Aula 12

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Persistência de Dados
  - Arquivos Binários

Arquivos (conforme visto na Aula 9)

- Programas **interativos** leem os dados de entrada do teclado e apresentam os dados de saída na tela.
  - Este tipo de programação é utilizada quando poucos dados são processados ou quando necessitam de interação humana.

Arquivos (conforme visto na Aula 9)

- Programas **interativos** leem os dados de entrada do teclado e apresentam os dados de saída na tela.
  - Este tipo de programação é utilizada quando poucos dados são processados ou quando necessitam de interação humana.

- Quando grandes quantidades de dados são processadas, **arquivos** são utilizados para armazenar os dados de entrada e os de saída.
  - Estes são chamados, respectivamente, arquivos de entrada e arquivos de saída.

Arquivo Binário

- Há basicamente dois tipos de arquivo: texto e binário.

Consórcio cederj Fundação CECIERJ

Arquivo Binário

- Há basicamente dois tipos de arquivo: texto e binário.

- Um **arquivo binário** é uma sequência de bytes que reside em uma área de armazenamento (e.g., disco rígido, pen drive, CD/DVD) sob um mesmo nome.
  - Essencialmente, arquivos texto também armazenam informação como uma sequência de bytes. Porém, esses são diretamente associados a caracteres.

Arquivo Binário

- Há basicamente dois tipos de arquivo: texto e binário.

- Um **arquivo binário** é uma sequência de bytes que reside em uma área de armazenamento (e.g., disco rígido, pen drive, CD/DVD) sob um mesmo nome.
  - Essencialmente, arquivos texto também armazenam informação como uma sequência de bytes. Porém, esses são diretamente associados a caracteres.

- Arquivos binários podem ser criados, editados e seu conteúdo recuperado por programas que conhecem a estrutura dos dados nele armazenados.

Operações Básicas

Abertura: Abre e associa um arquivo a uma variável (por exemplo, **arq**), permitindo a manipulação de seu conteúdo. Operação realizada através da função arq = **open**(...)

Operações Básicas

Abertura: Abre e associa um arquivo a uma variável (por exemplo, **arq**), permitindo a manipulação de seu conteúdo. Operação realizada através da função arq = **open**(...)

Fechamento: Fecha o arquivo associado a uma variável, liberando o recurso. Operação realizada através do método arq.**close**()

Operações Básicas

Abertura: Abre e associa um arquivo a uma variável (por exemplo, **arq**), permitindo a manipulação de seu conteúdo. Operação realizada através da função arq = **open**(...)

Fechamento: Fecha o arquivo associado a uma variável, liberando o recurso. Operação realizada através do método arq.**close**()

Leitura: Copia parte ou todo o conteúdo de um arquivo para a memória principal. Operação realizada através do método arq.**read**(...)

Escrita: Copia dados presentes na memória principal para dentro do arquivo. Operação realizada através do método arq.**write**(...)

Operações Básicas

Abertura: Abre e associa um arquivo a uma variável (por exemplo, **arq**), permitindo a manipulação de seu conteúdo. Operação realizada através da função arq = **open**(...)

Fechamento: Fecha o arquivo associado a uma variável, liberando o recurso. Operação realizada através do método arq.**close**()

Leitura: Copia parte ou todo o conteúdo de um arquivo para a memória principal. Operação realizada através do método arq.**read**(...)

Escrita: Copia dados presentes na memória principal para dentro do arquivo. Operação realizada através do método arq.**write**(...)

Navegação: Reposiciona e consulta a localização do cursor de escrita/leitura. Operações realizadas, respectivamente, através dos métodos arq.**seek**(...) e arq.**tell**()

Abertura e Fechamento de Arquivo Binário

A operação de abertura associa um arquivo a uma variável (e.g., **arq**), permitindo a manipulação de seu conteúdo. Ela é realizada pela função

arq = **open(***nome*, *modo***)**

- O parâmetro **nome** é um String que indica o local onde o arquivo reside.
- O parâmetro **modo** é um String que define como o arquivo deverá ser aberto. Ele pode assumir os valores: “rb”, “wb”, “r+b”, “w+b” e “a+b”

