Sistema de diretórios: entender como manipular diretórios lista-los utilizando o Python Menezes (2019).

Um diretório corrente em Python é considerado um ***path*** (caminho).

Trabalhar diretamente com arquivos no mesmo diretório quando o projeto é pequeno pode ser uma comodidade. Porém, normalmente um projeto começa com uma pequena ideia e vai aumentando de tamanho conforme o cliente vai percebendo o quanto o programa pode ajudá-lo, por isso é importante estar atendo ao fator escalabilidade.

- No Windows - considerando que temos o diretório base proj001, ative a política do segurança do windows.

```
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned
```

- Ative o Ambiente virtual no Windows com PowerShell:

```
.\venv\Scripts\Activate.ps1
```

- Ativar o Ambiente virtual no Windows com bash:

```
source venv/Scripts/activate
```

- Abra o *Visual Studio Code,* abra o path criado e após crie o arquivo principal.py.

Verifica o path – módulo os função **getcwd**()

```
import os
root_dir = os.getcwd()
print(f'\nO Path é: {root_dir} \n')
```

Em Python podemos utilizar o módulo **os** e a função **mkdir**(“nome”) para criar *path* e sub *path*. A partir do *Path*, usando o terminal do VSCode crie um sub *path* chamado clientes:

Cria os sub *path* **clientes** e **produtos** módulo **os** função **mkdir**()

```
import os
os.mkdir('clientes')
os.mkdir('produtos')
```

lista todos os path e arquivos relativos

```
import os
print(f'\nOs paths e arquivos são: {os.listdir(".")} \n')
```

Mostra o paths a partir do diretório de trabalho atual

```python
import os

root_dir = os.getcwd()
print(f'Diretório inicial do projeto: {root_dir} \n')

# Função para listar diretórios, excluindo o ambiente virtual
def directory_list(radix):
    # os.walk() é usado para gerar os nomes dos diretórios na árvore do diretório fornecido
    for radix, directories, _ in os.walk(radix):
        # Pula diretórios do ambiente virtual
        if 'env' in directories:
            directories.remove('env')
        for directory in directories:
            # os.path.join() é usado para criar o caminho completo para o diretório
            print(f'Diretórios do projeto: {os.path.join(radix, directory)} \n')

# Chamada da função com o diretório raiz como argumento
directory_list(root_dir)
```

Mostra os paths em forma de àrvore

```python
import os

def build_tree(root_dir, prefix="", skip_dirs=None):
    if skip_dirs is None:
        skip_dirs = ['env', '.venv']  # Lista de diretórios a serem ignorados
    files = sorted(os.listdir(root_dir))  # Ordena arquivos e diretórios

    # Filtra os arquivos para remover diretórios indesejados
    files_dirs = [f for f in files if os.path.isdir(os.path.join(root_dir, f)) and f not in skip_dirs]

    for i, filename in enumerate(files_dirs):
        path = os.path.join(root_dir, filename)
        is_last = i == (len(files_dirs) - 1)

        # Mostra o diretório
        print(f"{prefix}{'└── ' if is_last else '├── '}{filename}")

        # Se não for o último item, adicione uma linha vertical abaixo deste diretório
        extension = "    " if is_last else "│   "
        build_tree(path, prefix + extension, skip_dirs)

# Obtém o diretório de trabalho atual e inicia a construção da árvore a partir dele
root_dir = os.getcwd()
print(f"Diretório inicial do projeto: {root_dir}")
build_tree(root_dir)
```

Em Python podemos mudar o subdiretório de trabalho usando o módulo **os** e a função **chdir**(“nome”) para abri-lo.

Abrindo um sub path - módulo **os** função **chdir**()

```python
import os

# Verifica se a pasta "clientes" existe no diretório atual
if 'clientes' in os.listdir():
    os.chdir('clientes')
    print(f'O path é: {os.getcwd()}\n')
else:
    print("A pasta 'clientes' não foi encontrada no diretório atual.")
```

Note que o endereço do sub *path* **clientes** é relativo ao diretório proj001, ou seja, o path **proj001** é **pai** do sub *path* **clientes**. O que a função **chdir**() faz é mudar o subdiretório corrente permitindo o acesso aos arquivos que estão dentro dele mais facilmente.

