Bài: Kiểu dữ liệu số trong Python

Xem bài học trên website để ủng hộ Kteam: Kiểu dữ liệu số trong Python

Mọi vấn đề về lỗi website làm ảnh hưởng đến bạn hoặc thắc mắc, mong muốn khóa học mới, nhằm hỗ trợ cải thiện Website. Các bạn vui lòng phản hồi đến Fanpage How Kteam nhé!

Dẫn nhập

Trong các bài trước, bạn đã làm quen với khái niệm BIẾN TRONG PYTHON.

Ở bài này **Kteam** sẽ đề cập đến các bạn **KIỂU DỮ LIỆU SỐ**. Một trong những kiểu dữ liệu cực kì quan trọng của Python!

Nội dung chính

Để đọc hiểu bài này tốt nhất bạn cần:

- Cài đặt sẵn MÔI TRƯỜNG PHÁT TRIỂN CỦA PYTHON.
- Xem qua bài CÁCH CHẠY CHƯƠNG TRÌNH PYTHON.
- Nắm CÁCH GHI CHÚ và BIẾN TRONG PYTHON.

Trong bài học này, chúng ta sẽ cùng tìm hiểu các vấn đề:

- Số là gì?
- Một số kiểu dữ liệu số cơ bản trong Python.
- Các toán tử cơ bản với kiểu dữ liệu số trong Python.
- Thư viện math trong Python.

Số là gì?

Con số ở khắp nơi trong cuộc sống chúng ta. Bất cứ lúc nào bạn cũng có thể bắt gặp con số trong cuộc sống.

Tháng này có 30 hay 31 ngày? Mai đi chợ bó rau muốn 3000 đồng hay là 3500 đồng? Bài thi hôm bữa được 9,1 điểm hay là 1,9? Cái bánh này mình ăn ½ hay là ¾. Có thể thấy, số không còn là điều gì xa lạ với bạn. Và đương nhiên điều này tương tự với "con trăn" Python.

Trong Python cũng hỗ trợ rất nhiều kiểu dữ liệu số. Một số kiểu dữ liệu cơ bản như số nguyên (integers), số thực (floating-point), phân số (fraction), số phức (complex). Và những kiểu dữ liệu này sẽ được Kteam giới thiệu cho các bạn ngay sau đây!

Một số kiểu dữ liệu số cơ bản trong Python

Số nguyên

Số nguyên bao gồm các số nguyên dương (1, 2, 3, ..), các số nguyên âm (-1, -2, -3) và số 0. Trong Python, kiểu dữ liệu số nguyên cũng không có gì khác biệt.

Ví dụ: Gán giá trị cho một biến a là 4 và xuất ra kiểu dữ liệu của a.

Python:

```
>>> a = 4    # gán giá giá trị của biến a là số 4, là một số nguyên
>>> a
4
>>> type(a)  # số nguyên thuộc lớp 'int' trong Python
<class 'int'>
```

Một điểm đáng chú ý trong Python 3.X đó là kiểu dữ liệu số nguyên là vô hạn. Điều này cho phép bạn tính toán với những số cực kì lớn, điều mà đa số các ngôn ngữ lập trình khác KHÔNG THỂ.

Số thực

Về kiểu dữ liệu **số thực**, thì đây là tập hợp các số nguyên và số thập phân 1, 1.4, -123, 69.96,...

Ví dụ: Gán giá trị của biến f là 1.23 và xuất ra kiểu dữ liệu của f.

Python:

```
>>> f = 1.23   # gán giá trị của biến f là số 1.23, là một số thực
>>> f
1.23
>>> type(f)    # số thực trong Python thuộc lớp 'float'
<class 'float'>
>>> q = 1.0   # đây là số thực, không phải số nguyên
>>> q
1.0
>>> type(q)
<class 'float'>
```

> **Lưu ý:** Thường khi chúng ta viết số thực, phần nguyên và phần thập phân được tách nhau bởi dấu phẩy (,). Thế nhưng trong Python, dấu phẩy (,) này được thay thế thành dấu chấm (.)

Số thực trong Python có độ chính xác xấp xỉ 15 chữ số phần thập phân.

