

Revisão/Introdução ao Python

- Aula 03 -
Coleta, Preparação e
Análise de Dados



PUCRS

Pontifícia Universidade Católica
do Rio Grande do Sul

Prof. Me. Lucas R. C. Pessutto

Python

- * Linguagem de código aberto, lançada em 1991 por **Guido van Rossum**

- * **Simples:**

- Legibilidade
- Produtividade
- Indentação

- * **Flexível:**

- Tipos dinâmicos
- Script x Interativo
- Multiplataforma
- Multiparadigma

- * Página Oficial: <https://www.python.org/>



- * **Curiosidade:** Nome da linguagem inspirado em um grupo de comédia britânico: Monty Python



Python – Comunidade


pandas

 Keras

 TensorFlow



 NumPy

django

 Detectron2

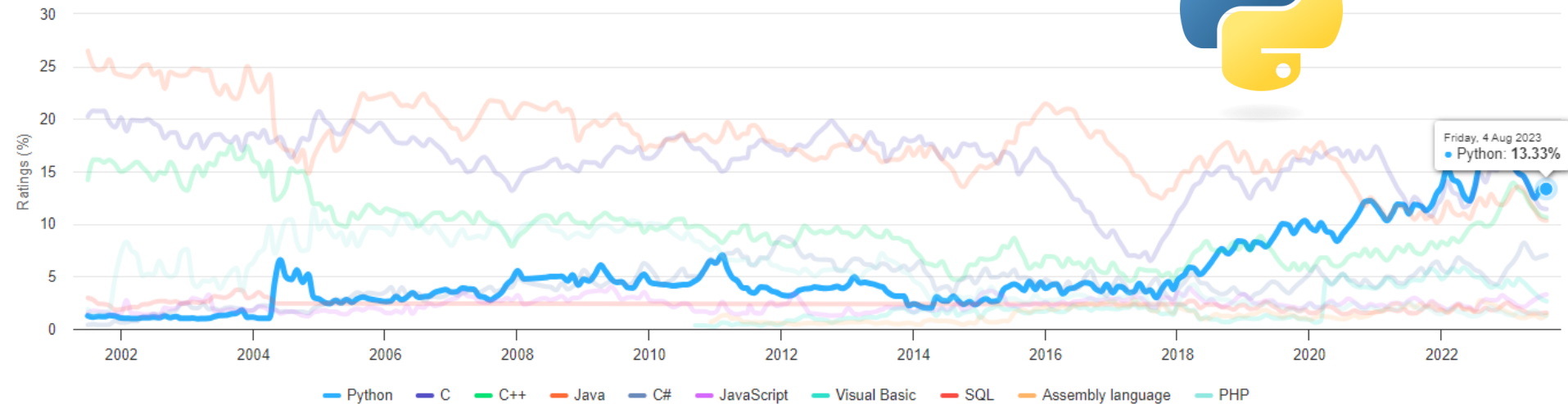
pygame

matplotlib

Python é Pop!

TIOBE Programming Community Index

Source: www.tiobe.com



Quem usa Python?



Instagram



Dropbox



Google



Pinterest



Spotify®

Uber

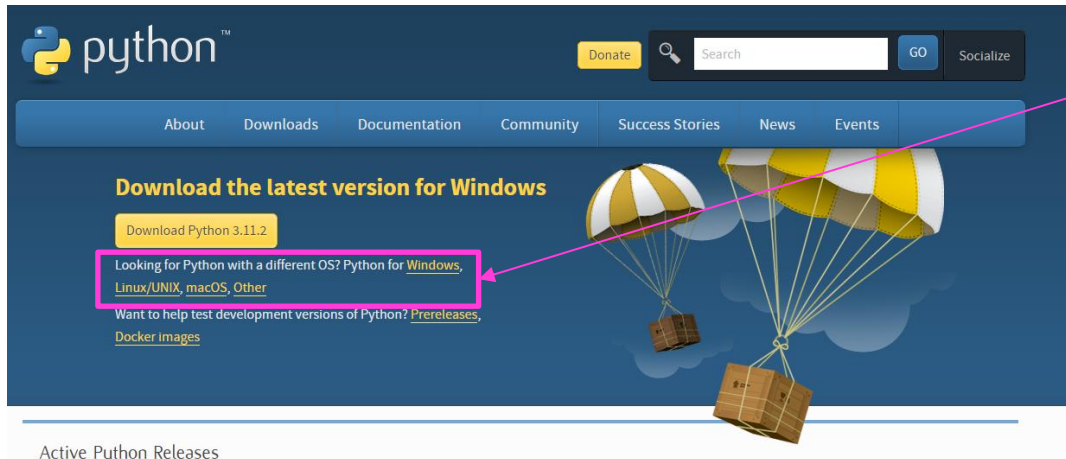
NETFLIX



reddit

Hands on: Instalando Python!

Acesse o site oficial do Python python.org/downloads e selecione a versão apropriada de acordo com o seu sistema operacional

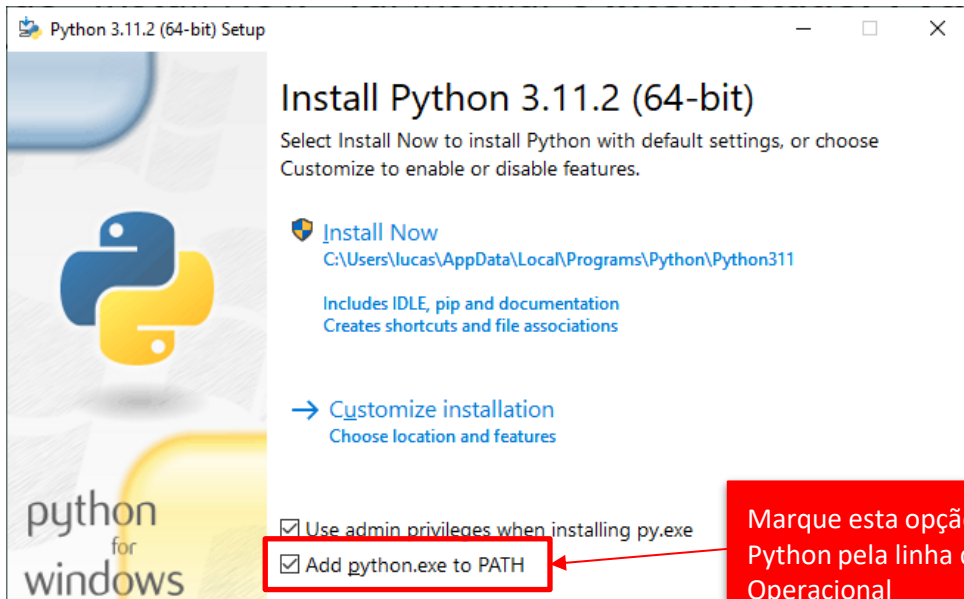


Se você não usa Windows, tem versões para **Mac OS X**, **Linux** e outras plataformas aqui!

