Вспоминаем функции.

Используя функции, довольно удобно записывать алгоритмы, результат которых `True` или `False`. Ведь в функциях есть волшебный оператор `return`!

Главное здесь — разобраться в том, какая проверка позволяет **сразу** вернуть логическое значение.

Например, проверка того, что последовательность чисел является арифметической прогрессией. Можно запомнить `delta` — разность первых двух элементов. Если затем сравнивать `delta` с остальными разностями, то равенство (каждое отдельное равенство) нам ничего не скажет. Зато неравенство позволяет сразу понять, что арифметической прогрессией тут не пахнет. И можно вернуть результат.

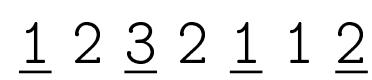
Почти все сдали задачу. Так что, наверное, каждому ясно, что надо написать вместо многоточия в следующей после цикла `for` строке.

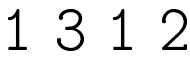
```
1  def ArithmProgression(x):
2      delta = x[1] - x[0]
3      for k in range(2, len(x)):
4          ...
5              return False, 0
6      return True, delta
7
8
9  N = int(input())
10 x = list(map(int, input().split()))
11 isProgression, delta = ArithmProgression(x)
12 if isProgression:
13     print(delta)
14 else:
15     print('NO')
```

Заодно обратите внимание, тут функция возвращает **два** значения. Это сделано исключительно ради того, чтобы избежать ситуации, когда функция возвращает в зависимости от обстоятельств **разные типы** значений. Здесь это были бы `False` (если `x` не прогрессия) и `delta` (если `x` прогрессия).

Посмотрите, как вызывается эта функция и как сохраняется результат её работы: в две переменные. Первая всегда логическая (bool), а вторая целое число (int).

Теперь не такая популярная задача — задача U про подпоследовательности. Вот картинка:





Надо определить — встречается ли в второй массив в первом как подпоследовательность.

Понятно (см. заголовок), что надо делать функцию. Второй массив надо перебрать весь, слева направо. Значит используем самый простой цикл:
`for elem in y`

Для каждого такого элемента надо найти его в массиве `x`. Если нашли, то со следующего индекса надо продолжать искать, а если не нашли (как проверить?), у нас есть волшебный `return`!
Возвращаем `False`, это уже не подпоследовательность.

```
1  def isSubsequence(x, y):
2      idx = 0
3      for elem in y:
4          while ... and ...:
5              idx += 1
6          if ...:
7              return False
8          idx += 1       # move right after match
9      return True
10
11 N = int(input())
12 x = list(map(int, input().split()))
13 M = int(input())
14 y = list(map(int, input().split()))
15 print('YES' if isSubsequence(x, y) else 'NO')
```

В обоих примерах функцию завершает `return` безо всяких условий. Сам факт того, что мы находимся в этой строке (номер 9) означает, что мы не вышли из функции ранее. А значит **все** `elem`енты из массива `y` были найдены и образуют подпоследовательность, т.к. продолжение мы искали правее ранее найденного (`idx` только увеличивается).

Неужели никто не придумал, что делать с задачей про строительство школы?

Можно для начала подумать, как быть, если домов 2, 3, 4, 5 (выберите **любые** координаты). Кажется, потом должно осенить. Если придумали, другим не рассказывайте, не лишайте удовольствия.