Strings

CIÊNCIA DA COMPUTAÇÃO

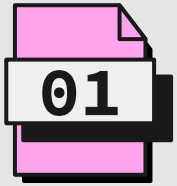
ATITUS EDUCAÇÃO

- Aula 10 -
Pensamento Computacional

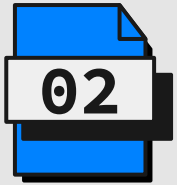
Prof. Me. Lucas R. C. Pessutto

ATITUS EDUCAÇÃO

Na aula de hoje...



Strings x Listas



Funções para lidar com strings

split / join
find / replace
Outras funções de strings

ALTUS EDUCAÇÃO

Recapitulando

* Variáveis são de um determinado tipo de acordo com o valor atribuído a elas em algum momento da execução do programa
* Em Python temos 4 tipos básicos principais:

Inteiro (int)

```
>>> i = 10
>>> type(i)
<class 'int'>
```

Decimal (float)

```
>>> f = 5.5
>>> type(f)
<class 'float'>
```

Texto (str)

```
>>> s = "texto"
>>> type(s)
<class 'str'>
```

Booleano (bool)

```
>>> b = True
>>> type(b)
<class 'bool'>
```

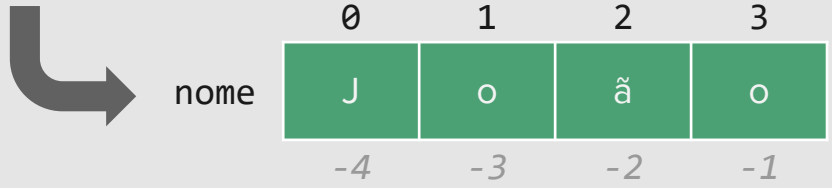
Nem tão básico assim!
O tipo `str` possui 45 métodos embutidos e é útil para resolver uma infinidade de problemas.

Strings x Listas

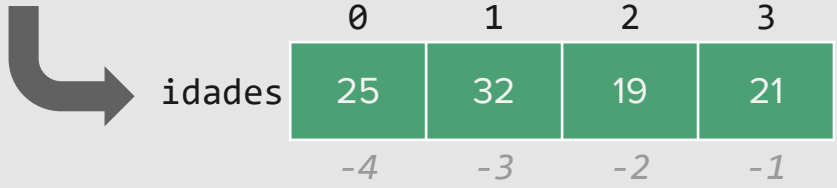
* Strings e listas são dois tipos de dados que compartilham uma série de similaridades e algumas diferenças.
* Vamos revisar as seguintes características e operações de listas que se aplicam também a strings destacando essas diferenças e similaridades:
  - → Criação
  - → Fatiamento
  - → Concatenação
  - → Iteração
  - → Mutabilidade

Criação

```
# Criando strings
>>> nome = "João"
```




```
# Criando listas
>>> idades = [25, 32, 19, 21]
```




```
# Criando strings com aspas simples
>>> outro_nome = 'Zé'
# Criando com três aspas (múltiplas linhas)
>>> estrofe = '''Minha terra tem palmeiras,
Onde canta o Sabiá;
As aves, que aqui gorjeiam,
Não gorjeiam como lá.'''
```

Fatiamento

```
# Fatiando strings
lang = "Python 3"

# Primeiras 6 letras
lang[:6]      'Python'

# Posição 6 até o fim
lang[6:]      ' 3'

# Do início menos a última letra
lang[:-1]     'Python '
```

```
# Fatiando listas
notas = [8.5, 7, 6, 10, 9.1]

# Primeiras 3 notas
notas[:3]     [8.5, 7, 6]

# Posição 3 até o fim
notas[3:]     [10, 9.1]

# Do início menos a última nota
notas[:-1]    [8.5, 7, 6, 10]
```

Tanto em listas quanto em strings, o fatiamento **gera uma cópia** parcial ou completa do objeto original.

Concatenação

```
# Concatenando strings
lang = "Python"
versao = "3"

# Concatenando lang e versao
lang + versao          'Python3'

# Agora com um espaço a mais
lang + " " + versao    'Python 3'
```

```
# Concatenando listas
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]

# Concatenando lista1 e lista2
lista1 + lista2 [1, 2, 3, 4, 5, 6]

# Agora com uma nova lista
[-2, -1, 0] + lista1
                [-2, -1, 0, 1, 2, 3]
```

Cuidado! Só se pode concatenar `str + str` e `list + list`. Não é permitido misturar.

