Повторяем пройденное

Задание 1. Написать программу, решающую уравнение вида

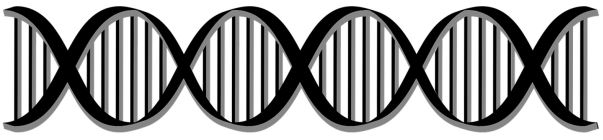
$$ax^2 + bx + c = 0$$

для любых значений коэффициентов a, b, c.

In [ ]:
```
a = float(input('a = '))
b = float(input('b = '))
c = float(input('c = '))

if a!=0:      # квадратное уравнение
    d = b**2 - 4*a*c
    if d > 0:
        x1 = (-b + d**0.5)/(2*a)
        x2 = (-b - d**0.5)/(2*a)
        print(f'{x1 = } { x2= }')
    elif d == 0:
        x = -b/(2*a)
        print(f'{x = }')
    else:
        print('Нет решений')
elif b!=0:    # линейное уравнение
    x = -c/b
    print(f'{x = }')
elif c!=0:    # неверное равенство c=0
    print('Нет решений')
else:         # тождество 0=0
    print('x - любое число')
```

Циклы



Цикл `while`

Вспомним вот эту задачу

Integer8. Дано двузначное число. Вывести число, полученное при перестановке цифр исходного числа.

и ее решение

```
n = int(input("двузначное число: "))
e = n%10
d = n//10
print(10*e + d)
```

Немного подумав и удлинив код, можно записать решение и для трехзначных чисел. И даже для четырехзначных. А для 150-значных? 😱🤷

На помощь приходят **циклы**. Они дают возможность выполнить одну и ту же последовательность действий много раз.

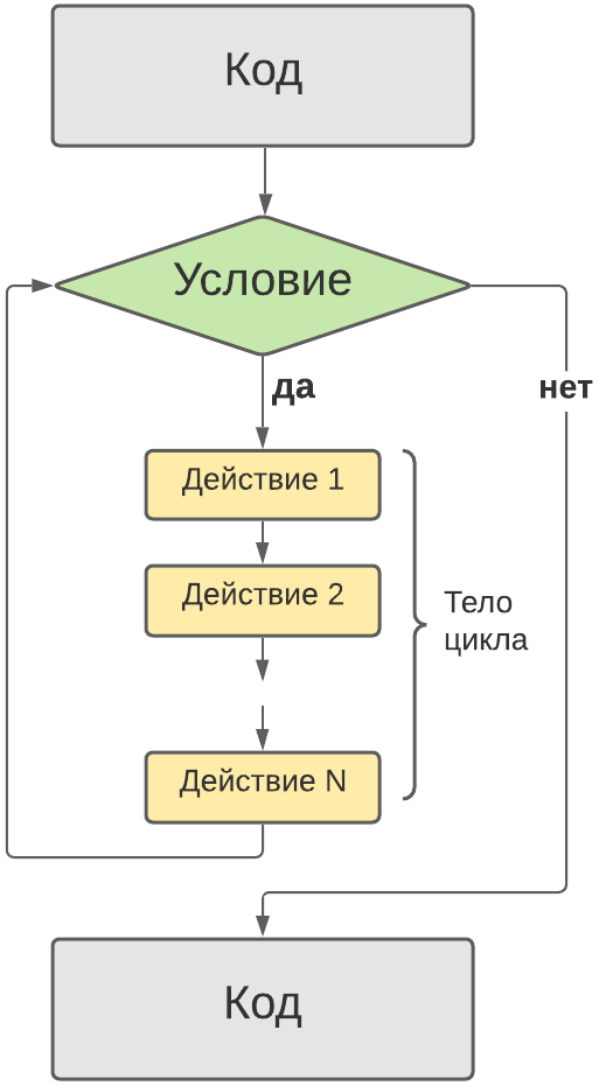
Как много? Столько, столько требуется (если, конечно, программа написана правильно).

Цикл while ("***пока***") позволяет выполнять последовательность одинаковых действий, пока проверяемое условие истинно.

Чуть позже решим задачу типа **Integer8** для числа с любым количеством десятичных цифр. А пока еще картинка и задачи попроще.

