



Powered by 



Aprofundamento no Python - Aula 1

- Conhecendo o projeto que vamos construir
- Criar o projeto e utilizar versionamento
- Conhecer tipos complexos: Listas e Dicionários
- Manipulação de strings e arquivos

Conhecendo o projeto

Estamos abrindo novas vagas para cursos online de programação,
houveram inscrições de pessoas de todo o brasil.

Recebemos arquivos com os dados dos inscritos de todo o brasil e para
facilitar o gerenciamento de todos os inscritos **precisamos desenvolver**
um sistema que irá auxiliar os na gestão do curso.



Especificações

- Receberemos um arquivo de extensão .csv com os dados dos inscritos.
- Precisamos armazenar os dados em um banco de dados relacional
- Não poderão serem salvas, pessoas com CPF inválido
- Precisamos disponibilizar API's para que nosso sistema se comunique com o mundo externo.

API's disponibilizadas:

- **Listar todos os participantes**
- Deletar um participante
- Alterar dados de um participante
- Adicionar um novo participante
- Detalhar um participante



E agora?

- Criar projeto no github
- Fazer o clone do projeto no seu computador
- Abrir o pycharm com o projeto



E agora?

Analizar o arquivo CSV que contém os dados de matrícula e responder:

- Os dados encontrados na planilha são referentes a que? Como é o formato do arquivo?
- Para armazenar estes dados, precisamos criar quantas tabelas?
- O que vamos precisar conhecer para conseguir colocar esse arquivo pra dentro da nossa aplicação python?



E agora?

Analizar o arquivo CSV que contém os dados de matrícula e responder:

- Os dados encontrados na planilha são referentes a que? Como é o formato do arquivo?

Resposta: Planilha contém dados dos alunos inscritos no curso, possui um cabeçalho e muitas linhas com os registros dos alunos

- Para armazenar estes dados, precisamos criar quantas tabelas?

Resposta: Pessoa, Endereço e Curso

- O que vamos precisar conhecer para conseguir colocar esse arquivo pra dentro da nossa aplicação python?

Resposta: Manipulação Strings, Listas, Dicionários, Manipulação de arquivos, Conectar python com banco de dados, API



Sim! Ainda vamos precisar aprender vários conteúdos até conseguir de fato implementar tudo o que precisamos.





Manipulação de Strings

Strings - Símbolos

Símbolo	Significado	Exemplo	Resultado
+	Concatenação	“bolinho de ” + “chuva ”+ “quero”	“bolinho de chuva quero”
*	Repetição	“Maria”*4	“MariaMariaMariaMaria”
[]	Indexação	“chuva”[1]	“h”
[:]	Fatiamento	“chuva”[0:1]	“c”
{}	Substituição	“chuv{}”.format(“inha”)	“chuvinha”
in	Verificação	“chuv” in “chuvinha”	True
not in	Verificação	“chuva” not in “chuvinha”	True



Strings - Funções

Função	Parâmetro	Aplicação	Resultado
upper	nenhum	“bolinho de chuva”.upper()	BOLINHO DE CHUVA
lower	nenhum	“BOLINHO DE CHUVA”.lower()	“bolinho de chuva”
capitalize	nenhum	“bolinho de chuva”.capitalize()	“Bolinho de chuva”
count	item	“bolinho de chuva”.count(“h”)	2
replace	(old, new)	“bolinho de chuva”.replace(“a”, “inha”)	“bolinho de chuvinha”
find	item	“bolinho de chuva”.find(“chu”)	1
len	item	len(“bolinho de chuva”)	16



Exercícios em Python [Strings-1]

Você é o responsável por gerar os certificados dos da trilha financeira da serasa, porém, sabemos que eventualmente os participantes não cooperam e escrevem seus nomes sem seguir a norma padrão, apenas com a primeira letra maiúscula. Para evitar certificados despadronizados, você resolveu escrever um programinha em Python para normalizar os nomes para você!



