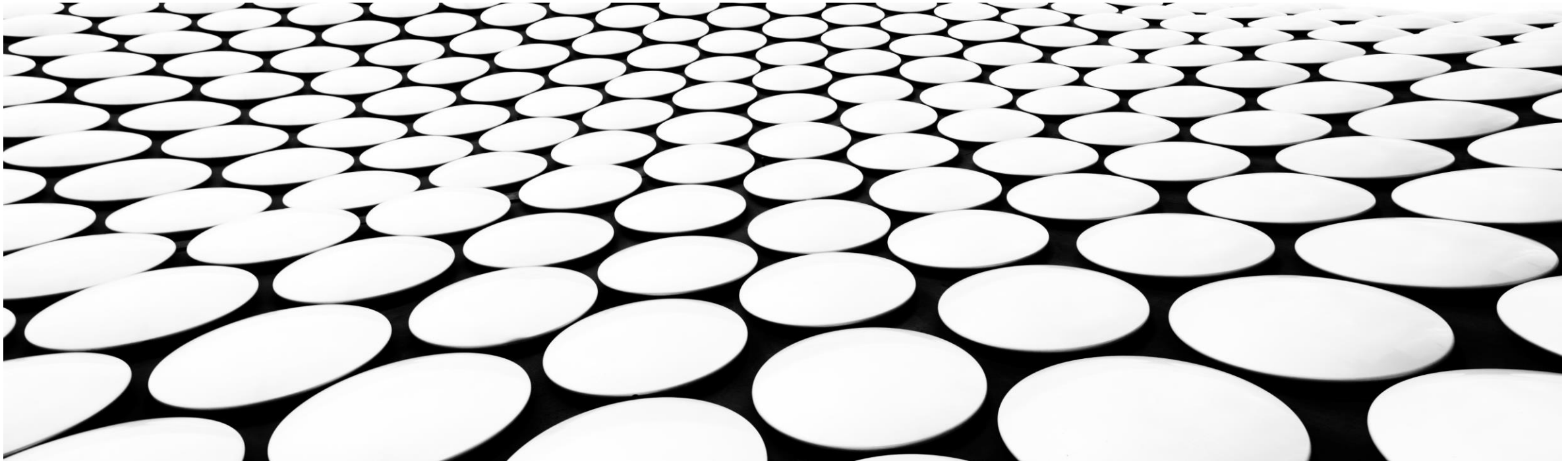
PROGRAMAÇÃO AVANÇADA

INFORMÁTICA 2023/2024

AULA 8



Python - enum

Enum é uma classe em python para criar enumerações, que são um conjunto de nomes simbólicos (membros) vinculados a valores únicos e constantes. Os membros de uma enumeração podem ser comparados por esses nomes simbólicos, e a própria enumeração pode ser iterada. Um enum tem as seguintes características.

- As enumerações são representações de string avaliáveis de um objeto também chamado repr().
- O nome do enum é exibido usando a 'name' palavra-chave.
- Usando type() podemos verificar os tipos de enumeração.

Exemplo:

```
import enum
# Using enum class create enumerations
class Days(enum.Enum):
   Sun = 1
   Mon = 2
   Tue = 3
# print the enum member as a string
print ("The enum member as a string is : ",end="")
print (Days.Mon)

# print the enum member as a repr
print ("he enum member as a repr is : ",end="")
print (repr(Days.Sun))
```

```
# Check type of enum member
print ("The type of enum member is : ",end ="")
print (type(Days.Mon))

# print name of enum member
print ("The name of enum member is : ",end ="")
print (Days.Tue.name)
```

Python - enum

- As enumerações são iteráveis . Eles podem ser iterados usando loops
- Enumerações suportam **hashing** . Enums podem ser usados em dicionários ou conjuntos.

```
import   enum
class   Animal(enum.Enum):
      dog =   1
      cat =   2
      lion =   3
print   ("All the enum values are : ")
for   Anim in   (Animal):
      print(Anim)
di =   {}
di[Animal.dog] =   'bark'
di[Animal.lion] =   'roar'
if   di=={Animal.dog : 'bark',Animal.lion : 'roar'}:
      print   ("Enum is hashed")
else:
      print   ("Enum is not hashed")
```

Output:

```
Todos os valores enum são:
Animal.dog
Animal.cat
Animal.lion
Enum está com hash
```

Modos de acesso: os membros Enum podem ser acedidos de duas maneiras:

1. Por valor : Neste método, o valor do membro enum é passado.
2. Por nome : Neste método, o nome do membro enum é passado.

