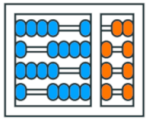


Instituto de Computação

UNIVERSIDADE ESTADUAL DE CAMPINAS

UNICAMP

MC102 - Aula 26 (Extra)

Arquivos e Expressões Regulares

Algoritmos e Programação de Computadores

Turmas

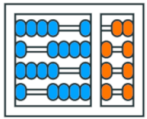
OVXZ

Prof. Lise R. R. Navarrete

lrommel@ic.unicamp.br

Quinta-feira, 30 de junho de 2022

19:00h - 21:00h (CB06)



Instituto de Computação

UNIVERSIDADE ESTADUAL DE CAMPINAS

UNICAMP

MC102 – Algoritmos e Programação de Computadores

Turmas

OVXZ

https://ic.unicamp.br/~mc102/

Site da Coordenação de MC102

Aulas teóricas:

Terça-feira, 21:00h - 23:00h (CB06)

Quinta-feira, 19:00h - 21:00h (CB06)

Conteúdo

- Arquivos
  - Dados em Memoria
  - Armazenamento permanente de dados
  - Tipos de Arquivos
  - Trabalhando com arquivos de texto em Python
  - Exercícios

- Expressões Regulares
  - Regras
  - Classes de Caracteres
  - Biblioteca re
  - Exercícios

Arquivos

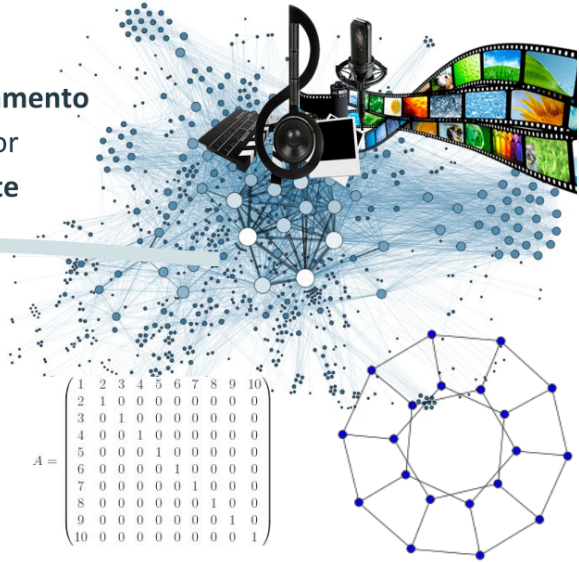
Arquivos

Dados em Memoria

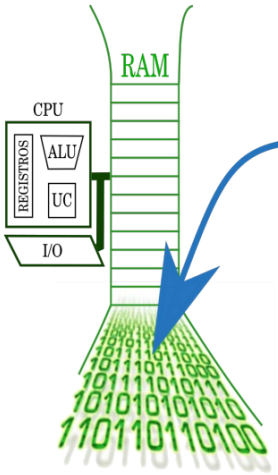
Estruturas de dados

São formas de **organizar o armazenamento** de **dados na memória** do computador com o objetivo de fazer mais **eficiente** seu uso.

vetores
listas
pilhas
....



$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 5 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 7 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 8 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$



Estruturas de dados

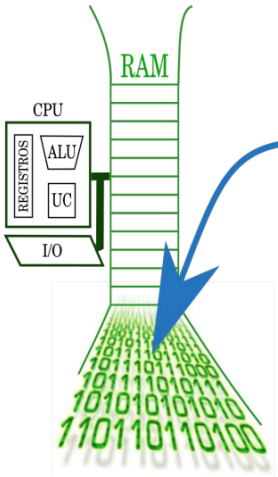
DADOS

um inteiro
um real
um pixel
um som

Programa

```
// criar

// inicializar
A = 4
// operação, pela lógica,  correta
A = A + 1
// mostrar o resultado
print (A)
```



Estruturas de dados

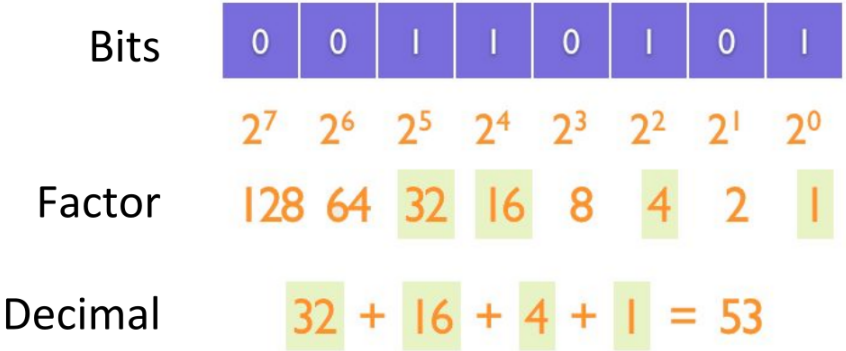
DADOS

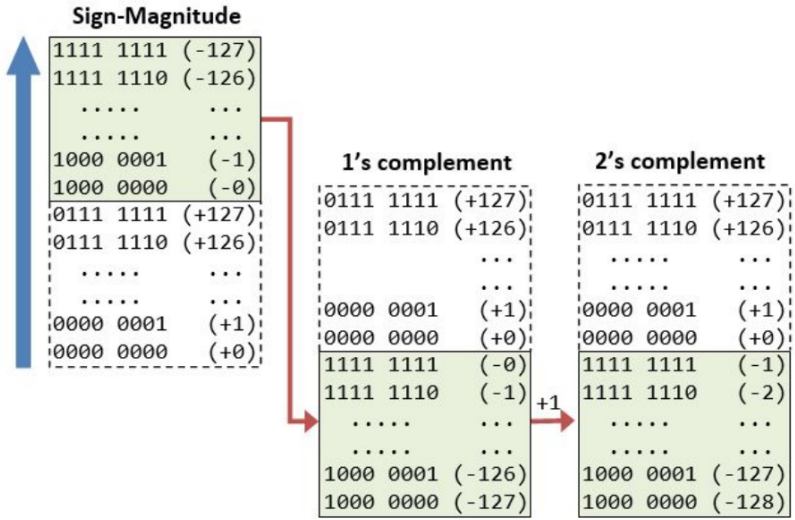
um inteiro
um real
um pixel
um som

Variável
(na linguagem de programação)

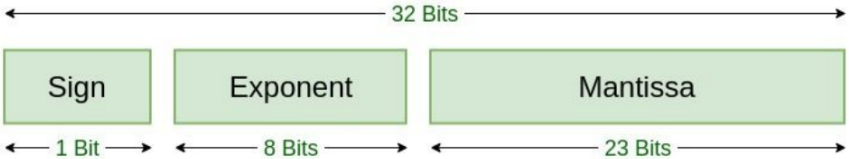
é um **mecanismo**
para poder **manipular**
uma **região de memória**
onde será **armazenado**
um **dado**.

inteiros


Bits
0 0 1 1 0 1 0 1
2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0
Factor
128 64 32 16 8 4 2 1
Decimal
32 + 16 + 4 + 1 = 53