Exercícios em Python [Strings-1]

```
nome = input("Digite o nome do participante:")
print(f"{nome.capitalize()}")
```

```
Digite o nome do participante:aliniiiii
Aliniiiii
```





Listas

Listas

É uma sequência de valores organizados entre colchetes [].

Esses dados podem ser de diferentes tipos:
inteiros, floats, strings e inclusive outras listas.

Exemplos:

dezenas = [10, 20, 30, 40]

pessoas= ['Alini', 'Pri', 'Lu', 'Jessica','Suzi']

lista_vazia = []



Listas

Os elementos de uma lista podem ser acessados pelo índice:

```
cores = ['amarelo', 'azul', 'branco', 'dourado']
```

Lembrando que a primeira posição de uma lista em Python é a posição 0:

```
>>> cores = ['amarelo', 'azul', 'branco', 'dourado']
>>> cores[0]
'amarelo'
>>> cores[2]
'branco'
>>
```



Listas

Além de ser possível **acessar** cada elemento da lista pelo índice, é possível, **inserir, alterar e deletar**.

Listas são **mutáveis**, ou seja, é possível alterar seus elementos: **adicionando, removendo e/ou substituindo-os**.



Listas

Formato da função	Explicação	Exemplo
lista.append(exemplo)	Adiciona um novo elemento ao final da lista.	inteiros = [1, 2, 3] inteiros.append(10)
lista.insert(posição, elemento)	Adiciona um novo elemento numa determinada posição da lista. Todos os elementos daquela posição em diante são deslocados uma posição para frente.	disciplinas = ["SO", "IA"] disciplinas.insert(0, "CAP")
lista.remove(elemento)	Remove o primeiro elemento especificado encontrado na lista.	nomes = ["Ana", "Bia"] nomes.remove("Ana")
lista.count(elemento)	Conta quantas vezes o elemento especificado aparece na lista. Retorna zero se o elemento não pertencer à lista.	frutas = ["maçã", "uva"] frutas.count("uva")



Listas

Formato da função	Explicação	Exemplo
lista.index(elemento)	Retorna a posição da primeira ocorrência do elemento especificado. Caso não seja encontrado, um erro será lançado.	tamanhos = ["M", "GG"] tamanhos.index("GG")
lista.pop(posição)	Remove e retorna o elemento da posição especificada. Caso não seja especificada uma posição, removerá o último elemento da lista.	Inteiros = [5, 6, 7, 8] inteiros.pop(2)
lista.sort()	Caso os elementos tenham implementação do operador menor (<) e sejam comparáveis (normalmente, de mesmo tipo), Python oferece uma função pronta para ordenar seu conteúdo. Caso os elementos não sejam comparáveis, um erro será lançado.	cores = ["lilás", "azul"] cores.sort()



Listas

Formato da função	Explicação	Exemplo
lista.reverse()	Inverte as posições dos elementos de uma lista. O último passa a ser o primeiro, o penúltimo passa a ser o segundo e assim por diante.	coffee = ["bolo", "refri"] coffee.reverse()
list("objeto iterável")	Cria uma lista a partir de um objeto iterável, por exemplo, uma string. Se não for objeto iterável, um erro será lançado	palavra = "PyLadies" list(palavra)
len(lista)	Retorna o tamanho de uma lista. Funciona para qualquer objeto iterável. Se não for um objeto iterável, um erro será lançado.	ladies = ["Nath", "Karol"] len(ladies)
elemento in lista	Verifica se um elemento está presente em uma lista.	trilhas = ["IA", "web"] "IA" in trilhas



Listas

Exemplo de como utilizar algumas das funções apresentadas:

```
coffee = ['refri', 'bolinha de queijo', 'bolo', 'coxinha']
coffee.append('café')
coffee.insert(0, 'água')
coffee.remove('refri')
```

Resultado esperado:

```
>>['água', 'bolinha de queijo', 'bolo', 'coxinha', 'café']
```



Iterando uma lista:

```
lista = [1, 2, 3, 4, 10]
for numero in lista:
    print(numero ** 2)
```

```
1
4
9
16
100
```

Se aplica a strings
também:

```
palavra = "casa"
for letra in palavra:
    print(letra)
```

```
c
a
s
a
```



Exercícios em Python [Listas-1]

Possuímos uma lista de cinco desenvolvedoras que participam do programa Maria vai com as devs.