Python - enum

Valores ou nomes separados também podem ser acedidos usando a palavra-chave “ nome ” ou “ valor ”.
Comparação: as enumerações suportam dois tipos de comparações

1. Identidade : São verificados com as palavras-chave “ é ” e “ não é ”.
2. Igualdade : Comparações de igualdade dos tipos “ == ” e “ ! = ” Também são suportadas.

```
import   enum

class   Animal(enum.Enum):
      dog =   1
      cat =   2
      lion =   3
print   ("The enum member associated with value 2 is : ",end="")
print   (Animal(2))
print   ("The enum member associated with name lion is : ",end="")
print   (Animal['lion'])
mem =   Animal.dog
print   ("The value associated with dog is : ",end="")
print   (mem.value)
print   ("The name associated with 1 is : ",end="")
print   (mem.name)
if   Animal.dog is   Animal.cat:
      print   ("Dog and cat are same animals")
else:
      print   ("Dog and cat are different animals")
if   Animal.lion !=   Animal.cat:
      print   ("Lions and cat are different")
else   :
      print   ("Lions and cat are same")
```

Output:

```
O membro enum associado ao valor 2 é: Animal.cat
O membro enum associado ao nome lion é: Animal.lion
O valor associado a cachorro é: 1
O nome associado a 1 é: cachorro
Cão e gato são animais diferentes
Leões e gatos são diferentes
```

Python – Tkinter (Exercícios)

Façam os seguintes exercícios utilizando a biblioteca Tkinter (modo gráfico):

1. Criem uma aplicação com menu que permita realizar as seguintes acções:
    a. Criar cliente (Número, nome e Morada)
    b. Altere dados do cliente (com excepção do número)
    c. Elimine cliente
    d. Criar produtos (Código, designação e preço)
    e. Registar compra do cliente (Número do cliente, Data, código do produto e preço)
    f. Mostrar histórico de compras realizadas pelo cliente, ordenado por data
    g. Escolhendo um cliente mostrar os dias da semana em que mais compra (usar o enum)
    h. Sair da aplicação
2. Criem um ficheiro com os gostos do cliente (cor, produto)
3. Alterem o programa para que a cor de fundo do ecrã seja a de gosto do cliente

Python com MySQL

O Python pode ser usado em aplicações que utilizem base de dados. Uma das bases de dados mais populares é o MySQL e é a que vamos utilizar para aprender todo o processo.

Install MySQL Driver

O Python precisa de um driver MySQL para aceder à base de dados MySQL. No nosso caso vamos usar o “mysql-connector-python”.
Devem instalar no PyCharm como fazem com qualquer outra biblioteca.
Para testar se a instalação foi bem-sucedida ou se você já possui o "MySQL Connector" instalado, crie uma página Python com o seguinte conteúdo:

```
import mysql.connector
```

Se o código acima foi executado sem erros, o "MySQL Connector" está instalado e pronto para ser usado.

Em seguida deveremos ir ao site: https://www.mysql.com/downloads/
Baixar a versão community do MySql e instalar no pc.

Python com MySQL

Criar uma ligação

Comecemos por criar uma conexão com a base de dados.

Use o nome do utilizador e password da sua base de dados MySQL:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)

print(mydb)
```

Python com MySQL

Criar uma Base de Dados

Comecemos por criar uma conexão com a base de dados.

Para criar uma base de dados no MySQL, use a instrução "CREATE DATABASE":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)

mycursor = mydb.cursor()
mycursor.execute("CREATE DATABASE IPT2024")
```

Python com MySQL

Vamos verificar se a BD foi bem criada, listando todas as bases de dados do sistema usando a instrução "SHOW DATABASES":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword"
)

mycursor = mydb.cursor()

mycursor.execute("SHOW DATABASES")

for x in mycursor:
  print(x)
```

Python com MySQL

Em alternativa podemos tentar aceder à base de dados quando fazemos a ligação:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="ipt2024"
)
```

Python com MySQL

Criar uma tabela

Para criar uma tabela no MySQL, use a instrução "CREATE TABLE".