Se você já possui o Python instalado em sua máquina, garanta que a versão instalada seja superior à versão 3.10

Hands on: Instalando Python!

A opção “Install Now” vai instalar o **interpretador Python** e o **ambiente de desenvolvimento IDLE** sem necessidade de muitas configurações manuais:

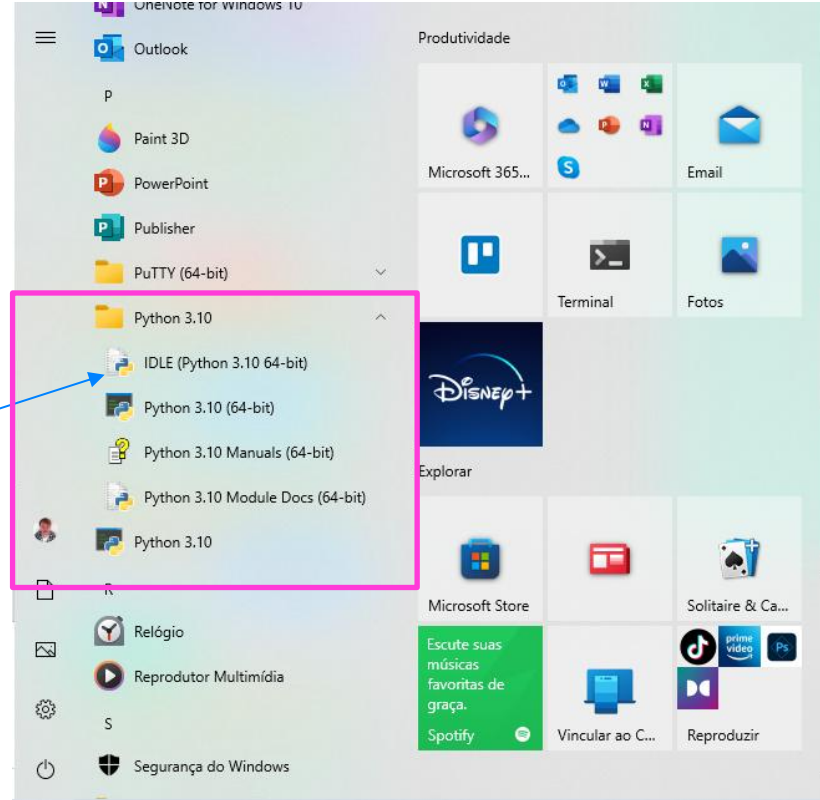


Marque esta opção! Ela vai permitir usar o Python pela linha de comando do Sistema Operacional

Hands on: Instalando Python!

Após instalados o interpretador Python e o IDLE podem ser acessados pelo menu do Windows

Normalmente vamos usar apenas o IDLE para editar e testar os nossos programas.



Hands on: Instalando Python!

```
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
>>> print("hello world!")
hello world!
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file: a file-like object (stream); defaults to the current sys
    sep: string inserted between values, default a space.
    end: string appended after the last value, default a newline
    flush: whether to forcibly flush the stream.

>>> |
```

Usamos o editor para criar **scripts** (programas de várias linhas) e executá-los de uma só vez. Normalmente, os scripts são mais utilizados pois vamos resolver problemas que precisarão de várias linhas de código.

Na interface interativa (shell) é possível digitar os comandos um após o outro e ver o resultado imediatamente no console.

```
aula_pratica_01.py - C:\Users\JulianoWickboldt\Google Drive\Teaching\UFRGS\INF01041 - Introdução à Programação em Python\Semana 01 - Introdução...
File Edit Format Run Options Window Help
# Meu primeiro programa
print("Bem-vindos!")
print("Esse é um primeiro programa implementado como script Python.")
print("Tchau!")

Ln: 5 Col: 0
```

Documentação de Referência

A documentação completa de referência da linguagem está disponível em docs.python.org/3/:

Python » Brazilian Portuguese » 3.11.2 » 3.11.2 Documentation »

Download

Baixar esses documentos

Documentação por versão

- Python 3.12 (in development)
- Python 3.11 (stable)
- Python 3.10 (stable)
- Python 3.9 (security-fixes)
- Python 3.8 (security-fixes)
- Python 3.7 (security-fixes)
- Python 3.6 (EOL)
- Python 3.5 (EOL)
- Python 2.7 (EOL)
- Todas versões

Outros recursos

- Índice das PEPs
- Guia do Iniciante
- Lista de Livros
- Material Audiovisual
- Guia do Desenvolvedor do Python

documentação Python 3.11.2

Seja bem-vindol Esta é a documentação oficial do Python 3.11.2,

Partes da documentação:

- [O que há de novo no Python 3.11?](#)
ou todos os documentos "O que há de novo" desde a 2.0
- [Tutorial](#)
comece por aqui
- [Referência da Biblioteca](#)
mantenha isso debaixo do seu travesseiro
- [Referência da Linguagem](#)
descreve sintaxe e elementos da linguagem
- [Configurações e Uso do Python](#)
como usar o Python em diferentes plataformas
- [Python HOWTOs](#)
documentos aprofundados sobre tópicos específicos
- [Instalando módulos Python](#)
instalando a partir do Python Package Index e outras fontes
- [Distribuindo módulos Python](#)
publicando módulos para instalação por outros
- [Estendendo e Incorporando](#)
tutorial para programadores C/C++
- [API Python/C](#)
referência para programadores C/C++
- [FAQs](#)
perguntas frequentes (com respostas!)

Índices e tabelas:

Tem um tutorial bem completo para leitura que você pode usar para estudar.

Essa é a principal referência para as **funções** e **tipos** de dados nativos da linguagem. Como a própria documentação sugere: “*mantenha isso debaixo do seu travesseiro*”.

