U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75789
ARIEL GUSTAVO RODRIGUES DA S
24PRG405 - 1 _____ / _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	`{0:10}`
alinhar à esquerda	`{0:<10}`
centralizar no espaço	`{0:^10}`
preencher o espaço com .	`{0:.<10}`
número: tamanho 5 com zeros	`{0:05}`
idem: 3 casas após a vírgula	`{0:10.3}`
com + ou -	`{0:-10}{1:+3}`
em binário	`{0:b}`
em hexadecimal*`{0:x} {1:X}`	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere FA="YMGTLNU"
Efetue todos os fatiamentos.
 a) FA[1:7:2]
 b) FA[1:-1]
 c) FA[-6:5]
 d) FA[3:7]
 e) FA[3:6]

2. Considere YG="KAIMTHS"
Efetue todos os fatiamentos.
 a) YG[3:7]
 b) YG[-6:6:3]
 c) YG[:5:3]
 d) YG[:5:-1]
 e) YG[3:-3]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

405-75789 - /

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75796
CASSIANO MAGNO CHAGAS E SA 24PRG405 - 2 _____ / _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	`{0:10}`
alinhar à esquerda	`{0:<10}`
centralizar no espaço	`{0:^10}`
preencher o espaço com .	`{0:.<10}`
número: tamanho 5 com zeros	`{0:05}`
idem: 3 casas após a vírgula	`{0:10.3}`
com + ou -	`{0:-10}{1:+3}`
em binário	`{0:b}`
em hexadecimal*	`{0:x} {1:X}`

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere PR="FAWPKV"
Efetue todos os fatiamentos.
 a) PR[:4]
 b) PR[:4:-3]
 c) PR[:4:-3]
 d) PR[1:5]
 e) PR[:6]

2. Considere MT="GMCHEFAS"
Efetue todos os fatiamentos.
 a) MT[:8:2]
 b) MT[2:6:3]
 c) MT[1:7:3]
 d) MT[3:6:2]
 e) MT[-6:7:2]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75808
CHRISTOPHER BUCH FILIPAK 24PRG405 - 3 / _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	`{0:10}`
alinhar à esquerda	`{0:<10}`
centralizar no espaço	`{0:^10}`
preencher o espaço com .	`{0:.<10}`
número: tamanho 5 com zeros	`{0:05}`
idem: 3 casas após a vírgula	`{0:10.3}`
com + ou -	`{0:-10}{1:+3}`
em binário	`{0:b}`
em hexadecimal*`{0:x} {1:X}`	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere QJ="DBWLMXR"
Efetue todos os fatiamentos.
 a) QJ[-5:5:2]
 b) QJ[:5:-3]
 c) QJ[3:7:2]
 d) QJ[-7:-2:3]
 e) QJ[:5]

2. Considere YF="SRFJUZ"
Efetue todos os fatiamentos.
 a) YF[3:-2:2]
 b) YF[3:4:3]
 c) YF[:6]
 d) YF[-4:5]
 e) YF[:5:2]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75815
ERAN MARTINEZ RAMOS 24PRG405 - 4
/ _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	{0:10}
alinhar à esquerda	{0:<10}
centralizar no espaço	{0:^10}
preencher o espaço com .	{0:.<10}
número: tamanho 5 com zeros	{0:05}
idem: 3 casas após a vírgula	{0:10.3}
com + ou -	{0:-10}{1:+3}
em binário	{0:b}
em hexadecimal*{0:x} {1:X}	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere PE="JXEMHKD"
Efetue todos os fatiamentos.
 a) PE[1:6:3]
 b) PE[:5:3]
 c) PE[2:5]
 d) PE[:5:-2]
 e) PE[3:7]

2. Considere SF="ALJMDEU"
Efetue todos os fatiamentos.
 a) SF[:5:-3]
 b) SF[-7:7]
 c) SF[-6:-3:2]
 d) SF[:6:2]
 e) SF[:7:2]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