Ví dụ: Số thực 10/3

Python:

```
>>> 10 / 3   # đây là một số vô hạn tuần hoàn 3.3333333333333333333..
3.3333333333333335
```

Nếu bạn muốn có kết quả được chính xác cao hơn, bạn nên sử dụng **Decimal**

Python:

```
>>> from fractions import Fraction   # lấy toàn bộ nội dung của thư viện Decimal,khuyến khích dùng: from fractions import Fraction
trong việc học để tạo thói quen trong công việc sau này.
>>> getcontext().prec = 30   # lấy tối đa 30 chữ số phần nguyên và phần thập phân Decimal
>>> Decimal(10) / Decimal(3)
Decimal('3. 333333333333333333333333333333')
>>> Decimal(100) / Decimal(3)
Decimal('33.333333333333333333333333333333')
>>> type(Decimal(5))   # các số Decimal thuộc lớp Decimal
<class 'decimal.Decimal'>
```

Tuy Decimal có độ chính xác cao hơn so với float tuy nhiên nó lại khá rườm rà so với float. Do đó, hãy cân bằng sự tiện lợi và chính xác để chọn kiểu dữ liệu phù hợp.

Phân số

Chúng ta biết đến phân số qua sách giáo khoa toán lớp 3. Phân số gồm hai phần là **tử số** và **mẫu số**.

Tạo một phân số

Để tạo phân số trong python, ta sử dụng hàm **Fraction** với cú pháp sau

> **Fraction**(<Tử_số>, <Mẫu_số>)

Ví dụ: Nhập phân số ¼, 3/9, ¾,

Python:

```
>>> from fractions import * # lấy toàn bộ nội dung của thư viện decimal
>>> Fraction(1, 4)   # phân số với tử số là 1, mẫu số là 4.
Fraction(1, 4)
>>> Fraction(3, 9)  # phân số sẽ được tối giản nếu có thể
Fraction(1, 3)
>>> type(Fraction(3, 4))  # các phân số thuộc lớp Fraction
<class 'fractions.Fraction'>
```

Tất nhiên, việc tạo một phân số với mẫu số bằng 0 sẽ gây lỗi.

Python:

```
>>> from fractions import *
>>> Fraction(1, 0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Users\PC\AppData\Local\Programs\Python\Python310\lib\fractions.py", line 156, in __new__
    raise ZeroDivisionError('Fraction(%s, 0)' % numerator)
ZeroDivisionError: Fraction(1, 0)
>>> Fraction(1.55, 0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "C:\Users\PC\AppData\Local\Programs\Python\Python310\lib\fractions.py", line 152, in __new__
    raise TypeError("both arguments should be "
TypeError: both arguments should be Rational instances
```

Số phức

Nếu bạn chưa biết đến **Số Phức**, Kteam khuyên bạn nên bỏ qua phần này.

Số phức gồm 2 thành phần :

> <Phần thực> + <Phần ảo> j

Trong đó

- <Phần thực> <Phần ảo> là số thực
- j là đơn vị ảo trong toán học với $j^2 = -1$

Tạo một số phức

Để tạo một số phức, bạn có thể sử dụng hàm **complex** với cú pháp sau

> **complex**(<Phần_thực>,<Phần_ảo>)

Gán giá trị số phức cho một biến

> <tên_biến> = <Phần_thực> + <Phần_Ảo>j

Xuất ra từng phần của một biến số phức

Để xuất ra phần thực, ta sử dụng cú pháp:

```
<tên_biến>.real
```

Để xuất ra phần ảo của biến số phức, ta dùng cú pháp:

```
<tên_biến>.imag
```

Ví dụ: Nhập một số số phức sau

1. 1 + 3j
2. Gán biến c có giá trị 2+1j. Xuất ra phần thực và phần ảo của biến c.
3. 4 +j (sẽ có lỗi vì kiểu dữ liệu nhập vào không đúng).
4. Tạo số phức có phần thực là 3, phần ảo là 1.
5. Tạo số phức chỉ có phần thực là 2.
6. Xuất ra kiểu dữ liệu của số 3+1j.