Abertura e Fechamento de Arquivo Binário

A operação de abertura associa um arquivo a uma variável (e.g., **arq**), permitindo a manipulação de seu conteúdo. Ela é realizada pela função

arq = **open(***nome*, *modo***)**

- O parâmetro **nome** é um String que indica o local onde o arquivo reside.
- O parâmetro **modo** é um String que define como o arquivo deverá ser aberto. Ele pode assumir os valores: “rb”, “wb”, “r+b”, “w+b” e “a+b”

O método arq.**close()** fecha o arquivo e libera o recurso. Todo arquivo aberto DEVE ser fechado após o uso!

```
# Abertura de arquivo para leitura, seguido pelo fechamento
try:
    arq = open(“arquivo.bin”, “rb”)
    print(“O arquivo está aberto.”)
    arq.close()
except IOError:
    print(“Erro ao abrir o arquivo.”)
```

Modos de Abertura de Arquivo Binário

arq = **open(**nome, modo**)**

O parâmetro **modo** pode assumir os seguintes valores:

"rb": Abre um arquivo binário especificado para leitura. O arquivo deve existir previamente.

Modos de Abertura de Arquivo Binário

arq = **open(**nome, modo**)**

O parâmetro **modo** pode assumir os seguintes valores:

"rb": Abre um arquivo binário especificado para leitura. O arquivo deve existir previamente.

"wb": Abre um arquivo binário especificado para escrita. Caso o arquivo não exista, este será criado. Caso exista, seus dados serão sobrescritos por completo.

Modos de Abertura de Arquivo Binário

arq = **open(**nome, modo**)**

O parâmetro **modo** pode assumir os seguintes valores:

"rb": Abre um arquivo binário especificado para leitura. O arquivo deve existir previamente.

"wb": Abre um arquivo binário especificado para escrita. Caso o arquivo não exista, este será criado. Caso exista, seus dados serão sobrescritos por completo.

"r+b": Abre um arquivo binário especificado para leitura e escrita. O arquivo deve existir previamente, podendo seu conteúdo ser alterado.

Modos de Abertura de Arquivo Binário

arq = **open(**nome, modo**)**

O parâmetro **modo** pode assumir os seguintes valores:

"rb": Abre um arquivo binário especificado para leitura. O arquivo deve existir previamente.

"wb": Abre um arquivo binário especificado para escrita. Caso o arquivo não exista, este será criado. Caso exista, seus dados serão sobrescritos por completo.

"r+b": Abre um arquivo binário especificado para leitura e escrita. O arquivo deve existir previamente, podendo seu conteúdo ser alterado.

"w+b": Abre um arquivo binário especificado para escrita e leitura. Caso o arquivo não exista, este será criado. Caso exista, seus dados serão sobrescritos por completo.

Modos de Abertura de Arquivo Binário

arq = **open(**nome, modo**)**

O parâmetro **modo** pode assumir os seguintes valores:

"rb": Abre um arquivo binário especificado para leitura. O arquivo deve existir previamente.

"wb": Abre um arquivo binário especificado para escrita. Caso o arquivo não exista, este será criado. Caso exista, seus dados serão sobrescritos por completo.

"r+b": Abre um arquivo binário especificado para leitura e escrita. O arquivo deve existir previamente, podendo seu conteúdo ser alterado.

"w+b": Abre um arquivo binário especificado para escrita e leitura. Caso o arquivo não exista, este será criado. Caso exista, seus dados serão sobrescritos por completo.

"a+b": Abre um arquivo binário no modo *append* (concatenação). Caso o arquivo não exista, este será criado. Caso existam seus dados serão mantidos e o novo conteúdo será anexado ao fim do arquivo.

Abertura e Fechamento de Arquivo Binário

Utilize a diretiva **with-as** para simplificar o script e para garantir que o arquivo será fechado, mesmo em caso de exceção na manipulação

```
try:
    arq = open("arquivo.bin", "rb")
    try:
        # Faça alguma manipulação no arquivo
    finally:
        arq.close()
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```

... simplifica para ...

