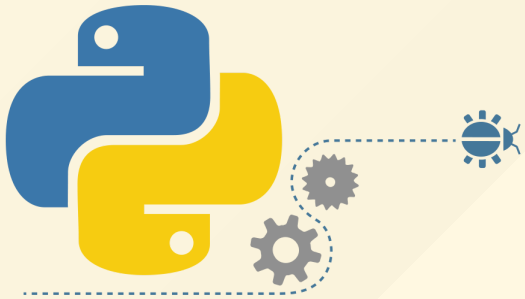

Python Fundamentals

Módulo 4

Clodonil Trigo (@clodonil)

Estruturada (Procedural)

> Estrutura a execução através de rotinas que são chamadas conforme a necessidade. Normalmente as chamamos de procedures/funções.

```
def soma(x,y):
    result = x + y
    return result


if __name__ == "__main__":
    x= soma(10,30)
    print(x)
```

import

- Importar todas as funções:

```
import calc

calc.soma(10,20)
```

- Importar funções especificas:

```
from calc import soma

soma(10,20)
```

Classes

- O método especial `__init__()` representa o construtor da classe e será executada somente quando a classe é criada;
- `Self` representa o endereçamento do objeto na mémoria, utilizado para acessar os atributos e métodos.

banco.py

```
class Conta:
  def __init__(self,nconta, titular, saldo, limite):
    print("Construindo objeto...")
    self.nconta = nconta
    self.titular = titular
    self.saldo = saldo
    self.limite = limite

  def deposito(self, valor):
    self.saldo += valor

  def saque(self, valor):
    self.saldo -= valor

  def extrato(self):
    print("Sr(a) {0} o Saldo eh {1}".format(self.titular,self.saldo))
```

Encapsulamento

podemos deixar o atributo no modo `privado`, e para isso adicionamos __ (underscore) antes do nome da variável.

```
class Conta:
   def __init__(self,nconta, titular, saldo, limite):
     print("Construindo objeto...")
     self.__nconta = nconta
     self.__titular = titular
     self.__saldo = saldo
     self.__limite = limite
```

Métodos para implementar os getters e setter para ter acesso ao atributo protegido.

```
@property
def saldo(self):
    return self.__saldo

@property
def titular(self):
    return self.__titular.capitalize()

@property
def limite(self):
    return self.__limite

@limite.setter
def limite(self, limite):
    self.__limite = limite
```

Herança

Criando a classe `Investimento` e herdando os métodos e atributos na classe `Conta`.

```
class Investimento(Conta):
    pass
```

- Validando a criação da classe `Investimento`.

```
>>> from conta import Conta, Investimento
>>> maria = Investimento(1321,'Maria',100,200)
Construindo objeto...
>>> maria.titular
'Maria'
>>> maria.limite
200
```

Módulo 4

Persistência de Dados

Date e Time

Para manipulação de `data` e `hora`, vamos um módulo interno do Python `datetime`.

```
import datetime
now = datetime.datetime.now()
print(now)  #2018-11-20 01:27:10.903625
```

Date e Time

Para formatar a impressão da data com as informações necessárias, podemos utilizar o método `strftime`.

```
import datetime

now = datetime.datetime.now())
print(now.strftime("%d/%m/%y"))
```

Date e Time

Os parâmetros que podemos utilizar no strftime são:

Parâmetros	Descriçao	Exemplo
%a	Apreviação do dia da semana.	Sun, Mon, ..., Sat
%A	Nome completo do dia da semana.	Sunday, Monday, ..., Saturday
%w	Dia da semana em numeral, sendo 0 para domingo e 6 para sábado.	0, 1, ..., 6
%d	Dia do mês em numeral.	01, 02, ..., 31
%b	Nome do mês abreviado.	Jan, Feb, ..., Dec
%B	Nome do mês completo.	January, February, ..., December
%m	Mês em númeral.	01, 02, ..., 12
%y	Ano com 2 casas decimal.	00, 01, ..., 99

Date e Time

Os parâmetros que podemos utilizar no strftime são:

Parâmetros	Descriçao	Exemplo
%Y	Ano com 4 casas decimal.	1970, 1988, 2001, 2013
%H	Hora (24-hora).	00, 01, ..., 23
%I	Hora (12-hora).	01, 02, ..., 12
%p	AM ou PM.	AM, PM
%M	Minuto.	00, 01, ..., 59
%S	Segundos.	00, 01, ..., 59
%f	Microsegundos.	000000, 000001, ..., 999999
%%	Escape do %.	%

