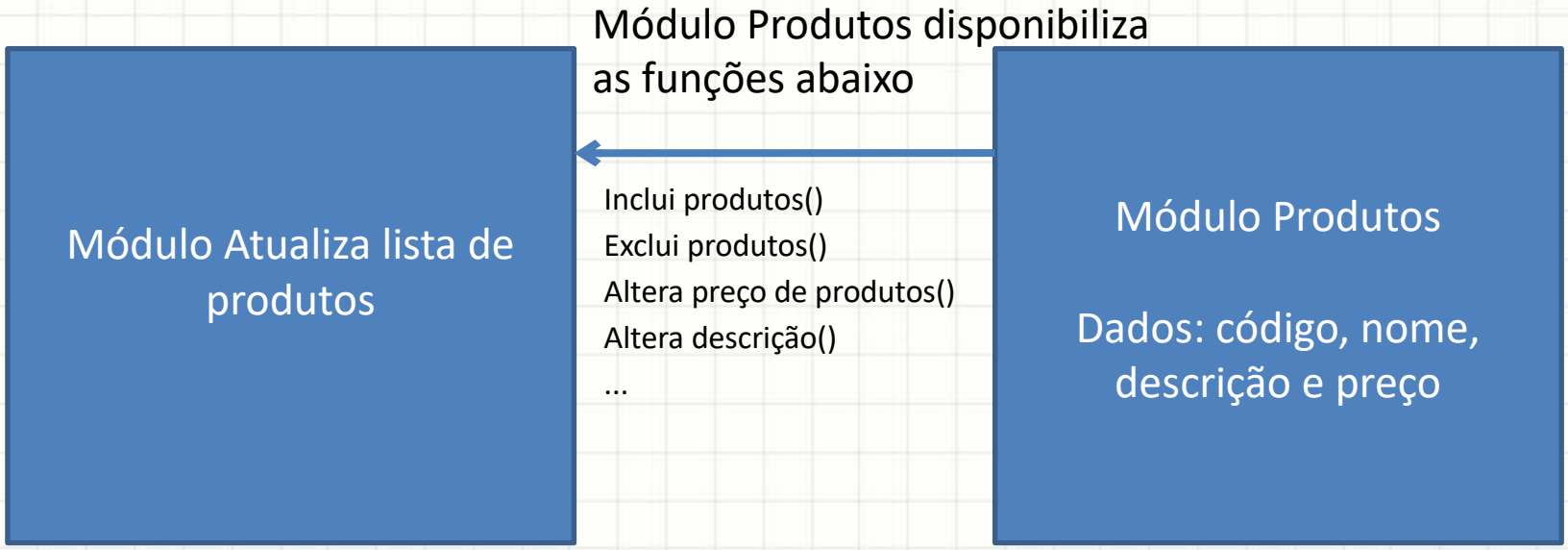
OVERVIEW DE PYTHON VOLTADO PARA MODULARIZAÇÃO

INF1301 – Programação Modular

Prof. Flavio Bevilacqua

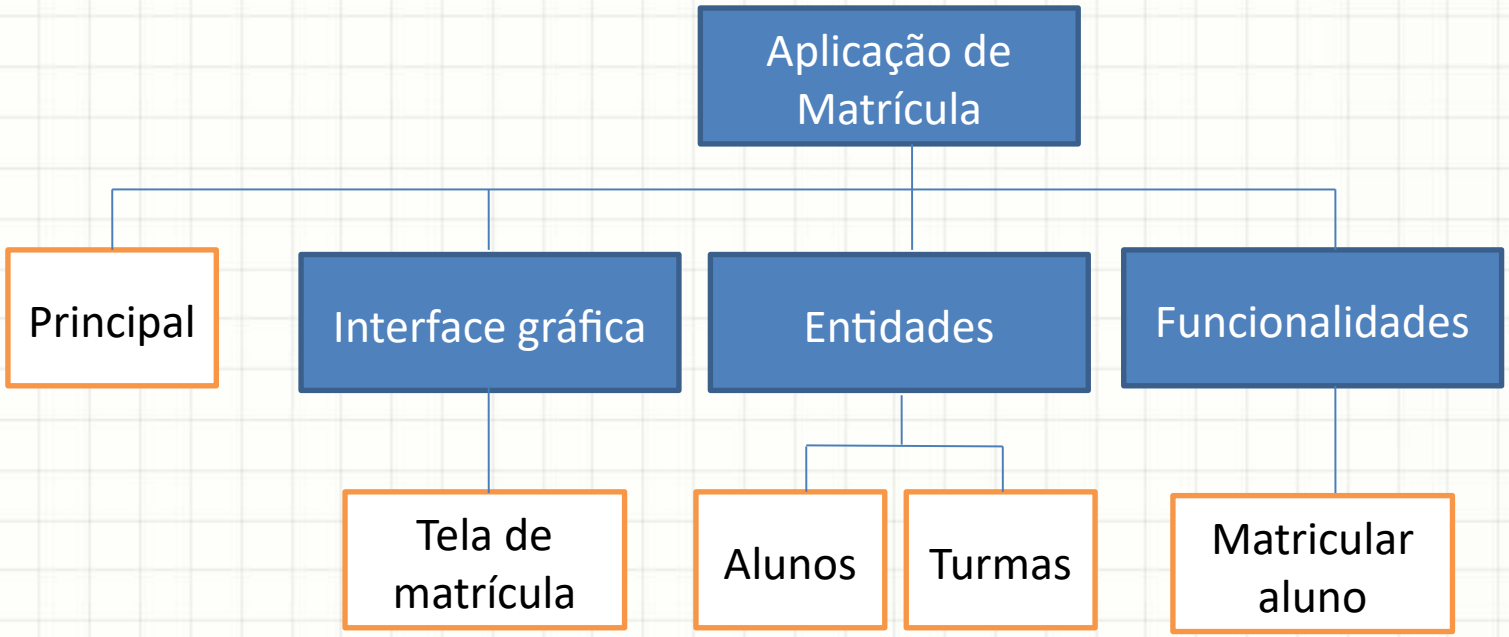
Módulos e Pacotes

Módulo contém uma abstração definida com os dados e as funções de acesso que manipulam estes dados de forma encapsulada



Módulos e Pacotes

Pacotes agrupam módulos em pastas acessadas com import



Abstrações e interfaces

professor.py (módulo Professor dentro do pacote entidades)

```
# Encapsulamento dos atributos e disponibilização apenas das funções de
acesso
__all__ = ["insere_professor","altera_professor",
"gera_relacao_professores","exclui_professor","consulta_professor"]

professores = []

def insere_professor(matricula, nome):
    matricula = matricula.strip()
    nome = nome.strip()
    if matricula == '' or nome == '':
        return 2
    for professor in professores:
        if professor['matricula'] == matricula:
            return 1
    novo_professor = {'matricula': matricula, 'nome': nome}
    professores.append(novo_professor)
    return 0
```

Abstrações e interfaces

```
def altera_professor(matricula, campo, valor):
    for professor in professores:
    if professor['matricula'] == matricula:
    professor[campo] = valor
    return 0
    return 1

def exclui_professor(matricula):
    for professor in professores:
    if professor['matricula'] == matricula:
    professores.remove(professor)
    return 0
    return 1
```

Abstrações e interfaces

```
def consulta_professor(matricula):
	for professor in professores:
		if professor['matricula'] == matricula:
			return professor
	return None

def gera_relacao_professores():
	relacao_professores = []
	for professor in professores:
		relacao_professores.append(professor)
	return relacao_professores
```

Proteção de dados e funções

__init__ - Contém o que será exportado dentro do pacote (pasta)

Exemplo:

__init__.py

```
from entidades.aluno import *
from entidades.criterio_aprovacao import *
from entidades.professor import *
from entidades.turma import *
from entidades.disciplina import *
```

Proteção de dados e funções

__all__ - Apresenta a lista de funções a serem disponibilizadas pelo módulo

Exemplo: professor.py

```
__all__ = ["insere_professor","altera_professor"]
```

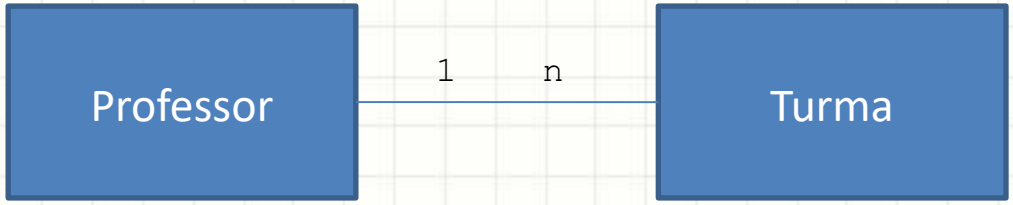
Armazenamento de Dados

Em um primeiro momento, os dados serão armazenados em uma lista de dicionários no módulo respectivo. Posteriormente os dados poderão serão persistidos em um base de dados.

```
aluno1 = {'matricula':20190202, 'nome':'andré', 'idade':20}
aluno2 = {'matricula':2014433, 'nome':'flavio', 'idade':22}
aluno3 = {'matricula':2015555, 'nome':'ana', 'idade':25}

lista_alunos = [aluno1, aluno2, aluno3]
```