Python:

```
>>> 3j + 1  # phần thực là 1, phần ảo là 3
(1 + 3j)
>>> c = 2 + 1j  # gán giá trị cho biến c là một số phức với phần thực là 2 còn phần ảo là 1
>>> c
(2 + 1j)
>>> 4 + j   # phần ảo là 1, tuy vậy chúng ta không được phép bỏ số 1 như trong toán
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'j' is not defined
>>> 4 + 1j
(4 + 1j)
>>> c.imag  # lấy phần ảo của số phức 2 + 1j mà ta đã gán cho biến c
1.0
>>> c.real  # lấy phần thực
2.0
>>> complex(3, 1)  # dùng hàm complex để tạo một số phức với phần thực là 3, ảo là 1
(3 + 1j)
>>> complex(2)      # chỉ có phần thực, phần ảo được mặc định là 0
(2 + 0j)
>>> type(3 + 1j)   # các số phức thuộc lớp complex
<class 'complex'>
```

Các toán tử cơ bản với kiểu dữ liệu số trong Python

Biểu thức chính là một **thực thể toán học**. Nói cách khác, nó là một sự kết hợp giữa 2 thành phần:

- **Toán hạng**: có thể là một hằng số, biến số (X , Y)
- **Toán tử**: xác định cách thức làm việc giữa các toán hạng (+,-,*,/)

Dưới đây là một số biểu thức toán học của kiểu dữ liệu số trong Python

BIỂU THỨC	MÔ TẢ
X + Y	Tổng của X với Y
X – Y	Hiệu của X với Y
X * Y	Tích của X với Y
X / Y	Thương của X với Y (kết quả luôn luôn là một số thức)
X // Y	Thương nguyên của X với Y (kết quả luôn luôn nhỏ hơn hoặc bằng X / Y)
X % Y	Dư của thương X với Y
X ** Y	Lũy thừa mũ Y với cơ số X

Ví dụ: Cho 2 biến a,b lần lượt bằng 8 và 3. Thực hiện các biểu thức toán học với a,b.

Python:

```
>>> a = 8
>>> b = 3
>>> a + b   # tương đương 8 cộng 3
11
>>> a – b    # tương đương 8 trừ 3
5
>>> a * b   # tương đương 8 nhân 3
24
>>> a / b   # tương đương 8 chia 3
2.6666666666666665
>>> a // b  # tương đương với 8 chia nguyên 3
2
>>> a % b   # tương đương với 8 chia dư 3
2
>>> a ** b   # tương đương 8 mũ 3
512
```

Tránh nhầm lẫn khi thực hiện các phép tính với số thực (float)

Đôi lúc, ta thực hiện các phép tính với số thực, và kết quả trả về thật "mù mắt":

Python:

```
>>> 0.1 + 0.2
0.30000000000000004
>>> 0.20 - 0.15
0.05000000000000002
>>> 0.05 * 0.2
0.010000000000000002
```

Không phải do lỗi của bạn, hay do python. Điều này xảy ra trên toàn bộ các ngôn ngữ lập trình. Một cách dễ hiểu, là cách lưu trữ số thập phân của máy tính tạo nên lỗi này.

Máy tính lưu trữ các số dưới dạng các dãy nhị phân. Do đó nên khi muốn lưu trữ bất kì một số nào đó, máy tính sẽ phải chuyển số đó về dạng nhị phân. Nhưng có nhiều số không thể được chuyển một cách chính xác hoàn toàn, và xuất hiện sai số.

Khi học tập cũng như khi làm việc, các bạn cần chú ý đến những lỗi như thế này để tránh nhầm lẫn.

Mức độ ưu tiên của các toán tử

Kteam xin lưu ý cho các bạn khi sử dụng toán tử: Các phép toán được thực hiện từ bên trái sang phải của biểu thức, đồng thời có độ ưu tiên thực hiện khác nhau (giống như khi ta thực hiện các phép tính bình thường)
Bảng sau sẽ đề cập đến một số toán tử với độ ưu tiên thực hiện từ trên xuống dưới (các biểu thức được đặt trong cặp ngoặc đơn được tính toán đầu tiên):

Toán tử	Ý nghĩa
**	Lấy lũy thừa
*, /, //, %	Phép nhân, chia, chia lấy phần nguyên và chia lấy dư
+, -	Cộng, trừ
in, not in, is, is not, <, <=, >, >=, !=, ==	Các toán tử logic (kết quả trả về là True hoặc False)
not	Biểu thức quan hệ not
or	Biểu thức quan hệ or
:=	Assignment expression