Para retornar ao *path* pai ou outro *path* relativo ao diretório corrente (***path*** corrente) aplicamos a função **chdir**('.')

retornando ao subdiretório pai - módulo os função **chdir('.')**

```
import os
os.chdir('..')
print(f'\nO path é: {os.getcwd()}\n')
```

Em Python podemos renomear um *path's* com a função **rename**() do módulo **os**.

Renomeando clientes para cli

```
import os

# Tentativa de renomear o diretório ou arquivo 'clientes' para 'cli'
try:
    os.rename('clientes', 'cli')
    print(f'\nO novo nome é: {os.listdir(".")}\n')
except FileNotFoundError:
    print("O diretório ou arquivo 'clientes' não foi encontrado.")
```

Para apagar um *path* aplique a função **rmdir**() do módulo **os**

Apagando **cli**

```
import os
os.rmdir('cli')
print(os.listdir('.'))
```

Uma função interessante do Python para manipular *path*, mas que deve ser utilizada com cautela, é a função **makedirs**() do módulo **os**. Essa função permite criar mais de um path de uma vez.

Cria os *path's* **teste** e o *path* relativo a teste chamado **test_cliente**

```python
import os

# Define o diretório base e os subdiretórios
base_dir = 'proj001'
test_dir = os.path.join(base_dir, 'teste')
test_client_dir = os.path.join(test_dir, 'test_client')

# Cria os diretórios, se eles não existirem
os.makedirs(test_client_dir, exist_ok=True)

# Função para listar a estrutura de diretórios de forma recursiva
def build_tree(root_dir, prefix=""):
    items = os.listdir(root_dir)
    items.sort()  # Ordena arquivos e diretórios
    for i, item in enumerate(items):
        path = os.path.join(root_dir, item)
        is_last = i == (len(items) - 1)
        # Checa se é um diretório e imprime de acordo
        if os.path.isdir(path):
            print(prefix + "└── " + item if is_last else prefix + "├── " + item)
            new_prefix = prefix + ("    " if is_last else "│   ")
            build_tree(path, new_prefix)
        else:  # Para arquivos
            print(prefix + "└── " + item if is_last else prefix + "├── " + item)

# Imprime a estrutura a partir do diretório base
print(f'Estrutura de diretórios a partir de {base_dir}:')
build_tree(base_dir)
```

Apague o sub path test_client

```python
import os

# caminho base que será ajustado para o sistema operacional atual
base_path = os.path.join('Users', 'Edados_2023', 'proj_22_11', 'teste', 'test_client')

# Verifica o sistema operacional e ajusta o caminho se necessário
if os.name == 'nt':  # Sistema operacional Windows
    path_to_remove = os.path.join('C:\\', base_path)  # Ajusta o caminho para o Windows
else:
    path_to_remove = os.path.join('/', base_path)  # Ajusta o caminho para sistemas Unix/Linux

# Remove o diretório
try:
    os.rmdir(path_to_remove)
    print(f"O diretório {path_to_remove} foi removido com sucesso.")
except OSError as error:
    print(f"Erro ao remover o diretório {path_to_remove}: {error}")
```

Atenção: o script verifica o sistema operacional onde está sendo executado (os.name == 'nt' indica que é Windows) e ajusta o caminho do diretório conforme necessário. Além disso, ele tenta remover o diretório e captura

qualquer erro que possa ocorrer durante o processo, exibindo uma mensagem apropriada.

Outra função importante do módulo **os** é a função **path**(). Ela serve obtermos o tamanho do subdiretório, data de criação, acesso e modificação de dados, ou verificar se a informação que estamos lidando é um subdiretório ou arquivo.

Verifica subdiretórios existentes módulo **os** função **path.isdir()**

```
import os

# Lista todos os subdiretórios no diretório atual
for subdiretorio in os.listdir('.'):
    if os.path.isdir(subdiretorio):
        print(f'{subdiretorio}')
```

Verifica se um subdiretório existe módulo **os** função **path.exists()**

```
import os.path

# Verifica se o diretório 'cli' existe no diretório atual
if os.path.exists('cli'):
    print('O diretório existe')
else:
    print('O diretório não existe')
```

Identifica o sistema operacional - – Linux/IOS/ Windows

```
import platform

# Imprime o nome do sistema operacional
print(platform.system())

# Imprime informações adicionais, como a versão do sistema operacional
print(platform.release())
```

ELEMENTOS DA ORIENTAÇÃO A OBJETOS: MÉTODOS

Método: comportamento do objeto. Segundo F.V.C. (2020, página 199), "... os métodos tentam representar a dinâmica das ações que podem ser admitidas" pelo objeto. Partindo desta premissa, dentro de uma classe podemos ter quantos métodos nossa regra de negócio precisar.

