Strings

- São seqüências de caracteres.
  - Em Python, o nome do tipo é **'str'** – em geral: string.
- Em Python, strings podem ser delimitados tanto por **aspas (")** quanto por **apóstrofos (')**.
  - Em algumas outras linguagens somente por aspas (apóstrofo usado para um só caracter).
- Em Python, **strings são objetos imutáveis!**
  - Não é possível *adicionar, remover ou modificar parte* de um string.
  - Para isto, é sempre necessário criar um outro string.

– Usar \", \' e \\ para inserir o símbolo após a \.
– Usar \n para mudar de linha (na impressão).
– Usar \t para dar um (na impressão).

Comparação

- **Operadores de comparação** funcionam também para strings!
  - Não só == e !=, também <, >, <= e >=.
- A ordem do sistema interno de representação é usada (ASCII, Unicode, etc.).
  - Ordem alfabética dentro do mesmo tipo de letra.
  - Mas, as **maiúsculas são menores que as minúsculas.**
- *Exemplos:*
  - *'B' < 'a' é True.*
  - *'b' < 'banana' é True.*

Acesso usando índices

- Cada um dos caracteres de um string pode ser acessado por um **índice posicional**:
  - Em Python, estes índices (posições) **começam a ser contados em zero** e são escritos entre **colchetes**. ⇒ [0] [1] [2] [3]
    - *Só em Python, índices negativos a partir de -1 podem ser usados para acessá-los de traz para a frente. [-4] [-3] [-2] [-1]*

💡 Usar um índice fora do intervalo válido de acordo com o tamanho do string causa um **erro fatal**! – IndexError.

- *OBS: Não é permitido usar comando de atribuição para uma posição de um string usando um índice.*

```
#Exemplos:
fruta = ‘bananas!’
primLet = fruta[0] #Resultado é ‘b’.
ultLet = fruta[-1] #Resultado é ‘!’.
penuLet = fruta[-2] #Resultado é ‘s’.
fruta [7] = ‘.’ #Erro fatal!
fruta = fruta[:7] + ‘.’ #Outro objeto é criado. Resultado é ‘ba
```

Comando for

- Em Python, o comando for pode ser usado para acessar os caracteres de um string “um a um”
  sem precisar usar índices!

```
#Sintaxe do comando for para percorrer strings.
for ch in qualquerString :
    comandoUsandoCh

#Exemplo:
fruta = "banana"
```

```
for c in fruta: # c recebe cada caracter da variável fruta.
    print(fruta, c) # c assume cada uma das letras...
```

Operadores e funções

- Operadores e/ou funções que visam facilitar a manipulação de strings.
- Em Python temos:
  - **len:** tamanho (quantidade de caracteres) do string.
  - **+**: concatenação de strings (para juntar/emendar...).
    - Conversão de valores de outros tipos não é automática.
  - *****: repetição de strings – concatenação repetida...
  - **in**: indica se um string está contido em outro.
  - ****: permite que um string muito longo possa continuar a ser digitado na linha seguinte do programa.

```
fruta = "banana"
print(fruta)
tam = len(fruta) # Resultado é 6.
print(tam)
qtd = 10
fruta2 = fruta + " comprida " + str(qtd)
print(fruta2)
spam = "Spam!" * 3 # spam = ‘Spam!Spam!Spam!’
print(spam)
res = fruta in fruta2 # res ficará com True.
print(res)
temA = "a" in fruta # temA também ficará com True.
print(temA)
alfabetos = "ABCDEFGHIJKLMNOPQRSTUVWXYZ\
abcdefghijklmnopqrstuvwxyz"
print(alfabetos)
```

Interpolação

- O operador de interpolação (**%**) (só em Python):
  - Versão mais eficiente de *concatenação;*
  - Geralmente usado para **combinar/intercalar valores de variáveis dentro de um string;**
  - Útil também em comandos print para formatar saída;
  - Usa "*máscaras*" para marcar os locais onde inserir os valores das variáveis – estes serão parâmetros para o operador %;
    - **%d: números inteiros – int e long.**
    - **%f: números reais – float.**
    - **%s: strings.**
      - O tamanho pode ser informado após o % e, no caso dos reais, também o número de casas decimais. Ex. *%5d ou %8.2f ou %.2f*
  - Pode-se informar um tamanho (em colunas - opcional) para incluir o valor do conteúdo mas, se o tamanho informado for insuficiente, ele será ignorado;

```
#Exemplos:
print 'A=%d e %s.' % (a, mensagem)
print "Média dos %3d alunos é %8.2f." % (qtd, media)
```

Substring

- Operador que serve para ter acesso (ou recuperar) um pedaço de um string:
  - Em Python são chamados de **slices**.
  - Na sintaxe de Python, usa-se dois índices separados por : (dois pontos) para significar um intervalo de índices **(fechado à esquerda e aberto à direita)**.
    - Cada um dos limites pode ser omitido e o significado será "a partir do início" e "até o fim", respectivamente.

    - Se o intervalo for vazio, ou seja, o segundo parâmetro for menor ou igual ao primeiro, o resultado é o string nulo – “”.

```
fruta = ‘banana’
#a contagem de posição começa do 0
#o intervalo é [fechado:aberto]
silaba2 = fruta [2:4] # Resultado é ‘na’.
silaba1 = fruta [0:2] # Resultado é ‘ba’.
silaba1 = fruta [:2] # Mesma coisa...
sufixo = fruta [3:6] # Resultado é ‘ana’.
sufixo = fruta [3:] # Mesma coisa...
tudo = fruta [:] # Válido mas não faz muito sentido...
tudo = fruta # Válido e mais eficiente.
vazio = fruta [3:3] # Resultado é o string nulo - “”.
```

Métodos

- São como operadores mas usam uma sintaxe diferente.
    - Isto é conseqüência de strings serem objetos!
    - Sintaxe é: **stringQualquer.nomeMétodo(parâmetros)**
- Alguns métodos comuns para string:
    - **upper:** troca todas as letras para maiúsculas.
    - **lower:** troca todas as letras para minúsculas.
    - **find:** procura posição de um substring em um string.
    - **replace:** troca ocorrências de um substring por outro.
    - **strip:** retira espaços no início, no final ou múltiplos, além de outros whitespaces como tab, newline, etc.

```
fr = ‘ Banana! ’
fr = fr.strip() # Resultado é ‘Banana!’
fr2 = fr.upper() # Resultado é ‘BANANA!’
fr3 = fr.lower() # Resultado é ‘banana!’
```

```
pos1 = fr.find (‘na’) # Resultado é 2.
pos2 = fr.find (‘na’,3) # Resultado é 4.
pos3 = fr.find (‘Ba’,2) # Resultado é -1.
pos4 = fr.find (‘!’) # Resultado é 6.
pos5 = fr.find (‘!’,1,5) # Resultado é -1.
fr4 = fr.replace(‘a’,’A’) # Resultado é ‘BAnAnA!’.
fr5 = fr.replace(‘a’,’A’,2) # Resultado é ‘BAnAna!’.
```

Exercícios

1. Ler o nome do usuário e imprimi-lo na vertical, ou seja, uma letra embaixo da outra.
2. Ler o nome do usuário e imprimi-lo em formato de escala, ou seja, só a primeira letra na primeira linha, as duas primeiras letras na segunda linha, e assim por diante.
3. Ler 2 strings e dizer quantas vezes o primeiro aparece no segundo.
4. Ler o nome completo do usuário e imprimi-lo com o primeiro e último nome escritos todo em maiúsculas.

```
print("1. Ler o nome do usuário e imprimi-lo na vertical, ou se
nome1 = input("Digite o seu nome:")
for s1 in nome1:
    print(f"{s1}")

print("2. Ler o nome do usuário e imprimi-lo em formato de escal
      "na primeira linha, as duas primeiras letras na segunda li
nome2 = input("Digite uma palavra:")
a2 = 0
b2 = 1
cont2 = 1
for s2 in nome2:
    print(nome2[a2:b2])
    cont2 += 1
    a2 = b2
```

```
    b2 += cont2

print("3. Ler 2 strings e dizer quantas vezes o primeiro aparece
nome3 = input("Digite uma palavra:")
nome33 = input("Digite outra palavra:")
resultado3 = nome33.find(nome3)
cont3 = 0
while resultado3 != -1:
    cont3 += 1
    resultado3 = nome33.find(nome3, resultado3+1)
if resultado3 == -1:
    resultado3 = 0
print(f"A palavra {nome3} aparece {cont3} vez(es) em {nome33}")

print("4. Ler o nome completo do usuário e imprimi-lo com o prin
nome4 = input("Digite seu nome completo:")
pposi4 = nome4.find(' ')
uposi4 = nome4.rfind(' ')
primeiro4 = nome4[:pposi4].upper()
ultimo4 = nome4[uposi4:].upper()
nomefinal4 = nome4.replace(nome4[:pposi4], primeiro4)
nomefinal4 = nomefinal4.replace(nome4[uposi4:], ultimo4)
print(nomefinal4)
```