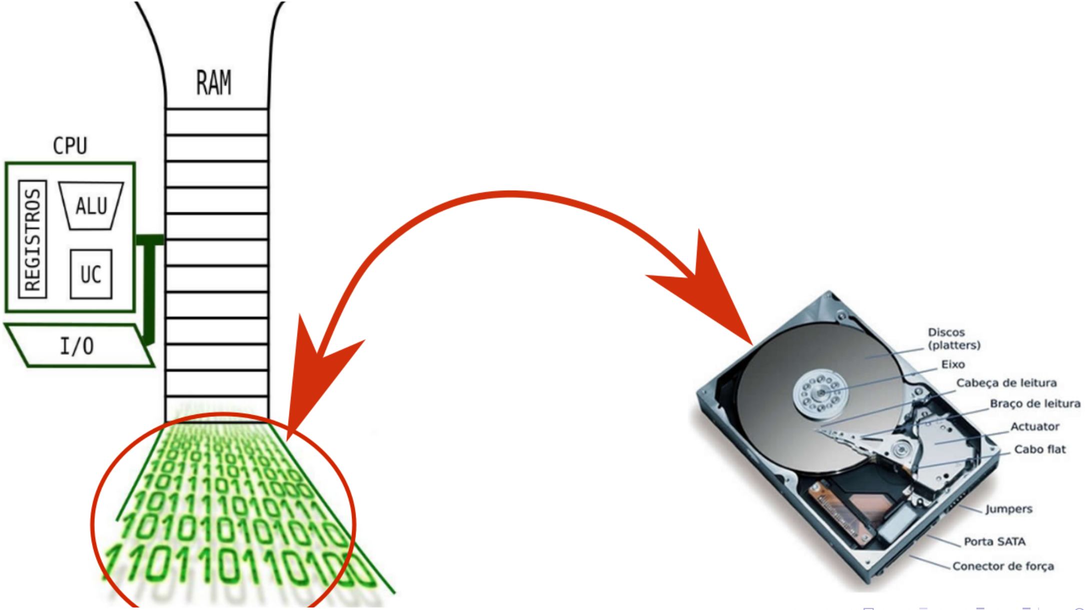
Ver: https://www3.ntu.edu.sg/home/ehchua/programming/java/datarepresentation.html

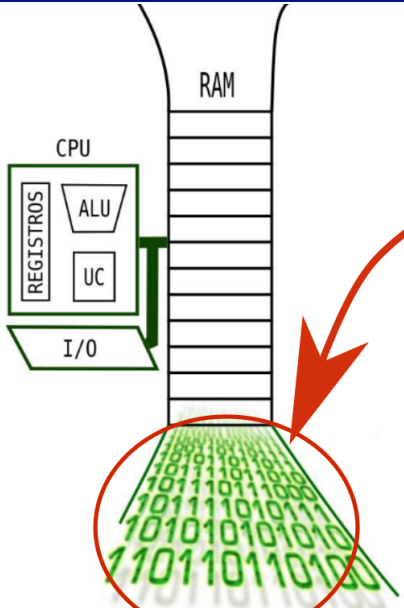


Single Precision
IEEE 754 Floating-Point Standard

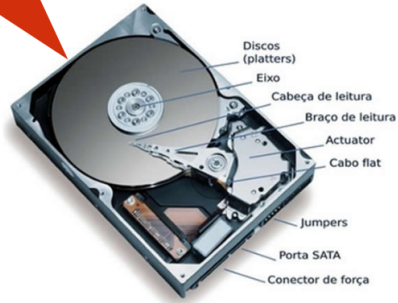
Arquivos

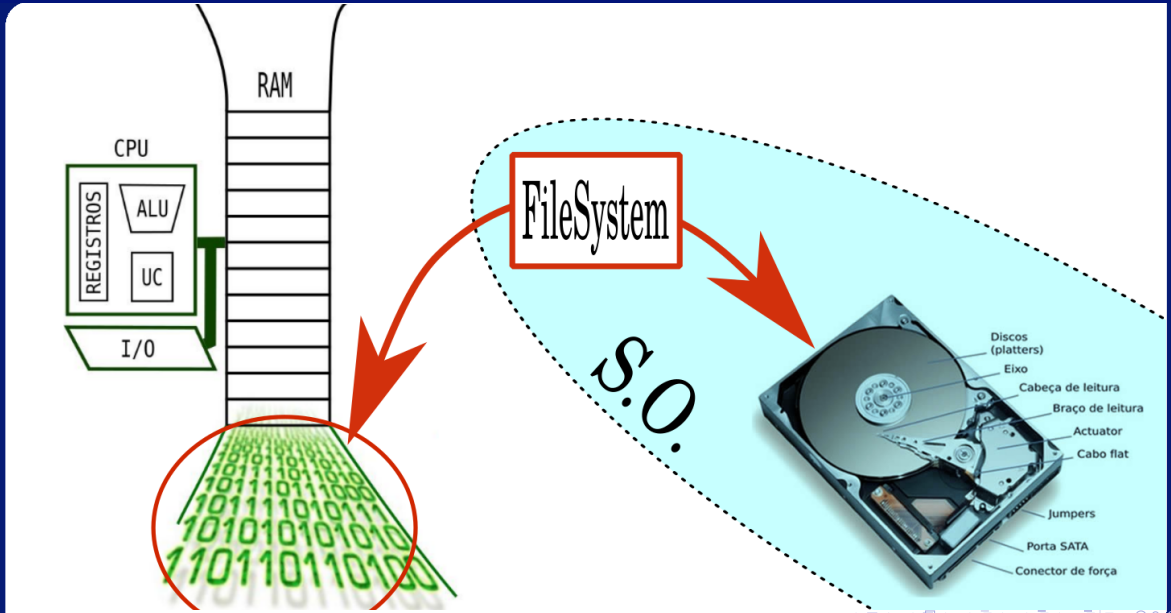
Armazenamento permanente de dados

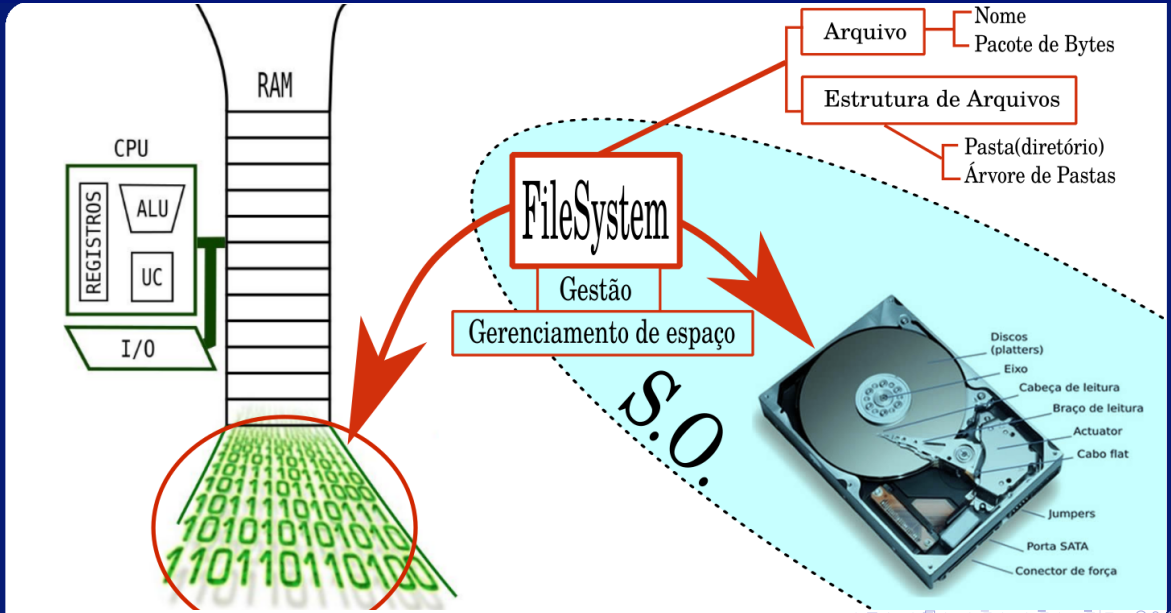

RAM
CPU
REGISTROS
ALU
UC
I/O
Discos
(platters)
Eixo
Cabeça de leitura
Braço de leitura
Actuator
Cabo flat
Jumpers
Porta SATA
Conector de força

