Python - Formatação de saída

Neste capítulo, serão discutidas diferentes técnicas para formatar a saída.

Operador de formatação de string

Um dos recursos mais interessantes do Python é o operador de formato de string%. Este operador é exclusivo para strings e compensa o conjunto de funções da família printf() do C. Símbolos de especificação de formato (%d %c %f %s etc) usados na linguagem C são usados como espaços reservados em uma string.

A seguir está um exemplo simples -

```
print ("My name is %s and weight is %d kg!" % ('Zara', 21))
```

Ele produzirá a seguinte **saída** -

```
My name is Zara and weight is 21 kg!
```

O método format()

Python 3.0 introduziu o método format() na classe str para lidar com formatação de strings complexas com mais eficiência. Desde então, este método foi portado para Python 2.6 e Python 2.7.

Este método de classe de string integrada oferece a capacidade de fazer substituições complexas de variáveis e formatação de valores. Esta nova técnica de formatação é considerada mais elegante.

Sintaxe

A sintaxe geral do método format() é a seguinte -

```
str.format(var1, var2,...)
```

Valor de retorno

O método retorna uma string formatada.

A própria string contém espaços reservados {} nos quais os valores das variáveis são inseridos sucessivamente.

Exemplo

```
name="Rajesh"
age=23
print ("my name is {} and my age is {} years".format(name, age))
```

Ele produzirá a seguinte **saída** -

```
my name is Rajesh and my age is 23 years
```

Você pode usar variáveis como argumentos de palavra-chave para o método format() e usar o nome da variável como espaço reservado na string.

```
print ("my name is {name} and my age is {age}
years".format(name="Rajesh", age=23))
```

Você também pode especificar símbolos de formatação de estilo C. A única mudança é usar: em vez de%. Por exemplo, em vez de %s use {:s} e em vez de %d use (:d}

```
name="Rajesh"
age=23
print ("my name is {:s} and my age is {:d} years".format(name, age))
```

cordas F

Em Python, strings f ou interpolação de string literal é outro recurso de formatação. Com este método de formatação você pode usar expressões Python incorporadas dentro de constantes de string. As strings F do Python são mais rápidas, mais legíveis, mais concisas e menos propensas a erros.

A string começa com um prefixo 'f', e nela são inseridos um ou mais placeholders, cujo valor é preenchido dinamicamente.

```
name = 'Rajesh'
age = 23

fstring = f'My name is {name} and I am {age} years old'
print (fstring)
```

Ele produzirá a seguinte **saída** -

```
My name is Rajesh and I am 23 years old
```

Sequências de modelo

A classe Template no módulo string fornece um método alternativo para formatar as strings dinamicamente. Um dos benefícios da classe Template é poder customizar as regras de formatação.

Uma string de modelo válida, ou espaço reservado, consiste em duas partes: O símbolo "$" seguido por um identificador Python válido.

Você precisa criar um objeto da classe Template e usar a string do template como argumento para o construtor.

A seguir, chame o método replace() da classe Template. Ele coloca os valores fornecidos como parâmetros no lugar das strings do modelo.

Exemplo

```
from string import Template

temp_str = "My name is $name and I am $age years old"
tempobj = Template(temp_str)
ret = tempobj.substitute(name='Rajesh', age=23)
print (ret)
```

Ele produzirá a seguinte **saída** -

```
My name is Rajesh and I am 23 years old
```


O Módulo de quebra de texto

A classe wrap no módulo textwrap do Python contém funcionalidade para formatar e agrupar textos simples ajustando as quebras de linha no parágrafo de entrada. Ajuda a deixar o texto bem formatado e bonito.

O módulo textwrap possui as seguintes funções convenientes -

textwrap.wrap(texto, largura=70)

Envolve o parágrafo único no texto (uma string) para que cada linha tenha no máximo caracteres de largura. Retorna uma lista de linhas de saída, sem novas linhas finais. Argumentos de palavras-chave opcionais correspondem aos atributos de instância do TextWrapper. a largura padrão é 70.

textwrap.fill(texto, largura=70)

Quebra o parágrafo único no texto e retorna uma única string contendo o parágrafo quebrado.

Ambos os métodos criam internamente um objeto da classe TextWrapper e chamam um único método nele. Como a instância não é reutilizada, será mais eficiente criar seu próprio objeto TextWrapper.

Exemplo

```
import textwrap

text = '''
Python is a high-level, general-purpose programming language. Its design philosophy e

Python is dynamically typed and garbage-collected. It supports multiple programming p
'''

wrapper = textwrap.TextWrapper(width=40)
wrapped = wrapper.wrap(text = text)

# Print output
for element in wrapped:
   print(element)
```

Ele produzirá a seguinte **saída** -

```
Python is a high-level, general-purpose
programming language. Its design
philosophy emphasizes code readability
with the use of significant indentation
via the off-side rule. Python is
dynamically typed and garbage-collected.
It supports multiple programming
paradigms, including structured
(particularly procedural), objectoriented and functional programming. It
is often described as a "batteries
included" language due to its
comprehensive standard library.
```

Os seguintes atributos são definidos para um objeto TextWrapper -

- **width** - (padrão: 70) O comprimento máximo das linhas quebradas.
- **expand_tabs** - (padrão: True) Se for verdadeiro, todos os caracteres de tabulação no texto serão expandidos para espaços usando o método expandtabs() de texto.

- **tabsize** - (padrão: 8) Se expand_tabs for verdadeiro, todos os caracteres de tabulação no texto serão expandidos para zero ou mais espaços, dependendo da coluna atual e do tamanho da tabulação fornecido.
- **replace_whitespace** - (padrão: True) Se for verdade, após a expansão da guia, mas antes da quebra, o método wrap() substituirá cada caractere de espaço em branco por um único espaço.
- **drop_whitespace** - (padrão: True) Se verdadeiro, os espaços em branco no início e no final de cada linha (após quebra automática, mas antes do recuo) são eliminados. O espaço em branco no início do parágrafo, entretanto, não será eliminado se não houver espaço em branco após ele. Se o espaço em branco eliminado ocupar uma linha inteira, a linha inteira será eliminada.
- **inicial_indent** - (padrão: '') String que será anexada à primeira linha da saída agrupada.
- **subseqüente_indent** - (padrão: '') String que será anexada a todas as linhas da saída agrupada, exceto a primeira.
- **fix_sentence_endings** - (padrão: False) Se verdadeiro, TextWrapper tenta detectar finais de frases e garantir que as frases sejam sempre separadas por exatamente dois espaços. Isso geralmente é desejado para texto em fonte monoespaçada.
- **break_long_words** - (padrão: True) Se for verdade, as palavras maiores que a largura serão quebradas para garantir que nenhuma linha seja maior que a largura. Se for falso, palavras longas não serão quebradas e algumas linhas poderão ser maiores que a largura.
- **break_on_hyphens** - (padrão: True) Se verdadeiro, a quebra ocorrerá preferencialmente em espaços em branco e logo após hífens em palavras compostas, como é habitual em inglês. Se for falso, apenas os espaços em branco serão considerados locais potencialmente bons para quebras de linha.

A função encurtar()

Recolha e trunque o texto fornecido para caber na largura especificada. O texto primeiro tem seu espaço em branco recolhido. Se couber na *largura*, será retornado como está. Caso contrário, tantas palavras quanto possível serão unidas e o espaço reservado será anexado -

Exemplo

```
import textwrap

python_desc = """Python is a general-purpose interpreted, interactive, object-oriente
```

```
my_wrap = textwrap.TextWrapper(width = 40)

short_text = textwrap.shorten(text = python_desc, width=150)
print('\n\n' + my_wrap.fill(text = short_text))
```

Ele produzirá a seguinte **saída** -

```
Python is a general-purpose interpreted,
interactive, object-oriented,and high
level programming language. It was
created by Guido van Rossum [...]
```


O módulo pprint

O módulo pprint na biblioteca padrão do Python permite uma aparência esteticamente bonita das estruturas de dados do Python. O nome pprint significa impressora bonita. Qualquer estrutura de dados que possa ser analisada corretamente pelo interpretador Python é formatada com elegância.

A expressão formatada é mantida em uma linha tanto quanto possível, mas é dividida em várias linhas se o comprimento exceder o parâmetro de largura da formatação. Um recurso exclusivo da saída pprint é que os dicionários são classificados automaticamente antes que a representação de exibição seja formatada.

Classe PrettyPrinter

O módulo pprint contém a definição da classe PrettyPrinter. Seu construtor segue o seguinte formato -

Sintaxe

```
pprint.PrettyPrinter(indent, width, depth, stream, compact)
```

Parâmetros

- **indent** - define o recuo adicionado em cada nível recursivo. O padrão é 1.
- **width** - o padrão é 80. A saída desejada é restrita por este valor. Se o comprimento for maior que a largura, ele será dividido em várias linhas.

- **profundidade** - controla o número de níveis a serem impressos.
- **stream** - é por padrão std.out - o dispositivo de saída padrão. Pode levar qualquer objeto de fluxo, como arquivo.
- **compact** - id definido como False por padrão. Se verdadeiro, apenas os dados ajustáveis na largura serão exibidos.

A classe PrettyPrinter define os seguintes métodos -

método print()

imprime a representação formatada do objeto PrettyPrinter.

Método pformat()

Retorna a representação formatada do objeto, com base nos parâmetros do construtor.

Exemplo

O exemplo a seguir demonstra um uso simples da classe PrettyPrinter -

```
import pprint
students={"Dilip":["English", "Maths", "Science"],"Raju":{"English":50,"Maths":60, "S
pp=pprint.PrettyPrinter()
print ("normal print output")
print (students)
print ("----")
print ("pprint output")
pp.pprint(students)
```

A saída mostra uma exibição de impressão normal e bonita -

```
normal print output
{'Dilip': ['English', 'Maths', 'Science'], 'Raju': {'English': 50, 'Maths': 60, 'Science': 70}, 'Kalpana':
----
pprint output
{'Dilip': ['English', 'Maths', 'Science'],
 'Kalpana': (50, 60, 70),
 'Raju': {'English': 50, 'Maths': 60, 'Science': 70}}
```

O módulo **pprint** também define as funções de conveniência pprint() e pformat() correspondentes aos métodos PrettyPrinter. O exemplo abaixo usa a função pprint().

```
from pprint import pprint
students={"Dilip":["English", "Maths", "Science"],
```

```
   "Raju":{"English":50,"Maths":60, "Science":70},
      "Kalpana":(50,60,70)}
print ("normal print output")
print (students)
print ("----")
print ("pprint output")
pprint (students)
```

Exemplo

O próximo exemplo usa o método pformat() e também a função pformat(). Para usar o método pformat(), o objeto PrettyPrinter é primeiro configurado. Em ambos os casos, a representação formatada é exibida usando a função print() normal.

```
import pprint
students={"Dilip":["English", "Maths", "Science"],
   "Raju":{"English":50,"Maths":60, "Science":70},
      "Kalpana":(50,60,70)}
print ("using pformat method")
pp=pprint.PrettyPrinter()
string=pp.pformat(students)
print (string)
print ('------')
print ("using pformat function")
string=pprint.pformat(students)
print (string)
```

Aqui está a saída do código acima -

```
using pformat method
{'Dilip': ['English', 'Maths', 'Science'],
 'Kalpana': (50, 60, 70),
 'Raju': {'English': 50, 'Maths': 60, 'Science': 70}}
------
using pformat function
{'Dilip': ['English', 'Maths', 'Science'],
 'Kalpana': (50, 60, 70),
 'Raju': {'English': 50, 'Maths': 60, 'Science': 70}}
```

A impressora bonita também pode ser usada com classes personalizadas. Dentro da classe o método __repr__() é substituído. O método __repr__() é chamado quando a função repr() é usada. É a representação oficial de string do objeto Python. Quando usamos o objeto como parâmetro para a função print(), ele imprime o valor de retorno da função repr().

Exemplo

Neste exemplo, o método __repr__() retorna a representação em string do objeto player -

```
import pprint
class player:
   def __init__(self, name, formats=[], runs=[]):
      self.name=name
      self.formats=formats
      self.runs=runs
   def __repr__(self):
      dct={}
      dct[self.name]=dict(zip(self.formats,self.runs))
      return (repr(dct))

l1=['Tests','ODI','T20']
l2=[[140, 45, 39],[15,122,36,67, 100, 49],[78,44, 12, 0, 23, 75]]
p1=player("virat",l1,l2)
pp=pprint.PrettyPrinter()
pp.pprint(p1)
```

A **saída** do código acima é -

```
{'virat': {'Tests': [140, 45, 39], 'ODI': [15, 122, 36, 67, 100, 49],
'T20': [78, 44, 12, 0, 23, 75]}}
```