Trabalhando com Strings na linguagem Python

Relembrando Strings

Uma string é uma sequência de caracteres.

- ✓ as posições dos caracteres na sequência são numeradas, iniciando em 0

U	m	a		f	r	a	s	e	!
0	1	2	3	4	5	6	7	8	9

Representamos strings usando aspas simples ou duplas:

'ana@gmail.com'

"ana@gmail.com"

'"Sim!", ele respondeu'

os caracteres " – aspas! - pertencem a essa sequência!

Operações com Strings

O operador **+** concatena strings:

```
nome = "laura"
dominio = "gmail.com"
print(nome + "@" + dominio)
```

→ **laura@gmail.com**

O operador ***** replica strings:

```
print(3 * nome)
```

→ **lauralauralaura**

A função len retorna o tamanho da string:

```
print(len(nome))
```

→ **5**

Indexando Strings

Selecionamos caracteres de uma string usando o operador de indexação [ ].

- ✓ em Python, um caractere é uma string de tamanho 1

U	m	a		f	r	a	s	e	!
0	1	2	3	4	5	6	7	8	9

```
frase = "Uma frase!"
print(frase[1])
print(frase[-1])

print(frase[10])
```

→ **m**

→ **!**

→ `IndexError: string index out of range`

Fatiamento de Strings

A seleção de uma fatia (*slice*) de uma string pode ser feita com a operação [n:m].

- ✓ o resultado é uma fatia da string original (também uma string!)

U	m	a		f	r	a	s	e	!
0	1	2	3	4	5	6	7	8	9

```
s[1:3]  ⟶ 'ma'      ⟶ fatia do caractere 1 ao caractere 2
s[:3]   ⟶ 'Uma'     ⟶ fatia do início ao caractere 2
s[4:]   ⟶ 'frase!'  ⟶ fatia do caractere 4 até o final

s[:]    ⟶           ⟶ qual é o resultado dessa operação?
```


Fatiamento de Strings: s[:]

A seleção de uma fatia (*slice*) de uma string pode ser feita com a operação [n:m].

- ✓ o resultado é uma fatia da string original (também uma string!)

U	m	a		f	r	a	s	e	!
0	1	2	3	4	5	6	7	8	9

```
s[1:3]  ⟶ 'ma'      ⟶ fatia do caractere 1 ao caractere 2
s[:3]   ⟶ 'Uma'     ⟶ fatia do início ao caractere 2
s[4:]   ⟶ 'frase!'  ⟶ fatia do caractere 4 até o final

s[:]    ⟶ 'Uma frase!'  ⟶ uma cópia da string original!
```

Strings são Imutáveis!

Podemos selecionar um caractere de uma string mas não podemos alterá-lo!

U	m	a		f	r	a	s	e	!
0	1	2	3	4	5	6	7	8	9

```
frase = "Uma frase!"
frase[4] = "c"
```

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

Imutáveis: alteração por fatiamento

Podemos selecionar um caractere de uma string mas não podemos alterá-lo!

U	m	a		f	r	a	s	e	!
0	1	2	3	4	5	6	7	8	9

```
frase = "Uma frase!"
frase[4] = "c"
```

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

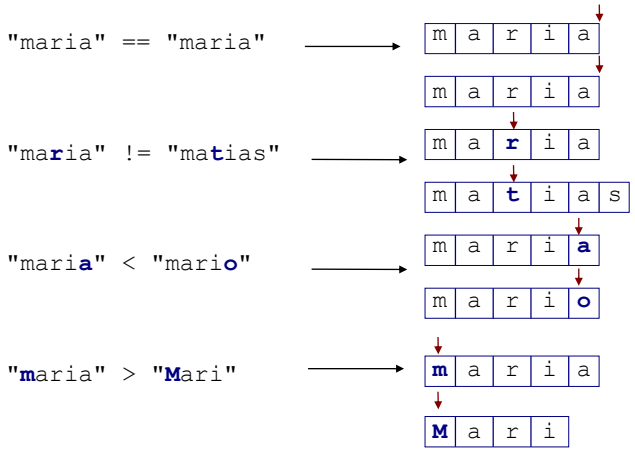
Uma solução é criar uma nova string, usando fatiamento:

```
frase = frase[:4] + "c" + frase[5:]
print(frase)
```

→ **`'Uma crase!'`**

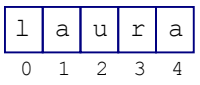
Comparando Strings

Strings são comparadas *lexicograficamente* (em "ordem alfabética"), caractere a caractere:



