Лекция 4
Числа с плавающей точкой

Давыдов Виталий Валерьевич (ГАИШ МГУ, Постгрес-ПРО)

Повторение пройденного

- Функции, задание аргументов, map, filter, reduce
- Списоковые выражения и генераторные включения
  - `[ x**2 for x in range(100) ]`
  - `( x**2 for x in range(100) )`
- Чтение и запись файлов (open, read, write, close)
- Обработка ошибок (исключения), try, except, finally
- Менеджеры контекста (with as)
- Регулярные выражения
- Рассмотрена практическая задача

Способы хранения чисел в памяти

- **Целые числа**
  - Точно представимы (тип: int)
- **Рациональные числа**
  - Представимы в виде обыкновенной дроби (числитель, знаменатель)
  - Хранение числителя и знаменателя как целые числа (int)
  - Модуль fractions: fractions.Fraction(1, 3)
  - Точное представление рациональных чисел (представимых в виде рац. дробей)
- **Числа с фиксированной точкой**
  - Размер целой и дробной части ограничен (фикс к-во цифр после запятой)
  - Модуль Decimal: Decimal(3.14)
  - Точная арифметика (в рамках заданной точности). Подсчет денежных сумм.
- **Числа с плавающей точкой (IEEE-754)**
  - Можно представить больший диапазон действительных чисел (не все числа точно представимы)
  - Тип float
  - Математические операции производятся процессором (или мат. сопроцессором)

Числа с плавающей точкой

- Позволяют представить широкий диапазон вещественных чисел
- Есть стандарт IEEE-754
- Используются в научных вычислениях
- Быстрые вычисления
- Современные процессоры поддерживают вычисления с плавающей точкой на аппаратном уровне
- От программиста требуется знание, умение и аккуратность при работе с вещественными числами

Вопросы?

- Какие мин/макс значения можно хранить в python.float (double)?
- Можно ли сравнивать числа с плавающей точкой?
- Все-ли числа точно представимы?
- Что такое денормализованные числа?
- Можно ли для сравнения с NaN использовать оператор '=='?
- Возникает ли ошибка при выполнении арифметических операций?
- С какой точностью вычисляется синус (math.sin)?

Информация об F64

- `import sys; print(sys.float_info)`
  - dig = 15
  - epsilon = 2.220446049250313e-16
  - mant_dig = 53
  - max = 1.7976931348623157e+308
  - max_10_exp = 308
  - max_exp = 1024
  - min = 2.2250738585072014e-308
  - min_10_exp = -307
  - min_exp = -1021

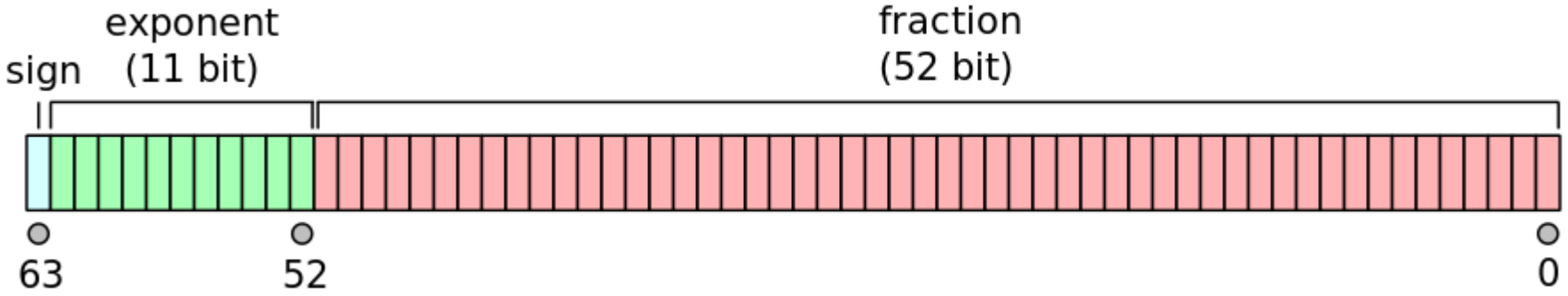
IEEE-754

- **Формат чисел с плавающей точкой:**
  - мантисса, экспонента, знак
  - F32, F64, F128 и др.
- **Представление особых значений:**
  - Положительный и отрицательный ноль
  - Положительная и отрицательная бесконечность
  - NAN
- **Методы, используемые при преобразовании чисел при выполнении мат операций**
- **Правила округления**
- **Исключительные ситуации (деление на ноль, переполнение, потеря значимости и др)**
- **Опарации над числами (арифметические и др)**