```
try:
    with open("arquivo.bin", "rb") as arq:
        # Faça alguma manipulação no arquivo
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```


Consórcio cederj Fundação CECIERJ

Leitura de Arquivo Binário

A operação de leitura é sequencial. Ela copia um bloco de bytes contido no arquivo, com início na posição atual do cursor de leitura/escrita, e move o cursor para o fim desse bloco. A leitura é realizada pelo método

bloco = arq.**read(***tamanho***)**

- O parâmetro **tamanho** é opcional. Quando não informado, resulta na leitura de todos os bytes, da posição atual até o fim do arquivo. Quando informado, resulta na leitura da quantidade de bytes indicada.

Leitura de Arquivo Binário

A operação de leitura é sequencial. Ela copia um bloco de bytes contido no arquivo, com início na posição atual do cursor de leitura/escrita, e move o cursor para o fim desse bloco. A leitura é realizada pelo método

bloco = arq.**read(***tamanho***)**

- O parâmetro **tamanho** é opcional. Quando não informado, resulta na leitura de todos os bytes, da posição atual até o fim do arquivo. Quando informado, resulta na leitura da quantidade de bytes indicada.

Utilize o método **unpack(***formato*, *bloco***)**, do módulo **struct**, para desempacotar o bloco de bytes na forma de tipos primitivos

Leitura de Arquivo Binário

A operação de leitura é sequencial. Ela copia um bloco de bytes contido no arquivo, com início na posição atual do cursor de leitura/escrita, e move o cursor para o fim desse bloco. A leitura é realizada pelo método

bloco = arq.**read(***tamanho***)**

- O parâmetro **tamanho** é opcional. Quando não informado, resulta na leitura de todos os bytes, da posição atual até o fim do arquivo. Quando informado, resulta na leitura da quantidade de bytes indicada.

Utilize o método **unpack(***formato*, *bloco***)**, do módulo **struct**, para desempacotar o bloco de bytes na forma de tipos primitivos

No caso de String, utilize o método **decode(***codificação***)** para converter o bloco de bytes em caracteres

Exemplos de Leitura de Arquivo Binário

```
# Leitura de todos os bytes contidos em um arquivo
try:
    with open("arquivo.bin", "rb") as arq:
        byte = arq.read(1)
        while byte != b"":
            # Faça alguma coisa com o byte lido
            byte = arq.read(1)
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```

```
import struct
# Leitura de um valor inteiro e um ponto flutuante com precisão dupla
try:
    with open("arquivo.bin", "rb") as arq:
        inteiro = struct.unpack("i", arq.read(4))[0]     # Note o acesso "[0]"
        real = struct.unpack("d", arq.read(8))[0]
        print("Valor inteiro:", inteiro, "Valor real:", real)
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```

Exemplos de Leitura de Arquivo Binário

```
# Leitura de um string binarizado que ocupa 16 bytes no arquivo
try:
    with open("arquivo.bin", "rb") as arq:
        bloco = arq.read(16)
        texto = bloco.decode("utf-8") # Conversão do bloco em String
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```

O parâmetro esperado pelo método **decode(***codificação***)** deve indicar qual o mapa de caracteres utilizado na conversão do bloco de bytes em String. Caso não seja informado, a codificação utilizada será "utf-8".

Dentre as dezenas de codificações disponíveis, junto à "utf-8", a codificação "ascii" é uma das mais comuns.

Escrita em Arquivo Binário

A operação de escrita é sequencial. Ela copia um bloco de bytes contido na memória principal (acessada por uma variável), para dentro de um bloco de mesmo tamanho no arquivo, com início na posição atual do cursor de leitura/escrita, e move o cursor para o fim desse bloco. A escrita é realizada pelo método

arq.**write(***bloco***)**

Escrita em Arquivo Binário

A operação de escrita é sequencial. Ela copia um bloco de bytes contido na memória principal (acessada por uma variável), para dentro de um bloco de mesmo tamanho no arquivo, com início na posição atual do cursor de leitura/escrita, e move o cursor para o fim desse bloco. A escrita é realizada pelo método

arq.**write(***bloco***)**

Utilize o método **pack(***formato*, *v1, v2, ...***)**, do módulo **struct**, para empacotar dentro de um bloco, conforme o formato especificado, os valores *v1*, *v2*, ...

Escrita em Arquivo Binário

A operação de escrita é sequencial. Ela copia um bloco de bytes contido na memória principal (acessada por uma variável), para dentro de um bloco de mesmo tamanho no arquivo, com início na posição atual do cursor de leitura/escrita, e move o cursor para o fim desse bloco. A escrita é realizada pelo método

arq.**write(***bloco***)**

Utilize o método **pack(***formato*, *v1, v2, ...***)**, do módulo **struct**, para empacotar dentro de um bloco, conforme o formato especificado, os valores *v1*, *v2*, ...

No caso de String, utilize o método **encode(***codificação***)** para converter os caracteres em um bloco de bytes

Exemplos de Escrita em Arquivo Binário

