**Định dạng bằng toán tử %**

Kiểu định dạng này sẽ là rất quen thuộc nếu bạn từng tiếp xúc với ngôn ngữ lập trình C. Hãy đến với một số ví dụ

>>> ‘My name **is** %s.’ %(‘Lucario’)

‘My name **is** Lucario’

>>> ‘%d. That **is** %s problem.’ %(1, ‘That’)

‘1. That **is** the problem.’

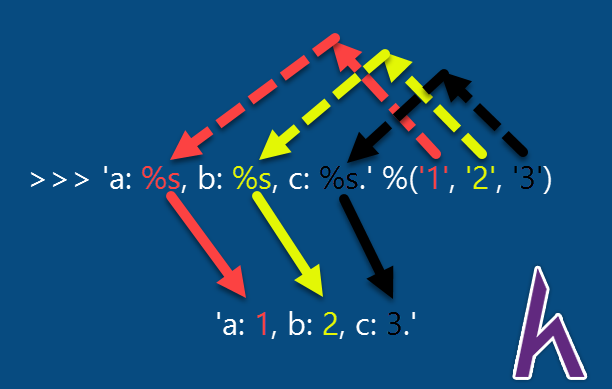
**Cú pháp:**

<chuỗi> **%**(giá trị thứ 1, giá trị thứ 2, .., giá trị thứ n – 1, giá trị thứ n)

**Lưu ý:**

Không hề có dấu ,, tách phần chuỗi và phần giá trị cần định dạng

Để hiểu rõ hơn cách hoạt động của cách định dạng này, mời các bạn xem hình sau



Với hình vẽ trên, bạn có thể dễ dàng biết được cách mà nó hoạt động. Đó là từng phần kí hiệu**%s** sẽ lần lượt được **thay thế**lần lượt bởi các **giá trị nằm trong cặp dấu ngoặc đơn** (Đây là kiểu dữ liệu Tuple, sẽ được Kteam giới thiệu ở bài [KIỂU DỮ LIỆU TUPLE](http://www.howkteam.vn/course/kieu-du-lieu-chuoi-trong-python--phan-3/kieu-du-lieu-tuple-trong-python-1550)).

Thêm một số ví dụ minh họa

>>> s = ‘%s %s’

>>> s %(‘one’, ‘two’)

‘one two’

>>> s %(‘a’, ‘b’)

‘a b’

>>> c = s %(‘c’, ‘cc’)

>>> c

‘c cc’

>>> s %(‘D’) # không được, vì trong chuỗi của biến d có dư kí hiệu % để thay thế

Traceback (most recent call last):

File "<stdin>", line 1, **in** <**module**>

TypeError: **not** enough arguments **for** format string

>>> d %(‘a’, ‘b’) # không thể, vì trong chuỗi của biến d không có đủ kí hiệu % để thay thế

Traceback (most recent call last):

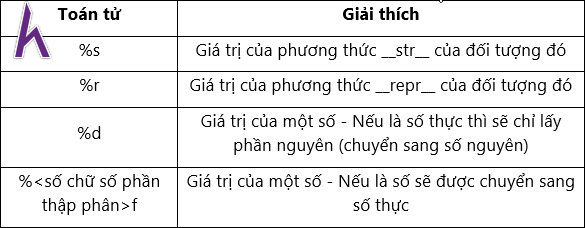
File "<stdin>", line 1, **in** <**module**>

TypeError: **not** all arguments converted during string formatting

Nếu các bạn để ý trong các ví dụ. Kteam không chỉ sử dụng mỗi kí hiệu %s, mà còn có %d. **Vậy sự khác nhau giữa %s và %d là gì? Liệu có còn kí hiệu % nào khác nữa không?**

Kteam sẽ giải đáp cho bạn ngay sau đây

Dưới đây là một số các toán tử % cơ bản trong Python



Có thể bạn sẽ cảm thấy khó hiểu ở hai toán tử**%s** và **%r**. Mọi thứ trong Python đều là các đối tượng của một lớp nào đó. Do đó nó đều có các phương  thước, thuộc tính riêng. Các đối tượng trong Python luôn luôn có hai phương thức đó là **\_\_str\_\_** và **\_\_repr\_\_**.

Tuy các bạn chưa tiếp xúc với hướng đối tượng bao giờ để hiểu được khái niệm này. Nhưng Kteam sẽ viết một lớp đơn giản để giải thích cho bạn hiểu sự khác biệt giữa **%r** và **%s**.