Variáveis em Python

São declaradas assim que recebem um valor, por exemplo:

`x = 5` indica que a variável recém criada `x` possui o valor `5`

`x` é o identificador, `=` é o símbolo de atribuição (versão de `←`) e `5` é uma constante

Variáveis possuem tipos de dados específicos:

inteiro (int): `42`

ponto flutuante/reais (float): `3.14`

textos (string): `"João da Silva"`

booleano (bool): `False`

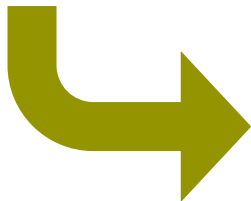
complexo (complex): `1.5 + 5.3j`

Suportam operações como adição, subtração, multiplicação, divisão e podem se relacionar logicamente

Case-sensitive: identificadores `nome` e `Nome` representam variáveis diferentes

Variáveis

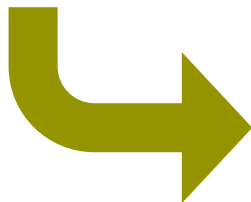
Para saber o tipo de uma variável utiliza-se a função `type(variável)`



```
x = 2
tipo = type(x)
print(tipo)

saída: <class 'int'>
```

Variáveis podem mudar seu tipo a qualquer momento



```
x = 4
tipo = type(x)
print(tipo)
x = 1.42
tipo = type(x)
print(tipo)

saída: <class 'int'>
saída: <class 'float'>
```

Boas Práticas para Identificadores

Geral: iniciar com letra minúscula + letras minúsculas ou números

Não usar caracteres especiais (&, *, \$, -, +, etc.) ou acentuação

Identificadores com múltiplas palavras devem ser separadas por sublinhado (_)

Utilizar identificadores significativos que descrevam o que representam

Identificadores desejáveis

x, y, endereco,
valor_do_pedido, jogador1,
jogador2



Identificadores indesejáveis/incorretos

Endereco, 12pessoas,
valorDoPedido, var&avel



Comentários

Estruturas **desconsideradas** pelo compilador

Buscam facilitar a leitura do código pelo programador

Importantes uma vez que programas complexos costumam ser muito mais vezes lidos do que escritos

Comentários em linha (#)

Exemplo de comentário em linha

Comentários em bloco (""" """)

"""

Exemplo de comentário em bloco

"""

Comando de Atribuição

Armazena um valor em uma variável pelo operador =

Sintaxe: `variável = expressão`

Ex:

```
idade = 21
```

```
altura = 1.82
```

```
resposta = "Sim"
```

```
outra_idade = idade
```

Endereço	Identificador	Valor	Tipo
0x0000	idade	21	int
0x0001	altura	1.82	float
0x0002	resposta	"Sim"	string
0x0003	outra_idade	21	int
...

Representação simplificada da memória (sem levar em conta tipos e tamanhos)

Saída de Dados

```
idade = 21  
print("O valor de idade é", idade)
```

O valor de idade é 21

O Python insere um espaço antes
de imprimir o valor da variável!

```
idade = 21  
print(idade, "O valor de idade é")
```

21 O valor de idade é

```
idade = 21  
print("O valor de idade é", "idade")
```

O valor de idade é idade

Saída Formatada

Python oferece uma série de formas de formatar texto com strings. Suponha que temos três variáveis para imprimir: nome (str), matricula (int), nota (float).

```
nome = "Lucas"  
matricula = 305016  
nota = 9.7539
```

```
print("Aluno:", nome, "(", matricula, ") - Nota:", nota)
```

```
Aluno: Lucas ( 305016 ) - Nota: 9.7539
```

Saída Formatada

Forma 3: f-string

strings prefixadas com um `f` permitem incluir variáveis ou expressões entre chaves `{}` diretamente no conteúdo da string

Atenção: O prefixo `f` fica antes das aspas

```
print(f"Aluno: {nome} ({matricula}) - Nota: {nota:.1f}")
```

A formatação da precisão do float funciona da mesma forma que nos outros métodos

Mais detalhes em: https://docs.python.org/pt-br/3/reference/lexical_analysis.html#f-strings

Funções da Biblioteca math

Função	Exemplo	Descrição
ceil	<code>ceil(x)</code>	Arredonda o número real para cima: <code>ceil(3.2) = 4</code>
floor	<code>floor(x)</code>	Arredonda o número real para baixo: <code>floor(3.8) = 3</code>
sin	<code>sin(x)</code>	Seno de x (em radianos)
cos	<code>cos(x)</code>	Cosseno de x (x em radianos)
tan	<code>tan(x)</code>	Tangente de x (em radianos)
exp	<code>exp(x)</code>	e elevado na potência x
log	<code>log(x)</code>	Logaritmo natural de x (base e)
log10	<code>log10(x)</code>	Logaritmo de x na base 10
pow	<code>pow(x, y)</code>	Calcula x elevado à potência y
sqrt	<code>sqrt(x)</code>	Raiz quadrada de x
fabs	<code>fabs(x)</code>	Valor absoluto de x – Retorna um float: <code>fabs(5.1) = 5.1</code>
fmod	<code>fmod(x, y)</code>	Resto da divisão de dois floats: <code>fmod(9.8, 4.0) = 1.8</code>

Lista completa de funções da biblioteca math: <https://docs.python.org/pt-br/3/library/math.html?highlight=math#module-math>

Funções da Biblioteca math

```
import math # Lembre-se de sempre importar a biblioteca
```

```
# Função ceil
```

```
print(f"ceil(9.2) = {math.ceil(9.2)}")
```

```
print(f"ceil(-9.8) = {math.ceil(-9.8)}")
```

```
# Função floor
```

```
print(f"floor(9.2) = {math.floor(9.2)}")
```

```
print(f"floor(-9.8) = {math.floor(-9.8)}")
```

```
# Funções Trigonométricas
```

```
print(f"sin(0) = {math.sin(0)}")
```

```
print(f"cos(0) = {math.cos(0)}")
```

```
print(f"tan(0) = {math.tan(0)}")
```

```
...
```

Comandos de Seleção

Condicional Simples

```
if condicao:  
    <comando>
```

Seleção Dupla

```
if condicao:  
    <comando>  
elif condicao:  
    <comando>  
else:  
    <comando>
```

Seleção Múltipla

```
match expressão:  
    case padrao2: <comando>  
    case padrao2: <comando>  
    ...
```

Indentação

- * Forma de hierarquizar as linhas do programa para tornar o código mais legível
- * Em Python a indentação é **obrigatória**
- * Utiliza determinado espaçamento a partir da margem esquerda

```
1  #Início do bloco 0
2  comandos do bloco 0
3      #Início do bloco 1
4      comandos do bloco 1
5          #Início do bloco 2
6          comandos do bloco 2
7          ...
8          #Fim do bloco 2
9      #Fim do bloco 1
10 #Fim do bloco 0
11
```

Comandos Iterativos

WHILE

```
while <expressão lógica>:  
    <comandos indentados>
```

FOR

```
for <variável> in <iterável>:  
    <comandos indentados>
```

Repetição com for

A **variável** especificada aqui assume cada um dos valores do objeto **iterável**