```
# Cópia do conteúdo de um arquivo binário para outro
with open("origem.bin", "rb") as entrada:
    with open("destino.bin", "wb") as saida:
        byte = entrada.read(1)
        while byte != b"":
            saida.write(byte)
            byte = entrada.read(1)
```

```
import struct
# Escrita de um valor inteiro e um ponto flutuante com precisão dupla
try:
    with open("arquivo.bin", "wb") as arq:
        bloco = struct.pack("i", 10)
        arq.write(bloco)
        bloco = struct.pack("d", 99.5)
        arq.write(bloco)
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```

Exemplo de Escrita em Arquivo Binário

```
# Escrita de um string binarizado no arquivo
try:
    with open("arquivo.bin", "wb") as arq:
        texto = "Sou aluno do CEDERJ"
        bloco = texto.encode("utf-8") # Conversão do String em bloco de bytes
        arq.write(bloco)
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```

Exemplo de Escrita em Arquivo Binário

```
# Escrita de um string binarizado no arquivo
try:
    with open("arquivo.bin", "wb") as arq:
        texto = "Sou aluno do CEDERJ"
        bloco = texto.encode("utf-8") # Conversão do String em bloco de bytes
        arq.write(bloco)
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```

O parâmetro esperado pelo método **encode(*codificação*)** deve indicar qual o mapa de caracteres utilizado na conversão de String para bloco de bytes. Caso não seja informado, a codificação utilizada será "utf-8".

Exemplo de Escrita em Arquivo Binário

```
# Escrita de um string binarizado no arquivo
try:
    with open("arquivo.bin", "wb") as arq:
        texto = "Sou aluno do CEDERJ"
        bloco = texto.encode("utf-8") # Conversão do String em bloco de bytes
        arq.write(bloco)
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```

O parâmetro esperado pelo método **encode(***codificação***)** deve indicar qual o mapa de caracteres utilizado na conversão de String para bloco de bytes. Caso não seja informado, a codificação utilizada será "utf-8".

Para evitar erros de interpretação de caracteres, o programador deve garantir que a mesma codificação seja utilizada tanto na escrita quanto na leitura do arquivo.

Deslocamento de Cursor de Leitura/Escrita

Ao abrir um arquivo em modo “rb”, “wb”, “r+b” ou “w+b”, o cursor de leitura/escrita é posicionado no primeiro byte do arquivo (endereço 0), enquanto que ao utilizar o modo “a+b”, o cursor é posicionado no byte seguinte ao último byte do arquivo existente.

Deslocamento de Cursor de Leitura/Escrita

Ao abrir um arquivo em modo “rb”, “wb”, “r+b” ou “w+b”, o cursor de leitura/escrita é posicionado no primeiro byte do arquivo (endereço 0), enquanto que ao utilizar o modo “a+b”, o cursor é posicionado no byte seguinte ao último byte do arquivo existente.

O cursor avança automaticamente *n* bytes a cada leitura ou escrita, onde *n* é a quantidade de bytes lidos ou escritos.

Deslocamento de Cursor de Leitura/Escrita

Ao abrir um arquivo em modo “rb”, “wb”, “r+b” ou “w+b”, o cursor de leitura/escrita é posicionado no primeiro byte do arquivo (endereço 0), enquanto que ao utilizar o modo “a+b”, o cursor é posicionado no byte seguinte ao último byte do arquivo existente.

O cursor avança automaticamente *n* bytes a cada leitura ou escrita, onde *n* é a quantidade de bytes lidos ou escritos.

O endereço atual do cursor é retornado pelo método arq.**tell()**.

Deslocamento de Cursor de Leitura/Escrita

Ao abrir um arquivo em modo “rb”, “wb”, “r+b” ou “w+b”, o cursor de leitura/escrita é posicionado no primeiro byte do arquivo (endereço 0), enquanto que ao utilizar o modo “a+b”, o cursor é posicionado no byte seguinte ao último byte do arquivo existente.

O cursor avança automaticamente *n* bytes a cada leitura ou escrita, onde *n* é a quantidade de bytes lidos ou escritos.

O endereço atual do cursor é retornado pelo método arq.**tell()**.

Utilize o método

arq.**seek(***endereço relativo*, *origem***)**

para posicionar o cursor no endereço desejado em relação à origem informada.

Deslocamento de Cursor de Leitura/Escrita

Ao abrir um arquivo em modo “rb”, “wb”, “r+b” ou “w+b”, o cursor de leitura/escrita é posicionado no primeiro byte do arquivo (endereço 0), enquanto que ao utilizar o modo “a+b”, o cursor é posicionado no byte seguinte ao último byte do arquivo existente.

O cursor avança automaticamente *n* bytes a cada leitura ou escrita, onde *n* é a quantidade de bytes lidos ou escritos.

O endereço atual do cursor é retornado pelo método arq.**tell()**.

Utilize o método

arq.**seek(***endereço relativo*, *origem***)**

para posicionar o cursor no endereço desejado em relação à origem informada. A origem é um parâmetro opcional que aceita os valores:

- **0**: início do arquivo (padrão, caso origem não seja informada)
- **1**: posição atual do cursor
- **2**: fim do arquivo

Exemplos de Navegação em Arquivo Binário