Construa um programa que descubra quantas delas possuem o nome começando com a letra “A”.

As desenvolvedoras se chamam: Andorinha, Azaléia, Amélia, Margarida e Rosa.



Exercícios em Python [Listas-1]

```
devs = ['amelia', 'azaleia', 'andorinha', 'margarida', 'rosa']
nome_com_a = 0

for nome in devs:
    if nome[0] == 'a':
        nome_com_a += 1

print(f"Foram encontradas {nome_com_a} desenvolvedoras que começam com a letra A")
```

```
Foram encontradas 3 desenvolvedoras que começam com a letra A
```



Exercícios em Python [Listas-2]

Leia um número N e, em seguida, leia N números inteiros. Imprima o maior e o menor entre os N inteiros lidos.

Exemplo de entrada: 10

10 -5 1 2 3 100 4 5 -17 1

Saída esperada: Maior: 100

Menor: -17



Exercícios em Python [Listas-2]

```
n = int(input("Digite a quantidade de números que deseja processar:"))
maior = -1000000
menor = 1000000
entrada = map(int, input(f"Digite {n} números, separado por vírgula:").split(","))

for atual in entrada:
    menor = min(menor, atual)
    maior = max(maior, atual)

print(f"Maior: {maior}")
print(f"Menor: {menor}")
```

Digite a quantidade de números que deseja processar:10

Digite 10 números, separado por vírgula:10, -5, 1, 2, 3, 100, 4, 5, -17,1

Maior: 100

Menor: -17



Exercícios em Python [Listas-3]

O RH da Serasa precisa controlar a presença dos membros nas atividades do MVCAD, isto é, precisa anotar na planilha de frequência todos os que participaram.

Como você está estudando Python, se voluntariou a ajudar a tornar essa tarefa um pouco mais fácil, você pretende entregar a lista de presentes já ordenada alfabeticamente para o RH!

Dado um número N, que representa quantas pessoas participaram, e N nomes, imprima os N nomes ordenados alfabeticamente.

Exemplo de entrada:

4

Suzi

Pri

Alini

Lu

Exemplo de entrada:

Alini

Lu

Pri

Suzi



Exercícios em Python [Listas-3]

```
presentes = int(input("Quantas pessoas estão presentes?"))
pessoas = []
for i in range(presentes):
    pessoas.append(input("Digite o nome do participante:"))
pessoas.sort()
print("Participantes por ordem alfabética:")
print("\n".join(pessoas))
```

```
Quantas pessoas estão presentes?4
Digite o nome do participante:Suzi
Digite o nome do participante:Alini
Digite o nome do participante:Lu
Digite o nome do participante:Pri
Participantes por ordem alfabética:
Alini
Lu
Pri
Suzi
```





Dicionários

Dicionários

Dicionário é um objeto que relaciona uma chave a um valor.

A **chave** é o que **indexa** um dicionário, isto é, ela corresponde ao conceito de índice. Desta forma, em vez de ter posições 0, 1, 2, 3 e assim por diante, podemos ter posições com os valores que desejamos, como “azul”, “amarelo” e “vermelho”.

Por se tratarem de índices, uma consequência direta é que as chaves de um dicionários precisam ser únicas, e, caso você declare mais de uma vez a mesma chave, o valor será sobreposto.