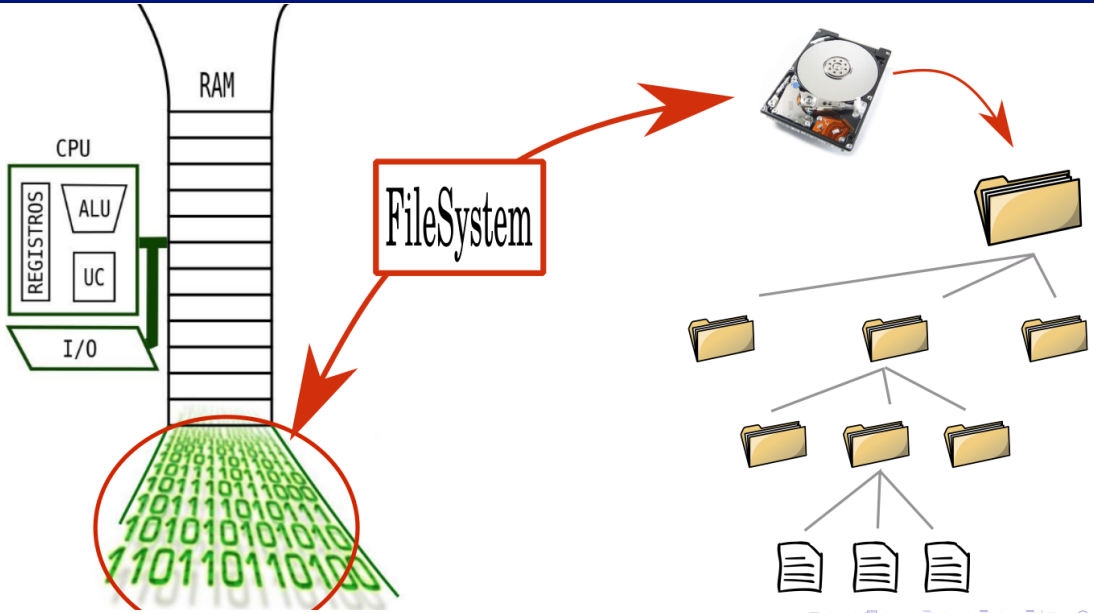

RAM
CPU
REGISTROS
ALU
UC
I/O

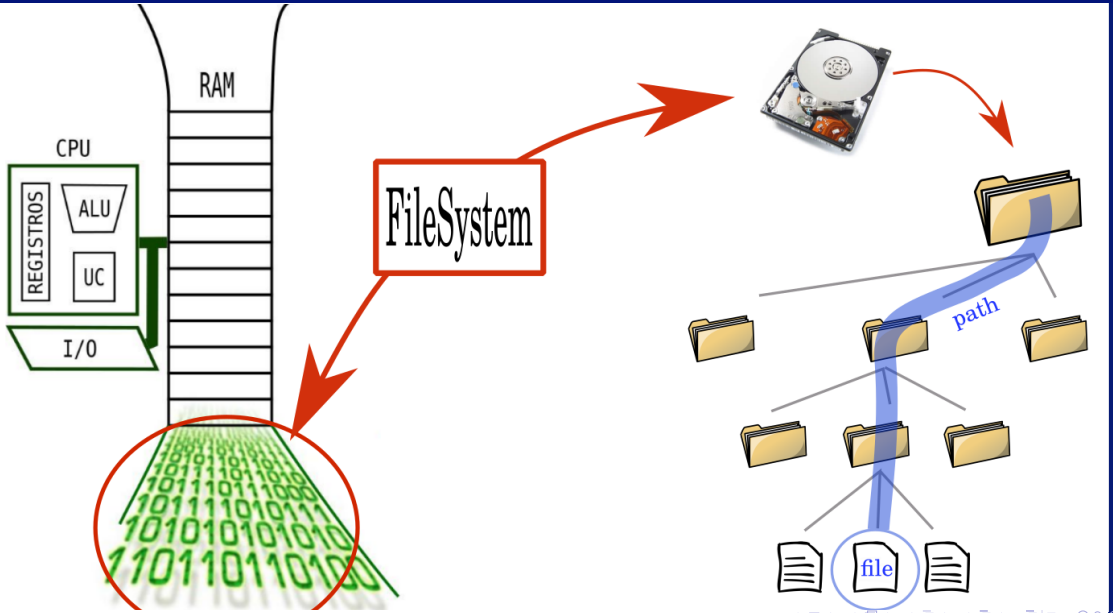
FileSystem

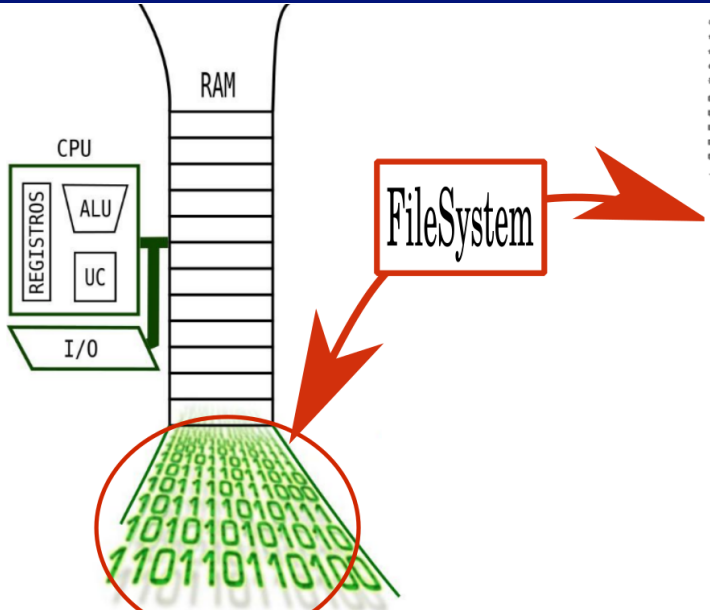

Discos (platters)
Eixo
Cabeça de leitura
Braço de leitura
Actuator
Cabo flat
Jumpers
Porta SATA
Conector de força


RAM
CPU
REGISTROS
ALU
UC
I/O
FileSystem
S.O.
Discos (platters)
Eixo
Cabeça de leitura
Braço de leitura
Actuator
Cabo flat
Jumpers
Porta SATA
Conector de força


RAM
CPU
REGISTROS
ALU
UC
I/O
Arquivo
Nome
Pacote de Bytes
Estrutura de Arquivos
Pasta(diretório)
Árvore de Pastas
FileSystem
Gestão
Gerenciamento de espaço
S.O.
Discos (platters)
Eixo
Cabeça de leitura
Braço de leitura
Actuator
Cabo flat
Jumpers
Porta SATA
Conector de força


RAM
CPU
REGISTROS
ALU
UC
I/O
FileSystem


RAM
CPU
REGISTROS
ALU
UC
I/O
FileSystem
path
file


RAM
CPU
REGISTROS
ALU
UC
I/O
FileSystem

Arquivo de texto

ou arquivo de **texto plano**, é um arquivo composto por **Bytes** que representam diretamente **símbolos de texto**, sob alguma **codificação** base:

ASCII,

UTF-8,

UNICODE,

...

