Softech®
Gather to thrive together

Aptech™
COMPUTER EDUCATION
Unleash your potential

BÀI 9

String in Python
Kiểu Chuỗi trong Python

Tóm Tắt Nội Dung

Trong bài học này chúng ta sẽ đi tìm hiểu lần lượt các nội dung

1. Tổng quan về Chuỗi (String)
2. Các phương thức xử lý với Chuỗi
3. Định dạng chuỗi với khuôn
4. Dãy thoát Escape trong Python

9.1 Tổng quan về Chuỗi (String)

Giới thiệu về Kiểu dữ liệu String

- Trong Python, kiểu dữ liệu chuỗi (string) là một dạng dữ liệu dùng để lưu trữ văn bản, chữ số, ký tự, hoặc bất kỳ dãy các ký tự nào

- Chuỗi trong Python được biểu diễn bằng các ký tự nằm trong cặp dấu nháy đơn ('...') hoặc nháy kép ("..."). Python hỗ trợ các thao tác và phương thức mạnh mẽ để làm việc với kiểu dữ liệu chuỗi

```
# Khai báo chuỗi bằng cặp dấu nháy đơn
string1 = 'Hello, World!'
# Khai báo chuỗi bằng cặp dấu nháy kép
string2 = "Python is awesome"
# Chuỗi có thể chứa các ký tự đặc biệt
string3 = 'Các ký tự đặc biệt: !@#$%^&*()'
# Chuỗi có thể chứa các số và chữ số
string4 = '12345'
```

9.1 Tổng quan về Chuỗi (String)

- Chuỗi nhiều dòng với 3 kí tự nháy kép """

```
a = """Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua."""
print(a)
```

- Hoặc Chuỗi nhiều dòng với 3 kí tự nháy đơn '''

```
a =  '''Lorem ipsum dolor sit amet,
consectetur adipiscing elit,
sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua.'''
print(a)
```

9.1 Tổng quan về Chuỗi (String)

- Trong Python String cũng được coi là một Mảng (Array) tập hợp của nhiều kí tự, vị trí sắp xếp bắt đầu là số 0

```
a = "Hello, World!"
print(a[1]) #output: e
```

- Vì là mảng nên bạn có thể lặp với vòng lặp **for**

```
for x in "banana":
  print(x)
```

- Dùng phương thức len() để kiểm tra độ dài của chuỗi có bao nhiêu kí tự

```
a = "Hello, World!"
print(len(a)) #output: 13
```

9.1 Tổng quan về Chuỗi (String)

- Kiểm tra sự tồn tại của một kí tự trong chuỗi với từ khóa **in**

```
txt = "The best things in life are free!"
print("free" in txt)

if "free" in txt:
  print("Có tồn tại")
```

- Hoặc kết hợp với **not** để kiểm tra không tồn tại

```
if "fresh" not in txt:
    print("Đúng, fresh không tồn tại trong chuỗi")
```

9.2 Các phương thức xử lí Chuỗi

Cắt chuỗi

Bạn có thể tách chuỗi dự vào vị trí index của nó trong chuỗi

Cú pháp: **string[start:end]**

- start mặc định là 0, có thể là số nguyên âm
- end mặc định là đến cuối chuỗi, có thể là số nguyên âm

```
b = "Hello, World!"
print(b[2:5]) #output: llo
```

Ví dụ này sẽ lấy kí tự từ vị trí thứ 2 đến 4 (5 – 1, không bao gồm vị trí số 5)

```
print(b[-2:-5]) #output: orl
```

Dùng số nguyên âm để đảo chiều vị trí lấy

9.2 Các phương thức xử lí Chuỗi

Cắt chuỗi

Ví dụ không điền **start**, thì start mặc định là số 0

```
b = "Hello, World!"
print(b[0:5]) #output: Hello
```

Ví dụ không điền **end**, thì start mặc định là đến vị trí cuối

```
print(b[2:]) #output: llo, World!
```

9.2 Các phương thức xử lí Chuỗi

Biến đổi chuỗi

- Biến thành chuỗi viết HOA

```
a = "Hello, World!"
print(a.upper()) #output: HELLO, WORLD!
```

- Biến thành chuỗi viết thường

```
print(a.lower()) #output: hello, world!
```

- Kí tự đầu tiên mỗi chữ viết HOA

```
txt = "Welcome to my world"
print(txt.title()) #output: Welcome To My World
```

9.2 Các phương thức xử lí Chuỗi

Biến đổi chuỗi

- Tách chuỗi với **split()** dựa vào một kí tự nằm giữa chuỗi

```
a = "Hello, World!"
print(a.split(",")) #returns list:  ['Hello', ' World!']
```

- Xóa khoảng trắng 2 đầu chuỗi với **strip()**

```
a = " Hello, World! "
print(a.strip())  # returns "Hello, World!"
```

- Thay thế kí tự trong chuỗi với **replace()**

```
txt = "Welcome to my world"
print(txt.replace("world", "home")) #output: Welcome To My home
```

9.3 Định dạng Chuỗi

Định dạng (đóng khuôn) chuỗi

- Trong python, không thể nối chuỗi với số với toán tử + cộng, nhưng chúng ta có thể làm điều đó với hàm **format()**

```
age = 36
txt = “Tôi là  John, Tôi {} tuổi"
print(txt.format(age))
#Kết quả in ra: Tôi là  John, Tôi 36 tuổi
```

Trong ví dụ trên kí tự {} đóng vai trò giữ chỗ cho biến age, và age sẽ hiển thị ra thay cho {}. Các vị trí {} này được gọi là **trường** (hay là field)

9.3 Định dạng Chuỗi

Định dạng (đóng khuôn) chuỗi

- **Thay thế đơn giản không chỉ số**

```
age = 36
name = ‘Jonh’
txt = “Tôi là  {}, Tôi {} tuổi"
print(txt.format(name, age))
#Kết quả in ra: Tôi là  John, Tôi 36 tuổi
```

- Khi đó biến **name** sẽ thay thế vào kí tự **{}** đầu tiên trong chuỗi txt
- Và biến **age** sẽ thay thế vào vị trí **{}** thứ 2 trong chuỗi txt
- Nếu có một biến khác thì biến và {} được tham chiếu một cách tuần tự như vậy khi bạn thêm vào hàm **format()**

Lưu ý: bạn có thể đưa nhiều tham số biến hơn số lượng {} cần thay thế

9.3 Định dạng Chuỗi

Định dạng (đóng khuôn) chuỗi

- **Thay thế có chỉ số**

```
quantity  = 2
price  = 49.95
txt = “Mua {0} cái bánh hết {1} đô la. Em tôi ăn hết {0} cái bánh"
print(txt.format(quantity, price))
#Kết quả in ra: Mua 2 cái bánh hết 49.95 đô la. Em tôi ăn hết 2 cái bánh
```

Với cách dùng chỉ số này thì nhìn vào dễ hiểu hơn:
- **quanlity:** sẽ tương ứng với field số {0}
- **price:** sẽ tương ứng với field số {1}

Và mình có thể đặt các field bất ký ở đâu chỉ cần ghi số vào là được.

Lưu ý: nếu đã đánh số thì tất cả các field đều đánh số, chứ không được cái thì {}, cái thì {0}. Python sẽ không xử lý đúng.

9.3 Định dạng Chuỗi

Chỉ lệnh trong field

<Filed> = {<Chỉ số> : <Chỉ lệnh>}

Chỉ số và chỉ lệnh đều là lựa chọn. Nên chúng ta có thêm 3 cách viết

<Filed> = {}	Thiếu cả chỉ số và chỉ lệnh
<Filed> = {<Chỉ số>}	Thiếu chỉ lệnh
<Filed> = { : <Chỉ lệnh>}	Thiếu chỉ số

Chỉ lệnh giúp tùy biến sâu hơn cách mà dữ liệu sẽ thay thế vào vị trí giữ chỗ

9.3 Định dạng Chuỗi

- **Chỉ lệnh thiết lập chiều rộng Width**

<Filed> = {<Chỉ số> : <width>}

```
s = “Cột {:5} Cột {:5} Cột {:5}”
print(s.format(1,2,3))
# Kết quả
Cột 1 Cột 2 Cột 3
```

Trong ví dụ trên **{:5}** được hiểu 5 chính là **width**. Nếu mà chỉnh lệnh là một số > 0 thì đó là chỉnh lệnh width quy định độ rộng của tham số sau khi thay thế hiển thị chiếm 5 kí tự.

Độ rộng 5 trong định dạng **{:5}** bao gồm cả giá trị và khoảng trắng. Nếu giá trị có độ dài nhỏ hơn 5, thì các khoảng trắng sẽ được thêm vào bên trái của giá trị để đạt được độ rộng mong muốn.

9.3 Định dạng Chuỗi

- **Chỉ lệnh thiết lập canh thẳng hàng**

Ký tự sẽ chèn vào các khoảng trống. Mặc định là kí tự trắng

<Filed> = { <Chỉ số>: <char>=<Witdh>}

= Canh đều 2 bên
< Canh trái
> Canh phải
^ Canh giữa. Mặc định

Trong ví dụ dưới: Khoảng trắng sẽ được bù vào theo hướng đã cấu hình cho các Fields

- Bù vào khoảng trắng bên phải Hà Nội → Canh trái
- Bù vào khoảng trắng 2 bên Đà Nẵng → Canh giữa
- Bù vào khoảng trắng bên trái TP Hồ chí minh → Canh phải

```
s = "{0:<10} {1:^10} {2:>10}"
print(s.format("Hà Nội", "Đà Nẵng", "TP Hồ Chính Minh"))
# Kết quả
Hà Nội Đà Nẵng TP Hồ Chính Minh
```

9.3 Định dạng Chuỗi

- **Chỉ lệnh canh chuỗi**

```
s = "{0:-<10} {1:*^10} {2:->10}"
print(s.format("Hà Nội", "Đà Nẵng", "TP Hồ Chính Minh"))
# Kết quả
Hà Nội---- *Đà Nẵng** TP Hồ Chính Minh
```

Trong ví dụ trên: Bạn có thể dùng các ký tự thay thế cho khoảng trắng

- Tổng chiều rộng của 3 tham số in ra là 30 kí tự
- Python sẽ tự động tính toán để chèn các kí tự đó theo hướng mà tham số đó hiển thị ra. Để lấp đầy các khoảng trống còn thiếu.

9.3 Định dạng Chuỗi

- **Chỉ lệnh thể hiện số gần đúng**

<Filed> = { <Chỉ số>: <char><=><Witdh>. <p>}

P là các chữ số thập phân phía sau dấu .

```
s = "{:-<10.5}"
print(s.format(0.956547678787))
print(s.format(49.956547678787))
#Kết quả in ra:
0.95655---
49.957----
```

- Nếu số thập phần < 1 thì p = số lượng các số thập phân sau dấu .
- Nếu số thập phần > 1 thì p = tổng số các chữ số trước và sau dấu .

9.3 Định dạng Chuỗi

- **Chỉ lệnh # (hash)**

<Filed> = { <Chỉ số>: <char><=><#><Witdh>. <p>}

Chỉ lệnh # được viết trước **<witdh>.<p>** sẽ có tác dụng điền thêm các số 0 bên phải để số thập phân có đủ chiều dài **<p>**

```
s = "{:-<10.5} {:*<#10.5}"
print(s.format(0.12, 0.4))
#Kết quả in ra: 0.12------ 0.40000***
```

Trong ví dụ trên bạn thấy:
- **Field đầu tiên:** chưa áp dụng hash, số thập phân < 1, Tuy p=5, nhưng số thập phân phía sau dấu . chỉ có 2 số nên hiển thị 2 số thập phân.
- **Filed thứ 2:** có dùng hash, số thập phân < 1, p = 5, số thấp phân phía sau dấu . chỉ có 1. Python tự thêm vào 4 số 0 nữa để cho được 5 chữ số thấp phân phía sau dấu .

9.3 Định dạng Chuỗi

- **Chỉ lệnh kiểu (type)**

<Filed> = { <Chỉ số>: <Type>}

Type là một trong các giá trị: **b,c,d,f,e,g** và nằm cuối cùng phần chỉ lệnh để quy định cách hiển thị các kiểu dữ liệu số khác nhau

b – biểu diễn số nhị phân
c – biểu diễn kí tự tương ứng trong bảng mã ASCII hoặc Unicode
d – biểu diễn số nguyên
f – biểu diễn số thập phân với các chữ số thập phân cố định
e – biểu diễn số thập phân dưới dạng lũy thừa của số e
g – biểu diễn số dưới dạng tốt quát nhất. Python sẽ tự chọn khuôn dạng

```
s = “{0:b} {0:d} {0:f}”
print(s.format(6))
#Kết quả in ra: 110 6 6.000000
```

9.3 Định dạng Chuỗi

- **Chỉ lệnh đặt dấu phẩy , vào các nhóm 3 chữ số**

<Filed> = { <Chỉ số>: <,>}

```
s = "{0:,.5f}"
print(s.format(12000123.1234567))
#Kết quả in ra:
12,000,123.12346
```

Trong ví dụ trên Python sẽ dùng dấu phẩy để phân cách theo từng nhóm 3 chữ số. Nhìn vào dễ đọc con số hơn.

9.3 Định dạng Chuỗi

- **Cách dùng khác của Lệnh format()**

format(<giá trị>, <"Chỉ lệnh điều khiển">)

Ngoài cách dùng **chỉ lệnh** trong các phần trước, bạn cũng có thể sử dụng chỉ lệnh ngay bên trong câu lệnh **format** theo cú pháp trên

```
print(format(12000123.1234567, ",.5f"))
print(format("Hà nội", "->15"))
#Kết quả in ra:
12,000,123.12346
---------Hà nội
```

9.3 Định dạng Chuỗi

- **Tạo khuôn bằng f<string>**

s = f"{<var1> : <Chỉ lệnh>} {<var2> : <Chỉ lệnh>}"

Đây là một cách đơn giản hơn để có thể trộn biến vào trong chuỗi

```
mark = 8
name = 'Toán'
txt = "Môn {} được {} điểm"
print(txt.format(name, age))
#Kết quả in ra:
Môn Toán được 8 điểm
```

```
mark = 8
name = 'Toán'
txt = f"Môn {name} được {mark:*>5.2f} điểm"
print(txt)
#Kết quả in ra:
Môn Toán được *8.00 điểm
```

Cách không tường minh

Chỉ lệnh vẫn được áp dụng với f<string>

Giúp code trở nên gọn hơn

9.4 Escape Character

Giới thiệu về khái niệm Escape

- Trong Python, ký tự escape (escape character) là một ký tự đặc biệt được sử dụng để thể hiện các ký tự không thể hiện hoặc có ý nghĩa đặc biệt trong chuỗi. Khi bạn sử dụng ký tự escape, nó cho phép bạn đưa vào chuỗi những ký tự đặc biệt như dấu nháy kép, dấu nháy đơn, dấu gạch chéo ngược, và các ký tự khác mà không bị xem là kết thúc của chuỗi hoặc gây ra sự hiểu lầm về cú pháp.

  Ký tự escape được ký hiệu bằng một dấu gạch chéo ngược ("") theo sau là ký tự đại diện cho ý nghĩa cụ thể. Dưới đây là một số ví dụ về ký tự escape phổ biến trong Python:

  \n: Ký tự xuống dòng (newline)
  \t: Ký tự tab ngang (tab)
  \": Dấu nháy kép
  \': Dấu nháy đơn

9.4 Escape Character

Ví dụ về cách sử dụng Escape

Sử dụng ký tự escape để in ra các ký tự đặc biệt trong chuỗi

```
print("Dòng 1\nDòng 2")
# In ra hai dòng cách nhau bởi dấu xuống dòng (\n)
print("Tab:\t\tKý tự tiếp theo sau tab")
# In ra ký tự tab (\t) để tạo khoảng cách
print("Dấu nháy kép: \"Ví dụ\"  ")
# In ra dấu nháy kép (") trong chuỗi
print('Dấu nháy đơn: I\'m a student ')
# In ra dấu nháy đơn (') trong chuỗi
print("Dấu gạch chéo ngược: \\")
# In ra dấu gạch chéo ngược (\) trong chuỗi
```

Tham khảo link sau để tìm hiểu tất cả phương thức xử lý chuỗi:
https://www.w3schools.com/python/python_strings_methods.asp

9.5 Nguồn tham chiếu

Nguồn tham chiếu

- Chuỗi trong Python

  https://www.w3schools.com/python/python_strings.asp

- Các phương thức Xử lý chuỗi trong Python

  https://www.w3schools.com/python/python_strings_methods.asp

Tổng kết lại bài

1. Hiểu được thế nào gọi là dữ liệu kiểu Chuỗi (String)
2. Nắm được các phương thức xử lý với Chuỗi
3. Biết cách định dạng chuỗi với khuôn format()
4. Hiểu thế nào là dãy thoát Escape trong Python và cách sử dụng