Dicionários

```
dicionario_vazio = {}
camiseta = {"M":12, "G": 10, "GG":10}

dados = {
    "grupo": "PyLadies São Carlos" ,
    "fundação":2014,
    "linguagem": "Python",
    "membros": 37
}

alunas_curso = {"Ana": "BCC", "Amanda": "EnC"}
alunas_curso["Lúcia"] = "BCC"
```



Dicionários

Para verificar se uma chave já existe, utilize o operador in:

```
>>> dic = {'chave': 'valor'}  
>>> 'chave' in dic  
True  
>>> 'chave_inexistente' in dic  
False  
>>>
```

Para deletar uma chave e seu valor de um dicionário, utilize o operador del:

```
>>> del(dic['chave'])  
>>> dic  
{}  
>>> _
```



Dicionários

Podemos obter todas as chaves de um dicionário com o método `keys()`:

```
dicionario.keys()
```

E também podemos obter todos os valores de um dicionário com o método `values()`:

```
dicionario.values()
```



Dicionários - Funções

Formato da função	Explicação	Exemplo
<code>dicionario.clear()</code>	Deleta todo o conteúdo do dicionário e retorna um dicionário vazio.	<code>mvcad= {"nome": "Ana"} mvcad.clear()</code>
<code>dicionario.copy()</code>	Retorna um novo dicionário, com uma cópia real do objeto. Se você fizer <code>dict1 = dict2</code> , ele criará apenas uma referência.	<code>mvcad= {"nome": "Ana"} mvcad.copy()</code>
<code>dicionario.fromkeys(objeto_iterável, valor_default)</code>	Cria um novo dicionário com as chaves contidas no objeto_iterável e valores baseadas no valor_default.	<code>dict().fromkeys([1, 2, 3, 4], None)</code>



Dicionários - Funções

Formato da função	Explicação	Exemplo
<code>dicionario.get(chave)</code>	Retorna o valor de uma dada chave, ou None caso ela não exista. É aconselhável o uso em vez de acesso direto em casos onde uma chave possa não existir, pois no acesso direto causaria erro.	<code>mvcad= {"nome": "Alini", "estado": "SC"} mvcad.get('nome')</code>
<code>dicionario.items()</code>	Retorna um objeto dict_items, que é composto por uma tupla que contém as chaves e outra que contém os valores. Útil em iterações para pegar chave e valor.	<code>mvcad= {"nome": "Alini", "estado": "SC"} mvcad.items()</code>
<code>dicionario.keys()</code>	Retorna um objeto dict_keys, que é composto por uma lista com as chaves.	<code>mvcad= {"nome": "Alini", "estado": "SC"} mvcad.keys()</code>



Dicionários - Funções

Formato da função	Explicação	Exemplo
dicionario.pop(chave)	Retorna o valor de uma dada chave e exclui todo o elemento do dicionário.	mvcad= {"nome": "Ana"} mvcad.pop('nome')
dicionario.popitem()	Retorna um elemento do dicionário e exclui ele do dicionário. Não é possível escolher o elemento que será retornado.	mvcad= {"nome": "Ana"} mvcad.popitem()
dicionario.update(dicionario)	Atualiza elementos existentes ou adiciona elementos caso estes não existam num dicionário base.	mvcad= {"nome": "Ana"} mvcad.update()



Dicionários

Para iterar sobre dicionários usando a chave como acesso:

```
for chave in dicionario:  
    dicionario[chave]
```



Recapitulando:

- Conhecemos o projeto que vamos construir
- Manipulação de Strings
- Listas
- Dicionários



E agora?

Analisar o arquivo CSV que contém os dados de matrícula e responder:

- Os dados encontrados na planilha são referentes a que? Como é o formato do arquivo?
Resposta: Planilha contém dados dos alunos inscritos no curso, possui um cabeçalho e muitas linhas com os registros dos alunos
- Para armazenar estes dados, precisamos criar quantas tabelas?
Resposta: Pessoa, Endereço e Curso
- O que vamos precisar conhecer para conseguir colocar esse arquivo pra dentro da nossa aplicação python?