Low Ascii

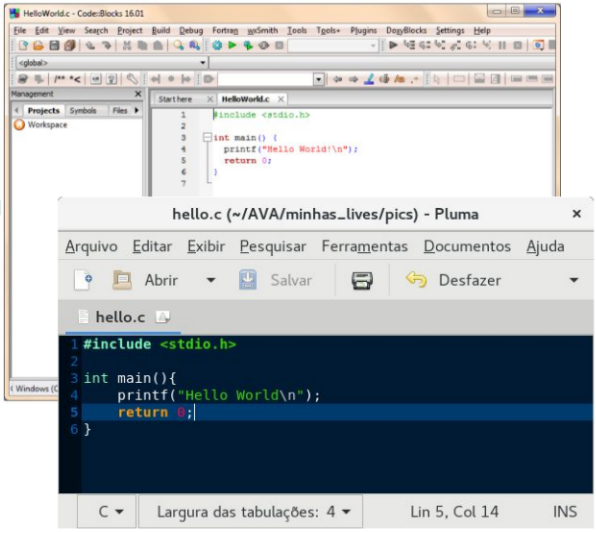
000:	013:♪	026:→	039:'	052:4	065:A	078:N	091:[	104:h	117:u
001:☺	014:♫	027:←	040:(	053:5	066:B	079:O	092:\	105:i	118:v
002:☻	015:☼	028:∟	041:)	054:6	067:C	080:P	093:]	106:j	119:w
003:♥	016:►	029:↔	042:*	055:7	068:D	081:Q	094:^	107:k	120:x
004:♦	017:◄	030:▲	043:+	056:8	069:E	082:R	095:_	108:l	121:y
005:♣	018:‼	031:▼	044:,	057:9	070:F	083:S	096:`	109:m	122:z
006:♠	019:‼	032:	045:-	058::	071:G	084:T	097:a	110:n	123:{
007:•	020:¶	033:!	046:.	059:;	072:H	085:U	098:b	111:o	124:|
008:◘	021:§	034:"	047:/	060:<	073:I	086:V	099:c	112:p	125:}
009:○	022:▬	035:#	048:0	061:=	074:J	087:W	100:d	113:q	126:~
010:◙	023:↨	036:$	049:1	062:>	075:K	088:X	101:e	114:r	127:⌂
011:♂	024:↑	037:%	050:2	063:?	076:L	089:Y	102:f	115:s	
012:♀	025:↓	038:&	051:3	064:@	077:M	090:Z	103:g	116:t	

High Ascii

128:Ç	141:ì	154:Ü	167:º	180:┤	193:┴	206:╬	219:█	232:Φ	245:J
129:ü	142:Ä	155:¢	168:¿	181:╡	194:┬	207:╧	220:▄	233:Θ	246:÷
130:é	143:Å	156:£	169:⌐	182:╢	195:├	208:╨	221:▌	234:Ω	247:≈
131:â	144:É	157:¥	170:¬	183:╖	196:─	209:╤	222:▐	235:δ	248:°
132:ä	145:æ	158:₧	171:½	184:╕	197:┼	210:╥	223:▀	236:∞	249:·
133:à	146:Æ	159:ƒ	172:¼	185:╣	198:╞	211:╙	224:α	237:φ	250:·
134:å	147:ô	160:á	173:¡	186:║	199:╟	212:╘	225:ß	238:ε	251:√
135:ç	148:ö	161:í	174:«	187:╗	200:╚	213:╒	226:Γ	239:∩	252:ⁿ
136:ê	149:ò	162:ó	175:»	188:╝	201:╔	214:╓	227:π	240:≡	253:²
137:ë	150:û	163:ú	176:▒	189:╜	202:╩	215:╫	228:Σ	241:±	254:■
138:è	151:ù	164:ñ	177:▓	190:╛	203:╦	216:╪	229:σ	242:≥	255:
139:ï	152:ÿ	165:Ñ	178:│	191:┐	204:╠	217:┘	230:μ	243:≤	
140:î	153:Ö	166:ª	179:┤	192:└	205:═	218:┌	231:τ	244:⌠	

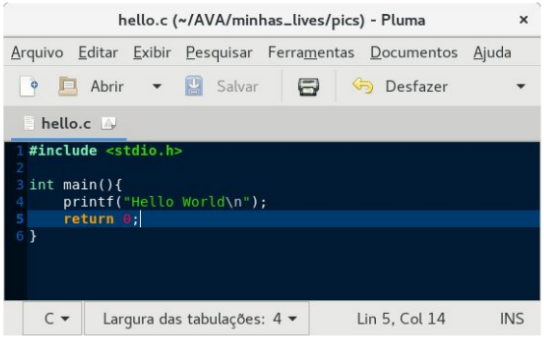
Editor de texto

É uma **ferramenta computacional** (ou **programa**) que permite **criar** e **modificar arquivos de texto**.



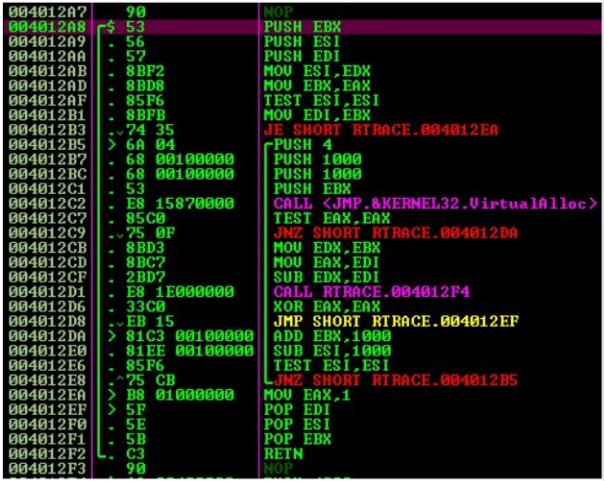
Arquivo fonte

É um **arquivo de texto** que representa um **programa**.
Está escrito seguindo a **sintaxe** de alguma **linguagem de programação**.



Arquivo executável

É um arquivo que contém **Bytes** que representam diretamente **instruções na linguagem maquina** e que pode ser **executado diretamente** pelo **sistema operacional**.



