

STRINGS

MC102 - Algoritmos e
Programação de
Computadores

Santiago Valdés Ravelo
[https://ic.unicamp.br/~santiago/
ravelo@unicamp.br](https://ic.unicamp.br/~santiago/ravelo@unicamp.br)

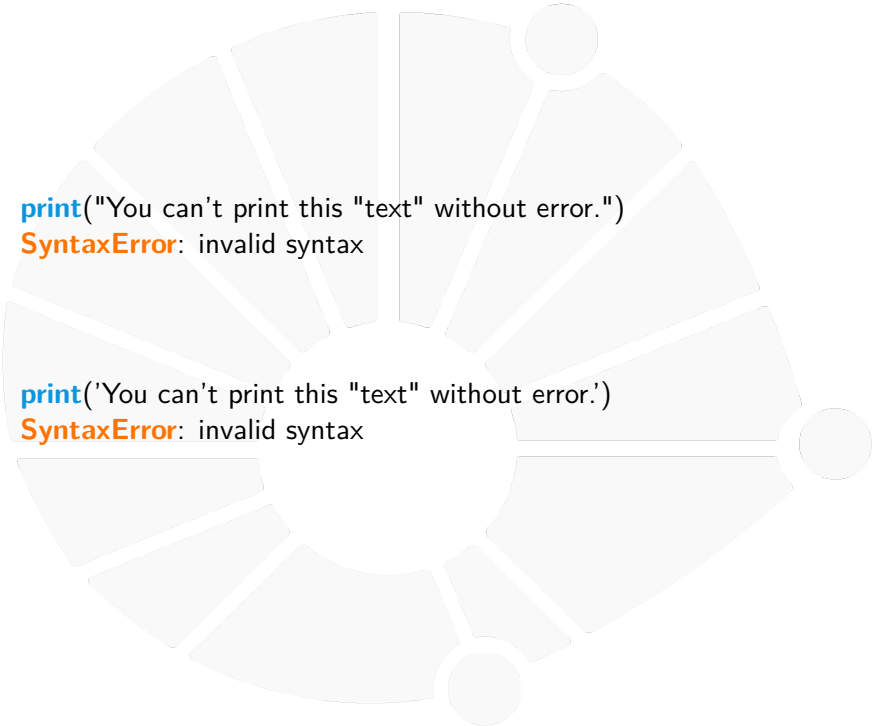
04/24

12



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

- ▶ Ao utilizar o raise ele apenas já identifica o erro, mas a impressão que ocorreu o erro é feita apenas pelo except?
- ▶ No último lab, em mais de um exercício era pedido para o usuário digitar uma lista onde o tamanho da lista era o primeiro elemento (sem contar o primeiro elemento), por exemplo [5, 3, 2, 0, 7, 2] o 5 é o tamanho da lista (sem contar o 5), e isso era dado em um map(int, input().split()), ou seja, em uma linha só, é possível limitar essa lista ao tamanho dado no primeiro elemento e fazer aparecer um erro caso o usuário digite uma lista inválida como por exemplo [6, 1, 3]?
- ▶ Python possui os operadores principais de manipulação de listas da programação funcional? (map, reduce, filter, group, any, all)
- ▶ Não entendi muito bem o assunto dos erros. Error é um tipo de dados? TypeError, RuntimeError são objetos dessa classe?
- ▶ Não entendi direito como usar a função lambda
- ▶ Pensando nessa situação: faço um código que contém um except específico e outro except genérico. Se ocorre um erro relacionado ao except específico, o programa roda somente o específico e não o genérico? Por quê?
- ▶ Qual o propósito de usar finally se eu posso simplesmente colocar o código fora da exceção?
- ▶ Se eu importar apenas uma função que criei em outro arquivo e essa função depender de funções específicas de outra biblioteca, ocorrerá algum erro?
- ▶ Qual a diferença na prática do raise e do except? Se não queremos que uma situação aconteça, como no exemplo dado quando foi explicado o raise, não é mais fácil colocar um if/else?
- ▶ É possível eu criar um erro novo para ser tratado no exception? Por exemplo, não quero dividir 4 por 2, então nesse caso, eu crio um erro específico para ele.
- ▶ Professor, pra mim não ficou muito claro sobre aquele cuidado que devemos ter ao importar bibliotecas pois o arquivo será executado. Como aquela solução do "__main__" funcionaria?
- ▶ Quando você atribui uma função a uma variável, não significa que a função mudou de nome, mas agora você pode chamá-la também por outro nome.
- ▶ Seria possível importar uma biblioteca dentro de uma função?