Uma **sequência** é um exemplo de **iterável**, mas existem outros, como listas, conjuntos, dicionários.

```
for <variável> in <iterável>:  
    <comandos indentados>
```

Os comandos do bloco indentado serão repetidos de acordo com a **quantidade** de elementos no objeto iterável

É uma construção similar a do “**para todo**” na matemática quando usado para definir relações de pertinência em conjuntos:

$$\forall x \in \{1, 2, \dots, n\} \dots$$

Listas - Revisão

O primeiro índice
é **sempre** zero

ÍNDICES: Identificação
de cada posição da lista

frutas

0

1

2

maçã

laranja

uva

NOME: Comum para
todos os elementos

TAMANHO: Quantidade
máxima de valores que podem
ser armazenados

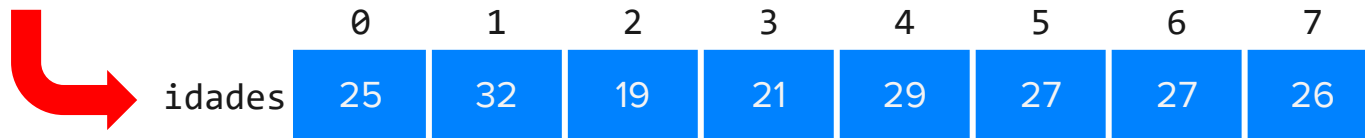
Declarando Listas

Lista vazia:

```
lista = []
```

Idade de pessoas participando de uma pesquisa:

```
idades = [25, 32, 19, 21, 29, 27, 27, 26]
```



Nomes dos times de um campeonato de futebol:

```
times = ["Grêmio", "Inter", "Flamengo", "Palmeiras", "...", "Santos"]
```



Inicializando uma lista

Lista vazia:

```
lista = []
```

Com valores pré-definidos:

```
idades = [25, 32, 19, 21, 29, 27, 27, 26]
```

	0	1	2	3	4	5	6	7
idades	25	32	19	21	29	27	27	26

Com um valor padrão

idades = [0] * 10

Lista de 10 elementos
inicializada com zeros

[illegible]

Acessando Elementos da Lista

	0	1	2	3	4	5	6	7
idades	25	32	19	21	29	27	27	26
	-8	-7	-6	-5	-4	-3	-2	-1

Para acessar cada elemento utilizamos seu **índice**.

Índices negativos também são válidos. Para acessar o **penúltimo** elemento podemos usar `idade[-2]`.

Inserção de Elementos

	0	1	2	3	4	5	6	7
idades	25	32	19	21	29	27	27	26

Adicionar uma idade (40) no final da lista:

```
idades.append(40)
```

Um **novo índice** é criado para referenciar o novo elemento.

	0	1	2	3	4	5	6	7	8
idades	25	32	19	21	29	27	27	26	40

Inserção de Elementos

	0	1	2	3	4	5	6	7
idades	25	32	19	21	29	27	27	26

Adicionar uma idade (21) no índice 5 da lista:

```
idades.insert(5, 21)
```

	0	1	2	3	4	5	6	7	8
idades	25	32	19	21	29	21	27	27	26

Os demais elementos
serão “deslocados” e
terão novos índices.

Remoção de Elementos

	0	1	2	3	4	5	6	7	8
idades	25	32	19	21	29	21	27	27	26

Remover o último elemento da lista:

```
idade_removida = idades.pop()
```

O índice e o elemento deixam de existir. O valor do elemento é **retornado**.

	0	1	2	3	4	5	6	7
idades	25	32	19	21	29	21	27	27

Remoção de Elementos

	0	1	2	3	4	5	6	7
idades	25	32	19	21	29	21	27	27

Remover o elemento de índice 3 da lista:

```
idade_removida = idades.pop(3)
```

	0	1	2	3	4	5	6
idades	25	32	19	29	21	27	27

Os demais elementos serão “deslocados” e terão novos índices. O índice que deixa de existir será o **último**.

Iterando sobre listas

Iteração com **while**:

```
idades = [25, 32, 19, 21, 29, 27, 27, 26]

i = 0
while i < 8:
    print(f"Idade {idades[i]} na posição {i}")
    i += 1
```

Com **while** é preciso manipular explicitamente os valores dos índices da lista usando, por exemplo, a própria variável (**i**) de controle do laço.

```
Idade 25 na posição 0
Idade 32 na posição 1
Idade 19 na posição 2
Idade 21 na posição 3
Idade 29 na posição 4
Idade 27 na posição 5
Idade 27 na posição 6
Idade 26 na posição 7
```

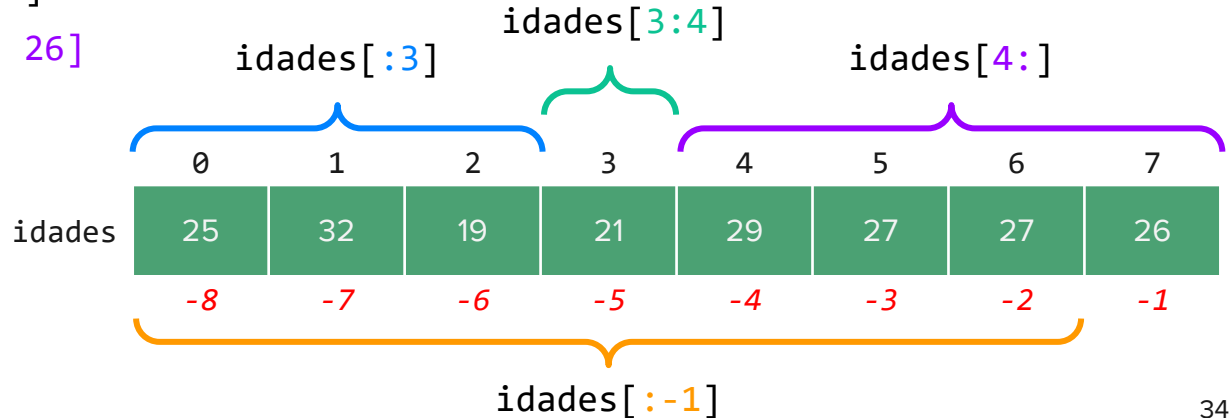
Fatiamento

- * A notação de colchetes `<início>` e `<fim>` são **índices** da lista, ambos **opcionais** e o índice `<fim>`, quando informado, **não é incluído** na fatia.
- * Ela também pode ser usada para criar sub-listas
- * Por exemplo, considere a seguinte lista:

```
>>> idades[3:4]    >>> idades[4:]  
[21]              [29, 27, 27, 26]
```

```
>>> idades[:3]  
[25, 32, 19]
```

```
>>> idades[:-1]  
[25, 32, 19, 21, 29, 27, 27]
```



List Comprehensions

- * Notação concisa (estilo funcional) para se criar novas listas.
- * Consegue compactar em uma única linha um laço de repetição inteiro

```
lista = []  
for i in range(1, 6):  
    lista.append(i)  
  
print(lista)
```



```
lista = [i for i in range(1, 6)]  
  
print(lista)
```

Para cada valor *i*, gerado pelo laço *for*, a expressão à esquerda é avaliada e o valor resultante (nesse caso o próprio *i*) é inserido na lista.