Assim como o fatiamento, a concatenação também **não altera o conteúdo** das listas/strings envolvidas gerando **novos objetos**.

Iteração

```python
# Iterando sobre strings
nome = "Fibonacci"
for letra in nome:
    print(f"Letra: {letra}")
```

```
Letra: F
Letra: i
Letra: b
Letra: o
Letra: n
Letra: a
Letra: c
Letra: c
Letra: i
```

O operador `in` pode ser usado também para verificar a existência de um ou mais caracteres em uma string.

```python
# Iterando sobre listas
fibo = [1, 1, 2, 3, 5, 8]
for f in fibo:
    print(f"Item: {f}")
```

```
Item: 1
Item: 1
Item: 2
Item: 3
Item: 5
Item: 8
```

Também é possível iterar com `while` usando os **índices** da string.

Mutabilidade

```
# Strings são imutáveis

erro = "Springs"
erro[1] = "t"
```

```
...
TypeError: 'str' object does not
support item assignment
```

```
# Listas podem ser alteradadas
letras = ["a", "b", "c", "d"]
print(letras)
letras[3] = "f"
print(letras)
```

```
['a', 'b', 'c', 'd']
['a', 'b', 'c', 'f']
```

Além de suportar atribuição direta vimos que listas oferecem métodos como append, pop, e remove. Não é possível fazer esse tipo de operação com strings.

Métodos Específicos de Strings

* Além das características e operações que já discutimos, strings ainda oferecem uma série de métodos úteis para diversas situações.

* A documentação completa apresenta um total de 45 métodos: docs.python.org/pt-br/3/library/stdtypes.html#string-methods.

* Exploraremos alguns deles agora...

Dividindo Strings

* É possível dividir uma string utilizando um caractere separador (ou mais de um) com o método split:

```
str.split(sep)
```

* **Retorna uma lista** de palavras na string, usando `sep` como a string delimitadora. Documentação completa em docs.python.org/pt-br/3/library/stdtypes.html#str.split.

Dividindo Strings

* Podemos, por exemplo, separar uma data digitada pelo usuário com seguinte código:

```
data = input("Informe uma data (dd/mm/aaaa):")
dia, mes, ano = data.split("/")
```

A **atribuição múltipla** vai separar automaticamente os itens da lista gerada pelo `split`. O comprimento da lista gerada precisa ser exatamente três, nesse caso, ou um `ValueError` será gerado.

Unindo Strings

* O processo inverso à divisão com `split` é a união de uma lista de strings em uma única string com o método `join`:

```
str.join(lista)
```

* **Retorna a string** que é a **concatenação** das strings em `list` (ou outro iterável). O separador entre elementos é a string `str`. Documentação completa em docs.python.org/pt-br/3/library/stdtypes.html#str.join.

Dividindo Strings

* Por exemplo, se quiséssemos converter uma data informada no formato dd/mm/aaaa para aaaa-mm-dd poderíamos usar o código:

```
data = input("Informe uma data (dd/mm/aaaa):")
dia, mes, ano = data.split("/")
print("-".join([ano, mes, dia]))
```

Um `TypeError` será levantado se existirem quaisquer valores que **não sejam strings** na lista.

Problema 1: Idade

Enunciado de um problema:

Fazer um programa onde o usuário informa a **data do seu nascimento** e deseja saber quantos **anos, meses e dias de idade** ele tem. A data deve ser informada no formato dd/mm/aaaa.

```
Informe seu aniversário (dd/mm/aaaa): 11/11/2011
Você já viveu 4198 dias
Você possui 11 anos, 6 meses e 3 dias
```

Problema 1: Idade

Descobrindo a data atual do sistema:

```
import datetime

# Lê o aniversário do usuário
data = input("Informe seu aniversário (dd/mm/aaaa): ")
dia, mes, ano = data.split("/")
dia, mes, ano = int(dia), int(mes), int(ano)

# Obtém o dia atual
hoje = str(datetime.date.today())
```

Documenação do módulo `datetime`:
docs.python.org/pt-br/3/library/datetime.html.