Resposta: ~~Manipulação Strings, Listas, Dicionários,~~

Manipulação de arquivos, Conectar python com banco de dados, API



Manipulando arquivos de texto

Podemos abrir um arquivo de duas maneiras:

- Para somente leitura ('r')
- Com permissão de escrita ('w')

```
#  
# leitura  
#  
f = open('nome-do-arquivo', 'r')  
  
#  
# escrita  
#  
f = open('nome-do-arquivo', 'w')
```

Manipulando arquivos de texto

Se não especificarmos o segundo parâmetro, a forma padrão leitura ('r') será utilizada:

```
>>> arquivo = open('nome-do-arquivo')
>>> arquivo
<_io.TextIOWrapper name='nome-do-arquivo.text' mode='r' encoding='UTF-8'>
```

O terceiro parâmetro é opcional e nele especificamos a codificação do arquivo:

```
arquivo = open(nome-do-arquivo, 'r', encoding="utf8")
```



Manipulando arquivos de texto

Se tentarmos abrir um arquivo para leitura que não existe, um erro será lançado:

```
>>> f = open('nome-errado.text', 'r')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    FileNotFoundError: [Errno 2] No such file or directory: 'nome-errado.text'
```

Se tentarmos abrir um arquivo para escrita que não existe, então ele será criado, porém, se ele já existir, todo seu conteúdo será apagado no momento em que abrimos o arquivo.

Devemos sempre fechar o arquivo aberto: `arquivo.close()`

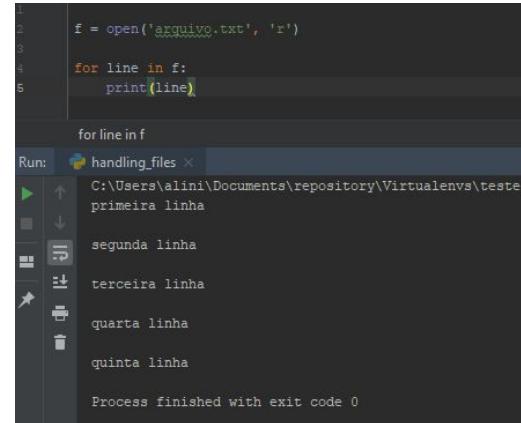


Exemplos

Como exemplo utilizaremos o arquivo de texto **seu- arquivo.text** que possui o seguinte conteúdo:

primeira linha
segunda linha
terceira linha
quarta linha
quinta linha

Podemos abrir um arquivo e iterar por cada linha conforme exemplo abaixo:



```
f = open('arquivo.txt', 'r')

for line in f:
    print(line)

for line in f
```

The screenshot shows a code editor with a Python script. The script opens a file named 'arquivo.txt' in read mode ('r'). It then uses a for loop to iterate over the file object 'f'. Inside the loop, it prints each line using the 'print' function. After the first iteration, the script tries to enter the loop again, which is commented out with a '#'. Below the code editor, the terminal window shows the output of the script: 'primeira linha', 'segunda linha', 'terceira linha', 'quarta linha', and 'quinta linha'. At the bottom of the terminal, it says 'Process finished with exit code 0'.



Exemplos

Se quisermos ler todo o conteúdo do arquivo em uma única string podemos utilizar a função `read()`:

```
>>> f = open('seu-arquivo.text', 'r')
>>> f.read()
'primeira linha\nsegunda linha\nterceira linha\nquarta linha\nquinta linha'
```

A função retornará uma lista vazia [] quando encontrar o final do arquivo (após a última linha ter sido lida).

```
>>> f = open('seu-arquivo.text', 'r')
>>> f.readline()
'primeira linha\n'
>>> f.readline()
'segunda linha\n'
>>> f.readline()
'terceira linha\n'
>>> f.readline()
'quarta linha\n'
>>> f.readline()
'quinta linha'
>>> f.readline()
''
```



Exemplos

Se quisermos ler todas linhas restantes em uma lista podemos utilizar a função readlines (estamos no plural).

```
>>> f = open('seu- arquivo.text', 'r')
>>> f.readlines()
['primeira linha\n', 'segunda linha\n', 'terceira linha\n', 'quarta linha\n', 'quinta linha']
>>> f.readlines()
[]
```



Exemplos

Repare que ao chamarmos pela segunda vez a função retornar uma lista vazia pois ela, na verdade, retorna as linhas restantes. Como, ao abrir o arquivo, restavam todas as linhas então ela retornou todas as linhas.