```
import struct
# Verificando o avanço do cursor de leitura/escrita
try:
    with open("arquivo.bin", "rb") as arq:
        pos = arq.tell()
        print("O cursor está no endereço", pos)
        x = struct.unpack("i", arq.read(4))[0]
        pos = arq.tell()
        print("Após ler 4 bytes, ele foi para o endereço", pos)
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```

```
# Verificando o tamanho do arquivo
try:
    with open("arquivo.bin", "rb") as arq:
        arq.seek(0, 2)
        tam = arq.tell()
        print("O arquivo possui", tam, "bytes")
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```

Exemplo de Navegação em Arquivo Binário

```
import struct
# Acesso randômico para leitura do conteúdo do arquivo
try:
    with open("arquivo.bin", "rb") as arq:
        print("Os 10 primeiros valores inteiros armazenados são")
        for x in range(0, 10):
            print(struct.unpack("i", arq.read(4))[0])
        print("Da posição atual, voltarei o cursor para reler os 5 últimos valores")
        arq.seek(-(4 * 5), 1)
        for x in range(0, 5):
            print(struct.unpack("i", arq.read(4))[0])
        print("Agora voltei para o início do arquivo para reler o primeiro valor")
        arq.seek(0)
        print(struct.unpack("i", arq.read(4))[0])
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```

Exemplo de Navegação em Arquivo Binário

```
import struct

# Subprograma para leitura dos "n" primeiros valores inteiros
def mostrarOsNPrimeirosValores(arq, n):
    print("Os primeiros valores contidos no arquivo são")
    arq.seek(0)
    for k in range(0, n):
        print(struct.unpack("i", arq.read(4))[0])

# Programa Principal para leitura e escrita com acesso randômico
try:
    with open("arquivo.bin", "w+b") as arq:
        print("Escrever os valores inteiro de 1 a 5 no arquivo")
        for x in range(1, 6):
            arq.write(struct.pack("i", x))
        mostrarOsNPrimeirosValores(arq, 5)
        print("Substituir o segundo valor inteiro contido no arquivo por 99")
        arq.seek(4)
        arq.write(struct.pack("i", 99))
        mostrarOsNPrimeirosValores(arq, 5)
except IOError:
    print("Erro ao abrir ou ao manipular o arquivo.")
```


Consórcio cederj Fundação CECIERJ

O Conceito de Registro

É conveniente organizar coleções de dados dentro de arquivos em unidades de tamanho fixo chamadas **registros**, mesmo que nem todo o espaço reservado para um registro seja efetivamente utilizado.