Certifiquem-se de definir o nome da base de dados ao criar a conexão.

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="ipt2024"
)

mycursor = mydb.cursor()

mycursor.execute("CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))")
```

Python com MySQL

Verificar se a tabela existe

Para verificar se uma tabela existe no MySQL, use a instrução "SHOW TABLES".

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="ipt2024"
)

mycursor = mydb.cursor()

mycursor.execute("SHOW TABLES")

for x in mycursor:
  print(x)
```

Python com MySQL

Chave primária

Ao criar uma tabela, vocês também devem criar uma coluna com uma chave exclusiva para cada registro (na maioria das situações).

Isso pode ser feito definindo uma CHAVE PRIMÁRIA.

Usamos a instrução "INT AUTO_INCREMENT PRIMARY KEY" que irá inserir um número único para cada registro. Começando em 1 e aumentado em um para cada registro.

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database=" IPT2024"
)

mycursor = mydb.cursor()

mycursor.execute("DROP TABLE customers")
mycursor.execute(“CREATE TABLE customers (id INT AUTO_INCREMENT PRIMARY KEY, name VARCHAR(255), address VARCHAR(255))")
```

Python com MySQL

Chave primária

Caso a tabela já exista, usem o comando “ALTER TABLE”:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database=“ipt2024"
)

mycursor = mydb.cursor()

mycursor.execute(“ALTER TABLE customers ADD COLUMN id INT AUTO_INCREMENT PRIMARY KEY")
```

Python com MySQL

Inserir registos numa tabela

Para preencher uma tabela no MySQL, use a instrução "INSERT INTO".

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="ipt2024"
)

mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = ("John", "Highway 21")
mycursor.execute(sql, val)

mydb.commit()

print(mycursor.rowcount, "record inserted.")
```

Importante!: Observe a instrução: mydb.commit(). É necessário, caso contrário, nenhuma alteração será feita na tabela.

Python com MySQL

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="mydatabase"
)

mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES
(%s, %s)"
val = [
  ('Peter', 'Lowstreet 4'),
  ('Amy', 'Apple st 652'),
  ('Hannah', 'Mountain 21'),
  ('Michael', 'Valley 345'),
  ('Sandy', 'Ocean blvd 2'),
  ('Betty', 'Green Grass 1'),
  ('Richard', 'Sky st 331'),
  ('Susan', 'One way 98'),
  ('Vicky', 'Yellow Garden 2'),
  ('Ben', 'Park Lane 38'),
  ('William', 'Central st 954'),
  ('Chuck', 'Main Road 989'),
  ('Viola', 'Sideway 1633')
]

mycursor.executemany(sql, val)

mydb.commit()

print(mycursor.rowcount, "was inserted.")
```

Python com MySQL

Vocês podem obter o id da linha que acabaram de inserir perguntando ao objeto cursor.

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="ipt2024"
)

mycursor = mydb.cursor()

sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"
val = ("Michelle", "Blue Village")
mycursor.execute(sql, val)

mydb.commit()

print("1 record inserted, ID:", mycursor.lastrowid)
```

Python com MySQL

Seleccionando registos de uma Table

Para selecionar de uma tabela no MySQL, usem a instrução "SELECT":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database=" ipt2024"
)

mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

Nota: Usamos o método fetchall(), que busca todas as linhas da última instrução executada.

Python com MySQL

Seleccionando colunas

Para selecionar apenas algumas das colunas em uma tabela, usem a instrução "SELECT" seguida do(s) nome(s) da(s) coluna(s):

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database=" ipt2024"
)

mycursor = mydb.cursor()

mycursor.execute("SELECT name, address FROM customers")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

Python com MySQL

Seleccionando colunas

Se vocês estiverem interessados apenas em uma linha, podem usar o método fetchone().
O método fetchone() retornará a primeira linha do resultado:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database=" ipt2024"
)

mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchone()

print(myresult)
```