Синтаксис цикла `while` в простейшем случае выглядит так:

```
# предыдущий код
while условие:
    действие 1
    действие 1
    ...
    действие N
# код после цикла
```



При выполнении цикла `while` сначала проверяется истинность или ложность условия. Если условие истинно, то выполняется цепочка действий (*тело цикла* или одна *итерация*), после чего условие проверяется снова и снова выполняется эта цепочка. Так продолжается до тех пор, пока условие будет истинно. Как только условие станет ложно, работа цикла завершится и управление передастся следующей инструкции после цикла.

Если условие ложно в самом начале, то цикл не начинается, управление сразу передается на следующую команду после тела цикла `while` .

В качестве примера рассмотрим задачу:
Дано натуральное число m. Вычислить m^2, не используя символ `*` .

Умножения придется заменить сложением:

$$m^2 = \underbrace{m + m + \ldots + m}_{m \text{ слагаемых}}$$

На каждом шаге цикла нужно увеличивать сумму, а также уменьшать оставшееся количество итераций.

In [ ]:
```
m = int(input("m: "))
iters_left = m      # количество оставшихся итераций
answer = 0          # переменная для накопления суммы
while iters_left>0:
    answer += m
    iters_left -= 1
print(f'm^2 = {answer}')
```

Устные упражнения: определите результат работы программы

In [1]:
```
# Пример 1. Что выведет программа?

num = 0
while num <= 5:
    print (num)
    num += 1

print ("Вне цикла")
print (num)
```

```
0
1
2
3
4
5
Вне цикла
6
```

In [2]:
```
# Пример 2. Что выведет программа?

number_of_loops = 0
number_of_apples = 2
while number_of_loops < 10:
    number_of_apples *= 2
    number_of_apples += number_of_loops
    number_of_loops -= 1
print ("Количество яблок: ", number_of_apples)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-2-537e921c8738> in <module>
      4 number_of_apples = 2
      5 while number_of_loops < 10:
----> 6     number_of_apples *= 2
      7     number_of_apples += number_of_loops
      8     number_of_loops -= 1

KeyboardInterrupt:
```

In [3]:
```
number_of_loops
```

Out[3]: -752926

In [4]:
```
# Пример 3. Что выведет программа?

num = 10
while num > 3:
    num -= 1
    print (num)
```

```
9
8
7
6
5
4
3
```

В языке Питон, как и во многих других языках, есть специальная команда, позволяющая прервать выполнение итераций цикла и сразу перейти к командам, расположенным после тела цикла. Это команда `break` .

In [5]:
```
# Пример 4. Что выведет программа?

num = 10
while True:
    if num < 7:
        print ('Прерываем цикл')
        break
    print (num)
    num -= 1
print ('После цикла')
```

```
10
9
8
7
Прерываем цикл
После цикла
```

In [ ]:
```
# Пример 5. Что выведет программа?

num = 100
while not False:
    if num < 0:
```

```
        break
print ('num is:', num)
```

Порешаем задачи

While4. Дано целое число n (> 0). Если оно является степенью числа 3, то вывести `True`, если не является — вывести `False`.

In [11]:
```
# Решение
n = int(input("целое число: "))
# вариант 1
while n%3==0:
    n //= 3
print(n==1)

# вариант 2
n = int(input("целое число: "))
p = 1
while p<n:
    p *= 3
print(p==n)
```

```
целое число: 515377520732011331036461129765621272702107522001
True
целое число: 515377520732011331036461129765621272702107522001
True
```

In [7]:
```
3**100
```

Out[7]:
```
515377520732011331036461129765621272702107522001
```

While18. Дано целое число n (> 0). Используя операции деления нацело и взятия остатка от деления, найти количество и сумму его цифр.

In [12]:
```
# Решение
n = int(input("целое число: "))
k = 0
s = 0
while n>0:
    k += 1
    s += n%10
    n //= 10
print(k, s)
```

```
целое число: 3864
4 21
```

Integer8 (усложненный вариант). Дано натуральное число. Вывести число, полученное при перестановке цифр исходного числа в обратном порядке.

In [13]:
```
# Решение
n = int(input("натуральное число: "))
s = 0
while n>0:
    s = s*10 + n%10
```

```
    n //= 10
print(s)
```

```
натуральное число: 3864
4683
```

While22. Дано целое число n (> 1). Если оно является простым, т. е. не имеет положительных делителей, кроме 1 и самого себя, то вывести `True`, иначе вывести `False`.

Работу программы можно проверить на числах 9973, 9999901 и 2147483647, которые являются простыми.