O Conceito de Registro

É conveniente organizar coleções de dados dentro de arquivos em unidades de tamanho fixo chamadas **registros**, mesmo que nem todo o espaço reservado para um registro seja efetivamente utilizado.

- Exemplo:

  Armazenar os dados de um grupo de alunos reservando, para cada aluno, *m* bytes do arquivo, isto é, um registro de *m* bytes.

O Conceito de Registro

É conveniente organizar coleções de dados dentro de arquivos em unidades de tamanho fixo chamadas **registros**, mesmo que nem todo o espaço reservado para um registro seja efetivamente utilizado.

- Exemplo:

  Armazenar os dados de um grupo de alunos reservando, para cada aluno, *m* bytes do arquivo, isto é, um registro de *m* bytes.

- Vantagens:
  - Podemos saber quantos registros existem no arquivo, ao tomar o tamanho do arquivo e dividi-lo pelo tamanho do registro.
  - Podemos calcular o endereço do início de cada registro utilizando

    $$endereço = chave\ do\ registro \times tamanho\ do\ registro$$

    e, assim, usar o método **seek(***endereço***)** para navegar entre os registros de maneira eficiente. A chave é um valor indexador que vai de zero à quantidade de registros, menos um.

O Conceito de Registro

Registros são compostos por **itens de dados**, também chamados de **campos**. É conveniente que os campos sejam preenchidos na mesma ordem em todo registro colocado em um arquivo.

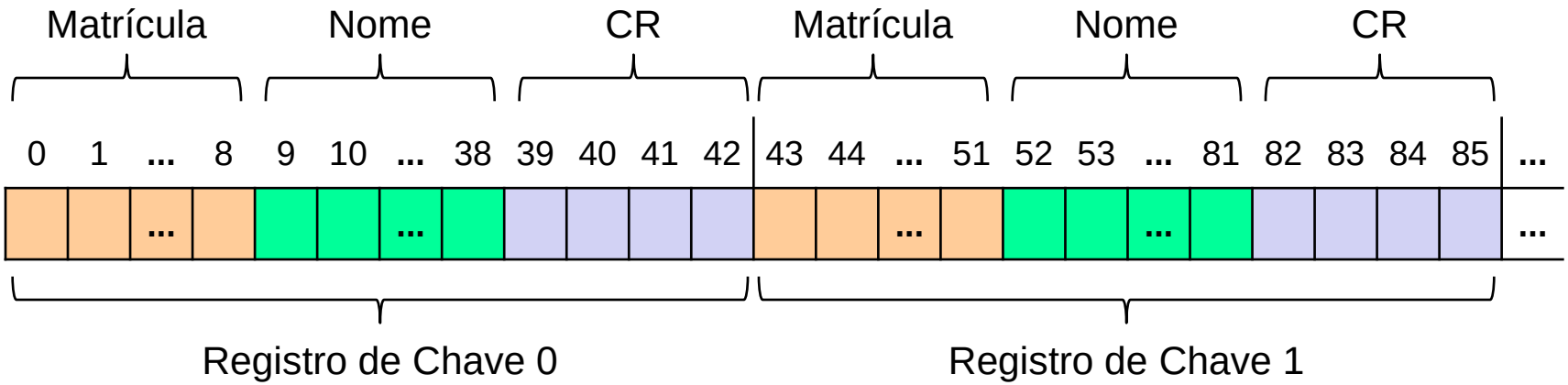
O Conceito de Registro

Registros são compostos por **itens de dados**, também chamados de **campos**. É conveniente que os campos sejam preenchidos na mesma ordem em todo registro colocado em um arquivo.

- Exemplo:

  O registro de cada aluno contém o código de matrícula (texto com exatamente 9 caracteres UTF-8; 9 bytes), o nome (texto com no máximo 30 caracteres UTF-8; 30 bytes) e o CR (valor float; 4 bytes).

  Logo, cada registro de aluno ocupa *m* = 43 bytes neste arquivo.