Python com MySQL

Seleccionando com filtros

Ao selecionar registros de uma tabela, vocês podem filtrar a seleção usando a instrução "WHERE":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database=" ipt2024"
)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address ='Park Lane 38'"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

Python com MySQL

Caracteres curinga

Vocês também podem selecionar os registos que começam, incluem ou terminam com uma determinada letra ou frase.
Usem o % para representar caracteres curinga:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database=" ipt2024"
)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers WHERE address LIKE '%way%'"

mycursor.execute(sql)
myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

Python com MySQL

Ordenando o resultado

Use a instrução ORDER BY para classificar o resultado em ordem crescente ou decrescente.

A palavra-chave ORDER BY classifica o resultado em ordem crescente por padrão. Para classificar o resultado em ordem decrescente, use a palavra-chave DESC.

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database=" ipt2024"
)

mycursor = mydb.cursor()
sql = "SELECT * FROM customers ORDER BY name"
mycursor.execute(sql)
myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

Python com MySQL

Ordenando o resultado

Use a palavra-chave DESC para classificar o resultado em ordem decrescente.

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database=" ipt2024"
)

mycursor = mydb.cursor()

sql = "SELECT * FROM customers ORDER BY name DESC"

mycursor.execute(sql)

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

Python com MySQL

Eliminando registos

Você pode excluir registos de uma tabela existente usando a instrução "DELETE FROM":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database=" ipt2024"
)

mycursor = mydb.cursor()
sql = "DELETE FROM customers WHERE address = 'Mountain 21'"
mycursor.execute(sql)
mydb.commit()

print(mycursor.rowcount, "record(s) deleted")
```

Observem a cláusula WHERE na sintaxe DELETE: A cláusula WHERE especifica quais registos devem ser excluídos. Se vocês omitirem a cláusula WHERE, todos os registos serão excluídos!

Python com MySQL

Apagar uma tabela

Vocês podem excluir uma tabela existente usando a instrução "DROP TABLE":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database=" ipt2024"
)

mycursor = mydb.cursor()

sql = "DROP TABLE customers"

mycursor.execute(sql)
```

Python com MySQL

Apagar uma tabela, apenas se existir

Se a tabela que vocês desejam excluir já foi excluída, ou por qualquer outro motivo não existe, vocês podem usar a palavra-chave IF EXISTS para evitar erros.

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database=" ipt2024"
)

mycursor = mydb.cursor()

sql = "DROP TABLE IF EXISTS customers"

mycursor.execute(sql)
```

Python com MySQL

Actualizar uma tabela

Vocês podem atualizar os registos existentes numa tabela usando a instrução "UPDATE":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database="ipt2024"
)

mycursor = mydb.cursor()

sql = "UPDATE customers SET address = 'Canyon 123' WHERE address = 'Valley 345'"

mycursor.execute(sql)

mydb.commit()

print(mycursor.rowcount, "record(s) affected")
```

Python com MySQL

Prevenir SQL INJECTION

É considerado uma boa prática escapar os valores de qualquer consulta, também em instruções de atualização.
Isso é para evitar injeções de SQL, que é uma técnica comum de hackers na Web para destruir ou usar indevidamente seu banco de dados.
O módulo mysql.connector usa o espaço reservado %s para valores de escape na instrução delete:

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database=" ipt2024"
)

mycursor = mydb.cursor()
sql = "UPDATE customers SET address = %s WHERE address = %s"
val = ("Valley 345", "Canyon 123")
mycursor.execute(sql, val)
mydb.commit()

print(mycursor.rowcount, "record(s) affected")
```

Python com MySQL

Limitar os resultados

Você pode limitar o número de registros retornados da consulta, usando a instrução "LIMIT":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database=" ipt2024"
)

mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers LIMIT 5")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

Python com MySQL

Limitar os resultados, começando de outra posição

Se vocês desejam retornar cinco registos, a partir do terceiro registo, vocês podem usar a palavra-chave "OFFSET":

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="yourusername",
  password="yourpassword",
  database=" ipt2024"
)

mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers LIMIT 5 OFFSET 2")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```