In [16]:
```
# Решение 1
n = int(input("натуральное число: "))
k = 2
while k<n:
    if n%k==0:
        break
    k += 1
print(k==n)
```

```
натуральное число: 9999901
True
```

In [20]:
```
# Решение 2
n = int(input("натуральное число: "))
k = 2
b = int(n**0.5)+1
while k<b:
    if n%k==0:
        break
    k += 1
print(k==b)
```

```
натуральное число: 2147483647
True
```

While24. Дано целое число n (> 1). Последовательность чисел Фибоначчи F_k определяется следующим образом:

$$F_1 = 1,\ F_2 = 1, F_k = F_{k-2} + F_{k-1},\ k = 3, 4, \ldots .$$

Проверить, является ли число n числом Фибоначчи. Если является, то вывести `True`, если нет — вывести `False`.

Например, 17711 - число Фибоначчи

In [22]:
```
# Решение
n = int(input("натуральное число: "))
a, b = 1, 1
while b<n:
    a, b = b, a+b
    print(b)
print(b==n)
```

```
натуральное число: 17711
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
10946
17711
True
```

«**Стандартное отклонение**» (pythontutor.ru)
Дана последовательность натуральных чисел x_1, x_2, ..., x_n. *Стандартным отклонением* называется величина

$$\sigma = \sqrt{\frac{(x_1 - s)^2 + (x_2 - s)^2 + \ldots + (x_n - s)^2}{n - 1}}$$

где $s = \frac{x_1 + x_2 + \ldots + x_n}{n}$ — среднее арифметическое последовательности.

Определите стандартное отклонение для данной последовательности натуральных чисел, завершающейся числом 0.

Комментарий к решению.
На первый взгляд, задачу нельзя решить, не сохраняя все значения переменных x_k, потому что для вычисления, например, $(x_1 - s)$ нужно знать s, то есть сначала нужно ввести все числа, а потом вернуться к x_1.

Решить задачу помогает школьная алгебра и знание формулы

$$(a - b)^2 = a^2 - 2ab + b^2.$$

Действительно,

$$(x_1 - s)^2 + (x_2 - s)^2 + \ldots + (x_n - s)^2 =$$

$$= x_1^2 - 2x_1 s + s^2 + x_2^2 - 2x_2 s + s^2 + \ldots x_n^2 - 2x_n s + s^2 =$$

$$= \sum_{i=1}^{n} x_i^2 - 2\sum_{i=1}^{n} x_i s + ns^2 =$$

$$= \sum_{i=1}^{n} x_i^2 - \frac{2}{n}\left(\sum_{i=1}^{n} x_i\right)^2 + \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2 =$$

$$= \sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2.$$

Таким образом, в процессе ввода необходимо накапливать значение суммы вводимых чисел и суммы их квадратов.

In [ ]:
```
# Решение
```

Цикл for

Цикл `for`, также называемый циклом с параметром, позволяет выполнить заранее известное количество итераций. Параметр цикла, или *счетчик*, — это переменная, указываемая в цикле `for`. В нем также задается множество (диапазон) значений, по которому будет пробегать эта переменная.

```
# ... какой-то код до цикла
for параметр_цикла in множество_значений:
    действие 1
    действие 2
    ...
    действие N
# ... код после цикла
```

Множество значений может быть задано, например, списком, строкой или диапазоном. Начнем с последнего — очень распространенного и полезного объекта `range()`.

In [23]:
```
# меняйте параметры функции range,
#         чтобы лучше понять, как она работает
for i in  range(1,10): # [1, 10)
    print(i, end=' ')
```

```
1 2 3 4 5 6 7 8 9
```

In [24]:
```
for i in range(0,10):
    print(i, end=' ')
```

```
0 1 2 3 4 5 6 7 8 9
```

In [25]:
```
for i in range(10):
    print(i, end=' ')
```

```
0 1 2 3 4 5 6 7 8 9
```

In [26]:
```
for i in range(0,10,2):
    print(i, end=' ')
```

```
0 2 4 6 8
```

In [27]:
```
for i in range(0,10,-1):
    print(i)
```

In [28]:
```
for i in range(10,0,-1): #[10, 0)
    print(i, end=' ')
```

```
10 9 8 7 6 5 4 3 2 1
```

In [29]:
```
range(10)
```

Out[29]:
```
range(0, 10)
```

In [30]:
```
# range умеет проверять
500 in range(1000)
```

Out[30]:
```
True
```

Примеры других вариантов задания множества значений параметра цикла.

