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

Nessas duas aulas, foi passado a prática do básico da linguagem python, onde aprendemos sobre:

Import: Utilizado para carregar módulos(arquivo) e bibliotecas(contêm 1 ou mais módulos) externas.
Sintaxis: import operadores.aritméticos(nome_da_pasta . nome_do_arq)
ou from operadores import aritméticos .

ABS: Qualquer valor, seja positivo ou negativo, é transformado em positivo, tirando o sinal “-” do número.

Concatenar: Para concatenar valores de diferentes tipos, utilizamos o format
Sintaxe: print(f”Meu nome é {nome} e eu tenho {idade}”)
- obs: com variáveis previamente definidas.

Input: Para o usuário digitar algo, é utilizado o input, muitas vezes tendo que formatar a variável de string para float para realizar operações matemáticas
Sintaxe: raio = float(input("informe o raio da circ? "))

Type: Dá para verificar o tipo da variável utilizando: print(type("texto"))

1 - Tipos:

1.1 - Listas: Após criar a lista, podemos utilizar o lista.append(3) para adicionar um elemento na lista e da mesma forma, atribuir valores com lista[2] = 100 (onde a terceira posição da lista receberá o valor 100).
A lista de forma geral pode ser modificada, tanto adicionar posição, mudar valor e adicionar posições.
Sintaxe: lista = [1,2,3].

1.2- Tuplas: Diferentemente das listas, as tuplas após serem criadas não podem ser modificadas, desde o tamanho e conteúdo dentro da mesma, assim o desempenho dela se tornando ligeiramente melhor.
Sintaxe: tupla = (1,2,3).

1.3 - Conjuntos: Nos conjuntos, valores não podem ser duplicados, eles são automaticamente removidos. Ele não é indexado, ou seja, não conseguimos acessar um elemento em uma posição específica.
Sintaxe: conj = {1,2,3}

1.4 - Dicionários: Podemos inserir nos dicionários dados em pares chave-valor, é diferente da lista ou tupla, acessamos cada elemento por chaves únicas

Sintaxe: dic = {
“nome” = “Higor”
“nota” = 9.0
}
print(aluno["nome"]) #assim acessamos o nome

2 - Operadores:

2.1 - Unários: Eles manipulam em apenas um operando, como, inverter sinais de positivos e negativos, o ‘’not’’ inverte o valor lógico de um operando na forma booleana.

2.2 - Aritméticos: Eles já manipulam em dois operandos: (x+y), (x-y) (x/y) (x*y) (x%y).

2.3 - Relacionais: Comparam dois valores e o retorno sempre vai ser um valor booleano(true ou false): (x>y), (x>=y), (x<y), (x<=y), (x!=y), (x==y)

2.4 - Atribuição: Atribuir valores nas variáveis, e como são tipos dinâmicos, conseguindo modificar os tipos nas atribuições, da mesma forma, conseguimos fazer atribuições matemáticas.

2.5 - Lógicos: Eles comparam as expressões booleanas retornando um valor também booleano, utilizando como operando o AND, NOT, OR, assim, podendo analisar as comparações retornando um bool.

2.6 - Ternário: Resulta em uma expressão condicional utilizando if e else na mesma linha, conseguindo fazer uma condição a partir de uma análise lógica.

3 - Controles:

3.1 - Estruturas de controle: Utilizado para obter respostas com tomadas de decisões, no caso do if, se a primeira condição for verdadeira, o bloco de código será executado, temos o elif que está como um intermediário que adiciona

verificações caso a primeira for falsa, e o else, caso todas as afirmações anteriores forem falsas ele será executado.

3.2 - Estruturas de repetição: Vimos duas estruturas de repetição, o for que atua fazendo iterações percorrendo uma lista,tuplas,dicionários entre outros, e executando um bloco de código para cada iteração.
Sintaxe: for i in range(5) - imprime de 0 a 4
E temos o while, onde enquanto uma condição ela for verdadeira, é executado o bloco de código, e quando ela se tornar falsa, sai do looping.