Arquivos

Tipos de Arquivos

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- Arquivos podem ter o mais variado conteúdo, mas do ponto de vista dos programas existem apenas dois tipos de arquivos:
  - Arquivo texto: Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples. Exemplos: código fonte Python, documento texto simples, páginas HTML (*HyperText Markup Language*), arquivos CSV (*Comma-Separated Values*).
  - Arquivo binário: Sequência de bits sujeita às convenções do programa que o gerou, não legíveis diretamente por um humano. Exemplos: arquivos executáveis, arquivos compactados, documentos do Word.

Arquivos

Trabalhando com arquivos de texto em Python

Apertura de arquivos:

`open()`

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- Para trabalharmos com arquivos devemos abri-lo e associá-lo com uma variável utilizando a função **open**.
- A função **open** recebe como parâmetros o nome do arquivo (incluindo o caminho até ele) e o modo desejado para abrir o arquivo.
  - `r` Leitura: nesse modo podemos somente ler os dados do arquivo.
  - `w` Escrita: nesse modo podemos escrever/modificar os dados do arquivo.
  - `r+` Leitura/escrita: nesse modo podemos ler e também escrever/modificar os dados do arquivo.
  - `a` Anexação: nesse modo podemos somente adicionar novos dados no final do arquivo.

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- Ao tentar abrir um arquivo inexistente para leitura (`r`), a função `open` gerará um erro.
- Ao abrir um arquivo para escrita (`w`), seu conteúdo é primeiramente apagado. Se o arquivo não existir, um novo arquivo será criado.
- Ao tentar abrir um arquivo inexistente para leitura leitura/escrita (`r+`), a função `open` gerará um erro. Se o arquivo existir, seu conteúdo não será primeiramente apagado.
- Ao tentar abrir um arquivo inexistente para anexação (`a`), um novo arquivo será criado.

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- Exemplo:

```
arq = open("teste1.txt", "r")
# abrindo o arquivo teste1.txt com modo leitura
arq = open("teste2.txt", "w")
# abrindo o arquivo teste2.txt com modo escrita
arq = open("teste3.txt", "r+")
# abrindo o arquivo teste3.txt com modo leitura/escrita
arq = open("teste4.txt", "a")
# abrindo o arquivo teste4.txt com modo anexação
```

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- Exemplo:

```
arq = open("MC102/teste.txt", "r")
# abrindo o arquivo teste.txt no diretório MC102
# usando modo de leitura
arq = open("arqs/arquivo.log", "r+")
# abrindo o arquivo arquivo.log no diretório arqs
# usando modo de leitura/escrita
```

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- A função `open` retorna um objeto do tipo `_io.TextIOWrapper` que possui métodos para ler e escrever em um arquivo.

```
1 arq = open("teste.txt", "r")
2 print(arq)
3 # <_io.TextIOWrapper name='teste.txt' mode='r'
4 #  encoding='UTF-8'>
5 print(type(arq))
6 # <class '_io.TextIOWrapper'>
```

Leitura de arquivos:

`read()`

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- O método **read** é utilizado para ler os dados de um arquivo.
- O método **read** recebe como parâmetro o número de caracteres que devem ser lidos.
- O método **read** retorna uma string compatível com a quantidade de caracteres especificados.
- Caso a quantidade de caracteres não seja especificada, o método **read** irá retornar o conteúdo completo do arquivo.
- Para utilizar o método **read**, o arquivo deve ser aberto no modo de leitura (**r**) ou leitura/escrita (**r+**).
- Considere o arquivo **teste.txt** com o seguinte conteúdo:

```
1 MC102
2 Unicamp - Python
```

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- Lendo o arquivo `teste.txt`:

```
arq = open("teste.txt", "r")
texto = arq.read()
print(texto, end = "")
# MC102
# Unicamp - Python
```

- Lendo os 5 primeiros caracteres do arquivo `teste.txt`:

```
arq = open("teste.txt", "r")
texto = arq.read(5)
print(texto)
# MC102
```

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- Quando um arquivo é aberto, um indicador de posição no arquivo é criado, e este recebe a posição do início do arquivo.
- Para cada dado lido ou escrito no arquivo, este indicador de posição é automaticamente incrementado para a próxima posição do arquivo.
- O método `read` retorna uma string vazia caso o indicador de posição esteja no fim do arquivo.

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- Exemplo de como ler os dados de um arquivo caractere por caractere:

```
arq = open("teste.txt", "r")
texto = ""
c = arq.read(1)

while c:
  texto = texto + c
  c = arq.read(1)

print(texto, end = "")
# MC102
# Unicamp - Python
```

Leitura de arquivos:

linha por linha

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- O método `readline` retorna uma string referente a uma linha do arquivo.
- Similar ao método `read`, o método `readline` retorna uma string vazia caso o indicador de posição esteja no fim do arquivo.
- Para utilizar o método `readline`, o arquivo deve ser aberto bo modo de leitura (`r`) ou leitura/escrita (`r+`).

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- Exemplo de como ler os dados de um arquivo linha por linha:

```
arq = open("teste.txt", "r")
linha = arq.readline()

while linha:
  print(linha, end = "")
  linha = arq.readline()

# MC102
# Unicamp - Python
```

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- Outra forma de ler os dados de um arquivo linha por linha:

```
arq = open("teste.txt", "r")

for linha in arq:
  print(linha, end = "")

# MC102
# Unicamp - Python
```

Leitura de arquivos:

`tell()`, `seek()`

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- O método `tell` retorna a posição atual no arquivo.
- Podemos alterar o indicador de posição de um arquivo utilizando o método `seek`.
- O método `seek` recebe a nova posição, em relação ao início do arquivo.
- Podemos usar os métodos `seek` e `tell` combinados para alterar a posição do arquivo com base na posição atual.

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- Lendo a primeira linha do arquivo `teste.txt` duas vezes:

```
1  arq = open("teste.txt", "r")
2
3  linha = arq.readline()
4  print(linha, end = "")
5  # MC102
6
7  arq.seek(0) # Voltando para o início do arquivo
8
9  linha = arq.readline()
10 print(linha, end = "")
11 # MC102
12
13 linha = arq.readline()
14 print(linha, end = "")
15 # Unicamp - Python
```

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- Avançando e retrocedendo num arquivo:

```
arq = open("teste.txt", "r")

linha = arq.readline()
print(linha, end = "")
# MC102
print("Posição =", arq.tell())
# Posição = 6

arq.seek(arq.tell() - 3)

linha = arq.readline()
print(linha, end = "")
# 02
print("Posição =", arq.tell())
# Posição = 6
```

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- Avançando e retrocedendo num arquivo:

```
arq = open("teste.txt", "r")

linha = arq.readline()
print(linha, end = "")
# MC102
print("Posição =", arq.tell())
# Posição = 6

arq.seek(arq.tell() + 3)

linha = arq.readline()
print(linha, end = "")
# camp - Python
print("Posição =", arq.tell())
# Posição = 23
```

Escrita de arquivos:

`write()`

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- Para escrevermos em um arquivo utilizamos o método `write`.
- O método `write` recebe como parâmetro a string que será escrita no arquivo.
- Para utilizar o método `write`, o arquivo deve ser aberto com o modo de escrita (`w`), leitura/escrita (`r+`) ou anexação (`a`).

fechar um arquivo:

```
close()
```

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- O método `close` deve sempre ser usado para fechar um arquivo que foi aberto.
- Quando escrevemos dados em um arquivo, este comando garante que os dados serão efetivamente escritos no arquivo.
- Ele também libera recursos que são alocados para manter a associação da variável com o arquivo.

