# Урок 09.01. Экспоненциальная форма вещественного числа

Мы уже хорошо поработали с функциями и узнали довольно таки много. И нам нужно вернуться к вещественным числам, а заодно и узнать как на самом деле дробные числа представляются в памяти компьютера. И как всегда будем учиться на примерах.

Разберем такую задачу «Очень маленькое число».

Один научно-исследовательский институт сейчас работает с очень большими и очень маленькими числами для вывода новых математических законов. Он дал нам задачу: какое число получится при делении единицы пополам 10 000 раз.

Такую задачу мы с легкостью можем решить вот так:

X = 1

for i in range(10000):

 x /= 2

print(x)

Запустим программу и… получим «0». Но почему? Выглядит очень странно. Давайте разбираться. Может это какая-то ошибка? Давайте уберем один нолик у 10 000, получится цикл на 1000 раз. И запустим программа. Вот теперь получили результат, правда, какой-то странный.

Давайте попробуем посмотреть как происходит деление. Для этого заменим цикл for на цикл while:

x = 1

count = 0

while x != 0:

 x /= 2

 count += 1

 print(x)

print(‘Итераций: ’, count

Запустим и получим 1075 раз. Как-то маловато до 10 000 раз. И теперь для того чтобы понять, что же произошло, давайте разбираться.

Начнем с того, что компьютер умеет работать только с целыми числами, а для работы с вещественными числами придумали хитрость: число хранится в памяти в виде двух целых чисел. Вообще вещественное число можно представить в классическом виде «4.5» или в виде «число с плавающей точкой»: 45∙10-1. Вот в последнем случае наше вещественное число задается как два целых числа: первое число показывает сами наши числа «45», а второе число задает степень десяти, или другими словами показывает, где нужно расположить точку. При этом число «45» называется мантисса, а «-1» – порядок или экспонента. В таком формате можно представить любое число. Именно в таком видео компьютер и хранит вещественные числа, только немного изменив запись. Так наше число 4.5 может быть выведено как «45e-1». И именно в таком виде нам показал компьютер результат.

И теперь с этими знаниями взглянем поподробнее на результаты нашей программы. Последнее число, которое мы увидели это 5e-324, это очень маленькое число, даже представить его очень сложно, а компьютер может.

И давайте решим еще одну задачу для закрепления.

Задача «Колонизация»

Ученые изучают планеты, и пытаются определить, какие пригодны для колонизации людьми, а какие нет и для этого они должны знать базовые характеристики планеты. Базовыми характеристиками планеты являются масса (m), объем (V) и плотность (p). У нас есть две планеты, объемы каждой из них нам известны, а данные о массе нам сообщают другие ученые. Нам нужно найти плотность каждой планеты, и понять какая из них ближе всего по плотности к нашей Земле (p = m / V).

Напишем такую программу

vFirstPlanet = 1.43128e15

vSecondPlanet = 6.254e13

pEarth = 5.5153e12

mFirstPlanet = float(input('Масса первой планеты: '))

mSecondPlanet = float(input('Масса второй планеты: '))

pFirstPlanet = mFirstPlanet / vFirstPlanet

pSecondPlanet = mSecondPlanet / vSecondPlanet

print('Плотность первой планеты: ', pFirstPlanet

print('Плотность второй планеты: ', pSecondPlanet

if abs(pEarth – pFirstPlanet) < abs(pEarth – pSecondPlanet):

 print('Первая планета ближе по плотности')

else:

 print('Вторая планета ближе по плотности')

Введем массы планет: 1.8985е27 и 1.0243е26. Получим следующие плотности: 1326435079090.0452 и 1637831787655.9004 числа больше. Но и плотность Земли не маленькая. В результате получим, что вторая планета ближе чем первая, что является правильно. И может сказать, что наша программа работает.

## Особенности работы с вещественными числами

Так, мы уже разобрались, как записываются большие и маленькие числа в python, однако мы до сих пор не знаем как они представляются внутри самого компьютера. Рассмотрим такое выражение

if (1.1 + 2.2) == 3.3:

 print('Равно')

else:

 print('На равно')

Для нас, очевидно, что такое эти выражения равны, но если запустить программу, то python с этим не согласен – странно, не так ли. И если данные числа сложить и вывести, то мы получим «3.000000000000000003», это число действительно не равно 3.3, но в чем же дело? И тут нам нужно еще кое что узнать про то как всё же на самом деле числа хранятся в компьютере. На самом деле тут играет роль то, что числа как мы уже знаем хранятся в памяти и на хранение чисел у компьютера есть ограничения и некоторые дробные числа (1/3, 2/3 и тд) мы можем хранить только приближённо. И сравнение дробных чисел всегда несет в себе риски потере точности, и тут проблема не в языке python, а в особенностях представления чисел и их обработкой процессором.

