Точность численных и временных типов в Питоне

Почему числа так важны?

- Математические **модели** и **алгоритмы** используют числа
- **Транзакции** представлены числами
- **Время** представлено в виде чисел
  - В торговле на бирже нужны **нано**секунды

Тестируем числа (и себя)

```
>>> 0.1 + 0.2 == 0.3
```

Тестируем числа (и себя)

```
>>> 0.1 + 0.2 == 0.3
```

False

Тестируем числа (и себя)

```
>>> 0.1 + 0.2 == 0.3
```

False

```
>>> 0.1 + 0.1 == 0.2
```

Тестируем числа (и себя)

```
>>> 0.1 + 0.2 == 0.3
```

False

```
>>> 0.1 + 0.1 == 0.2
```

True

Что каждый программист должен знать об арифметике с числами с плавающей точкой
http://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html

Почему?

одно число кодируется при помощи 64 бит

Почему?

1/3 -- бесконечная дробь в десятичной системе:
0.333333333...

1/10 – бесконечная дробь в двоичной системе:
0.000**1100110011001**10011001100…

Поскольку в нашем распоряжении только 64 бита (разряда),
то мы имеем только приблизительное значение 1/10 .

Почему?

1/3 -- бесконечная дробь в десятичной системе:
0.333333333...

1/10 – бесконечная дробь в двоичной системе:
0.000**1100110011001**10011001100…

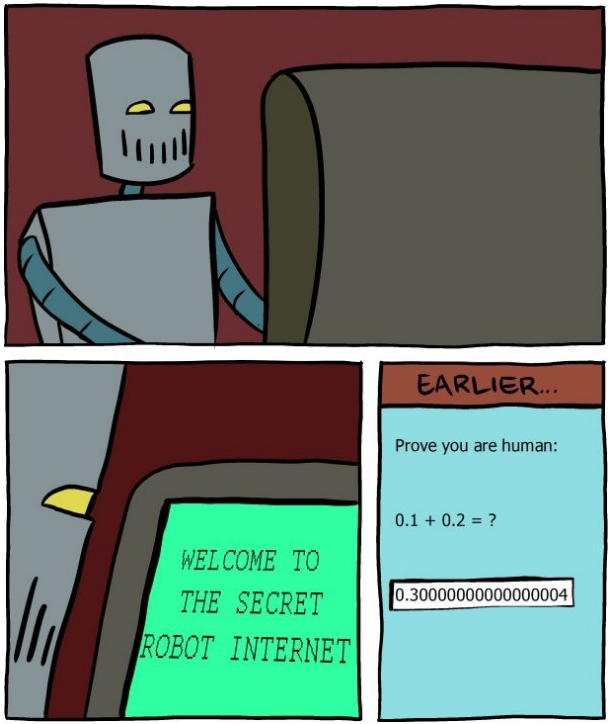
Поскольку в нашем распоряжении только 64 бита (разряда),
то мы имеем только приблизительное значение 1/10 ,

То есть: 0.1000000000000000055511151231257827021181583404541015625

Вывод

Арифметика над числами с плавающей точкой всегда имеет ошибку

Добро пожаловать в тайный интернет для роботов



Ранее:

Докажите, что Вы человек

0.1 + 0.2 = ?