Hãy cùng xem xét các ví dụ sau:

Python:

```
>>> 2 + 3 * 4 # Bằng 14 do phép nhân được thực hiện trước
14
>>> 2 * 3**2 # Bằng 18 do phép lũy thừa được thực hiện trước
18
>>> 2 * (2 + 3) - 3 # Bằng 7
7
>>> 2**3 * (1 + 3 % 2) # Bằng 16
16
```

Toán tử := (Assignment expression)

Nếu bạn để ý kĩ bảng trên, thì các bạn sẽ thấy một toán tử được gọi là **Assignment expression** (kí hiệu **:=**) ở hàng cuối cùng. Tại sao nó lại được gọi với cái tên như vậy ? Công dụng của nó là gì ? Kteam sẽ giúp các bạn hiểu rõ hơn về toán tử này.

Thông thường, để gán giá trị cho một biến nào đó, ta dùng toán tử "=" (phép gán). Nhưng việc sử dụng nó cũng có những hạn chế. Hãy xem xét ví dụ sau:

Python:

```
>>> a = 1
>>> b = (a += 1) + 2
  File "<stdin>", line 1
    b = (a += 1) + 2
           ^^
SyntaxError: invalid syntax
>>> b = 3
>>> print(b = 4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'b' is an invalid keyword argument for print()
```

Ta có thể thấy rằng, đối với toán tử "=", việc gán các giá trị chỉ được thực hiện khi bản thân lệnh gán đó được dùng trên **một dòng**. Việc thực hiện phép gán trong khi đang thực hiện các lệnh khác là không được cho phép.

Toán tử ":=" được sinh ra là vì lí do đó. Nó giúp khắc phục được điểm yếu của toán tử "=".

Mời các bạn cùng Kteam xem các ví dụ để hiểu rõ hơn:

Python:

```
>>> a = 3
>>> b = (a := a + 3) + 3 #Thay đổi giá trị của biến a, đồng thời khởi tạo biến b.
>>> a
6
>>> b
9
>>> print(c := 100) #Nếu cần, ta cũng có thể khởi tạo một biến bằng Assignment Expression
100
>>> (t := 4) #Việc khởi tạo biến bằng Assignment Expression bên ngoài lệnh cũng được cho phép, với điều kiện phép gán được đặt
trong cặp ngoặc
4
```

So sánh giữa số với số trong Python

Bạn chắc biết so sánh là gì nhờ các tiết học toán ở trường. Ví dụ như

- 3 > 1 là đúng
- 69 < 10 là sai
- 241 = 141 + 100 là đúng
- (5 x 0) ≠ 0 là sai.

Trong Python cũng có các toán tử như vậy. Tuy nhiên kí hiệu của chúng thì có khác đôi chút.

Bảng sau đây sẽ cho các bạn thông tin về những toán tử so sánh trong Python

Toán học	Python
>	>
<	<
=	==
≥	>=
≤	<=
≠	!=

Hãy xem ví dụ minh họa trong Python:

Python:

```
>>> 3 > 1 # 3 > 1 là đúng => True
True
>>> 69 < 10 # 69 < 10 là sai => False
False
>>> 241 == 141 + 100 # 241 = 141 + 100 là đúng => True
True
>>> (5 * 0) != 0  # 5 x 0 ≠ 0 là sai => False
False
```

Ngoài kiểu dữ liệu số, các toán tử so sánh còn có thể được thực hiện trên các kiểu dữ liệu khác – điều mà các bạn sẽ được tìm hiểu trong các bài tiếp theo

Thư viện math trong Python

Thư viện math trong Python hỗ trợ rất nhiều hàm tính toán liên quan đến toán học.

Để sử dụng một thư viện nào đó, ta dùng lệnh

```
import <tên_thư_viện>
```

Muốn sử dụng một hàm nào đó của thư viện, ta sử dụng cú pháp

```
<tên_thư_viện>.<tên_hàm>
```

Dưới đây là một số hàm thường được dùng trong việc tính toán cơ bản.