Exemplo: Criando um arquivo

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- Criando um arquivo `teste.txt`:

```
1  arq = open("teste.txt", "w")
2  arq.write("Hello World!\n")
3  arq.write("Hello World!\n")
4  arq.close()
5
6  arq = open("teste.txt", "r")
7  texto = arq.read()
8  arq.close()
9
10 print(texto, end = "")
11 # Hello World!
12 # Hello World!
```

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- Adicionando mais dados no arquivo `teste.txt`:

```
arq = open("teste.txt", "a")
arq.write("MC102\n")
arq.write("Unicamp - Python\n")
arq.close()

arq = open("teste.txt", "r")
texto = arq.read()
arq.close()

print(texto, end = "")
# Hello World!
# Hello World!
# MC102
# Unicamp - Python
```

Escrita de arquivos:

`print()`

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

- A função **print** também pode ser utilizada para escrever dados em um arquivo.
- Para isso, basta utilizar o parâmetro **file**, indicando em qual arquivo, adequadamente aberto, a mensagem deve ser escrita.
- Exemplo:

```
1  arq = open("teste.txt", "w")
2  print("Utilizando a função print", file = arq)
3  arq.close()
4
5  arq = open("teste.txt", "r")
6  texto = arq.read()
7  arq.close()
8
9  print(texto)
10 # Utilizando a função print
```

Tamanho de um arquivo

log.txt filesize.py

filesize.py > ...

```python
import os

size = os.path.getsize('log.txt')      #  usando "getsize"
print('Tamanho ', size, 'bytes')

stats = os.stat('log.txt')             #  usando "stats"
print('Tamanho ', stats.st_size, 'bytes')

f = open('log.txt')                    #  usando "open, seek e tell"
f.seek(0, os.SEEK_END)                 #  va até o final do arquivo
print('Tamanho ', f.tell(), 'bytes')  # lê a posição com tell
```

```
$ python3 filesize.py
Tamanho  662 bytes
Tamanho  662 bytes
Tamanho  662 bytes
$
```

Arquivos

Exercícios

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

Nos dois exercícios abaixo considere a existência de um arquivo `financeiro.log` com os registros financeiros de uma empresa, com o seguinte conteúdo inicial:

```
1000 capital inicial
-500 compra de matéria-prima
-200 mão de obra
400 venda do primeiro lote
300 venda do segundo lote
-300 aluguel da fábrica
```

1. Escreva um programa que leia o arquivo `financeiro.log` e imprima o saldo financeiro da empresa.
2. Escreva um programa que leia um valor e uma descrição, e inclua uma nova linha no arquivo `financeiro.log`, conforme o formato ilustrado acima.

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

```
arq = open("financeiro.txt", "r")
saldo = 0

for linha in arq:
  saldo = saldo + int(linha.split()[0])

print("Saldo =", saldo)

arq.close()
```

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

```
arq = open("financeiro.txt", "a")

valor = input()
descrição = input()

print(valor, descrição, file = arq)

arq.close()
```

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

3. Escreva um programa que leia o nome de dois arquivos e duas strings. Seu programa deve ler o conteúdo do primeiro arquivo e escrevê-lo no segundo arquivo, substituindo todas as ocorrências da primeira pela segunda string.

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

```
arq1 = input()
arq2 = input()
str1 = input()
str2 = input()

entrada = open(arq1, "r")
saída = open(arq2, "w")

for linha in entrada:
  nova = linha.replace(str1, str2)
  print(nova, end = "", file = saída)

entrada.close()
saída.close()
```

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

4. Escreva um programa que, dado o nome de um arquivo no formato CSV (*Comma-Separated Values*) e uma string representando o separador, leia e armazene o conteúdo do arquivo numa lista bidimensional.

CSV com separador = ","

```
Bulbasaur,0.7m,6.9kg,Seed,Overgrow
Charmander,0.6m,8.5kg,Lizard,Blaze
Squirtle,0.5m,9.0kg,Tyny Turtle,Torrent
Pikachu,0.4m,6.0kg,Mouse,Static
Jigglypuff,0.5m,5.5kg,Ballon,Cute Charm
Snorlax,2.1m,460.0kg,Sleeping,Immunity
```

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

```
csv = input()
sep = input()

arq = open(csv, "r")

m = []
for linha in arq:
  linha = linha.replace("\n", "")
  m.append(linha.split(sep))

arq.close()

print(m)
```

https://ic.unicamp.br/~mc102/aulas/aula14.pdf

```
[['Bulbasaur', '0.7m', '6.9kg', 'Seed', 'Overgrow'],
 ['Charmander', '0.6m', '8.5kg', 'Lizard', 'Blaze'],
 ['Squirtle', '0.5m', '9.0kg', 'Tyny Turtle', 'Torrent'],
 ['Pikachu', '0.4m', '6.0kg', 'Mouse', 'Static'],
 ['Jigglypuff', '0.5m', '5.5kg', 'Ballon', 'Cute Charm'],
 ['Snorlax', '2.1m', '460.0kg', 'Sleeping', 'Immunity']]
```

Expressões Regulares

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Expressões regulares são formas concisas de descrever um conjunto de strings que satisfazem um determinado padrão.
- Por exemplo:
  - Podemos criar uma expressão regular para descrever todas as strings que representam datas no formato `dd/dd/dddd`, onde `d` é um dígito qualquer.
  - Podemos verificar se uma string contém um número de telefone, descrito por uma expressão regular.

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Note que números de telefones e datas podem ser escritos em vários formatos diferentes.
- Números de telefones:
  - 19-91234-5678
  - (019) 91234 5678
  - (19)912345678
- Datas:
  - 09/10/2019
  - 09-10-19
  - 2019-10-09

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Expressões regulares constituem uma mini-linguagem, que permite especificar as regras de construção de um conjunto de strings.
- Essa mini-linguagem de especificação é muito parecida entre as diferentes linguagens de programação que possuem o conceito de expressões regulares (também chamado de RE, REGEX ou RegExp).
- Assim, aprender a escrever expressões regulares em Python será útil para descrever expressões regulares em outras linguagens de programação.
- Expressões regulares são frequentemente utilizadas para encontrar ou extrair informações de textos (*text parsing*).

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Exemplo de expressão regular:

```
'\d+\\\\'
```

- Essa expressão regular representa uma sequência de um ou mais dígitos seguidos por uma contrabarra (\).
- Vamos aprender regras de como escrever e usar expressões regulares.
- Geralmente escrevemos expressões regular iniciando com um caractere `r` para indicar uma `raw string`, ou seja, uma string onde o caractere \ é tratado como um caractere normal.
- Assim, a expressão regular resultante seria:

```
r'\d+\\'
```

Expressões Regulares

Regras

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Letras e números em uma expressão regular representam a si próprios.
- Assim a expressão regular `r'Python'` representa apenas a string `'Python'`.
- Os caracteres especiais (chamados de meta-caracteres) são:

```
. ^ $ * + ? \ | { } [ ] ( )
```

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- `.` um caractere qualquer.
- `^` o início da string.
- `$` o fim da string.
- `?` repetir zero ou uma vez.
- `*` repetir zero ou mais vezes.
- `+` repetir uma ou mais vezes.
- `\` usado para indicar caracteres especiais.