In [31]:
```
# "перечисление" (список или кортеж)
print('Цвета российского флага:')
i = 1
for color in 'белый', 'синий', 'красный':
    print(f'Цвет {i} - {color}')
    i += 1
```

```
Цвета российского флага:
Цвет 1 - белый
Цвет 2 - синий
Цвет 3 - красный
```

In [ ]:
```
# В этом примере ниже символ "\" позволил разорвать
# слишком длинную строку (для удобства чтения)
# После этого символа в строке не должно быть ничего,
#       даже пробела

i = 1
for цвет in 'красный','оранжевый',  \
            'желтый', 'зеленый', 'голубой', \
            'синий', 'фиолетовый':
    print(f'{i}-й цвет радуги - это {цвет}')
    i += 1

# И да, Python разрешает русские имена переменных.
# Но не стоит этим злоупотреблять!
```

In [32]:
```
# Цикл по символам строки
i = 1
for letter in 'КОЖЗГСФ':
    print(f'Название {i}-го цвета радуги начинается с буквы',
          letter)
    i += 1
```

```
Название 1-го цвета радуги начинается с буквы К
Название 2-го цвета радуги начинается с буквы О
Название 3-го цвета радуги начинается с буквы Ж
Название 4-го цвета радуги начинается с буквы З
Название 5-го цвета радуги начинается с буквы Г
Название 6-го цвета радуги начинается с буквы С
Название 7-го цвета радуги начинается с буквы Ф
```

Решаем задачи

Давно обещанное. С использованием модуля `time` выясните, наконец, какой из двух вариантов обмена значений двух переменных работает быстрее:

```
c = a
a = b
b = c
```

или

```
a,b = b,a
```

In [33]:
```
from time import time
n = 10**7
a = 10.4
b = 20
# 1 способ
start = time()
for i in range(n):
    c = a
    a = b
    b = c
finish = time()
print(finish-start)

# 2 способ
start = time()
for i in range(n):
    a, b = b, a
finish = time()
print(finish-start)
```

```
2.200002908706665
1.880002737045288
```

For7. Даны два целых числа a и b ($a < b$). Найти сумму всех целых чисел от a до b включительно.

In [34]:
```
a = int(input("введите целое число a "))
b = int(input("введите целое число b "))
# Решение с использованием цикла
s = 0
for i in range(a,b+1):
    s += i
print(s)


# Эффективное решение
s = sum(range(a, b+1))
```

```
введите целое число a 1
введите целое число b 5
15
```

In [35]:
```
# сравнение
from time import time
```

```
n = 10**7
start = time()
s = 0
for x in range(n):
    s += x
finish = time()
print(s, finish-start)



# Эффективное решение
start = time()
s = sum(range(n))
finish = time()
print(s, finish-start)
```

```
49999995000000 1.8900024890899658
49999995000000 0.5700008869171143
```

❓❓❓ Почему при решении задачи «**Стандартное отклонение**» не пользовались функцией `sum` ?

For19. Дано целое число n (> 0). Найти произведение

$$n! = 1 \cdot 2 \cdot \ldots \cdot n$$

In [39]:
```
# Решение
n = int(input("введите целое n(>0): "))
p = 1
for i in range(1, n+1):
    p *= i
print(p)
```

```
введите целое n(>0): 200
788657867364790503552363213932185062295135977687173263294742533244359449963403342920304284011984
623904177212138919638830257642790242637105061926624952829931113462857270763317237396988943922445
621451664240254033291864131227428294853277524242407573903240321257405579568660226031904170324062
351700858796178922227896237038973747200000000000000000000000000000000000000000000000000
```

In [40]:
```
import math
print(math.prod(range(1,n+1)))

print(math.factorial(n))
```

```
788657867364790503552363213932185062295135977687173263294742533244359449963403342920304284011984
623904177212138919638830257642790242637105061926624952829931113462857270763317237396988943922445
621451664240254033291864131227428294853277524242407573903240321257405579568660226031904170324062
351700858796178922227896237038973747200000000000000000000000000000000000000000000000000
788657867364790503552363213932185062295135977687173263294742533244359449963403342920304284011984
623904177212138919638830257642790242637105061926624952829931113462857270763317237396988943922445
621451664240254033291864131227428294853277524242407573903240321257405579568660226031904170324062
351700858796178922227896237038973747200000000000000000000000000000000000000000000000000
```

For17. Дано вещественное число x и целое число n (> 0). Используя один цикл (и не используя операцию возведения в степень), найти сумму

$$1 + x + x^2 + x^3 + \ldots + x^n.$$

дополнительно: а почему в этой задаче действительно важно не использовать возведение в степень?