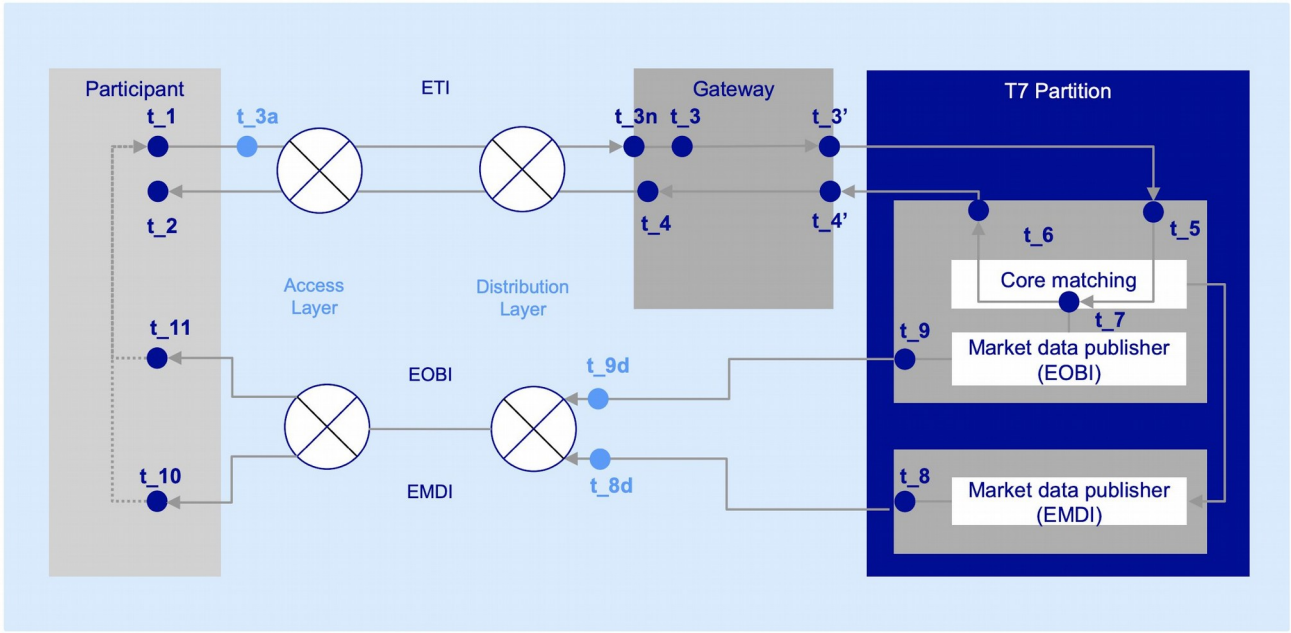
Решение: класс decimal.Decimal

```
>>> (0.1 + 0.2 - 0.3) * 10E20

55511.15123125783

>>> import decimal

>>> D = decimal.Decimal

>>> (D('0.1')+D('0.2') - D('0.3'))

Decimal('0.0')
```

Что насчет **времени**?

- Когда произошло событие
- Традиционно выражается числом, обозначающим количество секунд, прошедших с **1 января 1970 UTC**
- Полезно для сквозной сортировки (напр. входящих торговых ордеров)
- Обычно выражается в секундах (**s**) или миллисекундах (**ms**)
- Более высокая точность: в высокочастотной торговле (High Frequency Trading - HFT) используются наносекунды (**ns**)

Пример с Deutsche Boerse – время высокой точности



каждое “t” это записанное время

Пример с Deutsche Boerse – время высокой точности

	MarketSegmentId	SecurityId	...	RequestTime	ETICaptTime
1	589	2657557	...	**1527141846562083136**	**1527141846562079880**
2	589	2657557	...	**1527141846562082842**	**1527141846562079885**
3	589	2657557	...	**1527141846562083452**	**1527141846682079943**

https://www.mds.deutsche-boerse.com/mds-en/data-services/analytics/high-precision-timestamps

Поддержка времени в наносекундах в питоне

- Кодируется как **64-битное целое** без знака
  - **0 to 18,446,744,073,709,551,615 ((2**64)-1)**
- В Питоне 3.7 появился метод **`time.time_ns()`**, который возвращает время в наносекундах как целое число (**int**)

Время в наносекундах в Pandas

По умолчанию, **Pandas** преобразует числа в числа с плавающей точкой, когда зачитывает данные из **csv**, в котором есть пропущенные значения

- приводит к потере точности в случае целых чисел (int)
- **int**: кодируется 64 битами
- **float**: мантисса кодируется только 51 битом (M × 2^E)

Используйте собственный конвертер

```
def time_converter(num):
    try:
        return np.int64(num)
    except:
        return np.int64(0)
```

```
df = pd.read_csv(PATH,sep=";",

    converters = {
       'ETICaptTime'   : time_converter,
       'EOBICaptTime' : time_converter,
       'TransactTime' : time_converter,
       'AggressorTime': time_converter,
       'RequestTime'   : time_converter })
```

Практика

`numbers_precision.ipynb`