Представление дробных чисел в двоичной системе счисления

- **Не все дробные числа точно представимы**
- **Число 5/8 точно представимо**
  - $5/8 = 1/2 + 0/4 + 1/8 = 0.101_2$
- **Число 1/10 представимо неточно**
  - $1/10 = 0.0001100110011001100110011..._2$
  - $1/10 \approx 0.1000000000000000006$ (по IEEE-754, F64) – округление!

Представление F64 в памяти



$$(-1)^{\text{sign}}(1.b_{51}b_{50}\ldots b_0)_2 \times 2^{e-1023}$$

0 01111111111 0000000000000000000000000000000000000000000000000000$_2$ ≙ 3FF0 0000 0000 0000$_{16}$ ≙ $+2^0 \times 1 = 1$

0 10000000000 0000000000000000000000000000000000000000000000000000$_2$ ≙ 4000 0000 0000 0000$_{16}$ ≙ $+2^1 \times 1 = 2$

0 10000000000 1000000000000000000000000000000000000000000000000000$_2$ ≙ 4008 0000 0000 0000$_{16}$ ≙ $+2^1 \times 1.1_2 = 11_2 = 3$

Вопрос

Какой будет вывод?

```
vmin = 9007199254740990.0
vmax = vmin + 10.0
v = vmin
while v < vmax:
    print(v)
    v += 1.0
```

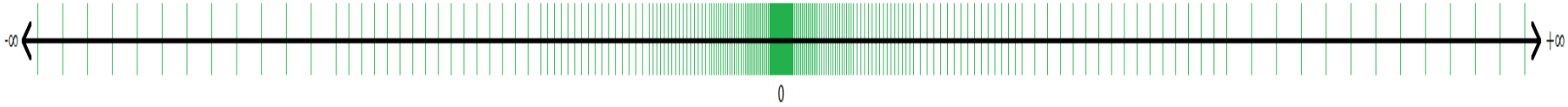
```
import numpy as np

vmin = 9007199254740990.0
vmax = vmin + 10.0
for v in np.arange(vmin, vmax, 1.0):
    print(v)
```

Почему? Какая реализация правильная?

Какие целые числа представимы точно?

Точное представление целых чисел в F64



- Точно представимы целые числа в непрерывном диапазоне от -9007199254740992 до 9007199254740992 (по модулю до 2**53).
- В этом диапазоне с F64 можно работать как с обычными числами типа int.
- За указанными пределами есть точно представимые целые числа, но не все: чем больше число, тем больше "дырки".
- Машинный эпсилон определяет ошибку задания числа
- Вопрос: Print(9007199254740993.0) - ?

9007199254740990.0
9007199254740991.0
9007199254740992.0
...
9007199254740994.0
...
9007199254740996.0
...
9007199254740998.0
...
9007199254741000.0

Вопросы

- Есть тип numpy.float128
- numpy.float128(9007199254740993.0) - ?
- numpy.float128('9007199254740993.0') - ?

Машинный эпсилон

- **Машинный эпсилон (ε) – разница между 1 и следующим представимым значением.**
- **Определения:**
  - 1 + ε > 1, ε – минимальное из возможных значений
  - 1 + ε = 1, ε – максимальное из возможных значений
- **Два (отличных от нуля) числа являются одинаковыми с точки зрения машинной арифметики, если их относительная разность по модулю меньше (при определении первого типа) или не превосходит (при определении второго типа) значение машинного эпсилона.**
- **Эпсилон (F32) = 2-23 ≈ 1.19e-07**
- **Эпсилон (F64) = 2-52 ≈ 2.20e-16**
- **sys.float_info.epsilon**

Машинный эпсилон

```
>>> 1.0 + sys.float_info.epsilon / 2
1.0
>>> 1.0 + sys.float_info.epsilon
1.0000000000000002
```

Сравнение чисел с плавающей точкой

Можно ли сравнивать два числа с плавающей точкой, используя операцию равенства?