Exemplo de Escrita e Leitura de Registros

```
import struct
TAM = 43          # Constante definindo o tamanho do registro, em bytes

# Subprogramas
def escrever(arq, matricula, nome, cr):
    arq.write(matricula.encode("utf-8"))
    arq.write(nome[:30].ljust(30, chr(0)).encode("utf-8"))
    arq.write(struct.pack("f", cr))

def ler(arq):
    matricula = arq.read(9).decode("utf-8")
    nome = arq.read(30).decode("utf-8").rstrip(chr(0))
    cr = struct.unpack("f", arq.read(4))[0]
    return matricula, nome, cr

# Programa Principal
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)    # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)        # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)       # chave 2

    arq.seek(2 * TAM)
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```


Consórcio cederj Fundação CECIERJ

Exemplo de Escrita e Leitura de Registros

```
import struct
TAM = 43          # Constante definindo o tamanho do registro, em bytes

# Subprogramas
def escrever(arq, matricula, nome, cr):
    arq.write(matricula.encode("utf-8"))
    arq.write(nome[:30].ljust(30, chr(0)).encode("utf-8"))
    arq.write(struct.pack("f", cr))

def ler(arq):
    matricula = arq.read(9).decode("utf-8")
    nome = arq.read(30).decode("utf-8").rstrip(chr(0))
    cr = struct.unpack("f", arq.read(4))[0]
    return matricula, nome, cr

# Programa Principal
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)    # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)        # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)       # chave 2

    arq.seek(2 * TAM)
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```


Consórcio cederj Fundação CECIERJ

Exemplo de Escrita e Leitura de Registros

```
import struct
TAM = 43          # Constante definindo o tamanho do registro, em bytes

# Subprogramas
def escrever(arq, matricula, nome, cr):
    arq.write(matricula.encode("utf-8"))
    arq.write(nome[:30].ljust(30, chr(0)).encode("utf-8"))
    arq.write(struct.pack("f", cr))

def ler(arq):
    matricula = arq.read(9).decode("utf-8")
    nome = arq.read(30).decode("utf-8").rstrip(chr(0))
    cr = struct.unpack("f", arq.read(4))[0]
    return matricula, nome, cr

# Programa Principal
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)    # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)        # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)       # chave 2

    arq.seek(2 * TAM)
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```


Consórcio cederj Fundação CECIERJ

Exemplo de Escrita e Leitura de Registros

```
import struct
TAM = 43          # Constante definindo o tamanho do registro, em bytes

# Subprogramas
def escrever(arq, matricula, nome, cr):
    arq.write(matricula.encode("utf-8"))
    arq.write(nome[:30].ljust(30, chr(0)).encode("utf-8"))
    arq.write(struct.pack("f", cr))

def ler(arq):
    matricula = arq.read(9).decode("utf-8")
    nome = arq.read(30).decode("utf-8").rstrip(chr(0))
    cr = struct.unpack("f", arq.read(4))[0]
    return matricula, nome, cr

# Programa Principal
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)    # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)        # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)       # chave 2

    arq.seek(2 * TAM)
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```


Consórcio cederj Fundação CECIERJ

Exemplo de Escrita e Leitura de Registros

```
import struct
TAM = 43          # Constante definindo o tamanho do registro, em bytes

# Subprogramas
def escrever(arq, matricula, nome, cr):
    arq.write(matricula.encode("utf-8"))
    arq.write(nome[:30].ljust(30, chr(0)).encode("utf-8"))
    arq.write(struct.pack("f", cr))

def ler(arq):
    matricula = arq.read(9).decode("utf-8")
    nome = arq.read(30).decode("utf-8").rstrip(chr(0))
    cr = struct.unpack("f", arq.read(4))[0]
    return matricula, nome, cr

# Programa Principal
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)    # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)        # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)       # chave 2

    arq.seek(2 * TAM)
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```


Consórcio cederj Fundação CECIERJ

Forma Alternativa para Lidar com Registros

Python oferece mais de uma maneira elegante de empacotar e desempacotar dados para serem escritos ou lidos como registros de tamanho fixo em arquivos binários.

A mais simples é através de estruturas criadas via módulo **struct**:

NomeDaEstrutura = struct.**Struct(***formato***)**

Forma Alternativa para Lidar com Registros

Python oferece mais de uma maneira elegante de empacotar e desempacotar dados para serem escritos ou lidos como registros de tamanho fixo em arquivos binários.

A mais simples é através de estruturas criadas via módulo **struct**:

NomeDaEstrutura = struct.**Struct(***formato***)**

NomeDaEstrutura é uma variável que referencia um objeto Struct capaz de empacotar e desempacotar valores, conforme o formato especificado

formato é uma String que determina a sequência em que os dados serão empacotados pelo método **pack(*v1*, *v2*, ...)** ou desempacotados em uma tupla pelo método **unpack(*bloco*)**

Exemplo de Escrita e Leitura de Registros