Localizar Texto

* O método find permite encontrar o índice de um caractere (ou vários) na string:

```
str.find(sub, start, end)
```

* **Retorna o índice** mais baixo na string onde o substring `sub` é encontrado dentro da fatia `str[start:end]` (`start` e `end` são opcionais). Retorna -1 se `sub` não for localizado. Documentação completa em docs.python.org/pt-br/3/library/stdtypes.html#str.find.

Localizar Texto

* Por exemplo, podemos fazer um sistema de busca textual:

```
texto = '''Alan Mathison Turing (Londres, 23 de junho de 1912 – Wilmslow, Cheshire, 7 de junho de 1954) foi um matemático, cientista da computação ...
'''

busca = input("Informe o texto a ser buscado: ")
ini = int(input("Informe a posição de início: "))
fim = int(input("Informe a posição final: "))

pos = texto.find(busca, ini, fim)

if pos != -1:
    print(f"Substring {busca} encontrada em {pos}: [{texto[pos-10:pos+len(busca)+10]}]")
else:
    print("Valor não encontrado!")
```

Localizar Texto – Exercício

* Modifique o programa para que ele mostre **todas as ocorrências** da palavra no texto.

```
texto = ''' ... '''

busca = input("Informe o texto a ser buscado: ")
ini = int(input("Informe a posição de início: "))
fim = int(input("Informe a posição final: "))

pos = texto.find(busca, ini, fim)

if pos != -1:
    print(f"Substring {busca} encontrada em {pos}: [{texto[pos-10:pos+len(busca)+10]}]")
else:
    print("Valor não encontrado!")
```

Substituir Texto

* O método `replace` permite fazer alteração de uma string substituindo partes dela:

```
str.replace(old, new)
```

* **Retorna uma cópia** da string com as ocorrências da substring `old` substituídas por `new`. Documentação completa em docs.python.org/pt-br/3/library/stdtypes.html#str.replace.

Substituir Texto

* O método `replace` permite fazer alteração de uma string substituindo partes dela:

```
str.replace(old, new)
```

* **Retorna uma cópia** da string com as ocorrências da substring `old` substituídas por `new`. Documentação completa em docs.python.org/pt-br/3/library/stdtypes.html#str.replace.

Substituir Texto

```
string = "Springs"
nova_string = string.replace("p", "t")
print(nova_string)

animal = "arara"
resultado = animal.replace("ra", "-")
print(resultado)
```

```
Strings
a--
```

Outras funções para Strings

Método	Parâmetros	Descrição
upper	nenhum	Retorna string com todas as letras maiúsculas
lower	nenhum	Retorna string com todas as letras minúsculas
capitalize	nenhum	Retorna string com o primeiro caractere em maiúscula e o resto em minúsculas
strip	nenhum	Retorna um string removendo caracteres em branco do início e do fim
lstrip	nenhum	Retorna um string removendo caracteres em branco do início
rstrip	nenhum	Retorna um string removendo caracteres em branco do fim
count	Item	Retorna o número de ocorrências de item

Outras funções para Strings

Método	Parâmetros	Descrição
center	largura	Retorna um string centrado em um campo de tamanho largura
ljust	largura	Retorna um string justificado à esquerda em um campo de tamanho largura
rjust	largura	Retorna um string justificado à direita em um campo de tamanho largura
find	Item	Retorna o índice mais à esquerda onde o substring item é encontrado ou -1.
rfind	Item	Retorna o índice mais à direita onde o substring item é encontrado
index	Item	Como find, mas causa um erro de execução caso item não seja encontrado
rindex	Item	Como rfind, mas causa um erro de execução caso item não seja encontrado

Problema 2: Texto Embaralhado

Enunciado de um problema:

* Os seers hnumaos são czapaes de ler ttxeos onde as lreats das prlaavas esãto ehabdamaarls, catntono que a permiria e a úmtila lrtea da paalvra sejam pdeeravrass.
* Usando seus conhecimentos sobre strings em Python, faça um programa que leia uma frase e a escreva com as letras do meio embaralhadas, mantendo intactas a primeira e a última letra.
* Dica: Utilizar a função `random.shuffle` para fazer o embaralhamento das letras