Funções: Utilizamos uma palavra para definir uma função “def”, elas são blocos onde podemos passar parâmetros e realizar operações dentro da mesma e retornar um valor final. São importantes para conseguirmos deixar o código mais organizado e não ficar repetindo trechos de códigos iguais caso precisar utilizar a mesma função, assim, deixando o código mais limpo e fácil de entender.
Também conseguimos colocar uma função dentro de outra, chamada de função interna, conseguindo utilizar variáveis que foram passadas como parâmetros da função externa, assim, conseguindo subdividir tarefas deixando o código mais dinâmico, e, neste caso, funções internas não podem ser utilizadas fora das funções externas, deixando o código menos poluído.
Sintaxe função: def(nome_da_funcao):
Há as funções lambdas, que são funções invisíveis, úteis quando precisamos de algo simples e efetivo.
Exemplo Lambda: soma = lambda x, y: x + y.

PEP(Python Enhancement Proposal): Guia de estilo de codificação do python, propondo melhorias para a escrita clara e compreensível, agindo como uma comunidade, para deixar o código mais limpo e para a compreensão ser melhor.

***args:** Permite passar para as funções como parâmetro quantos argumentos for necessário, sendo reconhecidos como uma tupla, estando vazia ou não, tornando a flexibilidade maior e abrindo várias possibilidades de execução com funções.

****kwargs:** Uma convenção usada para passar dicionários, diferentemente de args, que os argumentos podem ser apenas unitários. Kwargs oferece grandes possibilidades de forma dinâmica, conseguindo efetuar mudanças dentro dos dicionários.

Enumerate: Utilizado para enumerar uma lista de números, sendo uma lista e/ou tupla.

Map: Pega uma lista, converte em outra lista com o mesmo tamanho, porém mapeando uma informação em outro, ou seja, iterando algo ou algum valor.

Reduce: Passa como parâmetro uma função, valores e por onde começar, e reduz em apenas um argumento. Por exemplo, dá para pegar as notas dos alunos, e com uma função somar, reduzir em apenas um número onde está contido a soma das notas.
Filter: Ele é usado como filtro de um iterável, onde, ele passa por uma condição definida anteriormente, onde o retorno é apenas os elementos que a função retorna true.

Classes: As classes permite a programação orientada a objetos, onde inicialmente criamos objetos, onde dentro deles podemos declarar funções especiais(instâncias), como a __init__ , é chamado automaticamente quando um novo objeto é criado, usado para instanciar um objeto, é passado como primeira parâmetro o “self”, depois conseguimos criar outros parâmetros, quantos quiser, como nome, preco, altura. Com o self.nome = nome, conseguimos criar uma atribuição dinamicamente.

Métodos: Nos métodos, nos “damos vida” ao objeto anteriormente criado, fazendo funções nas quais podemos realizar operações matemáticas ou até “movimentos”, assim conseguindo utilizar no restante do código.

@property: Faz com que o método seja entendido como uma variável, facilitando a utilização, permitindo que usamos como atributos da classe.

Setter e Getter: Ambos permitem fazermos modificações no atributo de uma classe de forma controlada.
O getter retorna o valor de um atributo sem alterar e um setter atualiza o valor de um atributo.

@classmethod: O método de classe não precisa criar uma instância necessariamente para ser chamado como o primeiro argumento, conseguimos então, chamar o próprio método para utilizar como primeiro argumento, podendo acessar diretamente via classe o método.

@staticmethod: Usado caso não precisamos acessar nenhum atributo que pertence a classe, não podendo acessar e consequentemente modificar algo da classe.
Igualmente ao @classmethod, não precisamos criar uma instância para acessá-lo.

Conclusão: Nos dois vídeos, aprendemos conceitos básicos porém primordiais de python, desde operadores numéricos até a criação de classes e manipulação dos mesmo, nos permitindo conhecer a linguagem e futuramente aprofundar ainda mais, servindo como base para o os próximos cards que veremos.

Referências:

- PYTHON 3 Curso Rápido 🐍 Parte #1 2020 - 100% Prático!
- PYTHON 3 Curso Rápido 🐍 Parte #2 2020 - 100% Prático!