```
import struct

Aluno = struct.Struct("9s 30s f")   # Criar Struct com o formato do registro

#Subprogramas
def escrever(arq, matricula, nome, cr):
    bloco = Aluno.pack(matricula.encode("utf-8"), nome.encode("utf-8"), cr)
    arq.write(bloco)

def ler(arq):
    bloco = arq.read(Aluno.size)
    campos = Aluno.unpack(bloco)
    return campos[0], campos[1].decode("utf-8").rstrip(chr(0)), campos[2]

# Programa Principal
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)     # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)         # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)        # chave 2

    arq.seek(2 * Aluno.size)
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```


Consórcio cederj Fundação CECIERJ

Exemplo de Escrita e Leitura de Registros

```
import struct

Aluno = struct.Struct("9s 30s f")   # Criar Struct com o formato do registro

#Subprogramas
def escrever(arq, matricula, nome, cr):
    bloco = Aluno.pack(matricula.encode("utf-8"), nome.encode("utf-8"), cr)
    arq.write(bloco)

def ler(arq):
    bloco = arq.read(Aluno.size)
    campos = Aluno.unpack(bloco)
    return campos[0], campos[1].decode("utf-8").rstrip(chr(0)), campos[2]

# Programa Principal
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)     # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)         # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)        # chave 2

    arq.seek(2 * Aluno.size)
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```


Consórcio cederj Fundação CECIERJ

Exemplo de Escrita e Leitura de Registros

```
import struct

Aluno = struct.Struct("9s 30s f")   # Criar Struct com o formato do registro

#Subprogramas
def escrever(arq, matricula, nome, cr):
    bloco = Aluno.pack(matricula.encode("utf-8"), nome.encode("utf-8"), cr)
    arq.write(bloco)

def ler(arq):
    bloco = arq.read(Aluno.size)
    campos = Aluno.unpack(bloco)
    return campos[0], campos[1].decode("utf-8").rstrip(chr(0)), campos[2]

# Programa Principal
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)     # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)         # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)        # chave 2

    arq.seek(2 * Aluno.size)
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```


Consórcio cederj Fundação CECIERJ

Exemplo de Escrita e Leitura de Registros

```
import struct

Aluno = struct.Struct("9s 30s f")   # Criar Struct com o formato do registro

#Subprogramas
def escrever(arq, matricula, nome, cr):
    bloco = Aluno.pack(matricula.encode("utf-8"), nome.encode("utf-8"), cr)
    arq.write(bloco)

def ler(arq):
    bloco = arq.read(Aluno.size)
    campos = Aluno.unpack(bloco)
    return campos[0], campos[1].decode("utf-8").rstrip(chr(0)), campos[2]

# Programa Principal
with open("arquivo.bin", "w+b") as arq:
    escrever(arq, "213031001", "Alessandro", 5.4)    # chave 0
    escrever(arq, "114031012", "Dayana", 8.3)        # chave 1
    escrever(arq, "214031173", "Gustavo", 7.2)       # chave 2

    arq.seek(2 * Aluno.size)
    matricula, nome, cr = ler(arq)
    print("Matricula:", matricula, "Nome:", nome, "CR:", cr)
```

Faça os Exercícios Relacionados a essa Aula

Clique no botão para visualizar os enunciados:

Aula 12

Professores:

Dante Corbucci Filho
Leandro A. F. Fernandes

Conteúdo:

- Persistência de Dados
  - Arquivos Binários

Consórcio cederj
Fundação CECIERJ