List Comprehensions – Filtragem

- * **Filtragem** consiste em aplicar alguma condição nos valores gerados, de modo a produzir uma lista que contenha somente os valores que passarem por aquele filtro

```
lista = [i for i in range(1, 6) if i % 2 == 0]
print(lista)
```

[1 2 3 4 5] [~~1~~ 2 ~~3~~ 4 ~~5~~]

```
lista = [2 4]
```

Criando dicionários

Dicionário vazio: `dic = {}`

Dicionário com dados:

Palavras e seus respectivos significados (um dicionário ou glossário):

```
glossario = {"algoritmo": "maneira chique de falar passo a passo",  
             "bug": "problema no código do programa",  
             "estrutura de dados": "coleção de dados organizados",  
             "indentação": "recuos usados para organizar código",  
             "variável": "rótulo que referencia um valor"}
```

Número de matrícula e notas de alunos de uma turma:

```
turma = {1756: 9.5, 2025: 7.4, 2094: 7.2, 2132: 5.9, 1822: 7.6}
```

Apesar de numérico e único, o identificador do aluno não é sequencial nem inicia em zero, por isso é preferível usar um dicionário.

Acesso aos valores

Imprimir na tela o valor associado a chave “bug” do glossário:

```
print(glossario["bug"])
```

Para acessar cada valor utilizamos sua **chave**.

Calcular a média de 3 notas de alunos:

```
media = (turma[1756] + turma[2132] + turma[2094]) / 3
```

Cuidado! O acesso a chaves indefinidas gera erros:

```
print(turma["1756"])
```

```
...  
KeyError: '1756'
```

O **tipo de dado** usado no acesso deve ser o mesmo utilizado na criação.

Adicionar/modificar valores

Adicionando uma nova palavra (chave) e seu significado no glossário:

```
glossario["exceção"] = "erro na execução do programa"
```

Basta fazer uma **atribuição** direta que uma nova chave será criada e o respectivo valor associado a ela.

Modificar a nota do aluno com a matrícula 1756

```
turma[1756] = 10.0
```

Caso a chave já **exista** o valor é **modificado**.

Removendo pares (chave/valor)

Remover a entrada bug (chave) do glossário e seu significado (valor):

```
del glossario["bug"]
```

A instrução del pode ser usada para remover qualquer tipo de variável.

Tentar remover uma chave inexistente gera um KeyError.

Dicionários também possuem um método pop que remove e retorna o valor associado a uma chave:

```
nota = turma.pop(1756)
```

Não é possível remover apenas a chave ou apenas o valor. Em um dicionário uma **chave sempre terá um valor associado** e vice-versa.

Iteração sobre dicionários

Na iteração com **for** a variável de controle do laço vai assumir o valor de cada uma das chaves do dicionário:

```
for chave in glossario:  
    print(f"{chave} -> {glossario[chave]}")
```

Lembrando que com listas a variável do laço assume os valores contidos e não os índices.

```
algoritmo -> maneira chique de falar passo a passo  
estrutura de dados -> coleção de dados organizados  
indentação -> recuos usados para organizar código  
variável -> rótulo que referencia um valor  
exceção -> erro na execução do programa
```

Declarando uma função

def

Comando que identifica a definição de uma nova função

nome

Nome da função escolhido pelo programador

(argumentos) :

Dados de entrada recebidos pela função (opcional, veremos isso depois)

Chamando uma função

- * A **chamada** de uma função é feita através do seu **nome**
- * A chamada pode ser feita em **diversos pontos** do programa principal após a sua definição ou a partir de **outras funções**

nome **(argumentos)**

Mesmo nome da
função utilizado
na definição

Dados de entrada que
serão enviados para a
função (opcional)

Funções com Múltiplos Retornos

- * Servem para **retornar mais de um valor** como resultado de uma função, os quais podem ser:
 - Valor de uma variável dentro da respectiva função
 - Valor fixo
 - Resultado de um cálculo

Funções com Múltiplos Retornos

```
def nome(arg1, arg2, ..., argN):
```

→ Declaração da função

...

```
    return valor1, valor2, ..., valorN
```

Aqui os valores estão sendo retornados a partir de variáveis definidas no escopo da função.

Programa principal

...

```
r1, r2, ..., rN = nome(a1, a2, ..., aN)
```

→ Chamada da função

...

Cada valor retornado pela função está sendo **atribuído a uma variável** no programa principal.

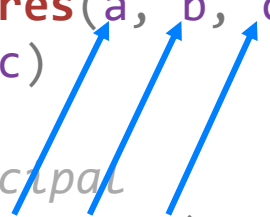
Argumentos Nomeados

- * Argumentos de funções vistos até agora são chamados de **argumentos posicionais**, onde cada valor é passado de acordo com sua posição na função

- * Exemplo:

```
def imprime_valores(a, b, c):  
    print(a, b, c)
```

```
## Programa Principal  
imprime_valores(1, 2, 3)
```



1 2 3

Argumentos Nomeados

- * Também podemos usar **argumentos nomeados**, onde cada valor é passado por uma associação de nome e valor
- * Dessa forma, os argumentos podem ser passados em qualquer ordem para a função
- * Utilizar o nome do argumento seguido de = e seu valor durante a chamada da função
- * Exemplo:

```
def imprime_valores(a, b, c):  
    print(a, b, c)
```

```
## Programa Principal
```

```
imprime_valores(c=1, b=2, a=3)
```

3 2 1

Argumentos Nomeados

- * Também podemos **misturar** argumentos posicionais e nomeados, em uma mesma função
- * Os argumentos posicionais devem **vir primeiro** que os argumentos nomeados
- * Os argumentos nomeados se tornam “opcionais” porque têm um valor padrão atribuído na declaração
- * Exemplo:

```
def imprime_valores(a, b, c, d=4):  
    print(a, b, c, d)
```

```
## Programa Principal  
imprime_valores(1, 2, 3)
```

```
imprime_valores(1, 2, 3, d=5)
```

```
1 2 3 4  
1 2 3 5
```


Classes em Python

Um programa pode **definir novas classes** (tipo de objeto) utilizando a primitiva **class**.