>>> **class** **SomeThing**:

… **def** \_\_repr\_\_(**self**):

… **return** ‘Đây là \_\_repr\_\_'

… def \_\_str\_\_(self):

… return 'Đây là \_\_str\_\_'

…

>>>

Vừa rồi, mình đã tạo một lớp với tên là SomeThing, giờ mình sẽ tạo một đối tượng thuộc lớp đó

>>> sthing = SomeThing()

Đừng vội bối rối! thật ra nó cũng là một giá trị bình thường thôi. Cũng giống như một chuỗi, một con số.

>>> **type**(sthing) # và nó thuộc lớp SomeThing

<class '\_\_main\_\_.SomeThing'>

Và giờ, hãy xem giá trị của đối tượng sthing nhé.

>>> sthing

Đây là \_\_repr\_\_

>>> print(sthing)

Đây là \_\_str\_\_

Nó có sự khác biệt. Và giờ, ta sẽ thấy **sự khác biệt giữa %s và %r**

>>> ‘%r’ %(sthing)

‘Đây là \_\_repr\_\_’

>>> ‘%s’ %(sthing)

‘Đây là \_\_str\_\_’

Đó là sự khác biệt giữa %s và %r.  Đây là một thứ mà nhiều bạn học Python nhầm lẫn.

Nếu bạn từng học **ngôn ngữ C** thì ngỡ %s là thay thế cho một chuỗi thì chưa đủ chính xác.

* %s thay thế cho giá trị của phương thức \_\_str\_\_ tạo nên đối  tượng đó.
* Còn về %r thì là phương thức \_\_repr\_\_.

Do đó, bạn có thể sử dụng %s hoặc %r với mọi đối tượng trong Python.

>>> ‘%s’ %(1) # số

‘1’

>>> ‘%r’ %(1)

‘1’

>>> ‘%s’ %([1, 2, 3]) # kiểu dữ liệu list

‘[1, 2, 3]’

>>> ‘%r’ %([1, 2, 3])

‘[1, 2, 3]’

>>> ‘%s’ %((1, 2, 3)) # kiểu dữ liệu tuple

‘(1, 2, 3)’

>>> ‘%r’ %((1, 2, 3))

‘(1, 2, 3)’

Ở kí hiệu %d, nó đơn giản dễ hiểu hơn với hai kí hiệu ta vừa biết qua ở trên. Kí hiệu này chỉ thay thế cho một số.

>>> ‘%d’ %(3)

‘3’

>>> ‘%d’ %(‘3’) # lỗi, vì ‘3’ không phải 3

Traceback (most recent call last):

File "<stdin>", line 1, **in** <**module**>

TypeError: %d format: a number is required, **not** str

>>> ‘%d’ %(3.9) # chỉ lấy phần nguyên

‘3’

>>> ‘%d’ %(10/3)

‘3’

Như bạn thấy, %d không phù hợp cho số thực, đó là lí do ta có **%f**

>>> ‘%f’ %(3.9)

'3.900000'

>>> ‘%f’ %(‘a’) # %f cũng yêu cầu một số, ngoài ra đều là lỗi

Traceback (most recent call last):

File "<stdin>", line 1, **in** <**module**>

TypeError: must be real number, **not** str

>>> ‘%f’ %(3)

‘3.000000’

>>> ‘%.2f’ %(3.563545) # chỉ lấy 2 số ở phần thập phân

‘3.56’

>>> ‘%.3f’ %(3.9999) # %f cũng có khả năng làm tròn

‘4.000’

**Định dạng bằng chuỗi f (f-string)**

Phương pháp định dạng này cho bạn khả năng thay thế một số chỗ ở trong một chuỗi bằng  giá trị của các biến mà bạn đã khởi tạo và có. Và để có thể sử dụng cách này, bạn phải có một chuỗi f.