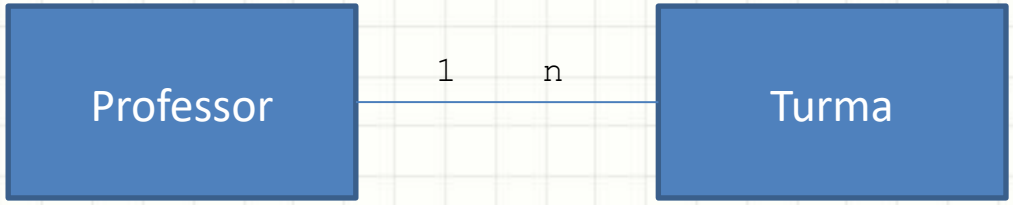
Relacionamento entre entidades




```
professor1 = {'matricula':'001', 'nome':'carlos'}
professor2 = {'matricula':'002', 'nome':'márcia'}

turma1 = {'codigo': '3WA', 'qtdAlunos':30, 'professor':'001'}
turma2 = {'codigo': '3WB', 'qtdAlunos':35, 'professor':'002'}
turma3 = {'codigo': '3WC', 'qtdAlunos':25, 'professor':'001'}
```

Relacionamento entre entidades




```
resultado_consulta =
[{'nome-professor':'carlos','turmas':
[{'codigo':'3WA','qtd-alunos':30},{'codigo':'3WC',
'qtd-alunos':25}]},
{'nome-professor':'márcia',
'turmas':
[{'codigo':'3WB','qtd-alunos':35}]}]

print (resultado_consulta)

print (resultado_consulta[0]['nome-professor'])

print (resultado_consulta[0]['turmas'][1]['codigo'])
```

PERSISTÊNCIA DE DADOS

INF1301 – Programação Modular

Prof. Flavio Bevilacqua

Definição

Persistir dados é a garantia de que um dado foi salvo e que poderá ser recuperado quando necessário no futuro

- Banco de Dados
- Arquivos

Persistência em Banco de Dados

Preparativos:

- Baixar e instalar o mysql
- Baixar e instalar o conector mysql para python (consultar no google mysql python connector)
- Criar o banco de dados
- Criar as tabelas

Persistência de Banco de Dados

- No nosso exemplo:

```
mysql –u root –p
use modular
create table alunos (
    matricula varchar(7) not null,
    nome      varchar(30),
    primary key(matricula)
    );
```

Conectando o banco de dados

```
import mysql.connector
from mysql.connector import Error

try:
    connection = mysql.connector.connect(host='localhost',
                                         database='Modular',
                                         user='root',
                                         password='root')
    if connection.is_connected():
        db_Info = connection.get_server_info()
        print("Conectado ao MySQL Server versão ", db_Info)
        cursor = connection.cursor()
        cursor.execute("select database()")
        record = cursor.fetchone()
        print("Conectado ao banco de dados ", record)
except Error as e:
    print("Erro de conexão", e)
finally:
    if (connection.is_connected()):
        cursor.close()
        connection.close()
        print("Conexão encerrada")
```

Inserindo dados na tabela

```
import mysql.connector
from mysql.connector import Error

try:
    connection = mysql.connector.connect(host='localhost',
                                         database='Modular',
                                         user='root',
                                         password='root')
    sql = """INSERT INTO Alunos (Matricula, Nome) VALUES ('2010801', 'Luiz')"""
    cursor = connection.cursor()
    cursor.execute(sql)
    connection.commit()
    print(cursor.rowcount, "Registro inserido")
    cursor.close()

except Error as e:
    print("Erro de conexão", e)
finally:
    if (connection.is_connected()):
        cursor.close()
        connection.close()
        print("Conexão encerrada")
```