Assim como funções (def), classes também devem ser definidas antes de usadas.

class **Carro:**

Comando que
identifica a definição
de uma nova classe

Nome da classe
escolhido pelo
programador

Por convenção, costuma-se
usar *CamelCase* nos nomes de
classes definidas pelo
programador.

Classes em Python

Toda classe deve conter um **método** (operação) especial `__init__` que normalmente define os **atributos** (dados) básicos de um objeto.

```
class Carro:  
    def __init__(self, marca, ano, cor):  
        self.marca = marca  
        self.ano = ano  
        self.cor = cor  
        self.velocidade = 0  
        self.marcha = 0  
        self.esta_ligado = False
```


O primeiro argumento é sempre `self` que identifica uma **instância** da classe.

Outros argumentos seguem as mesmas regras de funções.

Um método é uma espécie de “função local” a uma classe, isto é, só existe neste contexto.

Os atributos (dados) internos do objeto são identificados pelo prefixo `self`.

Os **argumentos** passados para o método `__init__`, em geral, são usados para inicializar **atributos**.



Métodos “Dunder”

- * Dunder = “Double underscore before and after”. (Também chamados de métodos mágicos)
- * São métodos especiais, que não são chamados diretamente pelo programador, mas sim pelo interpretador Python em momentos especiais
- * Veremos mais sobre métodos dunder quando falarmos sobre sobrecarga de operadores

Implementando Métodos

- * Outros métodos adicionam funcionalidade à classe Carro:

```
class Carro:
```

```
...
```

```
def ligar(self):
```

```
    self.esta_ligado = True
```

O argumento self segue sendo necessário.

Atributos do objeto definidos em outros métodos podem ser usados normalmente.

Argumentos e retorno funcionam da mesma forma como em funções.



- * Permitem que programas façam armazenamento **permanente** (ou **persistente**) de informações
- * Pode-se utilizar os mesmos dados entre execuções diferentes de um programa ou de programas diferentes
- * Arquivos são armazenados em dispositivos de **memória secundária** (disco rígido, CD, DVD, pendrives)
- * Podem armazenar mais dados que a memória principal comporta



Arquivos em Python

Um arquivo é um conjunto de *bytes*, colocados um após o outro (*stream*), armazenados em um dispositivo de memória secundária

- * A linguagem não impõe estrutura alguma aos arquivos
- * Arquivos podem ser usados de duas formas:
 - Como **fonte** de dados para o programa
 - Arquivo de entrada (**input**)
 - Como **destino** de dados de um programa
 - Arquivo de saída (**output**)

Arquivos de Dados

JSON (JavaScript Object Notation)

É um formato leve de troca de dados inspirado pela sintaxe de objetos JavaScript

```
{  
  "items": ["i1", "i2", "i3"],  
  "numero": 1234,  
  "ativo": true  
}
```

Padronizado pela RFC 7159

O módulo `json` oferece duas funções principais para carregar (`load`) e escrever (`dump`) arquivos nesse formato

Documentação completa em docs.python.org/pt-br/3/library/json.html

CSV (Comma-Separated Values)

Formato mais comum de importação e exportação de planilhas e bancos de dados

```
header-1, header-2, header-N  
dado-1.1, dado-1.2, dado-1.N  
dado-2.1, dado-2.2, dado-2.N  
...  
dado-M.1, dado-M.2, dado-M.N
```

Padronizado pela RFC 4180

O módulo `csv` em Python oferece objetos `reader` e `writer` para manipular arquivos nesse formato

Documentação completa em docs.python.org/pt-br/3/library/csv.html

Arquivos JSON

Chaves

```
{  
  "id": 1,  
  "Modelo": "Celta",  
  "Placa": "AAA1234"  
}
```

Valores

Os dados no arquivo json são organizados da mesma forma que em um dicionário!

Arquivos JSON

* Primeiramente, o módulo `json` não é “nativo” ele precisa ser importado em programas que o utilizem com `import json`

`json.load(fp, ...)`

* Desserializa um arquivo (`fp`) com suporte a `.read()` contendo um documento JSON, carregando seu conteúdo e retornando um objeto Python

`json.dump(obj, fp, ...)`

* Serializa um objeto Python (`obj`) como um stream JSON formatado descarregando em um arquivo (`fp`) com suporte a `.write()`

Problema 1: Hashmon®

Enunciado de um problema:

Você resolveu criar um jogo de cartas chamado **Hashmon®**. No seu jogo, cada carta representa um personagem com as seguintes características: **nome** (str), **tipo** (grama, água ou fogo) (str), **HP** (quantidade de vida) (int) e **fraqueza** (os mesmos valores do tipo) (str). Para auxiliar na criação do baralho você resolve fazer um programa em Python onde você vai digitando iterativamente as características das cartas e o programa gera um aviso em duas situações:

1. caso você tente inserir uma carta com nome repetido
2. caso você tente inserir uma carta com as outras características repetidas

Seu programa deve permitir salvar e recuperar de um arquivo seu baralho de Hashmons

Problema 3: Hashmon®



Chaves: nome (str)



Valores: características (dict)

Descrição dos Dados
e Algoritmos



```
baralho = {  
    "Hashzard": {  
        "hp": 20,  
        "tipo": "fogo",  
        "fraqueza": "grama"  
    },  
    "Squirrel":  
    {...}  
}
```

Carregando o Baralho

Vamos usar `json.load` para carregar o conteúdo de um arquivo JSON e definir o conteúdo inicial do baralho

```
import json
```

```
def carregar_baralho():
```

```
    try:
```

```
        with open("baralho.json", "r") as arquivo:
```

```
            return json.load(arquivo)
```

```
    except FileNotFoundError:
```

```
        print("O arquivo não foi encontrado!")
```

Se existir, o arquivo "baralho.json" será aberto para leitura ("r")

`json.load` vai tratar de desserializar todo o conteúdo do arquivo sem necessidade de um laço de iteração

Se o arquivo não existe (`FileNotFoundError`) não é um problema, pode ser a primeira execução do programa

Salvando o Baralho

Vamos usar `json.dump` para escrever a lista completa no arquivo texto `"baralho.json"` aberto para escrita (`"w"`)

```
import json
```

```
def salvar_baralho(dicionario):
```

```
    try:
```

```
        with open("baralho.json", "w") as arquivo:
```

```
            json.dump(dicionario, arquivo)
```

```
    except OSError:
```