- Com exceção do método do método **__init__** (método construtor). Na linguagem Python dividimos os métodos em dois grupos: **metodos de instâncias** e **metodos de classes.**

- Métodos são escritos com letras minúsculas e se o nome for composto, as palavras que o compõe devem ser separadas por underline (_) ou utilizar o estilo CamelCase.

- Evite usar o *dunder* para criar métodos que não seja o construtor, pois, o Python tem vários métodos que usam essa nomenclatura, sendo assim, corre-se o risco de mudar o comportamento de alguns métodos internos da linguagem.

 # 26 - Alguns métodos builtins do Python

```
print('\n',dir())
```

```
['__annotations__', '__builtins__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__spec__']
```

- MÉTODOS DE INSTÂNCIA: têm esse nome por precisar de uma classe para utilizá-lo, ou seja, ele está fica dentro da classe.

 # 27 - métodos de instancia

```
class Lampada:
    def __init__(self, cor, voltagem, luminosidade):
        self.__cor = cor
        self.__voltagem = voltagem
        self.__luminosidade = luminosidade
        self.__ligada = False
```

```
class ContaCorrente:

    contador = 6999

    def __init__(self, limite, saldo):
        self.__numero = ContaCorrente + contador +1
        self.__limite = limite
        self.__saldo  = saldo
        ContaCorrente.contador = self.__numero
```

- A primeira conta criada receberá contador mais 1
- A segunda conta criada será o número da última conta corrente mais 1...

```
class Produto:

    contador = 0

    def __init__(self,  nome, descricao, valor):
        self.__id = Produto.contador + 1
        self.__nome = nome
        self.__descricao = descricao
        self.__valor = valor
        Produto.contador = self.__id
```

```
class Usuario:

    def __init__(self, nome, email, senha):
        self.__nome = nome
        self.__email = email
        self.__senha = senha
```

Note que **todos** os atributos foram definidos como **privado**, para determinar um mínimo de segurança em nossa classe. Lembre-se os atributos devem ser acessados somente de dentro da classe.

 # 28 - Desenvolva um método de instância na classe Produto que retorne o valor do desconto de um produto será definido futuramente:

 # 29 - instancie o produto passando todos os dados da classe, porém, retorne apenas o valor do produto com desconto que foi definido. Fórmula desconto: valor * (100 - porcentagem) / 100)

 # 30 – Refatore a classe Usuario para receber o atributo sobrenome, depois desenvolva um método na mesma classe que retorne o nome completo e a senha.

 # 31 - Instancie 3 objetos e apresente na tela o nome completo e senha.

Sabemos que expor a senha do usuário é algo que se deve evitar, o melhor a fazer será criptografar a senha, para tanto, será necessário instalar uma biblioteca externa do Python. Abra o terminal e faça a instalação:

- Biblioteca para criptografar senha

```
pip install passlib
```

- Feita a instalação - importe a biblioteca para seu código:

```
from passlib.hash import pbkdf2_sha512 as cryptografa
```

- Para criptografar a senha a sintaxe é:

```
from passlib.hash import pbkdf2_sha512 as cryptografa

... = cryptografa.hash(senha, salt_size = 32, rounds = 12000)
```

- O comando **salt_size**(32), é o tamanho da parte de texto usada para criptografar;

- O comando **round = 12000**, embaralha duzentas mil vezes antes de gerar a senha.

 # 32 - Refatore a classe Usuario para receber o atributo senha criptografada, depois desenvolva um método na mesma classe que retorne o nome completo e a senha criptografada.

- Os exemplos e exercícios acima nos prepararam para trabalhar com os valores da instância do objeto e quem tem o valor nome, sobrenome, e-mail e senha é a instância e não a classe.

MÉTODOS DE CLASSE: não estão vinculados a nenhuma instância da classe, mas diretamente a classe. São considerados métodos estáticos.