**Một chuỗi f sẽ có cú pháp:**

f ’giá trị trong chuỗi’

**Ví dụ:**

>>> f’abc’ # đây là một f-string

‘abc’

>>> s = f’xyz’ # vẫn chưa có gì khác biệt so với chuỗi thông thường

>>> s

‘xyz’

>>> print(f’a\tb’)

a 4

Nhưng nó sẽ khác biệt, nếu bạn có một f-string theo kiểu này

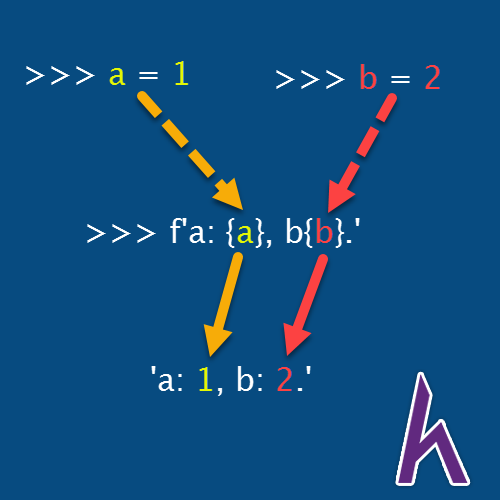
>>> variable = ‘**string**’

>>> f’This **is** a {variable}.’ # chú ý tới những thứ nằm trong cặp ngoặc nhọn

‘This **is** a **string**’

Đúng rồi đấy, giá  trị của biến variable được thay thế trong cặp dấu ngoặc nhọn chứa tên của nó. Nếu bạn có biết qua PHP, bạn sẽ thấy cách này tương tự với việc bạn sử dụng cặp dấu “” để định dạng.

Mời các bạn xem hình ảnh minh họa sau đây



Vậy, **khi bạn sử dụng chuỗi f, đặt một giá trị biến chưa được khai báo, hoặc có trong chương trình thì sao?**

>>> f’{variable\_2}’ # chưa khởi tạo biến có tên variable\_2

Traceback (most recent call last):

File "<stdin>", line 1, **in** <**module**>

NameError: name ' variable\_2' is **not** **defined**

Điều này đặt ra cho bạn một vấn đề, nếu như bạn muốn có chuỗi với nội dung như sau

1: {one}, 2: {two}, 3: {variable}

Và chỉ muốn định dạng mỗi chỗ **{variable}** thôi thì phải làm sao?

Cách giải quyết là hãy đặt thêm một dấu { kế bên {, còn với } là một dấu }. Tương tự như cách chúng ta muốn có một dấu \ mà để Python hiểu không phải là một kí tự bắt đầu kí tự [**escape sequence**](https://www.howkteam.com/course/kieu-du-lieu-so-trong-python/kieu-du-lieu-chuoi-trong-python--phan-1-1541) thì sẽ thêm một dấu \.

>>> variable = ‘three’

>>> f‘1: {{one}}, 2: {{two}}, 3: {variable}’

‘1: {one}, 2: {two}, 3: three’

Ngoài ra, chuỗi f còn hỗ trợ một cách in giá trị khá đặc biệt, cũng như là hỗ trợ toán tử :=.

>>> v = 1

>>> t = 2

>>> f'Two variables {v=} and {t=}'

'Two variables v=1 and t=2'

>>> f'Using operator := with c={(c:=3)}'

'Using operator := with c=3'

>>> c

3

**Định dạng bằng phương thức format**

Cách định dạng này cho phép Python định dạng chuỗi một cách tuyệt vời, không chỉ tốt về mặt nội dung mà còn về thẩm mĩ. Định dạng bằng phương thức format

Đầu tiên là đơn giản nhất

>>> ‘a: {}, b: {}, c: {}’.**format**(1, 2, 3)

‘a: 1, b: 2, c: 3’

>>> ‘a: %d, b: %d, c: %d’ %(1, 2, 3) # tương tự như dùng phương thức format trên

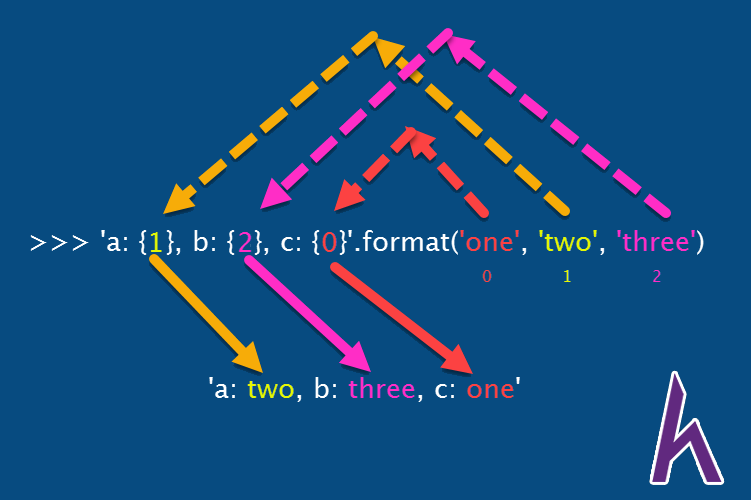
‘a: 1, b: 2, c: 3’