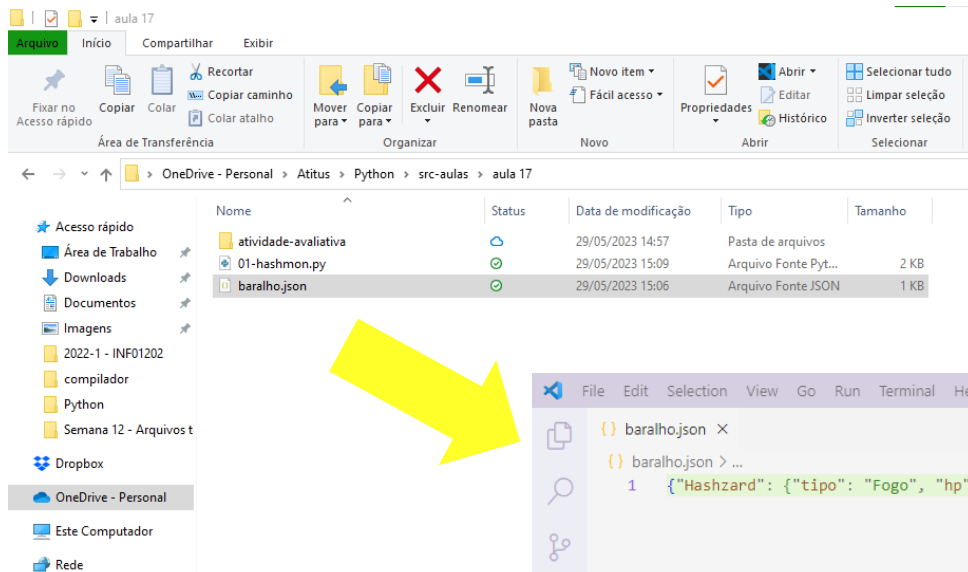
```
        print("Erro ao salvar")
```

Um arquivo chamado "baralho.json" será criado (ou sobrescrito) no mesmo diretório onde estiver rodando o programa em Python

Aqui, novamente, não há necessidade de iterar sobre a lista, todos os itens serão serializados pelo `json.dump` e salvos no arquivo

Arquivos JSON

* Após a execução do código você deve conseguir acessar o arquivo criado com o conteúdo da sua lista de compras em formato JSON



The screenshot shows a Windows File Explorer window with the address bar displaying the path: OneDrive - Personal > Atitús > Python > src-aulas > aula 17. The left sidebar shows the 'Acesso rápido' (Quick access) section with various folders. The main area displays a table of files and folders:

Nome	Status	Data de modificação	Tipo	Tamanho
atividade-avaliativa		29/05/2023 14:57	Pasta de arquivos	
01-hashmon.py		29/05/2023 15:09	Arquivo Fonte Pyt...	2 KB
baralho.json		29/05/2023 15:06	Arquivo Fonte JSON	1 KB

A large yellow arrow points from the 'baralho.json' file in the File Explorer to the Visual Studio Code editor window below.



The screenshot shows the Visual Studio Code editor with the file 'baralho.json' open. The editor displays the following JSON content:

```
{  
  "Hashzard": {  
    "tipo": "Fogo",  
    "hp": 20,  
    "fraqueza": "Grama"  
  },  
  "Squirrel": {  
    "tipo": "Grama",  
    "hp": 33,  
    "fraqueza": "Fogo"  
  }  
}
```

Exercício: Países/Cidades/Estados

- * Utilizando o arquivo `paises-estados-cidades.json`, que contém informações sobre todos os países, estados e cidades do planeta.
- * Crie um arquivo chamado `cidades_A.json`, que contém as informações das cidades cujo nome inicia com a letra “A”.

```
{  
  "Alegrete": {  
    "País": "Brasil",  
    "Estado": "Rio Grande do Sul"  
  },  
  "Altamira": {  
    "País": "Brasil",  
    "Estado": "Pará"  
  }  
}
```

Arquivos CSV

```
quantitativo-de-alunos.csv - Notepad
File Edit Format View Help
CodCurso;NomeCurso;Ano;Periodo;Vinculados;Matriculados;Ingressantes;Diplomados;Evadidos
298;ADMINISTRAÇÃO;2010;1;1501;1337;166;91;24
583;ADMINISTRAÇÃO - EAD;2010;1;351;345;0;0;6
299;AGRONOMIA;2010;1;495;441;47;31;20
300;ARQUITETURA E URBANISMO;2010;1;625;557;53;41;12
301;ARQUIVOLOGIA;2010;1;157;134;30;4;6
303;ARTES VISUAIS;2010;1;593;498;81;37;19
801;ARTES VISUAIS - ENSINO A DISTÂNCIA - EAD;2010;1;90;90;0;0;0
```

Nem sempre os arquivos CSV vão incluir cabeçalhos. O ideal é se orientar pelo dicionário de dados.

Um caractere específico é escolhido para delimitar os campos. Os mais comuns são vírgula, ponto e vírgula e tabulação (\t).

É preciso também verificar a codificação do arquivo CSV. Normalmente a informação é fornecida junto com o arquivo. No caso desse conjunto de dados específico a codificação é **utf-8**.

Arquivos CSV

* O módulo csv também não é “nativo” e precisa ser importado em programas que o utilizem com `import csv`

`csv.reader(csvfile, ...)`

* Retorna um objeto de leitura para iterar sobre as linhas do csvfile fornecido. No geral, csvfile deve ser um arquivo texto aberto para leitura, mas pode ser outro tipo de iterável.

`csv.writer(csvfile, ...)`

* Retorna um objeto de escrita capaz de converter os dados em strings delimitadas para gravação em arquivos. No geral, csvfile será um arquivo aberto para escrita.

Exemplo: Top-100 Videogames

```
video-games.csv - Bloco de Notas
Arquivo Editar Formatar Exibir Ajuda
Rank,Name,,Platform,Metascore,Date,Title,
1,The Legend of Zelda: Ocarina of Time,,Nintendo 64,99,23-Nov-98,"As a young boy, Link is tricked by Ganondorf, the
2,Tony Hawk's Pro Skater 2,,PlayStation,98,20-Sep-00,"As most major publishers' development efforts shift to any num
3,Grand Theft Auto IV,,PlayStation 3,98,29-Apr-08,"[Metacritic's 2008 PS3 Game of the Year; Also known as ""GTA IV""
4,SoulCalibur,,Dreamcast,98,8-Sep-99,"This is a tale of souls and swords, transcending the world and all its history
5,Grand Theft Auto IV,,Xbox 360,98,29-Apr-08,"[Metacritic's 2008 Xbox 360 Game of the Year; Also known as ""GTA IV""
6,Super Mario Galaxy,,Wii,97,12-Nov-07,"[Metacritic's 2007 Wii Game of the Year] The ultimate Nintendo hero is taking
aspects of the Wii Remote and Nunchuk controller, unleashing new moves as players shake the controller and even poin
7,Super Mario Galaxy 2,,Wii,97,23-May-10,"Super Mario Galaxy 2, the sequel to the galaxy-hopping original game, inclu
8,Red Dead Redemption 2,,Xbox One,97,26-Oct-18,"Developed by the creators of Grand Theft Auto V and Red Dead Redempt
9,Grand Theft Auto V,,Xbox One,97,18-Nov-14,"Grand Theft Auto 5 melds storytelling and gameplay in unique ways as pla
10,Grand Theft Auto V,,PlayStation 3,97,17-Sep-13,"Los Santos is a vast, sun-soaked metropolis full of self-help guri
```