Consultando dados na tabela

```
import mysql.connector
from mysql.connector import Error

try:
    connection = mysql.connector.connect(host='localhost',
                                         database='Modular',
                                         user='root',
                                         password='root')
    sql = """SELECT * FROM ALUNOS"""
    cursor = connection.cursor()
    cursor.execute(sql)
    rs = cursor.fetchall()
    print(rs)
    cursor.close()
except Error as e:
    print("Erro de conexão", e)
finally:
    if (connection.is_connected()):
        cursor.close()
        connection.close()
        print("Conexão encerrada")
```

tupla [('9010801','Luiz),('123456','Flavio')]

Consultando dados na tabela

```
matricula_ent = input("Digite uma matrícula:")
sql = """SELECT * FROM ALUNOS
WHERE Matricula = %s """
cursor = connection.cursor(prepared=True)
cursor.execute(sql,(matricula_ent,))
rs = cursor.fetchall()
print(rs)
if len(rs) == 0:
          print('Aluno não encontrado')
else:
          for aluno in rs:
                    print('Nome do aluno: ' + aluno[1])
cursor.close()
```

- Aspas triplas para conseguir colocar o comando SQL em várias linhas
- prepared=True para permitir instruções SQL parametrizadas

Persistência em arquivo xml

```
from xml.etree import ElementTree
from xml.etree.ElementTree import Element, SubElement, Comment
from xml.dom import minidom

funcionarios = Element('funcionarios')
comment = Comment('Dados de Funcionário')
funcionarios.append(comment)
funcionario = SubElement(funcionarios, 'funcionario')
nome = SubElement(funcionario, 'nome')
nome.text = 'José da Silva'
telefones = SubElement(funcionario, 'telefones')
telefone1 = SubElement(telefones, 'telefone')
telefone1.text = '(21)9111911'
telefone2 = SubElement(telefones, 'telefone')
telefone2.text = '(21)9111912'
telefone3 = SubElement(telefones, 'telefone')
telefone3.text = '(21)9111913'
nome_arquivo = 'funcionarios.xml'
with open(nome_arquivo, 'w') as file_object:
        file_object.write(formata_saida(funcionarios))
```

Gerando um arquivo xml

```
def formata_saida(elem):
    rough_string = ElementTree.tostring(elem, 'utf-8')
    reparsed = minidom.parseString(rough_string)
    return reparsed.toprettyxml(indent="  ")
```

```
<?xml version="1.0" ?>
<funcionarios>
  <!--Dados de Funcionário-->
  <funcionario>
    <nome>José da Silva</nome>
    <telefones>
      <telefone>(21)9111911</telefone>
      <telefone>(21)9111912</telefone>
      <telefone>(21)9111913</telefone>
    </telefones>
  </funcionario>
</funcionarios>
```

Lendo um arquivo xml para uma lista

```
from xml.etree import ElementTree

with open('funcionarios.xml', 'rt') as f:
    tree = ElementTree.parse(f)
    root = tree.getroot()
dict_funcionario = {}
lista_funcionario = []
for funcionario in root.findall('funcionario'):
          dict_funcionario = {}
          dict_funcionario['nome'] = funcionario.find('nome').text
          lista_telefones = []
          dict_telefone = {}
          for telefones in funcionario.iter('telefone'):
                    dict_telefone = {}
                    dict_telefone['numero telefone'] = telefones.text
                    lista_telefones.append(dict_telefone)
          dict_funcionario['telefones'] = lista_telefones
          lista_funcionario.append(dict_funcionario)
print(lista_funcionario)
```

Lendo um arquivo xml para uma lista

```
<?xml version="1.0" ?>
<funcionarios>
  <!--Dados de Funcionário-->
  <funcionario>
    <nome>José da Silva</nome>
    <telefones>
      <telefone>(21)9111911</telefone>
      <telefone>(21)9111912</telefone>
      <telefone>(21)9111913</telefone>
    </telefones>
  </funcionario>
  <funcionario>
    <nome>Maria</nome>
    <telefones>
      <telefone>(21)9111914</telefone>
      <telefone>(21)9111915</telefone>
      <telefone>(21)9111916</telefone>
    </telefones>
  </funcionario>
</funcionarios>
```

Lendo um arquivo xml para uma lista

```
[
{'telefones': [{'numero telefone': '(21)9111911'},
               {'numero telefone': '(21)9111912'},
               {'numero telefone': '(21)9111913'}],
  'nome': 'José da Silva'},
{'telefones': [{'numero telefone': '(21)9111911'},
               {'numero telefone': '(21)9111912'},
               {'numero telefone': '(21)9111913'}],
  'nome': 'Maria'}
]
```