Percorrendo uma String (1/7)

Podemos percorrer uma string caractere a caractere usando um laço while:



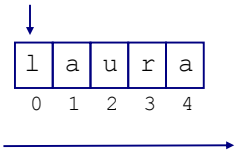

```
nome = "laura"
ind = 0
while ind < len(nome):
  letra = nome[ind]
  print(letra)
  ind = ind + 1
```

Exibir letras de uma string, uma em cada linha

Percorrendo uma String (2/7)

Podemos percorrer uma string caractere a caractere usando um laço while:

```
nome = "laura"
ind = 0
while ind < len(nome):
  letra = nome[ind]
  print(letra)
  ind = ind + 1
```



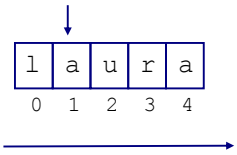

```
l
```

11

Percorrendo uma String (3/7)

Podemos percorrer uma string caractere a caractere usando um laço while:

```
nome = "laura"
ind = 0
while ind < len(nome):
  letra = nome[ind]
  print(letra)
  ind = ind + 1
```



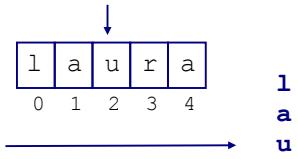

```
l
a
```

12

Percorrendo uma String (4/7)

Podemos percorrer uma string caractere a caractere usando um laço while:

```
nome = "laura"
ind = 0
while ind < len(nome):
  letra = nome[ind]
  print(letra)
  ind = ind + 1
```

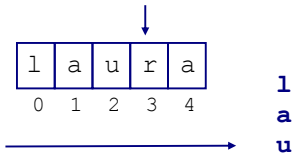



```
l
a
u
```


Percorrendo uma String (5/7)

Podemos percorrer uma string caractere a caractere usando um laço while:

```
nome = "laura"
ind = 0
while ind < len(nome):
  letra = nome[ind]
  print(letra)
  ind = ind + 1
```

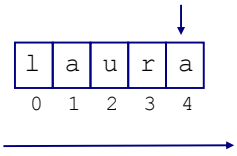



```
l
a
u
r
```

Percorrendo uma String (6/7)

Podemos percorrer uma string caractere a caractere usando um laço while:

```
nome = "laura"
ind = 0
while ind < len(nome):
  letra = nome[ind]
  print(letra)
  ind = ind + 1
```

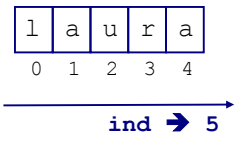



```
l
a
u
r
a
```


Percorrendo uma String (7/7)

Podemos percorrer uma string caractere a caractere usando um laço while:

```
nome = "laura"
ind = 0
while ind < len(nome):
  letra = nome[ind]
  print(letra)
  ind = ind + 1
```




```
l
a
u
r
a
```

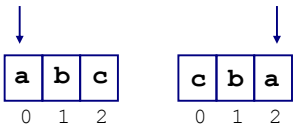
Invertendo uma String

Vamos escrever uma função booleana **eh_reversa** que receba duas strings e retorne True se a segunda string é o reverso da primeira, False caso contrário.

Exemplos:

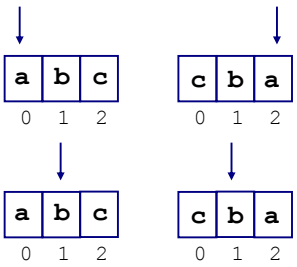
```
eh_reversa("abcde","edcba")  ──→ True
eh_reversa("ana","ana")      ──→ True
eh_reversa("abcde","xdcba")  ──→ False
eh_reversa("abcde","eddba")  ──→ False
eh_reversa("a","ba")         ──→ False
```


Desenvolvendo uma Solução (1/7)



caracteres iguais!

Desenvolvendo uma Solução (2/7)

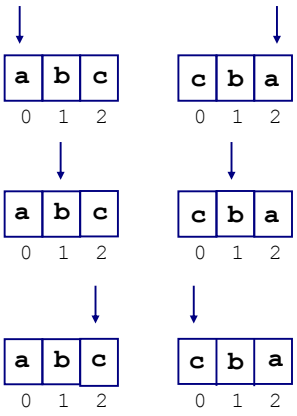


caracteres iguais!

caracteres iguais!