TÊN HÀM	CÔNG DỤNG
.trunc(x)	Trả về một số nguyên là phần nguyên của số x
.floor(x)	Trả về một số nguyên được làm tròn số từ số x, kết quả luôn luôn nhỏ hơn hoặc bằng x
.ceil(x)	Trả về một số nguyên được làm tròn số từ số x, kết quả luôn luôn lớn hơn hoặc bằng x
.fabs(x)	Trả về một số thực là trị tuyệt đối của số x
.sqrt(x)	Trả về một số thực là căn bậc hai của số x
.gcd(x, y)	Trả về một số nguyên là ước chung lớn nhất của hai số x và y

Ví dụ:

Python:

```
>>> import math   # lấy nội dung của thư viện math về sử dụng
>>> math.trunc(3.9)
3
>>> math.fabs(-3)
3.0
>>> math.sqrt(16)
4.0
>>> math.gcd(6, 4)
2
>>> math.lcm(4, 5)
20
>>> >>> math.ceil(9.4)
10
```

Câu hỏi củng cố

1. Kiểu dữ liệu số nguyên thuộc lớp nào?
2. Sự khác nhau giữa hai biến a và b dưới đây là gì?

Python:

```
>>> a = 0
>>> b = 0.0
```

3. Tại sao lại có sự khác nhau khi sử dụng hàm `trunc` ở thư viện math so với toán tử `//`

Python:

```
>>> import math
>>> math.trunc(15 / -4)
-3
>>> 15 // -4
-4
```

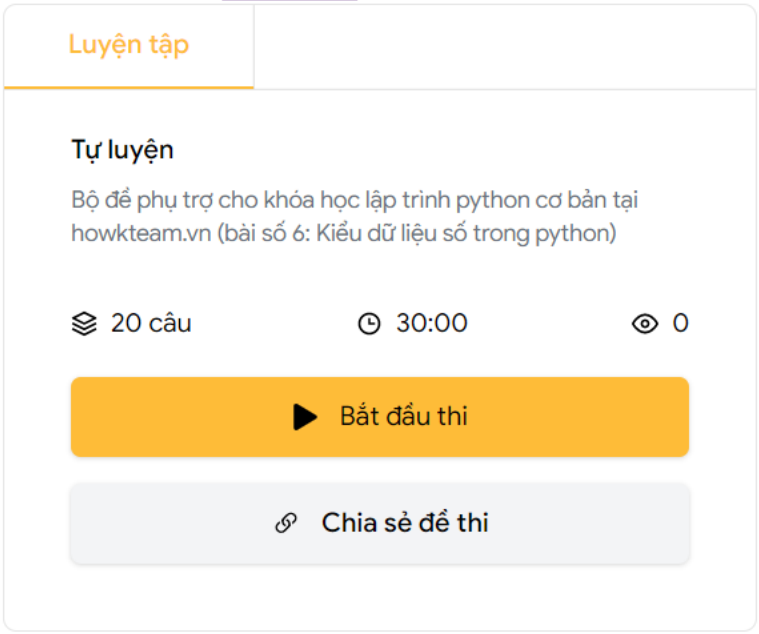
Trong khi chúng lại có trùng kết quả ở phép tính này.

Python:

```
>>> import math
>>> math.trunc(15 / 4)
3
>>> 15 // 4
3
```

Đáp án của phần này sẽ được trình bày ở bài tiếp theo. Tuy nhiên, Kteam khuyến khích bạn tự trả lời các câu hỏi để củng cố kiến thức cũng như thực hành một cách tốt nhất!

Bài tập trắc nghiệm rèn luyện Kquiz



> Các bạn làm bài tập trắc nghiệm tại đây hoặc nhấn vào ảnh để giải đề.

Kết luận

Bài viết này đã giới thiệu cho các bạn một số KIỂU DỮ LIỆU SỐ trong Python.

Ở bài sau, Kteam sẽ nói về KIỂU DỮ LIỆU CHUỖI TRONG PYTHON - một kiểu dữ liệu cũng cực kì quan trọng.

Cảm ơn bạn đã theo dõi bài viết. Hãy để lại bình luận hoặc góp ý của mình để phát triển bài viết tốt hơn. Đừng quên "**Luyện tập – Thử thách – Không ngại khó**".