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

`[]` indica um conjunto de caracteres.

- `r'[0-9]'`: um dígito.
- `r'[^0-9]'`: um caractere que não é um dígito.
- `r'[a-z]'`: uma letra minúscula de a até z.
- `r'[A-Z]'`: uma letra maiúscula de A até Z.
- `r'[a-zA-Z]*'`: zero ou mais letras.
- `r'[ACTG]+'`: uma sequência de DNA.

`{}` indica a quantidade de vezes que o padrão será repetido.

- `r'[0-9]{2}'`: dois dígitos.
- `r'[a-z]{3}'`: três letras minúsculas.
- `r'[A-Z]{2,3}'`: duas ou três letras maiúsculas.
- `r'.{4,5}'`: quatro ou cinco caracteres quaisquer.
- `r'[01]{3,}'`: pelo menos três bits.
- `r'[0-9]{,6}'`: no máximo seis dígitos.

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

`()` indica um grupo em uma expressão regular.

- `r'([0-9]{3}\.){2}[0-9]{3}-[0-9]{2}'`: um CPF.
- `r'([a-z]+, )*[a-z]+'`: uma sequência de uma ou mais palavras separadas por vírgulas (e espaços).

`|` similar ao operador lógico `or` para expressões regulares.

- `r'U(nicamp|nesp|SP)'`: uma das 3 universidades paulistas.
- `r'([0-9]{3}|[a-z]{4})'`: uma sequência de três dígitos ou uma sequência de quatro letras minúsculas.

Expressões Regulares

Classes de Caracteres

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Python possui algumas classes pré-definidas de caracteres:
  - `\d` um dígito, ou seja, `[0-9]`.
  - `\D` o complemento de `\d`, ou seja, `[^0-9]`.
  - `\s` um espaço em branco, ou seja, a `[ \t\n\r\f\v]`.
  - `\S` o complemento de `\s`, ou seja, `[^ \t\n\r\f\v]`.
  - `\w` um caractere alfanumérico, ou seja, `[a-zA-Z0-9]`.
  - `\W` o complemento de `\w`, ou seja, `[^a-zA-Z0-9]`.

Expressões Regulares

Biblioteca re

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Em Python, expressões regulares são implementadas pela biblioteca `re`.
- Sendo assim, para usar expressões regulares precisamos importar a biblioteca `re`:

```
1 import re
```

- Documentação da biblioteca `re`: `https://docs.python.org/3/library/re.html`

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- A principal função da biblioteca `re` é a `search`.
- Dada uma expressão regular e uma string, a função `search` busca na string a primeira ocorrência de uma substring com o padrão especificado pela expressão regular.
- Se o padrão especificado pela expressão regular for encontrado, a função `search` retornará um objeto do tipo `Match`, caso contrário retornará `None`.
- Objetos do tipo `Match` possuem dois métodos:
  - `span`: retorna uma tupla com o local na string (posição inicial, posição final) onde a expressão regular foi encontrada.
  - `group`: retorna a substring encontrada.

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Exemplo de uso da função `search`:

```python
import re
texto = "Algoritmos e Programação de Computadores"
result = re.search(r'(\w*)ama(\w*)', texto)
print(type(result))
# <class 're.Match'>
print(result.group())
# Programação
print(result.span())
# (13, 24)
print(re.search(r'^\w*', texto))
# <re.Match object; span=(0, 10), match='Algoritmos'>
print(re.search(r'\w*$', texto))
# <re.Match object; span=(28, 40), match='Computadores'>
print(re.search(r'(^|\s)\w{3,9}(\s|$)', texto))
# None
```

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Outro exemplo utilizando a função `search`:

```
1 import re
2 texto = "Algoritmos e Programação de Computadores"
3 result = re.search(r'\w+', texto)
4 print(result.group())
5 # Algoritmos
6 print(result.span())
7 # (0, 10)
```

- Note que a função `search` retorna apenas a primeira ocorrência do padrão especificado.

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Dada uma expressão regular e uma string, a função `findall` retorna uma lista com todas as ocorrências do padrão especificado pela expressão regular.
- Exemplo:

```
1 import re
2 texto = "Algoritmos e Programação de Computadores"
3 result = re.findall(r'\w+', texto)
4 print(result)
5 # ['Algoritmos', 'e', 'Programação', 'de', 'Computadores']
6 telefone = "(019) 91234-5678"
7 result = re.findall(r'[0-9]+', telefone)
8 print(result)
9 # ['019', '91234', '5678']
```

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Podemos construir uma expressão regular concatenando duas ou mais strings.
- Podemos usar o resultado das funções `search` e `findall` em expressões condicionais: `None` e `[]` são considerados `False`.

```
import re
texto = "Algoritmos e Programação de Computadores"
ini = "Algo"
meio = "ação"
fim = "dores"
regexp = r'^' + ini + r'.*' + meio + r'.*' + fim + r'$'
if re.search(regexp, texto):
  print("OK")
else:
  print("ERRO")
# OK
```

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Podemos construir uma expressão regular concatenando duas ou mais strings.
- Podemos usar o resultado das funções `search` e `findall` em expressões condicionais: `None` e `[]` são considerados `False`.

```
import re
texto = "Algoritmos e Programação de Computadores"
ini = "Algo"
meio = "ação"
fim = "dores"
regexp = r'^' + ini + r'.*' + meio + r'.*' + fim + r'$'
if re.findall(regexp, texto):
  print("OK")
else:
  print("ERRO")
# OK
```

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Expressões regulares podem ser utilizadas para dividir strings, similar ao método `split` visto na aula de strings.
- Dada uma expressão regular e uma string, a função `split` retorna uma lista com a divisão da string conforme especificado pela expressão regular.
- Exemplo:

```
import re
texto = "f1i1b2o3n5a8c13c21i"
letras = re.split(r'\d+', texto)
print(letras)
# ['f', 'i', 'b', 'o', 'n', 'a', 'c', 'c', 'i']
números = re.split(r'\D+', texto)
print(números)
# ['', '1', '1', '2', '3', '5', '8', '13', '21', '']
```

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Expressões regulares podem ser utilizadas para substituir substrings, similar ao método **replace** visto na aula de strings.
- Dados dois padrões (strings ou expressões regulares) e uma string, a função **sub** retorna uma string com a subtituição na string de toda ocorrência do primeiro padrão pelo segundo padrão.

```
1 import re
2 texto = "f1i1b2o3n5a8c13c21i"
3 letras = re.sub(r'\d+', "", texto)
4 print(letras)
5 # fibonacci
6 números = re.sub(r'(\D+)', ":", texto)
7 print(números)
8 # :1:1:2:3:5:8:13:21:
```

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Usando a função **sub**, podemos utilizar expressões regulares para indicar como a string será modificada, com base nos grupos da expressão regular (\1, \2, etc).
- Exemplo:

```
1  import re
2  data = "19/09/1975"
3  antigo = r'(\d{2})/(\d{2})/(\d{4})'
4  novo1 = r'\1-\2-\3'
5  data1 = re.sub(antigo, novo1, data)
6  print(data1)
7  # 19-09-1975
8  novo2 = r'\3/\2/\1'
9  data2 = re.sub(antigo, novo2, data)
10 print(data2)
11 # 1975/09/19
```