19

Desenvolvendo uma Solução (3/7)



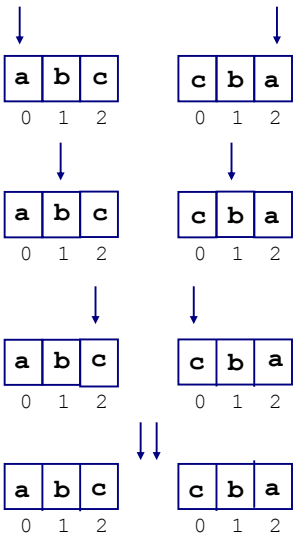
caracteres iguais!

caracteres iguais!

caracteres iguais!

20

Desenvolvendo uma Solução (4/7)



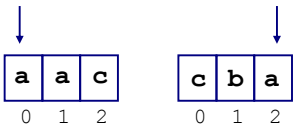
caracteres iguais!

caracteres iguais!

caracteres iguais!

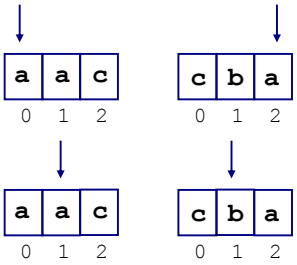
É reversa!!!

Desenvolvendo uma Solução (5/7)



caracteres iguais!

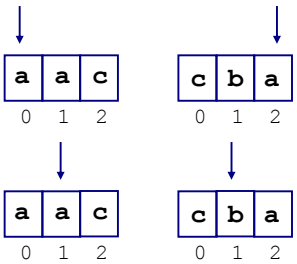
Desenvolvendo uma Solução (6/7)



caracteres iguais!

caracteres diferentes!

Desenvolvendo uma Solução (7/7)



caracteres iguais!

caracteres diferentes!

Não é reversa!!!

Eh_reversa: possível solução

```
def eh_reversa(s1,s2):
  if len(s1) != len(s2):
    return False
  i1 = 0
  i2 = len(s1)-1
  while i1 < len(s1):
    if s1[i1] != s2[i2]:
      return False
    i1= i1+1
    i2= i2-1
  return True
```


Strings: contagem e buscas

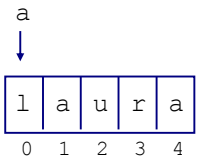
EXERCíCIOS

1) Escreva uma função **conta** que receba uma string e um caractere e retorne quantas vezes esse caractere ocorre na string.
   - ✓ lembre-se que você pode comparar strings com o operador relacional **==**

2) Escreva uma função **busca** que receba uma string e um caractere e retorne o índice da primeira ocorrência deste caractere na string.
   - ✓ se o caractere não está na string, a função deve retornar o valor **-1**

Solução Possível: conta (1/5)

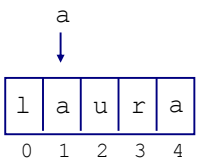
```
def conta(s,c):
    ind = 0
    vezes = 0
    while ind < len(s):
        if s[ind] == c:
            vezes = vezes + 1
        ind = ind + 1
    return vezes

num = conta("laura","a")
```



vezes ➔ 0

Solução Possível: conta (2/5)

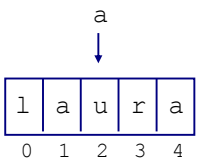
```
def conta(s,c):
    ind = 0
    vezes = 0
    while ind < len(s):
        if s[ind] == c:
            vezes = vezes + 1
        ind = ind + 1
    return vezes

num = conta("laura","a")
```



vezes ➔ 1

Solução Possível: conta (3/5)

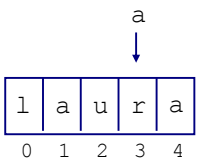
```
def conta(s,c):
    ind = 0
    vezes = 0
    while ind < len(s):
        if s[ind] == c:
            vezes = vezes + 1
        ind = ind + 1
    return vezes

num = conta("laura","a")
```



vezes ➔ 1

Solução Possível: conta (4/5)

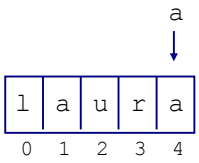
```
def conta(s,c):
    ind = 0
    vezes = 0
    while ind < len(s):
        if s[ind] == c:
            vezes = vezes + 1
        ind = ind + 1
    return vezes

num = conta("laura","a")
```



vezes ➔ 1

Solução Possível: conta (5/5)

