Python float

Summary: in this tutorial, you'll learn about the Python float type, how Python represents the floating-point numbers, and how to test the floating-point number for equality.

Introduction to the Python float type

Python uses the `float` class to represent the real numbers.

CPython implements float using C double type. The C double type usually implements IEEE 754 double-precision binary float (https://en.wikipedia.org/wiki/Double-precision_floating-point_format) , which is also called binary64.

Python float uses 8 bytes (or 64 bits) to represent real numbers. Unlike the integer type (https://www.pythontutorial.net/advanced-python/python-integers/) , the float type uses a fixed number of bytes.

Technically, Python uses 64 bits as follows:

- 1 bit for sign (positive or negative)
- 11 bits for exponent 1.5e-5 1.5 x 10^{-5} (exponent is `-5`) the range is `[-1022, 1023]` .
- 52 bits for significant digits

For the sake of simplicity, significant digits are all digits except leading and trailing zeros.

For example, 0.25 has two significant digits, 0.125 has three significant digits, and 12.25 has four significant digits.

$(1.25)_{10} = (1\times2^{0} + 0x2^{-1} + 1\times2^{-2})_{10} = (1.01)_{2}$

Some numbers have a finite binary representation, but some don't, e.g., `0.1` . It's `01.0001100110011...` in binary.

Because of this, Python can only use approximate float representations for those numbers.

Python float class

The `float()` returns a floating-point number based on a number or a string. For example:

```
>>> float(0.1)
0.1
>>> float('1.25')
1.25
```

If you pass an object (`obj`) to the `float(obj)` , it'll delegate to the `obj.__float__()` . If the `__float__()` is not defined, it'll fall back to `__index__()` .

If you don't pass any argument to the `float()` , it'll return `0.0`

When you use the `print()` function, you'll see that the number `0.1` is represented as `0.1` exactly.

Internally, Python can only represent `0.1` approximately.

To see how Python represents the `0.1` internally, you can use the `format()` function.

The following shows how Python represents the number 0.1 using 20 digits:

```
>>> format(0.1, '.20f')
'0.10000000000000000555'
```

As you can see, `0.1` is not exactly `0.1` but `0.10000000000000000555...`

Because Python can represent some floats approximately, it will cause many problems when you compare two floating-point numbers.

Equality testing

Let's take a look at the following example:

```
x = 0.1 + 0.1 + 0.1
y = 0.3

print(x == y)
```

Output:

```
False
```

Internally, Python cannot use a finite number of digits to represent the numbers `x` and `y` :

```
print(format(x, '.20f'))
print(format(y, '.20f'))
```

Output:

```
0.30000000000000004441
0.29999999999999998890
```

Note that the number of digits is infinite. We just show the first 20 digits.

One way to work around this problem is to round both sides of the equality expression to a number of sigificant digits. For example:

```
x = 0.1 + 0.1 + 0.1
y = 0.3
```

```python
print(round(x, 3) == round(y, 3))
```

Output:

```
True
```

This workaround doesn't work in all cases.

PEP485 (https://www.python.org/dev/peps/pep-0485/) provides a solution that fixes this problem by using relative and absolute tolerances.

It provides the `isclose()` function from the `math` module returns `True` if two numbers are relatively close to each other.

The following shows the `isclose()` function signature:

```python
isclose(a, b, rel_tol=1e-9, abs_tol=0.0)
```

For example:

```python
from math import isclose

x = 0.1 + 0.1 + 0.1
y = 0.3

print(isclose(x,y))
```

Output:

```
True
```

Summary

- Python uses `float` class to represent real numbers.

- Python uses a fixed number of bytes (8 bytes) to represent floats. Therefore, it can represent some numbers in binary approximately.
- Use the `isclose()` function from the math module to test equality for floating-point numbers.