Сравнение чисел с плавающей точкой

- **В общем случае сравнение с использованием операции равенства (=) может дать некорректный результат:**
  - `0.1 ** 2 ≠ 0.01`
- **Можно сравнивать с некоторой точностью, например:**
  - `math.fabs(A - B) < 1E-5`, где `1E-5` заданная точность
  - `math.isclose()`
  - `numpy.isclose(a, b)`, `numpy.allclose(arr1, arr2)`
- **Для сравнения двух чисел можно использовать машинный эпсилон**
- **Литералы можно сравнивать точно**
- **Можно почитать:**
  - https://floating-point-gui.de/errors/comparison/
  - https://realtimecollisiondetection.net/blog/?p=89

$$fabs(A - B) <= E_{abs}$$
$$fabs(A - B) <= E_{rel} * MAX(A, B)$$

Вопрос

Результат арифметической операции над числами с плавающей точкой точен?

Потеря значащих разрядов

- **Вычитание двух близких чисел:**
  - 1.2345432 - 1.23451 = 0.0000332 – точный результат
  - 1.23454 - 1.23451 = 0.00003 – результат в случае 6-ти значной мантиссы
- **Сложение или вычитание большого и малого числа:**
  - 12 345.6 + 0.123456 = 12 345.72345 (точный результат)
  - 12 345.6 + 0.123456 = 12 345.7 – результат в случае 6-ти значной мантиссы
- **Баг gcc 323 (optimized code gives strange floating point results), https://gcc.gnu.org/bugzilla/show_bug.cgi?id=323**

Накопление ошибки

- При вычислениях ошибка накапливается
- Алгоритм компенсационного суммирования (math.fsum)
- Метод сортировки и суммирования с малых значений

```
>>> a = 0
>>> for x in range(10000000):
...     a += 0.1
...
>>> a
999999.9998389754
```

```
>>> a = 0
>>> for x in range(10000000):
...     a += 0.5
...
>>> a
5000000.0
```

Агоритм компенсационного суммирования:

```
>>> import math
>>> math.fsum(0.1 for x in range(10000000))
1000000.0
```

Обращение в машинный ноль

- **Underflow**
- **Денормализованные числа (потеря точности)**
  - Subnormal
  - Мантисса начинается с нуля (нет явной единицы)
  - Экспонента – минимально возможная (нулевые биты экспоненты)
  - Машинный ноль – денормализованное число
- **Иногда это является проблемой для вычислений**
- **Что можно сделать?**
  - Использовать типы с большей разрядностью
  - Умножить все числа на большую константу (например, 1E+100)

```
>>> x = 1
>>> for i in range(101):
...     print(x)
...     x *= 5.E-4
...
1
0.0005
2.5e-07
1.25e-10
6.250000000000001e-14
3.1250000000000001e-17
```

```
4.038967834731588e-301
2.019483917365794e-304
1.0097419586828971e-307
5.0487097934146e-311
2.5243548965e-314
1.2621776e-317
6.31e-321
5e-324
0.0
0.0
```

Переполнение

- Overflow
- Специальные значения:
  - INF, -INF
- math.inf, -math.inf
- math.isinf()
- math.isfinite()

```
>>> import numpy as np
>>> for x in range(1, 40, 4):
...     print('exp({}) = {}'.format(x**2, np.exp(x**2)))
...
exp(1) = 2.718281828459045
exp(25) = 72004899337.38588
exp(81) = 1.5060973145850306e+35
exp(169) = 2.487524928317743e+73
exp(289) = 3.2441824460394912e+125
exp(441) = 3.340923407659982e+191
exp(625) = 2.7167594696637367e+271
<stdin>:2: RuntimeWarning: overflow encountered in exp
exp(841) = inf
exp(1089) = inf
exp(1369) = inf
```

NAN

- NaN – Not a Number
- Нельзя сравнивать!
- math.nan
- math.isnan()
- math.isfinite()
-
- Вопросы:
  - math.inf / math.inf
  - math.inf + math.inf
  - math.inf - math.inf
  - math.nan == math.nan

Подумать

- **Вычисление тригонометрических функций?**
  - Как вычисляется sin?
  - С какой точностью?
- **Почему `math.sin(math.pi)` ≠ 0?**
- **Что надо сделать, чтобы `math.sin(math.pi)` == 0?**