Rank	Name	Platform	Metascore	Date	Title
1	The Legend of Zelda: Ocarina of Time	Nintendo 64	99	23-Nov-98	As a young boy, Link is tricked by Ganondorf, t
2	Tony Hawk's Pro Skater 2	PlayStation	98	20-Sep-00	As most major publishers' development efforts
3	Grand Theft Auto IV	PlayStation 3	98	29-Apr-08	[Metacritic's 2008 PS3 Game of the Year; Also k
4	SoulCalibur	Dreamcast	98	8-Sep-99	This is a tale of souls and swords, transcending
5	Grand Theft Auto IV	Xbox 360	98	29-Apr-08	[Metacritic's 2008 Xbox 360 Game of the Year; ,
6	Super Mario Galaxy	Wii	97	12-Nov-07	[Metacritic's 2007 Wii Game of the Year] The ul
7	Super Mario Galaxy 2	Wii	97	23-May-10	Super Mario Galaxy 2, the sequel to the galaxy-
8	Red Dead Redemption 2	Xbox One	97	26-Oct-18	Developed by the creators of Grand Theft Auto
9	Grand Theft Auto V	Xbox One	97	18-Nov-14	Grand Theft Auto 5 melds storytelling and gam
10	Grand Theft Auto V	PlayStation 3	97	17-Sep-13	Los Santos is a vast, sun-soaked metropolis full

Exemplo: Top-100 Videogames

* Lendo o arquivo e escrevendo cada linha na tela:

```
import csv
```

```
with open("video-games.csv", "r", encoding="utf-8") as arquivo:
    csv_reader = csv.reader(arquivo, delimiter=",")
    titulos = next(csv_reader) # "pula" os cabeçalhos
    for linha in csv_reader:
        print(linha)
```

O método `csv.reader` retorna um objeto leitor para iterar sobre as linhas do arquivo `_csv`. O delimitador de campos do arquivo é `,`.

A variável `linha` vai representar cada linha do arquivo CSV como uma lista

Primeira Linha do arquivo:

```
['1', 'The Legend of Zelda: Ocarina of Time', '', 'Nintendo 64', '99',  
'23-Nov-98', 'As a young boy, Link is ... of the Seven Sages.', '']
```

Exemplo: Top-100 Videogames

- * Lendo o arquivo e escrevendo cada linha na tela:

```
import csv

with open("video-games.csv", "r", encoding="utf-8") as arquivo:
    csv_reader = csv.reader(arquivo, delimiter=",")
    titulos = next(csv_reader) # "pula" os cabeçalhos
    for linha in csv_reader:
        print(f"| Rank      : {linha[0]}")
        print(f"| Título    : {linha[1]}")
        print(f"| Plataforma: {linha[3]}")
        print(f"| Lançamento : {linha[5]}")
        print('-' * 30)
```

Exemplo: Top-100 Videogames

- * Lendo os dados do arquivo para dentro de um dicionário

```
import csv

with open("video-games.csv", "r", encoding="utf-8") as
arquivo:
    csv_reader = csv.DictReader(arquivo, delimiter=",")
    for linha in csv_reader:
        print(linha)
```

Primeira Linha do arquivo:

```
{'Rank': '1', 'Name': 'The Legend of Zelda: Ocarina of Time', ':',
'Platform': 'Nintendo 64', 'Metascore': '99', 'Date': '23-Nov-98',
'Title': 'As a young boy, Link is ... of the Seven Sages.'}
```

Exemplo: Top-100 Videogames

- * Lendo os dados do arquivo para dentro de um dicionário

```
import csv

with open("video-games.csv", "r", encoding="utf-8") as arquivo:
    csv_reader = csv.DictReader(arquivo, delimiter=",")
    for linha in csv_reader:
        print(f"| Rank      : {linha['Rank']}")
        print(f"| Título    : {linha['Name']}")
        print(f"| Plataforma : {linha['Platform']}")
        print(f"| Lançamento : {linha['Date']}")
        print('-' * 30)
```

Exemplo de Gravação de Dados

* Salvando dados linha a linha no arquivo:

```
import csv
```

Esse parâmetro faz com que o arquivo não escreva o \n no arquivo duas vezes

```
with open("alunos.csv", "w", newline='', encoding="utf-8") as arquivo:
```

```
    csv_writer = csv.writer(arquivo, delimiter=";")
```

```
    csv_writer.writerow(["Nro", "Nome", "Nota"])
```

```
    csv_writer.writerow([1, "Pedro Carlos", 9.5])
```

```
    csv_writer.writerow([2, "Maria Clara", 9.9])
```

```
    csv_writer.writerow([3, "Enzo Gabriel", 9.2])
```

"Linha de Cabeçalho"

Delimitador pode ser personalizado

```
alunos.csv
1  Nro;Nome;Nota
2  1;Pedro Carlos;9.5
3  2;Maria Clara;9.9
4  3;Enzo Gabriel;9.2
```

Exemplo de Gravação de Dados

- * Salvando dados de um dicionário no arquivo:

```
import csv
```

```
aluno1 = {"Nro": 1, "Nome": "Pedro Carlos", "Nota": 9.5}
```

```
aluno2 = {"Nro": 2, "Nome": "Maria Clara", "Nota": 9.9}
```

```
aluno3 = {"Nro": 3, "Nome": "Enzo Gabriel", "Nota": 9.2}
```

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

```
cabecalho = ['Nro', 'Nome', 'Nota']
```

```
with open("alunos_dict.csv", "w", newline='', encoding="utf-8") as arquivo:  
    csv_writer = csv.DictWriter(arquivo, delimiter="$", fieldnames=cabecalho)  
    csv_writer.writeheader()  
    csv_writer.writerows(alunos)
```

Grava a lista de cabeçalhos

Grava o dicionário