Date

O `import datetime` importa o módulo inteiro de date e hora, entretando muitas vezes é necessário apenas a trabalhar com `date`, não sendo necessário importar a classe date.

```
from datetime import date

agora = date.today()

print(agora.day)
print(agora.month)
print(agora.year)
```

Converter uma string em datetime.

```
from datetime import datetime

data_em_texto = '01/03/2018 12:30'
data_e_hora = datetime.strptime(data_em_texto,
                                '%d/%m/%Y %H:%M')
print(data_e_hora)
```

#Operação com Data

```
import datetime
#data atual
now = datetime.datetime.now()
# data nascimento
nasc = datetime.datetime.strptime('01/03/2018 12:30',
                                  '%d/%m/%Y %H:%M')
print ( now - nasc)
print ( now > nasc)
```

Timedelta

A classe `timedelta` também é útil na manipulação das datas. Com ela podemos movimentar as datas em semanas (`weeks`), dias (`days`), horas (`hours`), minutos (`minutes`) ou segundos (`seconds`).

Timedelta (Exemplo1)

somar duas semanas em um data especifica.

```
import datetime
data = datetime.datetime.strptime('01/03/2018 12:30',
                                  '%d/%m/%Y %H:%M')
print(data)
data += datetime.timedelta(weeks=2)
print(data)
```

Timedelta (Exemplo2)

Vamos subtrair 100 dias de um data especifica.

```
import datetime
data = datetime.datetime.strptime('01/03/2018 12:30',
                                  '%d/%m/%Y %H:%M')
print(data)
data -= datetime.timedelta(days=100)
print(data)
```

Trabalhando com Arquivos

Podemos tornar os nossos programas muito mais interessante quando podemos gravar e ler conteúdo de arquivos. No Python podemos utilizar a função `open()` para `ler` o conteúdo de um arquivo e também para `escrever` nesse arquivo.

`open()`

A função `open()` tem a seguinte sintaxe:

open(`filename`, `mode`)`:

mode	descrição
'r'	Abre o arquivo apenas para leitura;
'w'	Cria o arquivo para escrita.
'a'	Abre o arquivo para escrita e adiciona o conteúdo no final do arquivo.
'r+'	Modo especial de leitura e gravação.

Se nenhuma opção for declarado, por padrão vai ser `r`.

`open()`

As funções de leitura e escrita:

- `readlines()` : A função readlines retorna todas as linhas em forma de lista, sendo uma linha em cada posição.
- `read()` : Retorna todo o conteúdo de um texto;
- `write()` : Escreve uma stringo no arquivo.

open()

O programa vai ler o conteudo do arquivo e imprimir na tela.

```
#abrir o arquivo
farq = open('file.txt')
#Lendo todo o conteúdo do arquivo
conteudo = farq.readlines()
#Fechando o arquivo
farq.close()
#Imprimindo o conteúdo do arquivo
print(conteudo)
```

O arquivo `file.txt` está no mesmo diretório do código em python.

`open()` (Exemplo)

Nesse exemplo vamos escrever uma linha no arquivo `file2.txt`.

```
linha = "Primeira linha do arquivo"
fcon = open('file2.txt','w')
fcon.write(linha)
fcon.close()
```

JSON

O Python manipula dados em JSON (`JavaScript Object Notation`) através da biblioteca interna `json`.

Principais funções da classe json são:

- `loads`: Transforma um json em dicionário
- `dumps`: Transforma um dicionário no formato JSON

JSON

Transformar um json que está na string para dicionário usando a função `json.loads`:

```
import json
json_string = '{"first_name": "Guido", "last_name":"Rossum"}'
parsed_json = json.loads(json_string)
print(parsed_json['first_name'])
```

JSON

Função `dumps` para transformar um dicionário em json.

```
import json
dados = {
          "nome" : "Jose",
                   "end"   : "rua de  baixo",
                   "idade": "40"
         }
json_dados = json.dumps(dados)
print(json_dados)
```

#JSON

Uma combinação bastante utilizada e interessante é obter dados de API em JSON e manipular esse dados.

```
import json
import requests

response = requests.get(
           "https://jsonplaceholder.typicode.com/todos")
todos = json.loads(response.text)

print(todos)
```

JSON

Escrever o JSON em um arquivo. O processo é semelhante a ler um arquivo texto comum.

```
import json

dados = { 'nome': 'jose',
          'cidade': 'SP',
                    'partido': 'NOVO'
                    }

j_file = open('filename.json','w')
j_file.write(json.dumps(dados))
j_file.close()
```

JSON

Ler o conteúdo e realizar a transformação para dicionário.

```
import json

j_file = open('filename.json','r')
raw = j_file.read()
j_file.close()

dados =  json.loads(raw)

print(dados['nome'])
print(dados)
print(json.dumps(dados, indent=4, sort_keys=True))
```

CSV

Outra forma bastante comum para armazenar os dados os dados de forma estruturada é utilizando arquivos CSV (`Comma Separated Values`).