Para declarar um método de classe devemos usar um ***decorator*** junto com o comando `@classmethod` classmethod

Ao utilizar um decorador `@classmethod` por convenção, o parâmetro do método que será decorado é: (cls) uma abreviação da palavra classe. Exemplo:

```
@classmethod
def conta_usuario(cls):
    print(f'\nClasse: {cls}')
```

Para acessar os métodos de classe, segue-se o mesmo princípio dos atributos de classe, ou seja, não utilizamos a instancia da classe para fazer o acesso, mas sim, a própria classe. Exemplo:

 # 33 - Aplicando métodos de Classe

```
from passlib.hash import pbkdf2_sha512 as cryptografa

class Usuario:

    controle = 0

    @classmethod
    def qtd_usuario(cls):
        print(f'\nTotal de usuários cadastrados no sistema: {cls.controle}')

    def __init__(self, nome, sobrenome, email, senha):
        self.__id = Usuario.controle + 1
        self.__nome = nome
        self.__sobrenome = sobrenome
        self.__email = email
        self.__senha = cryptografa.hash(senha, salt_size = 32, rounds = 12000)
        Usuario.controle = self.__id

usuario_um = Usuario('Gnomica', 'Cristófina','Gnomica@gmail.com', '654321')

usuario_um.qtd_usuario()
```

```
Total de usuários cadastrados no sistema: 1
```

MÉTODOS DE CLASSE PRIVADOS: são métodos que só podem ser acessados dentro da classe. São iniciados com duplo *underline*. Exemplo:

 # 34 - Métodos de classe privados

```
class Usuario:

    controle = 0

    @classmethod
    def qtd_usuario(cls):
        print(f'\nTotal de usuários cadastrados no sistema: {cls.controle}')

    @classmethod
    def Disciplina(cls):
        print('Estrutura de Dados')

    def __init__(self, nome, sobrenome, email, senha):
        self.__id = Usuario.controle + 1
        self.__nome = nome
        self.__sobrenome = sobrenome
        self.__email = email
        self.__senha = cryptografa.hash(senha, salt_size = 32, rounds = 12000)
        Usuario.controle = self.__id
        print(f'\nNome do e-mail capturado: {self.__captura_nome_email()} \n')

    def nome_completo_senha(self):
        return f'Nome.....: {self.__nome} {self.__sobrenome} - Senha: {self.__senha}'

    def __captura_nome_email(self): # métododе classe privado
        return self.__email.split('@')[0]

usuario_dois = Usuario('Gertrudez','Sancré','Gertrudez@gmail.com', '01010101')
```

```
Nome do e-mail capturado: Gertrudez
```

MÉTODOS DE CLASSE ESTÁTICOS: para cria-lo usamos o *decorator* @staticmethod, que é usado para retornar uma constante. Nesse tipo de método não há acesso a classe ou a instância. Exemplo:

 # 35 - Métodos de classe estático

```
class Usuario:

    controle = 0

    @classmethod
    def qtd_usuario(cls):
        print(f'\nTotal de usuários cadastrados no sistema: {cls.controle}')

    @classmethod
    def disciplina(cls):
        print('Estrutura de Dados')

    @staticmethod
    def instituicao():
        return 'Instituto Federal de Rondônia - Campus Ariquemes - 2022'

    def __init__(self, nome, sobrenome, email, senha):
        self.__id = Usuario.controle + 1
        self.__nome = nome
        self.__sobrenome = sobrenome
        self.__email = email
        self.__senha = cryptografa.hash(senha, salt_size = 32, rounds = 12000)
        Usuario.controle = self.__id
        print(f'\nNome do e-mail capturado: {self.__captura_nome_email()} \n')

    def nome_completo_senha(self):
        return f'Nome.....: {self.__nome} {self.__sobrenome} - Senha: {self.__senha}'

    def __captura_nome_email(self): # métododе classe privado
        return self.__email.split('@')[0]

Usuario.qtd_usuario()

Usuario.disciplina()

print(Usuario.instituicao())

usuario_dois = Usuario('Gertrudez','Sancré','Gertrudez@gmail.com', '01010101')
```

```
Total de usuários cadastrados no sistema: 0
Estrutura de Dados
Instituto Federal de Rondônia - Campus Ariquemes - 2022

Nome do e-mail capturado: Gertrudez
```

Poste aqui todos os códigos e exercícios do material 28-02-2024 até o dia -06-03-2024 às 17:00 - não vale pontos.