405-75815 - /

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75822
HENRIQUE IVANKIO GUSSE 24PRG405 - 5 / _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	{0:10}
alinhar à esquerda	{0:<10}
centralizar no espaço	{0:^10}
preencher o espaço com .	{0:.<10}
número: tamanho 5 com zeros	{0:05}
idem: 3 casas após a vírgula	{0:10.3}
com + ou -	{0:-10}{1:+3}
em binário	{0:b}
em hexadecimal*{0:x} {1:X}	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere KP="LVJXIPK"
Efetue todos os fatiamentos.
 a) KP[-7:-2:2]
 b) KP[1:-3:2]
 c) KP[-7:-3:2]
 d) KP[:7]
 e) KP[2:-3:2]

2. Considere PI="PYFAZSTD"
Efetue todos os fatiamentos.
 a) PI[:7:2]
 b) PI[:7:2]
 c) PI[2:7:2]
 d) PI[3:6:2]
 e) PI[-8:7:2]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75839 IAN DOLABELLA OHATA 24PRG405 - 6 _____ / / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	`{0:10}`
alinhar à esquerda	`{0:<10}`
centralizar no espaço	`{0:^10}`
preencher o espaço com .	`{0:.<10}`
número: tamanho 5 com zeros	`{0:05}`
idem: 3 casas após a vírgula	`{0:10.3}`
com + ou -	`{0:-10}{1:+3}`
em binário	`{0:b}`
em hexadecimal*`{0:x} {1:X}`	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere KH="ZBEDHR"
Efetue todos os fatiamentos.
 a) KH[-6:5:2]
 b) KH[1:-1:3]
 c) KH[:6]
 d) KH[1:6]
 e) KH[:4:2]

2. Considere WF="KIFJLD"
Efetue todos os fatiamentos.
 a) WF[1:4:2]
 b) WF[-6:-3]
 c) WF[2:4]
 d) WF[-5:-3:2]
 e) WF[3:-2]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75846
JOAO BONFIN LINO 24PRG405 - 7 _____ / / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	`{0:10}`
alinhar à esquerda	`{0:<10}`
centralizar no espaço	`{0:^10}`
preencher o espaço com .	`{0:.<10}`
número: tamanho 5 com zeros	`{0:05}`
idem: 3 casas após a vírgula	`{0:10.3}`
com + ou -	`{0:-10}{1:+3}`
em binário	`{0:b}`
em hexadecimal*`{0:x} {1:X}`	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere XZ="FAODVIP"
Efetue todos os fatiamentos.
 a) XZ[:5:2]
 b) XZ[-5:7]
 c) XZ[-5:7:2]
 d) XZ[3:5]
 e) XZ[-6:5:3]

2. Considere TO="LHYIERON"
Efetue todos os fatiamentos.
 a) TO[:6:-2]
 b) TO[:6:2]
 c) TO[-6:7:2]
 d) TO[:6:-3]
 e) TO[:7]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75853
JOAO LUCA FERNANDES DE OLIVE
24PRG405 - 8 _____ / _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	{0:10}
alinhar à esquerda	{0:<10}
centralizar no espaço	{0:^10}
preencher o espaço com .	{0:.<10}
número: tamanho 5 com zeros	{0:05}
idem: 3 casas após a vírgula	{0:10.3}
com + ou -	{0:-10}{1:+3}
em binário	{0:b}
em hexadecimal*{0:x} {1:X}	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere XJ="YNDUKFA"
Efetue todos os fatiamentos.
 a) XJ[-6:7]
 b) XJ[3:5:2]
 c) XJ[:5:2]
 d) XJ[-6:5:2]
 e) XJ[:7:3]

2. Considere ZC="POCUMF"
Efetue todos os fatiamentos.
 a) ZC[:6:3]
 b) ZC[2:4:3]
 c) ZC[:5]
 d) ZC[2:-1]
 e) ZC[1:5:3]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

405-75853 - /

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75860
JOAO PEDRO DE BRITO DUARTE 24PRG405 - 9 _____ / _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	`{0:10}`
alinhar à esquerda	`{0:<10}`
centralizar no espaço	`{0:^10}`
preencher o espaço com .	`{0:.<10}`
número: tamanho 5 com zeros	`{0:05}`
idem: 3 casas após a vírgula	`{0:10.3}`
com + ou -	`{0:-10}{1:+3}`
em binário	`{0:b}`
em hexadecimal*`{0:x} {1:X}`	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere CB="OWBINMHGV"
Efetue todos os fatiamentos.
 a) CB[2:8:2]
 b) CB[-8:7:2]
 c) CB[1:9:2]
 d) CB[3:9:2]
 e) CB[:8:3]

2. Considere XL="FCNHMPJ"
Efetue todos os fatiamentos.
 a) XL[2:6]
 b) XL[2:7]
 c) XL[-6:7:2]
 d) XL[-7:5:2]
 e) XL[:5:2]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75877
JOAO PEDRO MARIANO SUEKE 24PRG405 - 10 _____ / _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	`{0:10}`
alinhar à esquerda	`{0:<10}`
centralizar no espaço	`{0:^10}`
preencher o espaço com .	`{0:.<10}`
número: tamanho 5 com zeros	`{0:05}`
idem: 3 casas após a vírgula	`{0:10.3}`
com + ou -	`{0:-10}{1:+3}`
em binário	`{0:b}`
em hexadecimal*`{0:x} {1:X}`	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere DU="YHECFJIQN"
Efetue todos os fatiamentos.
 a) DU[:7:-1]
 b) DU[1:7:2]
 c) DU[1:9:2]
 d) DU[:9:2]
 e) DU[3:-2:2]

2. Considere BH="PTGJHS"
Efetue todos os fatiamentos.
 a) BH[3:6]
 b) BH[-5:5:2]
 c) BH[:6]
 d) BH[2:4:2]
 e) BH[3:5]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

405-75877 - /

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75884
JOAO VICTOR GOMES VILELA GER
24PRG405 - 11 _____ / _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	`{0:10}`
alinhar à esquerda	`{0:<10}`
centralizar no espaço	`{0:^10}`
preencher o espaço com .	`{0:.<10}`
número: tamanho 5 com zeros	`{0:05}`
idem: 3 casas após a vírgula	`{0:10.3}`
com + ou -	`{0:-10}{1:+3}`
em binário	`{0:b}`
em hexadecimal*`{0:x} {1:X}`	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere WT="TDZVUICKM"
Efetue todos os fatiamentos.
 a) WT[3:-1:2]
 b) WT[:7:2]
 c) WT[-7:-2:2]
 d) WT[:7:-1]
 e) WT[-8:9:2]

2. Considere ZC="DEKOXJF"
Efetue todos os fatiamentos.
 a) ZC[3:-2:2]
 b) ZC[1:6:3]
 c) ZC[-5:-3:2]
 d) ZC[:7]
 e) ZC[:7]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75989
LUCAS ANTOSZCZYSZEN 24PRG405 - 12
/ _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	{0:10}
alinhar à esquerda	{0:<10}
centralizar no espaço	{0:^10}
preencher o espaço com .	{0:.<10}
número: tamanho 5 com zeros	{0:05}
idem: 3 casas após a vírgula	{0:10.3}
com + ou -	{0:-10}{1:+3}
em binário	{0:b}
em hexadecimal*{0:x} {1:X}	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere ZC="LDHZOCAM"
Efetue todos os fatiamentos.
 a) ZC[-8:6:3]
 b) ZC[3:7:3]
 c) ZC[:6:2]
 d) ZC[-7:-3:2]
 e) ZC[:7]

2. Considere NY="GTEYLA"
Efetue todos os fatiamentos.
 a) NY[2:5]
 b) NY[2:-2:2]
 c) NY[-5:4:3]
 d) NY[:5:2]
 e) NY[2:-2]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

405-75989 - /

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75891
LUIZ FELIPE GUEDES BUCCHERI 24PRG405 - 13 _____ / _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	`{0:10}`
alinhar à esquerda	`{0:<10}`
centralizar no espaço	`{0:^10}`
preencher o espaço com .	`{0:.<10}`
número: tamanho 5 com zeros	`{0:05}`
idem: 3 casas após a vírgula	`{0:10.3}`
com + ou -	`{0:-10}{1:+3}`
em binário	`{0:b}`
em hexadecimal*`{0:x} {1:X}`	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere AF="CENOFQ"
Efetue todos os fatiamentos.
 a) AF[-6:5]
 b) AF[2:-1:2]
 c) AF[2:5:2]
 d) AF[-5:6:2]
 e) AF[:4]

2. Considere YW="MJQVLIZGU"
Efetue todos os fatiamentos.
 a) YW[2:8:3]
 b) YW[3:-3]
 c) YW[3:7]
 d) YW[-8:7]
 e) YW[3:-2]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75903
MANUELLA LEAL DE MEIRELLES F
24PRG405 - 14 _____ / _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	`{0:10}`
alinhar à esquerda	`{0:<10}`
centralizar no espaço	`{0:^10}`
preencher o espaço com .	`{0:.<10}`
número: tamanho 5 com zeros	`{0:05}`
idem: 3 casas após a vírgula	`{0:10.3}`
com + ou -	`{0:-10}{1:+3}`
em binário	`{0:b}`
em hexadecimal*`{0:x} {1:X}`	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere PZ="CRQIUGFY"
Efetue todos os fatiamentos.
 a) PZ[:8]
 b) PZ[-8:6:3]
 c) PZ[2:6:2]
 d) PZ[3:6:2]
 e) PZ[-7:6]

2. Considere SN="IUDKJCH"
Efetue todos os fatiamentos.
 a) SN[3:6:3]
 b) SN[:7]
 c) SN[1:5]
 d) SN[:6:2]
 e) SN[3:6:3]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75910
MARIA EDUARDA DA SILVA 24PRG405 - 15 / _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	`{0:10}`
alinhar à esquerda	`{0:<10}`
centralizar no espaço	`{0:^10}`
preencher o espaço com .	`{0:.<10}`
número: tamanho 5 com zeros	`{0:05}`
idem: 3 casas após a vírgula	`{0:10.3}`
com + ou -	`{0:-10}{1:+3}`
em binário	`{0:b}`
em hexadecimal*`{0:x} {1:X}`	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma ”code page”(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere UR="RGQEVTIK"
Efetue todos os fatiamentos.
 a) UR[2:-2:2]
 b) UR[:6:-2]
 c) UR[-6:6:3]
 d) UR[3:7:2]
 e) UR[-7:6]

2. Considere NY="IUNBDXMP"
Efetue todos os fatiamentos.
 a) NY[1:6]
 b) NY[1:-2]
 c) NY[:7:3]
 d) NY[3:6:2]
 e) NY[-7:8:2]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

405-75910 - /

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75927
MATHEUS AITA FABRICIO DE CAR
24PRG405 - 16 _____ / _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	{0:10}
alinhar à esquerda	{0:<10}
centralizar no espaço	{0:^10}
preencher o espaço com .	{0:.<10}
número: tamanho 5 com zeros	{0:05}
idem: 3 casas após a vírgula	{0:10.3}
com + ou -	{0:-10}{1:+3}
em binário	{0:b}
em hexadecimal*{0:x} {1:X}	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere ID="HSGZDXQR"
Efetue todos os fatiamentos.
 a) ID[-7:6:2]
 b) ID[2:6:2]
 c) ID[1:6]
 d) ID[:6:2]
 e) ID[-7:-1]

2. Considere HW="ONEYQKBS"
Efetue todos os fatiamentos.
 a) HW[:8:2]
 b) HW[:6:-2]
 c) HW[2:7:3]
 d) HW[-8:-3:2]
 e) HW[3:8]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75934
MATHEUS LEANDRO DE BITENCOUR
24PRG405 - 17 _____ / _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	`{0:10}`
alinhar à esquerda	`{0:<10}`
centralizar no espaço	`{0:^10}`
preencher o espaço com .	`{0:.<10}`
número: tamanho 5 com zeros	`{0:05}`
idem: 3 casas após a vírgula	`{0:10.3}`
com + ou -	`{0:-10}{1:+3}`
em binário	`{0:b}`
em hexadecimal*`{0:x} {1:X}`	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma ”code page”(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere HP="GRFSANVUM"
Efetue todos os fatiamentos.
 a) HP[3:7:2]
 b) HP[-9:-2]
 c) HP[2:-3]
 d) HP[3:8:2]
 e) HP[3:-2:3]

2. Considere IL="TEXHORSI"
Efetue todos os fatiamentos.
 a) IL[-8:6:3]
 b) IL[3:7:2]
 c) IL[:6:-2]
 d) IL[-6:8]
 e) IL[3:6]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75941
MAYCON TAVARES WOLPE 24PRG405 - 18 / _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	`{0:10}`
alinhar à esquerda	`{0:<10}`
centralizar no espaço	`{0:^10}`
preencher o espaço com .	`{0:.<10}`
número: tamanho 5 com zeros	`{0:05}`
idem: 3 casas após a vírgula	`{0:10.3}`
com + ou -	`{0:-10}{1:+3}`
em binário	`{0:b}`
em hexadecimal*`{0:x} {1:X}`	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere QO="JDHSGURC"
Efetue todos os fatiamentos.
 a) QO[:6:-2]
 b) QO[1:-1]
 c) QO[2:8:2]
 d) QO[1:8:2]
 e) QO[-6:-2]

2. Considere KC="BHTNPFJDX"
Efetue todos os fatiamentos.
 a) KC[3:-2:2]
 b) KC[:7:-1]
 c) KC[2:7:3]
 d) KC[1:-1:2]
 e) KC[2:9:3]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

405-75941 - /

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75958
PEDRO TOLEDO LEAL 24PRG405 - 19
/ _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	`{0:10}`
alinhar à esquerda	`{0:<10}`
centralizar no espaço	`{0:^10}`
preencher o espaço com .	`{0:.<10}`
número: tamanho 5 com zeros	`{0:05}`
idem: 3 casas após a vírgula	`{0:10.3}`
com + ou -	`{0:-10}{1:+3}`
em binário	`{0:b}`
em hexadecimal*`{0:x} {1:X}`	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere EA="SOHUZNM"
Efetue todos os fatiamentos.
 a) EA[3:-2:2]
 b) EA[:5:-2]
 c) EA[:7:2]
 d) EA[:6]
 e) EA[2:-3]

2. Considere CV="EKYVLU"
Efetue todos os fatiamentos.
 a) CV[:4]
 b) CV[:6:2]
 c) CV[:4:2]
 d) CV[1:5]
 e) CV[:4:2]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

405-75958 - /

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75965
PHELIPE GABRIEL LIMA DA SILV 24PRG405 - 20 _____ / _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	`{0:10}`
alinhar à esquerda	`{0:<10}`
centralizar no espaço	`{0:^10}`
preencher o espaço com .	`{0:.<10}`
número: tamanho 5 com zeros	`{0:05}`
idem: 3 casas após a vírgula	`{0:10.3}`
com + ou -	`{0:-10}{1:+3}`
em binário	`{0:b}`
em hexadecimal*`{0:x} {1:X}`	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere TM="QDGZSK"
Efetue todos os fatiamentos.
 a) TM[3:5:2]
 b) TM[:6:2]
 c) TM[:4:2]
 d) TM[3:5:2]
 e) TM[-5:5:3]

2. Considere OL="QRTSLFVEG"
Efetue todos os fatiamentos.
 a) OL[:7:2]
 b) OL[1:-2:2]
 c) OL[2:8:2]
 d) OL[-9:-1:2]
 e) OL[:9:3]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

405-75965 - /

U Positivo - UTFPR - PUCPr – 09/09/2024 - 09:49:21.5
Prof Dr P Kantek (pkantek@gmail.com)
Strings em Python VIVX931a, V: 1.05 75972
REBECA CABRAL DOS SANTOS 24PRG405 - 21 _____ / _____ / _____

Strings em Python

Além de números, Python também pode manipular strings (sequências de caracteres), que podem ser expressas de diversas formas. Elas podem ser delimitadas por aspas simples ou duplas:

```
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
"doesn't"
>>> "doesn't"
"doesn't"
>>> '"Yes," he said.'
'"Yes," he said.'
```

Strings podem ser acessados como listas, mas eles são imutáveis, não podem ser alterados. Para alterar letras em strings, deve-se antes transformar o string em lista. Veja a ida e a volta

```
>>> L = list("Curitiba")
>>> L[0]='X'
>>> print(L)
['X', 'u', 'r', 'i', 't', 'i', 'b', 'a']
>>> s=''.join(L)
>>> s
'Xuritiba'
>>> print(s)
Xuritiba
```

Como se viu a função `list` transforma cada caractere da string em um elemento da lista retornada. O método `join` faz o inverso, transformando uma lista em um string.

O interpretador exibe o resultado de operações com strings da mesma forma como elas são digitadas na entrada: dentro de aspas, e com aspas, caracteres acentuados e outros caracteres especiais representados por sequências de escape com barras invertidas (como `'\t'`, `'\xc3\xa9'` etc.), para mostrar o valor preciso. A string é delimitada entre aspas simples, exceto quando ela contém uma aspa simples e nenhuma aspa dupla. O comando print produz uma saída mais legível para tais strings com caracteres especiais.

Strings que contém mais de uma linha podem ser construídas de diversas maneiras. Linhas de continuação podem ser usadas, com uma barra invertida colocada na última posição para indicar que a próxima linha física é a continuação de uma linha lógica:

```
oi = "Eis uma string longa contendo\n\
diversas linhas assim como se faria em C.\n\
```

Não existe um tipo específico para caracteres; um caractere é simplesmente uma string de tamanho 1. Substrings podem ser especificadas através da notação de slice (fatiamento ou intervalo): dois índices separados por dois pontos. É exatamente o mesmo conceito estudado lá nas listas.

```
>>> palavra[4]
'a'
>>> palavra[0:2]
'Aj'
>>> palavra[2:4]
'ud'
```

Diferentemente de C, strings em Python não podem ser alteradas. Tentar atribuir valor a uma posição (índice ou fatia) dentro de uma string resulta em erro:

```
>>> palavra[0] = 'x'
=== Erro ===
>>> palavra[:1] = 'Splat'
=== Erro ===
```

Entretanto, criar uma nova string combinando conteúdos é fácil e eficiente:

```
>>> 'x' + palavra[1:]
'xjudaZ'
>>> 'Splat' + palavra[5]
'SplatZ'
```

Eis uma invariante interessante das operações de fatiamento: s[:i] + s[i:] é igual a s.

Intervalos fora de limites: um índice maior que o comprimento é trocado pelo comprimento, um limite superior menor que o limite inferior produz uma string vazia.

```
>>> palavra[1:100]
'judaZ'
>>> palavra[10:]
''
>>> palavra[2:1]
''
```

Indices podem ser números negativos, para iniciar a contagem pela direita. Observe que -0 é o mesmo que 0, logo neste caso não se conta a partir da direita!

```
>>> palavra[-0]
'A'
```

A função built-in (embutida) len() devolve o comprimento de uma string:

```
>>> s = 'anticonstitucionalissimamente'
>>> len(s)
29
```

Métodos de String

Tanto strings comuns quanto Unicode oferecem um grande número de métodos para busca e transformações básicas. Eis alguns

.capitalize() Transforma o primeiro caractere de uma string em maíusculo. Ignora caracteres acentuados ou que não sejam letras.

```
>>> "curitiba".capitalize()
'Curitiba'
>>> "ágora".capitalize()
ágora
```

.center() Centraliza a string usando espaços à esquerda e à direita, até atingir a largura.

```
>>>"paraná".center(20)
'       paraná       '
>>> "paraná".center(2)
'paraná'
```

.count() Conta a quantidade de substrings dentro da string

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

.find() Retorna a primeira ocorrência do substring dentro da string. Caso nada seja encontrado, retorna -1.

```
>>> "abracadabra".count("ab")
2
>>> "abracadabra".count("a")
5
```

Avaliadores São eles: `.isalpha()`, `.isdecimal()`, `.isdigit()`, `.islower()`,`.isnumeric()`, `.isspace()`, `.istitle()`, `.isupper()`. Retornam `True` caso a string seja composta exclusivamente de alfanuméricos, decimais, dígitos, minúsculos, numéricos, espaços, títulos (a primeira letra de cada palavra em maiúsculo) e maiúsculas. Todos ignoram as letras acentuadas.

.join() Concatena todos os elementos dentro de iterável usando a string especificada como concatenador. A string pode ser vazia. Os elementos do iterável devem ser string.

```
>>> a=['agua', 'fogo', 'xabu']
>>> ''.join(a)
'aguafogoxabu'
>>> '\n'.join(a)
'agua\nfogo\nxabu'
>>> print('\n'.join(a))
agua
fogo
xabu
```

.partition() Divide a string em uma tupla com 3 elementos: a parte anterior à primeira ocorrência do `separador`, o separador e a parte posterior ao separador

```
>>> "curitibaparanabrasil".partition('ra')
('curitibapa', 'ra', 'nabrasil')
>>> "curitibaparanabrasil".partition('z')
('curitibaparanabrasil', '', '')
```

.replace() Substitui n ocorrências da substring `antiga` pela `nova`. Se n não é fornecido, ele substitui todas.

```
>>> "curitibaparanabrasil".replace("ra","agora")
'curitibapaagoranabagorasil'
>>> "curitibaparanabrasil".replace("ra","zz",1)
'curitibapazznabrasil'
```

.split() Divide a string transformando-a em uma lista usando um separador n vezes. Se o separador não é informado ele usa um espaço em branco.

```
>>> "curitibaparanabrasil".split('a')
['curitib', 'p', 'r', 'n', 'br', 'sil']
>>> "curitiba parana brasil".split()
['curitiba', 'parana', 'brasil']
```

Muitos outros Aqui estão só alguns, talvez os mais úteis e/ou principais. Consulte a documentação da versão de Python que você está usando. Estão todos lá.

Formatação

Além da formatação padrão C (com %d, %s, %f, etc), Python introduziu a formatação representando os parâmetros por números em chaves. Veja o exemplo

```
>>> "{0} traga {1} aqui".format("Ze", 4)
'Ze traga 4 aqui'
```

Graças a esta facilidade, pode-se repetir o mesmo valor na frase e reordenar os parâmetros. Eis algumas coisas que se podem fazer

estabelecer o tamanho	`{0:10}`
alinhar à esquerda	`{0:<10}`
centralizar no espaço	`{0:^10}`
preencher o espaço com .	`{0:.<10}`
número: tamanho 5 com zeros	`{0:05}`
idem: 3 casas após a vírgula	`{0:10.3}`
com + ou -	`{0:-10}{1:+3}`
em binário	`{0:b}`
em hexadecimal*`{0:x} {1:X}`	

Strings Unicode

Um novo tipo para armazenar textos foi introduzido: o tipo unicode. Ele pode ser usado para armazenar e manipular dados no padrão Unicode (veja http://www.unicode.org/) e se integra bem aos objetos string pré-existentes, realizando conversões automáticas quando necessário.

Unicode tem a vantagem de associar um único número ordinal a cada caractere, para todas as formas de escrita usadas em textos modernos ou antigos. Anteriormente, havia somente 256 números ordinais possíveis para identificar caracteres. Cada texto era tipicamente limitado a uma "code page"(uma tabela de códigos) que associava ordinais aos caracteres. Isso levou a muita confusão especialmente no âmbito da internacionalização de software. Unicode resolve esses problemas ao definir uma única tabela de códigos para todos os conjuntos de caracteres.

Criar strings Unicode em Python é tão simples quanto criar strings normais:

```
>>> u'Hello World !'
u'Hello World !'
```

O u antes das aspas indica a criação de uma string Unicode. Se você desejar incluir caracteres especiais na string, você pode fazê-lo através da codificação Unicode-Escape de Python. O exemplo a seguir mostra como:

```
>>> u'Hello\u0020World !'
u'Hello World !'
```

O código de escape `\u0020` insere um caractere Unicode com valor ordinal 0x0020 (o espaço em branco) naquela posição.

☞ Para você fazer

```
1. Considere PO="ZPMLHUC"
Efetue todos os fatiamentos.
 a) PO[-7:-3:2]
 b) PO[-5:5:2]
 c) PO[:5:-1]
 d) PO[1:6]
 e) PO[:7:3]

2. Considere LJ="QPVFGI"
Efetue todos os fatiamentos.
 a) LJ[1:5]
 b) LJ[3:5:2]
 c) LJ[1:-3]
 d) LJ[-4:-2:2]
 e) LJ[3:4:2]
```

Responda aqui:

1a	1b	1c	1d	1e
2a	2b	2c	2d	2e

405-75972 - /