Como exemplo, vamos criar um arquivo CSV manualmente.

Salve com o nome `arquivo.csv`.

```
id;nome;idade;partido
1;jose;90;PT
2;maria;18;PSDB
3;carlos;20;NOVO
```

CSV

Arquivo CSV utiliza um delimitador para separar os dados, no exemplo o delimitador utilizado é o `;`.

Para manipulação de dados em CSV, vamos utilizar a biblioteca `csv`. Primeiramente vamos lêr os dados do arquivo e transformar em uma lista.

CSV

O método `csv.reader` faz todo o trabalho que é ler o arquivo e transformar em uma lista que python facilmente consegue tratar.

```
import csv
csvfile = open('arquivo.csv')
spamreader = csv.reader(csvfile, delimiter=';')

for item in spamreader:
  print(item)
```

CSV

Salvar os dados em um arquivo CSV.

O método `csv.write` abre o arquivo para escrita e define o delimitador. Já o `writerow` salva os dados de uma linha.

```
import csv
csvfile = open('file2.csv', 'w')
writer = csv.writer(csvfile, delimiter=';')

dados = [  ['nome','idade','UF'],
           ['Maria','19','SP'],
           ['Jose','40','RJ'],
           ['Pedro','21','MG'],]
for linha in dados:
    writer.writerow(linha)
csvfile.close()
```

Banco de Dados

A biblioteca `sqlalchemy` é muito interessante porque transforma a estrutura de banco de dados em código.

`SQLAlchemy` é um SQL toolkit ORM (`Object Relational Mapper`), ela abstrae as funções que são executadas em um banco de dados.

Muitos banco de dados são suportados, alguns deles são:

DB	
SQLite	Firebird
Postgresql	Sybase
MySQL	Muitos outros....
Oracle	
MS-SQL	

Banco de Dados

Para utilizar o `SQLAlchemy` precisamos instalar a biblioteca:

```
pip install SQLAlchemy
```

Banco de Dados

Com a biblioteca instalado, vamos criar um banco de dados e realizar as operações básicas utilizado CRUD.

```
from sqlalchemy import Column, Integer, String
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import create_engine
from sqlalchemy.orm import sessionmaker

Base = declarative_base()

class User(Base):
    __tablename__ = 'users'
    id = Column(Integer, primary_key=True)
    nome   = Column(String(250), nullable=False)
    idade  = Column(String(250), nullable=False)
    cidade = Column(String(250), nullable=False)

banco = "banco_de_dado.db"
engine = create_engine("sqlite:///{0}".format(banco))

if not os.path.exists(banco):
   Base.metadata.create_all(engine)

#Base.metadata.bind = engine
DBSession = sessionmaker(bind=engine)
session = DBSession()
```

Continuando....

```
# Adicionando um novo registro
new_person = User(nome='clodonil',idade='10',cidade='SP')
session.add(new_person)
session.commit()
```

```
# Lista todos
def lista_todos():
    for user in session.query(User).all():
        print(user.nome, user.idade, user.cidade)
```

Continuando....

```
# Pesquisar por um elemento espeficio
user1 = session.query(User).filter(User.nome == 'clodonil').first()

lista_todos()
#Alterar um Registro
user1.nome = "jose"
user1.idade = "20"
session.commit()
lista_todos()
```

```
# Delete
user1 = session.query(User).filter(User.nome == 'clodonil').first()
session.delete(user1)
session.commit()
```

Packages

As bibliotecas externas podem ser instaladas facilmente utilizando o aplicativo `pip`.

Commando	descrição
install	Instalação de pacotes.
download	Download de pacotes.
uninstall	Desinstalação de pacotes.
freeze	Gera um arquivo `requirements` com os pacotes instalados.
list	Lista todos os pacotes instalados.
show	Mostra informação sobre um pacote instalado.
check	Verify installed packages have compatible dependencies.
search	Procura no PyPI por um pacote.
help	Mostra ajuda.

`pip`

O `pip` utiliza o site `https://pypi.org/` como repositório para instalação dos pacotes.

Laboratório Módulo 4