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Podemos referenciar os grupos dentro da própria expressão regular para construir padrões mais complexos.
- Exemplo:

```
import re
dna = "AGTTAGTGCACACACTGAGGTTC"
print(re.search(r'(G[ACTG]{2})(.*)\1', dna).group())
# GTTAGTGCACACACTGAGGTT
# 11122222222222222111
print(re.search(r'([ACTG]{2})(.*)\1(.*)\1', dna).group())
# AGTTAGTGCACACACTGAG
# 1122113333333333311
print(re.sub(r'([ACTG]{2})(.*)\1(.*)\1', r'\1\3\1\2\1',
      dna))
# AGTGCACACACTGAGTTAGGTTC
# 1133333333333112211----
```

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Podemos recuperar cada um dos grupos de uma expressão regular com a função `group`.
- Exemplo:

```
1  import re
2  texto = "Data de Nascimento: 19/09/1975"
3  result = re.search(r'(\d{2})/(\d{2})/(\d{4})', texto)
4  print(result.group())
5  # 19/09/1975
6  print("Dia:", result.group(1))
7  # Dia: 19
8  print("Mês:", result.group(2))
9  # Mês: 09
10 print("Ano:", result.group(3))
11 # Ano: 1975
12 print(result.group(1, 2, 3))
13 # ('19', '09', '1975')
```

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Por padrão, os operadores `+`, `*`, `?` e `{,}` são executados de forma gulosa, ou seja, eles tentam casar com o maior número possível de caracteres.
- Usando o caractere `?` na frente daqueles operadores, eles são executados de forma não gulosa.
- Exemplo:

```
1 import re
2 texto = "Algoritmos e Programação de Computadores"
3 print(re.search(r'o(.*)e(.*)o', texto).group())
4 # oritmos e Programação de Computado
5 print(re.search(r'o(.*)e(.*?)o', texto).group())
6 # oritmos e Programação de Co
7 print(re.search(r'o(.*?)e(.*?)o', texto).group())
8 # oritmos e Pro
```

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

- Por padrão, os operadores `+`, `*`, `?` e `{,}` são executados de forma gulosa, ou seja, eles tentam casar com o maior número possível de caracteres.
- Usando o caractere `?` na frente daqueles operadores, eles são executados de forma não gulosa.
- Exemplo:

```
1 import re
2 texto = "Removendo as <em>marcas</em> do <pre>texto</pre>."
3 print(re.sub(r'<.*>', "", texto))
4 # Removendo as .
5 print(re.sub(r'</.*>', "", texto))
6 # Removendo as <em>marcas.
7 print(re.sub(r'<.*?>', "", texto))
8 # Removendo as marcas do texto.
```

Expressões Regulares

Exercícios

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

1. Escreva um programa para determinar se uma string representa um número (inteiro ou real) válido. Exemplos de números válidos: 10, +5, −3, −10.3, 0.80, 2.8033.
2. Escreva um programa para determinar se uma string representa um número de telefone (fixo ou celular) válido. Exemplos de números de telefones válidos:
   - (19) 3123-4567
   - 193123-4567
   - (019)3123-4567
   - (19)31234567
   - 1931234567
   - (019) 91234 5678
   - (019)91234 5678
   - 019912345678
   - 19 91234 5678
   - 1991234 5678

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

3. Com base no exercício anterior, escreva uma função que recebe como parâmetro uma string que representa um número de telefone (fixo ou celular). Caso o número não seja válido, sua função deve retornar `None`. Caso contrário, ela deve retornar uma string no formato `(XX) XXXX-XXXX` (no caso de telefone fixo) ou `(XX) XXXXX-XXXX` (no caso de telefone celular), onde X representa um dígito do telefone.
4. Escreva um programa que, dada uma palavra e uma frase, verifique se as letras da palavra aparecem na frase, na mesma ordem, mas não necessariamente de forma consecutiva. Exemplos:
   - "palavra" e "capa, lata, livro e caderno"
   - "escondida" e "mar, pesca, ondas e bebidas"

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

```
import re
regexp = r'^[+-]?[0-9]+(\.[0-9]+)?$'

while True:
  número = input()

  if not(número):
    break

  if re.search(regexp, número):
    print("OK")
  else:
    print("ERRO")
```

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

```
import re
ddd = r'^(0?[1-9]{2}[- ]?|\(0?[0-9]{2}\) ?)'
tel = r'[2-9]?[0-9]{4}[- ]?[0-9]{4}$'
regexp = ddd + tel

while True:
  telefone = input()

  if not(telefone):
    break

  if re.search(regexp, telefone):
    print("OK")
  else:
    print("ERRO")
```

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

```
import re

def padroniza_telefone(telefone):
  ddd = r'^(0?[1-9]{2}[- ]?|\(0?[0-9]{2}\) ?)'
  tel = r'([2-9]?[0-9]{4})[- ]?([0-9]{4})$'
  regexp = ddd + tel

  if re.search(regexp, telefone):
    return None

  dígitos = re.sub(r'[^0-9]', "", telefone)
  grupos = r'^0?([0-9]{2})([0-9]{4,5})([0-9]{4})$'
  formato = r'(\1) \2-\3'

  return re.sub(grupos, formato, dígitos)
```

https://ic.unicamp.br/~mc102/aulas/aula15.pdf

```
import re

palavra = input()
frase = input()

regexp = ".*".join(list(palavra))

if re.search(regexp, frase):
  print("OK")
else:
  print("ERRO")
```

Perguntas ....

Referências

- Zanoni Dias, MC102, Algoritmos e Programação de Computadores, IC/UNICAMP, 2021. https://ic.unicamp.br/~mc102/

  Aula Introdutória [slides] [vídeo]
  Primeira Aula de Laboratório [slides] [vídeo]
  Python Básico: Tipos, Variáveis, Operadores, Entrada e Saída [slides] [vídeo]
  Comandos Condicionais [slides] [vídeo]
  Comandos de Repetição [slides] [vídeo]
  Listas e Tuplas [slides] [vídeo]
  Strings [slides] [vídeo]
  Dicionários [slides] [vídeo]
  Funções [slides] [vídeo]
  Objetos Multidimensionais [slides] [vídeo]
  Algoritmos de Ordenação [slides] [vídeo]
  Algoritmos de Busca [slides] [vídeo]
  Recursão [slides] [vídeo]
  Algoritmos de Ordenação Recursivos [slides] [vídeo]
  Arquivos [slides] [vídeo]
  Expressões Regulares [slides] [vídeo]
  Execução de Testes no Google Cloud Shell [slides] [vídeo]
  Numpy [slides] [vídeo]
  Pandas [slides] [vídeo]

- Panda - Cursos de Computação em Python (IME -USP) https://panda.ime.usp.br/

  Como Pensar Como um Cientista da Computação https://panda.ime.usp.br/pensepy/static/pensepy/
  Aulas de Introdução à Computação em Python https://panda.ime.usp.br/aulasPython/static/aulasPython/

- Fabio Kon, Introdução à Ciência da Computação com Python http://bit.ly/FabioKon/
- Socratica, Python Programming Tutorials http://bit.ly/SocraticaPython/
- Google - online editor for cloud-native applications (Python programming) https://shell.cloud.google.com/
- w3schools - Python Tutorial https://www.w3schools.com/python/
- Outros, citados nos Slides.