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 caractere...
 - ▶ `s[1] = 'A'`.
 - ▶ **`TypeError: 'str' object does not support item assignment.`**
- ▶ Nem remover um caractere (**`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 "?

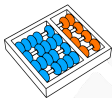
- ▶ `s = "Olá "Mundo""` dá **SyntaxError**.
- ▶ `s = 'Olá "Mundo"'` funciona!

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:

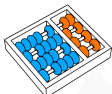
- ▶ `'I\'m "nice" to people'`.
- ▶ `"I'm \"nice\" to people"`.

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:

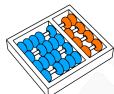
- ▶ Poder escrever `'` e `"` em uma string.
- ▶ Inserir uma quebra de linha em um texto: `\n`.
- ▶ Inserir um tab em um texto: `\t`.

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`.



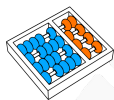
Ordem lexicográfica (ou alfabética)

Na ordem alfabética (lexicográfica) temos que:

- ▶ **ana** vem antes de **bet**o.
- ▶ **abacate** vem antes de **ana**.
- ▶ **ana** vem antes de **anamaria**.

A ordem lexicográfica é definida da seguinte forma:

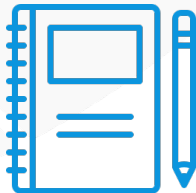
- ▶ Seja $p = p_1p_2 \dots p_n$ uma palavra de n letras
 - ▶ p_1, p_2, \dots , são as letras dessa palavra
- ▶ Seja $q = q_1q_2 \dots q_m$ uma palavra de m letras
- ▶ p precede q na ordem (escrevemos $p < q$) se:
 - ▶ p é prefixo próprio de q
 - ▶ Existe $1 \leq i \leq \min\{n, m\}$ tal que
 - ▶ $p_j = q_j$ para $0 \leq j < i$ e
 - ▶ $p_i < q_i$

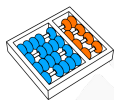


Listas e repetição



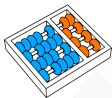
Vamos fazer alguns exercícios?





Exercício: menor lexicograficamente

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



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"]**.



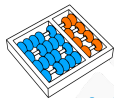
Comparação direta entre strings no Python

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

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

Exemplo:

- ▶ `"ana" < "beto"`.
- ▶ `"ana" < "anamaria"`.
- ▶ `[1, 2, 3] < [1, 2, 4]`.
- ▶ `[1, 2, 3, 5] < [1, 2, 4]`.
- ▶ `[1, 2] < [1, 2, 4]`.



Alguns métodos úteis

`sep.join(lista)`:

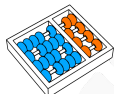
- ▶ Concatena uma **lista** usando **sep** como separador.
- ▶ Ex: `",".join(["A", "B", "C"])` resulta em `"A,B,C"`.
- ▶ Ex: `"".join(["A", "B", "C"])` resulta em `"ABC"`.

`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 (ou meio) 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.



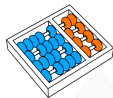
format mais legível

'**{}******{}**' 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).

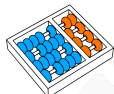
 - ▶ Não vejo motivo para usar esse...



Formatação dos dados

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

- ▶ `'{}'.format(x)` é `'0.300000000000000004'`.
- ▶ `'{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%'`.



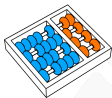
Formatação dos dados

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.



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

Santiago Valdés Ravelo
[https://ic.unicamp.br/~santiago/
ravelo@unicamp.br](https://ic.unicamp.br/~santiago/ravelo@unicamp.br)

04/24

12



UNICAMP