Nếu chỉ tương tự với toán tử %, phương thức này sẽ không có gì nổi bật. Vậy hãy đến với ví dụ tiếp theo

>>> ‘**a**: {1}, **b**: {2}, **c**: {0}’.format(‘**one**’, ‘**two**’, ‘**three**’)

‘**a**: **two**, **b**: **three**, **c**: **one**‘

Nếu vẫn còn mơ hồ, bạn hãy xem hình ảnh minh họa sau đây



**Giá trị ở vị trí thứ nhất**sẽ thay thế cho **vị trí thứ nhất** ở trong chuỗi, và cứ thế với các giá trị sau.

Và với phương thức này, cũng không quá khắt khe việc số các giá trị bằng số các nơi cần định dạng trong chuỗi. Ta có thể cho dư giá trị

>>> ‘only one **value**: {0}’.format(1, 2)

‘only one **value**: 1’

>>> ‘only one **value**: {1}’.format(1, 2)

‘only one **value**: 2’

>>> ‘two same **value**: {0}, {0}’.format(1, 2) # và cũng có thể lặp lại

‘two same **value**: 1, 1’

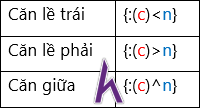
Vẫn chưa thỏa mãn, vì các vị trí đánh số còn chưa đủ rõ ràng, và bạn có khả năng bị nhầm lẫn. Phương thức format vẫn chiều lòng được bạn.

>>> ‘1: {one}, 2: {two}’.format(one=111, two=222)

‘1: 111, 2: 222’

Như đã nói, không chỉ định dạng về nội dung, mà nó còn giúp tang tính thẩm mĩ. Cụ thể là phương thức này giúp bạn định dạng căn lề một cách tuyệt vời.Cách này khá tương tự với việc sử dụng f-string đúng không nào?

Dưới đây là 3 cách căn lề cơ bản của phương thức format



Trong đó

* **c** là kí tự bạn muốn thay thế vào chỗ trống, nếu để trống thì sẽ là kí tự khoảng trắng
* **n** là số kí tự dùng để căn lề.

Để hiểu rõ hơn hãy đến với ví dụ:

>>> ‘{:^10}’.format(‘aaaa’) # căn giữa

' aaaa '

>>> '{:<10}'.format('aaaa') # căn lề trái

'aaaa '

>>> '{:>10}'.format('aaaa') # căn lề phải

' aaaa'

>>> '{:\*>10}'.format('aaaa') # căn lề trái, thay thế khoảng trắng bằng kí tự \*

'\*\*\*\*\*\*aaaa'

>>> '{:\*<10}'.format('aaaa') # căn lề phải, thay thế khoảng trắng bằng kí tự \*

'aaaa\*\*\*\*\*\*'

>>> '{:\*^10}'.format('aaaa') # căn giữa, thay thế khoảng trắng bằng kí tự \*

'\*\*\*aaaa\*\*\*'

Nhờ việc căn lề bằng phương thức này, bạn sẽ dễ dàng hơn để có thể cho kết quả của bạn đẹp mắt.

**Ví dụ\*:** Hãy tạo một file Python với nội dung sau.

# phần định dạng

row\_1 = '+ {:-<6} + {:-^15} + {:->10} +'.**format**('', '', '')

row\_2 = '| {:<6} | {:^15} | {:>10} |'.**format**('ID', 'Ho va ten', 'Noi sinh')

row\_3 = '| {:<6} | {:^15} | {:>10} |'.**format**('123', 'Kteam', 'TP HCM')

row\_4 = '| {:<6} | {:^15} | {:>10} |'.**format**('6969', 'Kquiz', 'Da Lat')

row\_5 = '+ {:-<6} + {:-^15} + {:->10} +'.**format**('', '', '')

# phần xuất kết quả

**print**(row\_1)

**print**(row\_2)

**print**(row\_3)

**print**(row\_4)

**print**(row\_5)

Khi chạy file Python đó, bạn sẽ có kết quả là