И для проверки мы можем рассмотреть такой пример:

a = 1

while True:

 a += 1e-324

 print(a)

Запустим программу и получим, что переменная а не меняется, всегда равна 1. Тогда уменьшим прибавляемое число и поставим «1e-100», запустим, и всё ещё ничего не происходит. Тогда уменьшим еще – «1е-10», так, число а начало увеличиваться. Тогда возьмём «1е-20» – ничего. Возьмем «1е-15» – работает, тогда берем «1е-16» и уже ничего не работает.

И дело тут в том, что когда числа записываются в памяти, то они занимают конкретное число ячеек (бит) памяти, и если все цифры нашего числа начинают не помещаться в эти ячейки, то компьютер считает их не нужными и отбрасывает. И тут мы подходим к тому, что у нашей переменной а все разряды заняты, правда нулями, и когда мы пытаемся к ней прибавить число, которое меньше чем разрядов может хранить наша переменная, то мы и получаем отбрасывание меньших разрядов. А когда мы стали прибавлять «1е-16», то нашил минимальное возможное значение количества цифр, которые запоминаются в данном случае. Такое минимальное число называется машинный эпсилон – это минимально различимое для программы число.

И тогда, возвращаясь к нашей первой задачи, мы попробуем воспользоваться этими знаниям. А именно, мы понимаем, что наши выражения различаются где-то в районе эпсилона, тогда мы можем предположить, что если из одного выражения (1.1 + 2.2) вычесть другое выражение (3.3) и наша разница будет меньше эпсилона, то эти числа равны, так и запрограммируем и попутно все числа перенесем в переменные:

a = 1.1

b = 2.2

c = 3.3

d = a + b

if abs(d – c) < 1e-16:

 print('Равно')

else:

 print('На равно')

Запустим. Опять не получилось, однако тут все просто – наша разница не может быть меньше чем минимально различимое число, тогда нам просто нужно увеличить, то с чем мы сравниваем. Например, увеличить степень с 16 на 15:

a = 1.1

b = 2.2

c = 3.3

d = a + b

if abs(d – c) < 1e-15:

 print('Равно')

else:

 print('На равно')

Вот теперь у нас все получилось. И такой способ называется задание точности вычислений.

И нужно запомнить, что необходимо избегать сравнения двух вещественных чисел, однако если нам всё же нужно их сравнить, то нужно пользоваться показанным способом с заданием точности.

## Алгоритмы с заданной точностью

Продолжим изучать связь математики и программированием.

Иногда в задачах необходимо получить точный ответ, но получить точный ответ мы не можем, а только приблизительно.

Например, банки очень любят деньги и поэтому дают кредиты на самых разных условиях. И одним из условий может быть способ платежа – аннуитетный. Это такой платеж, когда ты всегда платишь одну и туже сумму, но часть суммы идет на погашение долга, а часть суммы идет на погашение процентов. Очень легко составить программу, которая посчитает сколько нужно платить если вы платите определенную сумму. Однако в жизни, как правило, всех интересует сколько мне нужно платить, чтобы расплатиться за 2 года.

Данная задача математически уже давно решена, и выглядит это решение так:



И когда мы видим подобные дроби, то сразу чисел после запятой будет очень много. А так как это работа с деньгами, то и считать мы должны точно, и при этом не забывать о том, что денег меньше копеек у нас ничего нет. И здесь используются специальные алгоритмы, которые так и называют «алгоритмы с заданной точностью».

Подобная задача у вас будет в практических заданиях.

А разбираться мы будем на примере попроще.

В математике очень часто используется число Эйлера (это один из великих математиков), и чтобы его получить мы можем подключить библиотеку математики и получить его (math.e). Однако сегодня мы его сами посчитаем, и для этого нам нужно знать, как оно считается, и считать нам его надо будет с заданной точностью. А формула следующая:

$$e=\frac{1}{0!}+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+…$$

Пусть пользователь вводит точность, а программа уже будет вычислять это число с заданной точностью. Начнем

import math

Def exp(precision):

 result = 0

 while addMember > precision:

 addMember = 1 / math.factorial(i)

 result += addMember

 return result

my\_precision = float(input('Введите точность: '))

print('Результат:', exp(my\_precision))

print('Число из библиотеки:', math.e)