível como C. Esses ajudam a construir uma base sólida ao apresentar tópicos como sintaxe básica, variáveis, estruturas de controle e funções,

essenciais para resolver problemas e avançar no aprendizado. Além disso,os vídeos oferecem exemplos práticos, tornando o conteúdo mais dinâmico e compreensível

Referências

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

https://www.hashtagtreinamentos.com/dicionarios-em-python?

https://vaiprogramar.com/quais-sao-os-operadores-usados-em-python/