Confundiu? Veja se este exemplo clareia as coisas.

```
>>> f = open('seu- arquivo.text', 'r')
>>> f.readline()
'primeira linha\n'
>>> f.readline()
'segunda linha\n'
>>> f.readlines()
['terceira linha\n', 'quarta linha\n', 'quinta linha']
```



Exemplos

Para escrever em um arquivo sem apagar seu conteúdo, ou seja, adicionando (incluído) novo conteúdo seguimos 3 passos:

- Ler todo o conteúdo do arquivo (Cursor vai para o final do arquivo)
- efetuar a adição do conteúdo
- escrever as linhas

```
# Abra o arquivo (leitura)
arquivo = open('musica.txt', 'r')
conteudo = arquivo.readlines()

# insira seu conteúdo
# obs: o método append() é proveniente de uma lista
conteudo.append('Nova linha')

# Abre novamente o arquivo (escrita)
# e escreva o conteúdo criado anteriormente nele.
arquivo = open('musica.txt', 'w')
arquivo.writelines(conteudo)
arquivo.close()
```





Leitura e escrita de arquivos CSV

Arquivos CSV

CSV (Comma Separated Values, ou valores separados por vírgulas) correspondem ao idioma comum de troca de informações entre planilhas de cálculo ou base de dados.

Embora cada aplicativo tenha o seu método específico de **codificar seus dados**, todos possuem uma forma de **exportar e importar** dados em formato csv.

- Cada linha em um arquivo csv é um registro de dados.
- Cada registro consiste de um ou mais campos, separados por vírgulas.



Biblioteca CSV

A linguagem Python possui um módulo, não por acaso denominado csv, que facilita em muito o trabalho com este tipo de arquivo em Python.

Este módulo implementa funções que realizam a leitura e escrita sobre arquivos csv.

<https://cadernodelaboratorio.com.br/arquivos-csv-em-python>
n/



Biblioteca CSV

As funções que iremos conhecer são:

- **csv.reader()**: Retorna um objeto “reader” que permite a iteração sobre as linhas do arquivo csv
- **csv.writer()**: Retorna um objeto “writer” que converte o dado do usuário numa linha do arquivo csv.

- **csv.DictReader()**: Com esta classe trabalhamos com um dicionário no qual as chaves são dadas pelos nomes das colunas. Estes nomes podes ser dados através do parâmetro “fieldnames” ou obtidos diretamente da primeira linha do arquivo csv.
- **csv.DictWriter()**: Cria um objeto que permite a gravação de linhas no arquivo cvs a partir de um dicionário passado como parâmetro.



Exemplo CSV.reader()

```
with open('teste.csv') as csfile:  
    reader = csv.reader(csfile)
```

Exemplo CSV.writer()

```
with open('teste.csv', 'w') as csfile:  
    writer = csv.writer(csfile)  
    writer.writerow("nova linha")
```



Exemplo CSV.DictWriter()

```
# w -> write, a-> append
with open('meu_arquivo.csv', 'a', newline='') as csvfile:
    fieldnames = ['nome_coluna1', 'nome_coluna2']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames, delimiter=";")

    writer.writeheader()
    writer.writerow({'nome_coluna1': 'boing 737', 'nome_coluna2': 'Ferrari'})
    writer.writerow({'nome_coluna1': 'airbus4', 'nome_coluna2': 'Volks6'})
```



Exemplo CSV.DictReader()

```
with open('meu_arquivo.csv') as csvfile:  
    reader = csv.DictReader(csvfile, delimiter=";")  
    for row in reader:  
        print(row['nome_coluna1'], row['nome_coluna2'])
```





Recapitulando:

- Conhecemos como funciona a manipulação de arquivos de texto no Python
- Conhecemos leitura e escrita de CSV no Python





Importando arquivo .csv para nossa projeto

E agora?

- Vamos voltar para nosso projeto
- Importar o arquivo CSV
- Separar os dados dos alunos em uma lista