In [41]:
```
# Решение
x = float(input('x: '))
n = int(input("введите целое n(>0): "))
s = 1
p = 1
for i in range(1, n+1):
    p *= x
    s += p
print(s)
```

```
x: 1
введите целое n(>0): 10
11.0
```

For24. Дано вещественное число x и целое число n (> 0). Найти значение выражения

$$1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots + (-1)^n \frac{x^{2n}}{(2n)!}$$

Полученное число является приближенным значением функции $\cos$ в точке x.

In [44]:
```
# Решение
x = float(input("введите x: "))
n = int(input("введите целое n(>0): "))
s = 0
p = 1
d = 1
for i in range(n+1):
    p *= d
    s += p
    d = -x*x/(2*i+1)/(2*i+2)
print(f'{s = }')
from math import cos
print(f'cos({x}) = {cos(x)}')
```

```
введите x: 1.57
введите целое n(>0): 5
s = 0.0007958647579494599
cos(1.57) = 0.0007963267107332633
```

For39. Даны целые положительные числа m и n ($m < n$). Вывести все целые числа от m до n включительно; при этом каждое число должно выводиться столько раз, каково его значение (например, число 3 выводится 3 раза).

In [46]:
```
# Решение
m = int(input("введите целое m(>0): "))
n = int(input("введите целое n(>0): "))
for i in range(m, n+1):
    for _ in range(i):
        print(i, end = ' ')
    print()
```

```
введите целое m(>0): 5
введите целое n(>0): 12
5 5 5 5 5
6 6 6 6 6 6
7 7 7 7 7 7 7
8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9
10 10 10 10 10 10 10 10 10 10
11 11 11 11 11 11 11 11 11 11 11
12 12 12 12 12 12 12 12 12 12 12 12
```

Minmax2. Дано целое число n и набор из n прямоугольников, заданных своими сторонами — парами чисел (a, b). Найти минимальную площадь прямоугольника из данного набора.

In [4]:
```
float('inf')*10
```

Out[4]:
```
inf
```

In [5]:
```
# Решение
n = int(input("введите целое n(>0): "))
min_s = float('inf')
for i in range(n):
    a = float(input("a:"))
    b = float(input("b:"))
    s = a*b
    if s<min_s:
        min_s = s
print(min_s)
```

```
введите целое n(>0): 5
a:10
b:8
a:6
b:9
a:7
b:7
a:3
b:4
a:2
b:12
12.0
```

Minmax6. Дано целое число n и набор из n целых чисел. Найти номера первого минимального и последнего максимального элемента из данного набора и вывести их в указанном порядке.

In [6]:
```
# Решение
n = int(input("введите целое n(>0): "))
min_el = float('inf')
max_el = -float('inf')
for i in range(1, n+1):
    x = int(input('x:'))
    if x<min_el:
        min_el = x
        min_no = i
    if x>=max_el:
```

```
        max_el = x
        max_no = i
print(min_no, max_no)

# 1 2 3 1 3 1 2
```

```
введите целое n(>0): 7
x:1
x:2
x:3
x:1
x:3
x:1
x:2
1 5
```

Задачи с сайта pythontutor.ru

Лесенка. По данному натуральному $n \leq 9$ выведите лесенку из n ступенек, i-я ступенька состоит из чисел от 1 до i без пробелов. Например, при $n = 5$ должно получиться

```
1
12
123
1234
12345
```

In [8]:
```
# Решение
n = int(input("введите n(<=9): "))
for i in range(1, n+1):
    for j in range(1, i+1):
        print(j, end='')
    print()
```

```
введите n(<=9): 9
1
12
123
1234
12345
123456
1234567
12345678
123456789
```

Карточка. Для настольной игры используются карточки с номерами от 1 до n. Одна карточка потерялась. Найдите ее, зная номера оставшихся карточек.

Дано число n, далее $n - 1$ номер оставшихся карточек (различные числа от 1 до n). Программа должна вывести номер потерянной карточки.

In [9]:
```
# Решение
n = int(input("введите натуральное n: "))
s = n*(n+1) // 2
```

```
for i in range(n-1):
    x = int(input("карточка:"))
    s -= x
print(s)
```

```
введите натуральное n: 6
карточка:5
карточка:3
карточка:6
карточка:1
карточка:2
4
```

Второй максимум. Последовательность состоит из различных натуральных чисел и завершается числом 0. Определите значение второго по величине элемента в этой последовательности. Гарантируется, что в последовательности есть хотя бы два элемента.

In [10]:
```
# Решение
max1, max2 = 0, 0
while True:
    x = int(input('x:'))
    if x==0:
        break
    if x>max1:
        max2 = max1
        max1 = x
    elif x>max2:
        max2 = x
print(max2)
```

```
x:6
x:7
x:1
x:4
x:8
x:2
x:5
x:0
7
```

Готовимся к контрольной

Финальная задача демонстрационного варианта контрольной работы

Назовем положительное число n *квадратно-четным*, если все цифры его квадрата – четные. Среди однозначных таких чисел два (2 и 8), примером двухзначных могут служить числа 20 и 68 (202 = 400, 682 = 4624), 498 – это пример трехзначного квадратно-четного числа (4982 = 248004). Разработайте программу, которая для заданного натурального числа m ($m < 8$) определяет, сколько существует m -значных квадратно-четных чисел. Примерная схема взаимодействия программы с пользователем:

Введенное число	Сообщение программы
m = -1	Введенное Вами число должны быть положительно. Учтите это при следующем запуске

Введенное число	Сообщение программы
	программы
m = 10	Ваше число больше семи. Программа может работать слишком долго. В следующий раз введите число поменьше
m = 2	Кол-во 2-значных квадратно-четных чисел = 6
m = 3	Кол-во 3-значных квадратно-четных чисел = 18
m = 7	Кол-во 7-значных квадратно-четных чисел = 850

In [12]:
```
# Решение
m = int(input('m:'))
if m<=0:
    print('Введенное Вами число должны быть положительно. Учтите это при следующем запуске прогр
elif m>7:
    print('Ваше число больше семи. Программа может работать слишком долго. В следующий раз введи
else:
    k = 0
    for num in range(10**(m-1), 10**m):
        k += 1
        square = num*num
        while square:
            c = square%10
            if c%2 != 0:
                k -= 1
                break
            square //= 10
    print(k)
```

```
m:7
850
```

Первое знакомство с методом половинного деления. Он же метод бинарного поиска. Или дихотомии.

Дано вещественное число z. Найдите кубический корень из этого числа с заданной точностью ε.

Найти с заданной точностью означает "найти такое число x, что $|x - \sqrt[3]{z}| < \varepsilon$".

Сначала рассмотрим вспомогательную задачу:

Дано натуральное число N. Проверьте, является ли это число кубом

In [15]:
```
N = int(input('Введите натуральное число: '))
ans = 0
while ans**3 < N:
    ans += 1
if ans**3 == N:
    print('кубический корень равен',ans)
else:
    print('число не является точным кубом')
```

```
Введите натуральное число: 700
число не является точным кубом
```

Для вычисления кубического корня из вещественного числа можно использовать эту же идею. Только прибавлять нужно не единицу, а ε.

In [18]:
```
z = float(input("z (>0): "))
eps = 1.0e-8
ans = 0
while ans**3 < z:
    ans += eps
print(ans)
```

```
z (>0): 2
1.2599210507102045
```

In [17]:
```
2**(1/3)
```

Out[17]:
```
1.2599210498948732
```

К сожалению, эта программа работает очень медленно: если $\varepsilon = 10^{-12}$, то результата можем не дождаться (впрочем, желающие могут попробовать)

Идея: кубический корень из положительного числа z точно находится на отрезке [0, max(1,z)].

Сначала возьмем середину этого отрезка и обозначим ее через c. Корень слева от c, если $c^3 > z$, или справа, если $c^3 < z$. Выберем ту половину, где корень, и найдем ее середину. Повторяем процесс, пока очередная половинка не станет меньше, чем ε

In [24]:
```
# Решение
z = float(input('z='))
eps = 1e-15
a = 0
b = max(1,z)
while abs(b-a)>eps:
    c = (a+b)/2
    if c**3<z:
        a = c
    else:
        b = c
print(c)
print(0.125**(1/3))
```

```
z=0.125
0.4999999999999991
0.5
```

Подумать:

- что произойдет, если `c**3` окажется точно равным `z` ?
- Как исправить программу, чтобы корень извлекался и из положительных, и из отрицательных чисел?

In [ ]: