A01. Новогодние подарки

Основной вариант.
Выполнение заданного условия возможно, если только общее (суммарное) количество конфет в подарках кратно трём (делится на три без остатка). Для определения минимального количества перемещаемых конфет можно вычислить минимум из двух чисел: абсолютной величиной (модулем) между ⅓ суммы (количество конфет в меньшей кучке после перемещения) и количеством конфет в первом подарке, абсолютной величиной (модулем) между ⅓ суммы (количество конфет в меньшей кучке после перемещения) и количеством конфет во втором подарке.

```
x1, x2 = map(int, input().split())
if (x1 + x2) % 3:
   print('impossible')
else:
   x = (x1 + x2) // 3
   print(min(abs(x1 - x), abs(x2 - x)))
```

Дополнительный вариант.

```
# решение участника
a, b = map(int, input().split())
if (a + b) % 3 != 0:
   print('impossible')
else:
   print(abs(min(a, b) - (a + b) // 3))
```

A02. До встречи

Основной вариант.

Требуется определить минимальное расстояние между положением участников по окончании бега. Оно выбирается из двух значений: абсолютной величиной (модулем) разности между координатами участников, измеренной по двум направлениям (в прямом и обратном): найденное расстояние, 400 - найденное расстояние. Первоначально определяется положение (координаты) участников по окончании бега с учётом противоположных направлений их бега.

```
n, k = map(int, input().split())
print(min(abs(n % 400 - -k % 400), 400 - abs(n % 400 - -k % 400)))
```

Дополнительный вариант.

Можно определить остаток при делении общего (суммарного) пробега участников на длину круга (400), который будет соответствовать расстоянию между участниками по окончании бега и найти минимум из двух значений (остаток, 400 - остаток).

```
# решение участника
n, k = map(int, input().split())
print(min((n + k) % 400, 400 - ((n + k) % 400)))
```

A03. Между двумя нулями

Основной вариант.

Задача решается с помощью цикла while. Цикл повторяется, пока счётчик введённых нулей меньше 2. Подсчёт суммы производится, когда счётчик введённых нулей равен 1.

```
st = s = 0

while st < 2:
   a = int(input())
   if a == 0:
       st += 1
   else:
       if st == 1:
           s += a

print(s)
```

Дополнительный вариант.

Можно организовать два последовательных цикла while и решать задачу с помощью алгоритма “пока a != 0”, сперва определив место, с которого необходимо искать требуемую сумму.

```
# решение участника
k = int(input())
while k != 0:
   k = int(input())
k = 1
s = 0
while k != 0:
   k = int(input())
   s += k
print(s)
```

A04. Угадайка

Основной вариант.

Задача решается с помощью подсчёта количества каждой из букв в первой строке (счётчик буквы увеличивается на 1) и во второй строке (счётчик буквы уменьшается на 1). Для этого создаётся список счётчиков. Если после подсчёта каждый из счетчиков равен 0, то буквы определить невозможно (в обоих случаях убиралась одна и та же буква). Иначе определяется, у какой буквы счётчик равен 1 (эта буква присутствует в первой, но отсутствует во второй строке, то есть убиралась во второй раз), а у какой буквы счётчик равен -1 (эта буква отсутствует в первой строке, но присутствует во второй строке, то есть убиралась в первый раз).

```
a = input()
b = input()
ans1 = ans2 = ''
cnt = [0] * 91
for el in a:
   cnt[ord(el)] += 1

for el in b:
   cnt[ord(el)] -= 1

for i in range(91):
   if cnt[i] == -1:
       ans1 = chr(i)
   elif cnt[i] == 1:
       ans2 = chr(i)

if ans1:
   print(ans1, ans2)
else:
   print('unknown')
```

Дополнительный вариант (с использованием для подсчёта словаря).

```
a = input()
b = input()
d = {}
for el in a:
   d[el] = d.get(el, 0) + 1
for el in b:
   d[el] = d.get(el, 0) - 1

ans1 = ans2 = ''
for el in d:
   if d[el] == -1:
       ans1 = el
   elif d[el] == 1:
       ans2 = el

if ans1:
   print(ans1, ans2)

else:
   print('unknown')
```

Дополнительный вариант

```
# решение участника
a = input()
b = input()
s = [chr(el) for el in range(ord('A'), ord('Z') + 1)]
pr1 = ''
pr2 = ''
for el in s:
   if a.count(el) > b.count(el):
       pr2 = el
   elif a.count(el) < b.count(el):
       pr1 = el
if pr1 == pr2:
   print('unknown')
else:
   print(pr1, pr2)
```

A05. Между двумя числами (v.4.0)*

Основной вариант.

Можно определить общее количество чисел в промежутке и вычесть из него количество чисел, кратных трём. Для выполнения второй части решения определяется кратное трём число, не большее начала промежутка (с него начинается подсчёт таких чисел), и учитывается кратность трём начала промежутка.

```
x1, x2 = map(int, input().split())
print(x2 - x1 + 1 - (x2 - (x1 - x1 % 3)) // 3 - (x1 % 3 == 0))
```

Дополнительный вариант.

```
# решение участника
x1, x2 = map(int, input().split())
print(x2 - x1 + 1 - x2 // 3 + (x1 - 1) // 3)
```

Дополнительный вариант (решение в одну строку).

```
# решение участника
print((lambda a, b: b-a+1 - (b // 3 - (a-1) // 3))(*map(int, input().split())))
```

A06. Думай как опытный программист*

Основной вариант.

Требовалось понять, что указанная операция соответствует сложению чисел в двоичном (бинарном) коде. Решение сводится к применению функций bin() и int() (во втором случае с именованным параметром base, равным 2).

```
# решение участника
print(bin(sum([int(el, 2) for el in input().split()]))[2::])
```

Дополнительный вариант.

```
# решение участника
a, b = input().split()
print(bin(int(a, 2) + int(b, 2))[2:])
```

Дополнительный вариант.

```
# решение участника
s1, s2 = input().split()
x1 = x2 = 0
for i in range(len(s1)):
   x1 *= 2
   x1 += int(s1[i])
for i in range(len(s2)):
   x2 *= 2
   x2 += int(s2[i])
print(bin(x1 + x2)[2:])
```

Дополнительный вариант (решение в одну строку).

```
print((lambda x: bin(int(x[0], 2) + int(x[1], 2))[2:])(input().split()))
```

A07. Квадрат или не квадрат?

Основной вариант.

Определяется максимум из представленных размеров - сторона квадрата (max_). Далее определяется граничный прямоугольник, одна из сторон которого совпадает с max_. С помощью сторон этого граничного прямоугольника определяется размер остальной (составленной из двух оставшихся прямоугольников) части квадрата (у этой части квадрата один размер max_, второй размер x1 = max_ - вторая сторона граничного прямоугольника). У двух других прямоугольников определяется общая сторона. Если их общая сторона совпадает с max_, то имеет место случай разделения квадрата двумя параллельными линиями, в этом случае проверяется равенство суммы других сторон значению x1. Если такого совпадения нет, то их общая сторона совпадает с x1, в этом случае проверяется равенство суммы других сторон значению max_.

```
a1, a2, b1, b2, c1, c2 = map(int, input().split())
max_ = max(a1, a2, b1, b2, c1, c2)
if max_ ** 2 != a1 * a2 + b1 * b2 + c1 * c2:
   print('no')
else:
   if max_ in [a1, a2]:
       x1 = 2 * max_ - (a1 + a2)
       tmp = [b1, b2, c1, c2]
   elif max_ in [b1, b2]:
       x1 = 2 * max_ - (b1 + b2)
       tmp = [a1, a2, c1, c2]
   else:
       x1 = 2 * max_  - (c1 + c2)
       tmp = [a1, a2, b1, b2]

   x2 = set(tmp[:2]) & set(tmp[2:])
   tmp2 = -1
   if max_ in x2:
       tmp2 = sum(tmp) - 2 * max_ - x1
   elif x1 in x2:
       tmp2 = sum(tmp) - 2 * x1 - max_
   if tmp2 == 0:
       print('yes')
   else:
       print('no')
```

Дополнительный вариант (с предварительной сортировкой прямоугольников по их размерам).

```
# решение участника
def sort(el_mas):
   return -el_mas[0], -el_mas[1]


a1, b1, a2, b2, a3, b3 = map(int, input().split())
mas = [[max(a1, b1), min(a1, b1)], [max(a2, b2), min(a2, b2)], [max(a3, b3), min(a3, b3)]]
mas.sort(key=sort)
if mas[0][0] == mas[1][0] == mas[2][0] and mas[0][0] == mas[0][1] + mas[1][1] + mas[2][1]:
   print('yes')
elif mas[1][0] == mas[2][0] == mas[0][0] - mas[0][1] and mas[1][1] + mas[2][1] == mas[0][0]:
   print('yes')
elif mas[1][1] == mas[2][0] == mas[0][0] - mas[0][1] and mas[1][0] + mas[2][1] == mas[0][0]:
   print('yes')
elif mas[1][1] == mas[2][1] == mas[0][0] - mas[0][1] and mas[1][0] + mas[2][0] == mas[0][0]:
   print('yes')
else:
   print('no')
```

A08. Правда ли?

Основной вариант.

Наиболее целесообразным является использование дополнительного множества, составленного из удвоенных значений входных данных. Пересечение данного множества и множества, составленного из входных данных, свидетельствует о наличии пары чисел, соответствующих условиям задачи.

```
n, *a = map(int, input().split())
a = set(a)
b = {el * 2 for el in a}
if a & b:
   print('yes')
else:
   print('no')
```

Дополнительный вариант (с использованием одного множества, составленного из входных данных, и проверкой наличия в этом множестве элемента, равного удвоенному значению одного из элементов).

```
# решение участника
n, *lst = [int(el) for el in input().split()]
lst = set(lst)
for el in lst:
   if el * 2 in lst:
       print('yes')
       break
else:
   print('no')
```

A09. Приоритет операций

Основной вариант.

Для решения первоначально требуется дополнить каждую из операций её порядковым номером и отсортировать полученную структуру в порядке невозрастания (убывания) в зависимости от приоритета операции.

```
def f(x):
   return D[x[0]]


D = {'**':0, '*':1, '/':1, '//':1, '%':1, '+':2, '-':2}

n = int(input())
a = input().split()
a = [(a[i], i + 1) for i in range(n)]

a.sort(key=f)

print(*(el[1] for el in a))
```

Дополнительный вариант (с использованием дополнительных списков для каждого из приоритетов и фактической сортировки подсчётом).

```
n = int(input())
a = input().split()
s0 = []
s1 = []
s2 = []

for i in range(len(a)):
   if a[i] == '**':
       s0.append(i + 1)
   elif a[i] in ('*', '/', '//', '%'):
       s1.append(i + 1)
   else:
       s2.append(i + 1)

print(*(s0 + s1 + s2))
```

Дополнительный вариант (без сортировки с использованием внешнего цикла, задающего приоритет операции от большего к меньшему, и определения принадлежности приоритета текущей операции задаваемому внешним циклом).

```
# решение участника
n = int(input())
lst = input().split()
d = {1: {'**'}, 2: {'*', '/', '//', '%'}, 3: {'+', '-'}}
k = 1
for key in range(1, 4):
   for i in range(n):
       if lst[i] in d[key]:
           print(i+1, end=' ')
```

A10. Арбузы

Основной вариант.

Возможно решение с помощью стека, при котором (после сортировки) проверяется разность между проверяемым элементом и элементом в вершине стека с последующим либо удалением элемента из стека если разность меньше или равна 1), либо добавлением к стеку (в противном случае).

```
n, *a = map(int, input().split())
a.sort()
res = 0
st = []
for el in a:
   if st and el - st[-1] <= 1:
       res += 1
       st.pop()
   else:
       st.append(el)

print(res + len(st))
```

Дополнительный вариант (с обходом отсортированного списка с помощью цикла while).

```
# решение участника
n, *ar = [int(el) for el in input().split()]
ar.sort()
i = 0
s = 0
while i < n:
   if i < n - 1 and ar[i + 1] - ar[i] <= 1:
       i += 1
   i += 1
   s += 1
print(s)
```

A11. Ёлочка (v.2.0)*

Основной вариант.

Первоначально требовалось определить зависимость между количеством рядов и количеством звёздочек (второе есть квадрат первого, для ответа необходимо найти квадратный корень из количества звёздочек). Особенностью задачи является большой диапазон входных данных, из-за чего нахождение ответа с помощью стандартного возведения в степень 0.5 или применения функции sqrt() модуля math приводит к округлению ответа в большую сторону при больших значениях аргумента . Задачу можно решить с использованием функции isqrt() модуля math (возвращает целочисленный квадратный корень аргумента, округлённый вниз).

```
# решение участника
import math
print(int(math.isqrt(int(input()))))
```

Дополнительный вариант (с использованием правостороннего двоичного (бинарного) поиска).

```
n = int(input())
le = 0
ri = n + 1
while le < ri - 1:
   mi = (le + ri) // 2
   if mi ** 2 <= n:
       le = mi
   else:
       ri = mi

print(le)
```

Дополнительный вариант (проверка найденного квадратного корня на округление в большую сторону с последующей корректировкой ответа).

```
n = int(input())
x = int((n ** 0.5))
if x ** 2 > n:
   print (x - 1)
else: print(x)
```

Дополнительный вариант (решение в одну строку).

```
print((lambda n: int((n ** 0.5)) - 1 if int((n ** 0.5)) ** 2 > n else int((n ** 0.5)))(int(input())))
```

A12. Домашний квас (v.2.0)

Основной вариант.

Особенностью задачи является использование для решения исключительно целых чисел (используя в качестве единицы измерения грамма), поскольку использование вещественных чисел при заданном диапазоне входных данных даст погрешность вычислений. Задача решается с помощью правостороннего бинарного (двоичного) поиска по ответу с округлением количества банок/пакетов/упаковок в большую сторону при их определении с помощью целочисленного деления (если требуется хотя бы единица ингредиента, то необходимо брать целую банку/пакет/упаковку).

```
s, p1, p2, p3 = map(int, input().split())
l = -1
r = 10 ** 18 + 1
while l < r - 1:
   mi = (l + r) // 2
   if ((mi * 26 - 1) // 650 + 1) * p1 + ((mi * 667 - 1) // 10000 + 1) * p2 + ((mi * 12 - 1) // 500 + 1) * p3 > s:
       r = mi
   else:
       l = mi

print(l)
```

Дополнительный вариант (с использованием цикла for вместо цикла while).

```
# решение участника
l = 0
r = 10 ** 20
s, p1, p2, p3 = map(int, input().split())
for i in range(20000):
   m = (r + l) // 2
   s1 = m * 26 // 650
   if m * 26 % 650 > 0:
       s1 += 1
   s2 = m * 667 // 10000
   if m * 667 % 10000 > 0:
       s2 += 1
   s3 = m * 12 // 500
   if m * 12 % 500 > 0:
       s3 += 1
   if p1 * s1 + p2 * s2 + p3 * s3 > s:
       r = m
   else:
       l = m
print(l)
```

A13. Максимальный перепад температур

Основной вариант.

Задача решается с помощью нахождения минимума/максимума в скользящем окне с использованием двухсторонней очереди (дека) с последующим определением максимума разности и позиции окна с максимальной разностью.

```
from collections import deque

n, k, *a = map(int, input().split())
st_min = deque()
st_max = deque()

res_min = []
res_max = []


for i in range(n):
   while st_min and a[i] <= a[st_min[-1]]:
       st_min.pop()
   st_min.append(i)

   if i - k == st_min[0]:
       st_min.popleft()

   if i >= k - 1:
       res_min.append(st_min[0])

   while st_max and a[i] >= a[st_max[-1]]:
       st_max.pop()
   st_max.append(i)

   if i - k == st_max[0]:
       st_max.popleft()

   if i >= k - 1:
       res_max.append(st_max[0])

ans = 0
for i in range(len(res_min)):
   if a[res_max[i]] - a[res_min[i]] > a[res_max[ans]] - a[res_min[ans]]:
       ans = i

print(ans + 1, a[res_max[ans]] - a[res_min[ans]])
```

A14. Числа на «К»

Основной вариант.

В задаче речь идет о нахождении чисел Каталана любым способом, в частности, с помощью динамического программирования.

```
n = int(input())
dp = [1] * (n + 1)
for i in range(2, n + 1):
   dp[i] = sum(dp[j] * dp[i - 1 - j] for j in range(i))

print(dp[-1])
```

Дополнительный вариант (с использованием комбинаторных методов).

```
import math

n = int(input())
if n == 0:
   print(1)
else:
   print(math.comb(2 * n, n) - math.comb(2 * n, n - 1))
```

Дополнительный вариант.

```
# решение участника
import math

n = int(input())
print(math.factorial(2 * n) // math.factorial(n + 1) //  math.factorial(n))
```

A15. Легендарная матрица (v.22.0)

Основной вариант.

Для поиска ответа требуется найти максимум из следующих значений: сумма двух наибольших значений сумм строк (строка + строка), сумма двух наибольших значений сумм столбцов (столбец + столбец), сумма по каждой строке и каждому столбцу (строка + столбец).

```
n, k = map(int, input().split())
m = [[int(el) for el in input().split()] for i in range(n)]

sp = [sum(el) for el in m]

sc = [sum(el[j] for el in m) for j in range(k)]

res = [sum(sorted(sp)[-2:])] + [sum(sorted(sc)[-2:])]

for i in range(n):
   for j in range(k):
       res.append(sp[i] + sc[j] - m[i][j])

print(max(res))
```

Дополнительный вариант.

```
# решение участника
n, k = map(int,input().split())
mas = []
s1 = [0] * n
s2 = [0] * k
for i in range(n):
   a1 = [int(el) for el in input().split()]
   mas.append(a1)
   s1[i] = sum(a1)
   for j in range(k):
       s2[j] += a1[j]
s3 = []
for i in range(n):
   for j in range(k):
       s3.append(s1[i] + s2[j] - mas[i][j])
s1.sort()
s2.sort()
print(max(max(s3), s1[-1], s2[-1], s1[-1] + s1[-2], s2[-1] + s2[-2]))
```