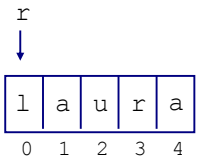
```
def conta(s,c):
    ind = 0
    vezes = 0
    while ind < len(s):
        if s[ind] == c:
            vezes = vezes + 1
        ind = ind + 1
    return vezes

num = conta("laura","a")
```



vezes ➔ 2

Função busca: uma solução (1/4)

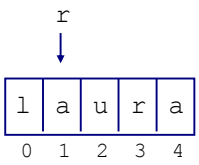
```
def busca(s,c):
  ind = 0
  while ind < len(s):
    if s[ind] == c:
      return ind
    ind = ind + 1
  return -1

pos = busca("laura","r")
```



ind ➔ 0

Função busca: uma solução (2/4)

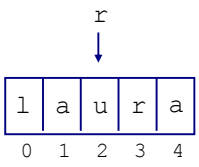
```
def busca(s,c):
  ind = 0
  while ind < len(s):
    if s[ind] == c:
      return ind
    ind = ind + 1
  return -1

pos = busca("laura","r")
```



ind ➔ 1

Função busca: uma solução (3/4)

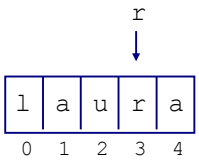
```
def busca(s,c):
  ind = 0
  while ind < len(s):
    if s[ind] == c:
      return ind
    ind = ind + 1
  return -1

pos = busca("laura","r")
```



ind ➔ 2

Função busca: uma solução (4/4)

```
def busca(s,c):
  ind = 0
  while ind < len(s):
    if s[ind] == c:
      return ind
    ind = ind + 1
  return -1

pos = busca("laura","r")
```



ind ➔ 3

valor retornado!

Experimente fazer um "chinês" para o caso do caractere "buscado" não existir na string!

Métodos de String

Assim como uma tartaruga, uma string é um **objeto**.

- ✓ objetos do tipo string possuem métodos com diversos tipos de ações

```
pat = turtle.Turtle()
pat.forward(100)

s = "Objeto"
s_mai = s.upper()
print(s_mai)
s_min = s.lower()
print(s_min)

print(s.count("o"))
```

OBJETO

objeto

2

Exemplos: UPPER, LOWER e COUNT

Procurando uma Substring: FIND

Obtemos o índice do início de uma "substring" usando o método find:

```
                5          15    20                    39
s = "Quem parte e reparte, fica com a maior parte."
```

```
s.find("parte")   ------>  5
s.find(",")       ------>  20
s.find("menor")   ------>  -1   <-- substring não foi encontrada!
```

Podemos especificar a região (início, fim) onde é feita a procura da substring:

```
s.find("parte",6)       ------>  15
s.find("parte",20)      ------>  39
s.find("parte",6,20)    ------>  15
```


Substituindo Substrings: REPLACE

Substituimos todas as ocorrências de uma substring usando o método replace:

```
s = "Quem parte e reparte, fica com a maior parte."
print(s.replace("parte","liga"))
```

```
Quem liga e religa, fica com a maior liga.
```

Podemos substituir apenas um número de ocorrências:

```
print(s.replace("parte","liga",2))
```

```
Quem liga e religa, fica com a maior parte.
```

Eliminando caracteres *em branco*

O método strip remove os caracteres *em branco* do início e fim de uma string;

- ✓ são considerados caracteres *em branco*:
  - o espaço (" "),
  - tab ("\t"), e
  - "nova linha" ("\n")

Os métodos lstrip e rstrip removem, respectivamente, os caracteres *em branco* do início ou do fim da string

```
s = " Olá, mundo!  "
print("[" + s + "]")          ⟶ [ Olá, mundo!  ]
print("[" + s.strip() + "]")  ⟶ [Olá, mundo!]
print("[" + s.lstrip()+ "]")  ⟶ [Olá, mundo!  ]
print("[" + s.rstrip()+ "]")  ⟶ [ Olá, mundo!]
```


Mais Métodos de String

Existem muitos outros métodos úteis para trabalharmos com textos!

- ✓ eles são descritos na documentação de Python
- ✓ http://docs.python.org/py3k

Exercício: itens em um texto

Suponha um tipo de texto composto por diversos itens separados por vírgulas, como nos exemplos de texto t1 e t2 abaixo

```
t1 = "um item"
t2 = "primeiro,segundo,terceiro,quarto"
```

Escreva uma função **num_itens** que receba uma string com um texto como descrito acima e retorne o número de itens presentes no texto.

- ✓ você pode assumir que o texto tenha pelo menos um item mas certifique-se que sua função funciona corretamente para um texto que tenha apenas um item!

Uma Solução: itens em textos

```
def num_itens(s):
    itens = 0
    pos = 0
    while pos != -1:
        itens = itens + 1
        pos = s.find(",",pos+1)
    return itens
```

Exercício: obtém item em textos

Escreva agora uma função **obtem_item** que receba uma string com um texto como o do exercício anterior e um inteiro que indica o número do item a ser retornado.

- ✓ considere que os itens são numerados a partir de 0
- ✓ você pode considerar que o texto recebido tem pelo menos um item
- ✓ você deve usar a sua função **num_itens** para verificar se existe um item com o número indicado. Se não existir, retorne uma string vazia.

Obtem_item: uma solução

```
def obtem_item(s,n):
  # testa se o numero do item é válido
  if n < 0 or n > num_itens(s) - 1:
    return ""

  # procura a "," antes do item
  item = 0
  ini = 0
  while item != n:
    ini = s.find(",",ini) + 1 # posicao depois da ","
    item = item + 1

  # procura a "," depois do item
  fim = s.find(",",ini)

  if fim == -1:        # ultimo item
     return s[ini:]
  else:
      return s[ini:fim]
```

Exercício: remove substrings

Escreva uma função **remove_sub** que receba duas strings e retorne uma cópia da primeira string onde foram removidas todas as ocorrências da segunda string.

Exemplos:

```
remove_sub("abcabbcabcbca","bc")
    ⟶ 'aabaa'
remove_sub("abcabbbcbc","bc")
    ⟶ 'aabb'
remove_sub("bcabbbc","bc")
    ⟶ 'abb'
```

```
remove_sub("abac","bc")
    ⟶ 'abac'
remove_sub("bcbc","bc")
    ⟶ ''
```

Você consegue escrever essa função sem usar o método replace?

Soluções remove_sub

COM:

```
def remove_sub(s, sub):
  return s.replace(sub,"")
```

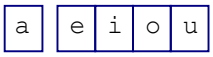
SEM:

```
def remove_sub(s,sub):
  nova = ""
  prox = 0          # proxima posição a copiar
  ini = 0           # início da substring

  while ini != -1:
    ini = s.find(sub,prox)
    if ini != -1:
      nova = nova + s[prox:ini]
      prox = ini + len(sub)
  nova = nova + s[prox:]
  return nova
```

Desafio: reverte recursiva

Podemos ver uma string como uma concatenação do seu primeiro caractere com o resto da sequência.

- ✓ esse resto é a fatia [1:]



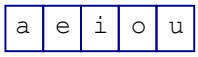
Usando essa "visão recursiva" de uma string, você consegue escrever uma função **reverte** recursiva?

Dicas

- ✓ o caso "base" ocorre quando a string é vazia
- ✓ no passo recursivo, podemos encontrar o reverso da "fatia" que começa depois da primeira letra da string ...

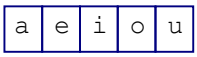
Solução Reverte Recursiva (1/14)

```
def reverte_rec(s):
  if len(s) == 0:
    return ""
  else:
    return reverte_rec(s[1:]) + s[0]
```



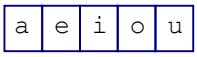
Solução Reverte Recursiva (2/14)

```
def reverte_rec(s):
  if len(s) == 0:
    return ""
  else:
    return reverte_rec(s[1:]) + s[0]


reverte_rec("aeiou")
```



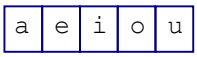
Solução Reverte Recursiva (3/14)

```
def reverte_rec(s):
  if len(s) == 0:
    return ""
  else:
    return reverte_rec(s[1:]) + s[0]


reverte_rec("aeiou")
  reverte_rec("eiou")
```



Solução Reverte Recursiva (4/14)

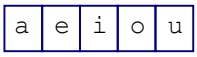
```
def reverte_rec(s):
  if len(s) == 0:
    return ""
  else:
    return reverte_rec(s[1:]) + s[0]
```




```
reverte_rec("aeiou")
  reverte_rec("eiou")
    reverte_rec("iou")
```


Solução Reverte Recursiva (5/14)

```
def reverte_rec(s):
  if len(s) == 0:
    return ""
  else:
    return reverte_rec(s[1:]) + s[0]
```




```
reverte_rec("aeiou")
  reverte_rec("eiou")
    reverte_rec("iou")
      reverte_rec("ou")
```

Solução Reverte Recursiva (6/14)

```
def reverte_rec(s):
  if len(s) == 0:
    return ""
  else:
    return reverte_rec(s[1:]) + s[0]
```

a	e	i	o	u

```
reverte_rec("aeiou")
  reverte_rec("eiou")
    reverte_rec("iou")
      reverte_rec("ou")
        reverte_rec("u")
```

53

Solução Reverte Recursiva (7/14)

```
def reverte_rec(s):
  if len(s) == 0:
    return ""
  else:
    return reverte_rec(s[1:]) + s[0]
```

a	e	i	o	u

```
reverte_rec("aeiou")
  reverte_rec("eiou")
    reverte_rec("iou")
      reverte_rec("ou")
        reverte_rec("u")
          reverte_rec("")
```

54

Solução Reverte Recursiva (8/14)

```
def reverte_rec(s):
  if len(s) == 0:
    return ""
  else:
    return reverte_rec(s[1:]) + s[0]
```

a	e	i	o	u

```
reverte_rec("aeiou")
  reverte_rec("eiou")
    reverte_rec("iou")
      reverte_rec("ou")
        reverte_rec("u")
          reverte_rec("")  ——————→  ""
```

55

Solução Reverte Recursiva (9/14)

```
def reverte_rec(s):
  if len(s) == 0:
    return ""
  else:
    return reverte_rec(s[1:]) + s[0]
```

a	e	i	o	u

```
reverte_rec("aeiou")
  reverte_rec("eiou")
    reverte_rec("iou")
      reverte_rec("ou")
        reverte_rec("u")  ——————→  ""  + "u"
          reverte_rec("")  ——————→  ""
```

56

Solução Reverte Recursiva (10/14)

```
def reverte_rec(s):
  if len(s) == 0:
    return ""
  else:
    return reverte_rec(s[1:]) + s[0]
```

a	e	i	o	u

```
reverte_rec("aeiou")
  reverte_rec("eiou")
    reverte_rec("iou")
      reverte_rec("ou")      ──────►  "u"  + "o"
        reverte_rec("u")     ──────►  ""   + "u"
          reverte_rec("")    ──────►  ""
```

57

Solução Reverte Recursiva (11/14)

```
def reverte_rec(s):
  if len(s) == 0:
    return ""
  else:
    return reverte_rec(s[1:]) + s[0]
```

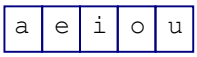
a	e	i	o	u

```
reverte_rec("aeiou")
  reverte_rec("eiou")
    reverte_rec("iou")     ──────►  "uo" + "i"
      reverte_rec("ou")    ──────►  "u"  + "o"
        reverte_rec("u")   ──────►  ""   + "u"
          reverte_rec("")  ──────►  ""
```

58

Solução Reverte Recursiva (12/14)

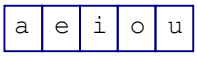
```
def reverte_rec(s):
  if len(s) == 0:
    return ""
  else:
    return reverte_rec(s[1:]) + s[0]
```




```
reverte_rec("aeiou")
  reverte_rec("eiou")      ⟶ "uoi" + "e"
    reverte_rec("iou")     ⟶ "uo" + "i"
      reverte_rec("ou")    ⟶ "u"  + "o"
        reverte_rec("u")   ⟶ ""   + "u"
          reverte_rec("")  ⟶ ""
```


Solução Reverte Recursiva (13/14)

```
def reverte_rec(s):
  if len(s) == 0:
    return ""
  else:
    return reverte_rec(s[1:]) + s[0]
```




```
reverte_rec("aeiou")     ⟶ "uoie" + "a"
  reverte_rec("eiou")    ⟶ "uoi" + "e"
    reverte_rec("iou")   ⟶ "uo" + "i"
      reverte_rec("ou")  ⟶ "u"  + "o"
        reverte_rec("u") ⟶ ""   + "u"
          reverte_rec("") ⟶ ""
```

Solução Reverte Recursiva (14/14)

```
def reverte_rec(s):
  if len(s) == 0:
    return ""
  else:
    return reverte_rec(s[1:]) + s[0]
```

a	e	i	o	u

```
                                  ──────────→ "uoiea"
reverte_rec("aeiou")              ──────────→ "uoie" + "a"
  reverte_rec("eiou")             ──────────→ "uoi" + "e"
    reverte_rec("iou")            ──────────→ "uo" + "i"
      reverte_rec("ou")           ──────────→ "u"  + "o"
        reverte_rec("u")          ──────────→ ""   + "u"
          reverte_rec("")         ──────────→ ""
```

61