Strings

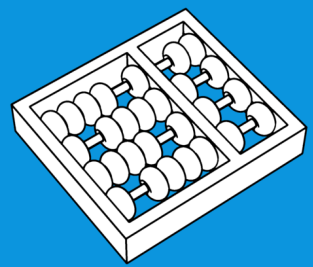
MC102 - Algoritmos e Programação de Computadores

04/25

13

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

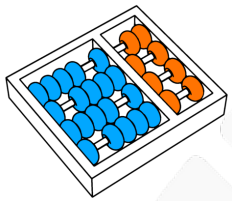
UNICAMP



```
print("You can't print this "text" without error.")
SyntaxError: invalid syntax
```

```
print('You can't print this "text" without error.')
SyntaxError: invalid syntax
```

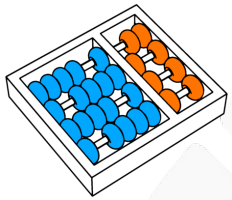
Dúvidas da aula anterior



Dúvidas selecionadas

- É possível programar algo como o Flake8 para que ele se adapte a alguma "norma"específica? (Caso em um empresa, por exemplo, a norma seja q dois espaços devem ser dados entre operadores)
- Qual a vantagem da tipagem dinâmica? Ela só parece deixar o código mais lento e mais suscetível a erros.
- Apenas consigo instalar, importar e utilizar as ferramentas mostradas em aula em um ambiente próprio pra programação em python?
- Há algum modo sistemático e prático de saber ou consultar se dois tipos de dados são intercambiáveis? Por exemplo, é possível converter uma string para list, mas não o oposto (pelo menos no caso que eu testei, a conversão recíproca não funcionou)
- Os monitores reclamam que o meu código fica muito "grudado". Como melhorar isso?
- Qual a diferença do pytest pro assert?
- Em grandes projetos, é mais importante ter um código complexo e eficiente ou simples e mais lento?
- Quando sei se devo definir uma função pra algo específico ou não?
- Para quê serve assert?
- Prof, modularizar demais pode acabar prejudicando a legibilidade ou o desempenho do código?
- O que exatamente é o pip que chamamos para instalar os pacotes de teste?
- Tem alguma extensão do VScode que faça algo parecido com MyPy e semelhantes?
- Não entendi como faço os testes no meu código.

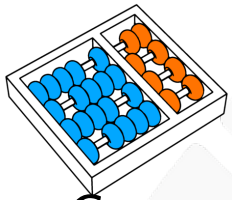
Strings



Strings

A classe **str** é parecida com a classe **list**.

- Podemos acessar os caracteres da string usando índice:
  - Um caractere é simplesmente uma string de tamanho 1.
  - Ex: **s[1]** é o segundo caractere de **s**.
  - Inclusive você pode escrever **for letra in string:**
  - E você pode usar *slices* também.
- Mas você não pode alterar um caracter...
  - **s[1] = 'A'**.
  - **TypeError: 'str' object does not support item assignment**.
- Nem remover um caracter (**del s[1]**).
- **str** é imutável.



Strings com ' e "

Como escrever uma string que contém '?

- **s = 'I'm a coder'** dá **SyntaxError**.
- **s = "I'm a coder"** funciona!

Como escrever uma string que contém "?

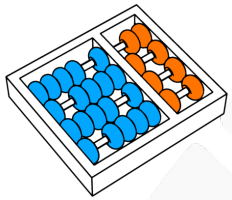
- **s = "Olá "Mundo""** dá **SyntaxError**.
- **s = 'Olá "Mundo"'** funciona!

Mas e se a string tiver tanto ' quanto "?

- **s = "I'm "nice" to people"** dá **SyntaxError**.
- **s = 'I'm "nice" to people'** dá **SyntaxError**.

Soluções:

- **s = '''I'm "nice" to people'''** funciona.
- **s = """I'm "nice" to people"""** funciona.



Outra solução

Podemos escrever também:

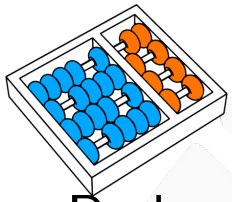
- **'I\'m "nice" to people'**.
- **"I'm \"nice\" to people"**.

Dizemos ao Python que:

- Não interpretar o ' ou o " como final de string.
- Deve considerar como um caractere.

Estamos ***escapando*** o ' ou o ":

- A \ modifica a interpretação do símbolo a seguir.



Escapando

Podemos usar a \ para:

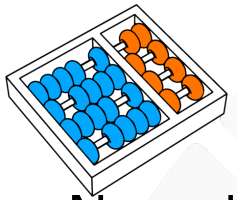
- Poder escrever ' e " em uma string.
- Inserir uma quebra de linha em um texto: **\n**.
- Inserir um tab em um texto: **\t**.

Mas e se eu quiser usar a \ na minha string?

- '\' dá **SyntaxError**...
- '\\' é o correto.

Lembre que \:

- Transforma caracteres especiais em caracteres normais.
  - Ex: ', " e \.
- Transforma caracteres normais em especiais.
  - Ex: **\n** e **\t**.



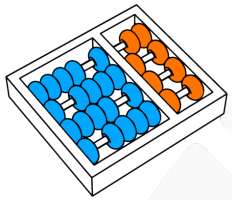
Ordem lexicográfica (ou alfabética)

Na ordem alfabética (lexicográfica) temos que:

- **ana** vem antes de **beto**.
- **abacate** vem antes de **ana**.
- **ana** vem antes de **anamaria**.

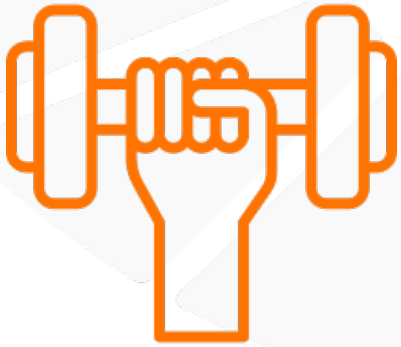
A ordem lexicográfica é definida da seguinte forma:

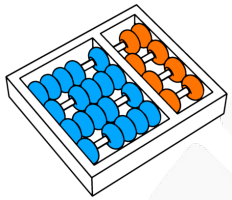
- Seja $p = p_1 p_2 \ldots p_n$ uma palavra de n letras
  - p_1, p_2, ..., são as letras dessa palavra
- Seja $q = q_1 q_2 \ldots q_m$ uma palavra de m letras
- p precede q na ordem (escrevemos $p \prec q$) se:
  - p é prefixo próprio de q
  - Existe $1 \leq i \leq \min\{n, m\}$ tal que
    - $p_j = q_j$ para $0 \leq j < i$ e
    - $p_i < q_i$



Listas e repetição

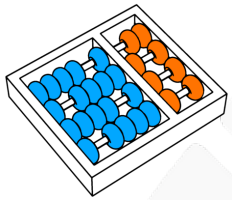
Vamos fazer alguns exercícios?





Exercício: menor lexicograficamente

Faça uma função que, dadas duas listas **l1** e **l2**, nos diz se **l1** é menor ou igual (lexicograficamente) a **l2**.



Comparação direta entre strings no Python

Na verdade, bastaria escrever **p <= q**!

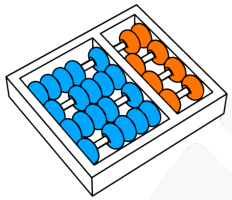
Exemplo:

- **[1, 2, 3] < [1, 2, 4]**.
- **[1, 2, 3, 5] < [1, 2, 4]**.
- **[1, 2] < [1, 2, 4]**.

- E isso vale também para strings!
- E você pode usar: **<=**, **<**, **>=**, **>**, **==** e **!=**

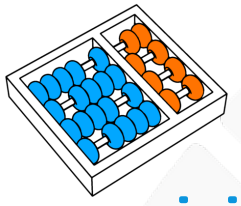
Exemplo:

- **"ana" < "beto"**.
- **"ana" < "anamaria"**.



Exercícios

1. Faça uma função que, dado uma string **sep** e uma lista **l** de strings, concatena as strings de **l** usando **sep** como separador:
   - Ex: Se **l == ["A", "B", "C"]** e **sep == ","**, então o resultado deve ser **"A,B,C"**.
2. Faça uma função que, dadas duas strings **s** e **t**, verifica se **s** é prefixo de **t**.
3. Faça uma função que, dadas duas strings **s** e **t**, verifica se **s** é substring de **t**.
4. Faça uma função que, dada uma string **s** e uma string **sep**, devolve uma lista resultante da quebra de **s** em uma ou mais strings, em cada ocorrência de **sep**:
   - Ex: Se **s == "A,B,C"** e **sep == ","**, então o resultado deve ser **["A", "B", "C"]**.



Alguns métodos úteis

sep.join(lista):

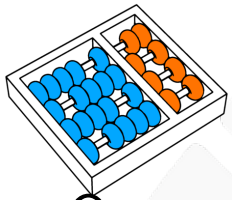
- Concatena uma **lista** usando **sep** como separador.
- Ex: **",".join(["A", "B", "C"])** resulta em **"A,B,C"**.
- Ex: **"".join(["A", "B", "C"])** resulta em **"ABC"**.

s.split(sep):

- Quebra string **s** em cada ocorrência de **sep** em uma lista.
- Ex: **"A,B,C".split(",")** resulta em **["A", "B", "C"]**.
- Ex: **palavras = input().split(' ')**.

Outros métodos que fazem o mesmo que fizemos nos exercícios:

- **s.startswith** para verificar se uma string é substring no começo de **s**.
- **s.find** para achar o índice onde uma substring começa em **s**.



Formatação de Strings

Queremos montar uma string a partir de cálculos que fizemos:
s = str(x) + '' + str(y) + " é " + str(x ** y)**.

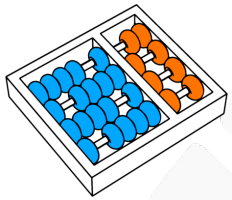
- Temos que converter para **str** para concatenar...
- O que deixa a expressão longa

Podemos escrever, simplesmente:
s = f'{x}{y} é {x ** y}'**.

- O **f** indica que essa é uma string formatada.
- Entre { } podemos colocar qualquer expressão Python.
- Será convertida para **str** automaticamente.

Podemos também escrever:
s = '{}{} é {}'.format(x, y, x ** y)**.

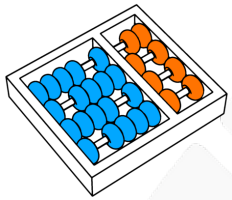
- **format** troca o **i**-ésimo { } pelo **i**-ésimo parâmetro.



format mais legível

'{}{} é {}'.format(x, y, x ** y)** não é tão legível...

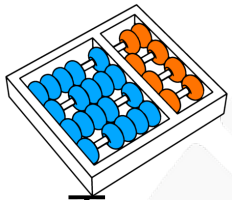
- Opção 1: **'{0}**{1} é {2}'.format(x, y, x ** y)**.
  - **{i}** é trocado pelo **i**-ésimo parâmetro.
  - Não precisa ser na mesma ordem...
  - Ex: **'{1} {0} {1}'.format('A', 'B') == 'B A B'**.
- Opção 2: **'{base}**{exp} é {res}'.format(base=x, exp=y, res=x ** y)**.
  - Usa a ideia de nome de parâmetro.
- Opção 3: Combinar os dois... o que talvez seja meio estranho. **'{0}**{exp} é {res}'.format(x, exp=y, res=x ** y)**.
  - Não vejo motivo para usar esse...



Formatação dos dados

Imagine que **x = 0.1 + 0.2** e queremos colocar **x** em uma string:

- **'{}'.format(x)** é **'0.30000000000000004'**.
- **'{num:.1f}'.format(num=x)** é **'0.3'**.
  - O **f** indica que queremos imprimir um **float**.
  - E o **.1** indica que queremos uma casa de precisão.
  - Ex: **'{0:.2f}'.format(x)** é **'0.30'**.
  - Ex: **'{:.0f}'.format(x)** é **'0'**.
- **'{:e}'.format(x)** é **'3.000000e-01'**
  - **e** indica que queremos notação científica.
  - Com **E** maiúsculo, fica **'3.000000E-01'**.
- **'{:%}'.format(x)** é **'30.000000%'**.



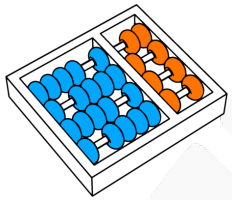
Formatação dos dados

Temos opções para **int** também:

- **'{:d}'.format(42)** é **'42'** (decimal).
- **'{:x}'.format(42)** é **'2a'** (hexadecimal).
- **'{:X}'.format(42)** é **'2A'** (hexadecimal).
- **'{:o}'.format(42)** é **'52'** (octal).
- **'{:b}'.format(42)** é **'101010'** (binário).
- E com um **#** antes de **x**, **o**, e **b** você obtém:
  - **'0x2a'**, **'0o52'**, e **'0b101010'**

Há algumas opções (**<**, **>** e **^**) para controlar o tamanho da string e alinhar o conteúdo:

- Além de mais algumas opções de impressão de número.
  - Ex: Em números positivos, o sinal deve aparecer ou não?
- Pesquise sobre isso!



Uma última dica

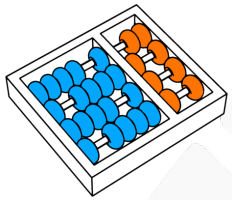
Muitas vezes queremos imprimir o valor de uma variável:

- Ao invés de escrever **print(f'x = {x}')**.
- Posso escrever **print(f'{x = }')**.
- Na verdade, o **print** não importa para a string.

Outro exemplo (com **x = 10** e **y = 3**):
f'{x + 3 * y + 2 = }' é a string **'x + 3 * y + 2 = 21'**

Isso não é muito bonito para o usuário:

- Mas pode ser útil para o programador.



str tem muitos métodos úteis

capitalize • **casefold** • **center** • **count** • **encode** • **endswith** • **expandtabs** • **find** • **format_map** • **format** • **index** • **isalnum** • **isalpha** • **isascii** • **isdecimal** • **isdigit** • **isidentifier** • **islower** • **isnumeric** • **isprintable** • **isspace** • **istitle** • **isupper** • **join** • **ljust** • **lower** • **lstrip** • **partition** • **replace** • **rfind** • **rindex** • **rjust** • **rpartition** • **rsplit** • **rstrip** • **split** • **splitlines** • **startswith** • **strip** • **swapcase** • **title** • **translate** • **upper** • **zfill**

Leia a documentação!

Strings

MC102 - Algoritmos e Programação de Computadores

04/25

13

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP