Я продолжая упорно отказываться от идеи брать код из видео мужика, потому что, во-первых, он слишком понтовский, а во-вторых, слишком сложный. Многое из этого я могу реализовать только более простыми путями. И мне кажется, что так лучше.

Перейдём ближе к теме.

Всё ещё мучаясь в поисках нормального пути решения головоломки с клетками, я решил объединить все клетки в класс клетка, а каждая клетка будет функцией, которую мы будем просить предоставить нам информацию о том, какие фигуры на ней стоят.

То есть на выходе функция будет выдавать нам массив, состоящий из двух элементов.

def A2():

A2 = ['A', '2', 'WHITE', PAWN, '♙']

color = A2[2]

piece = A2[3]

Cell\_info = [color, piece]

return Cell\_info

На вход чуть позже, когда понадобится делать ходы, мы будем запрашивать изменения. И производить замену.

Условно, например, пешку на А2 забрали ладьёй. Мы запрашиваем изменения, которые тоже будут приходить в клетку доски в виде массива, состоящего из всё тех же двух элементов, то есть цвет и фигура.

def A2 (changes):

A2 = ['A', '2', 'WHITE', 'PAWN']

if (changes == []):

Cell\_info = [A2[2], A2[3]]

return Cell\_info

else:

del A2[2]

del A2[2]

A2.append(changes[0])

A2.append(changes[1])

Cell\_info = [A2[2], A2[3]]

return A2

changes = ['BLACK', 'ROOK']

print (A2(changes))

А вот так выглядит программа клетки с учётом того, что будут поступать какие-то изменения.

То есть мы задаём глобально для функции массив с информацией о клетке и если массив изменений для этой клетки пуст, то ничего не происходит и она возвращает два значения. А если изменения всё же есть, функция последовательно удаляет два элемента и встраивает новые элементы.

Немного доработаем эту программу.

def A2 (changes):

A2 = ['A', '2', 'WHITE', 'PAWN', '♙']

if (changes == []):

Cell\_info = [A2[2], A2[3], A2[4]]

return Cell\_info

else:

del A2[2:]

#del name[x:] удаляет все элементы, начиная с х

A2.append(changes[0])

A2.append(changes[1])

A2.append(changes[2])

Cell\_info = [A2[2], A2[3], A2[4]]

return Cell\_info

changes = ['BLACK', 'ROOK', '♜']

print (A2(changes))

Теперь хорошо бы ещё добавление элементов массива сделать в одно действие. Но, видимо, такой функции нет. Ничего страшного.

Теперь надо понять, как мы будем обращаться к этим клеткам. Видимо посредством названия их.

То есть мы с помощью перекодировки строк будем прибавлять значения, потом суммой строк складывать необходимое обращение и из него получать непосредственно само обращение к клетке.

Далее. Можно назвать массив, который будет содержать всю информацию о клетке доски, одинаково везде. Чтобы не мучаться с названиями. Ведь эти массивы и так содержат информацию о том, какие координаты они имеют. С другой стороны, тогда придётся разбираться с ними. Хотя, я наверное, за то, чтобы было одно название. Пусть это будет CL, а нужная нам информация Cell\_info.

В перспективе, когда я лучше освою классы, необходимо вынести общее в один большой класс, и от этого класса строить все клетки. Потому что большая часть кода абсолютная одинакова, и отличается только разными по содержанию массивами.

Хотя, давайте прямо сейчас это и сделаем. Попытаемся сделать.

Также необходимо изучить двумерные массивы в Питоне.

class Cell(object):

def \_\_init\_\_(self, changes):

self.changes = changes

def self (changes):

if (changes == []):

Cell\_info = [CL[2], CL[3], CL[4]]

else:

del CL[2:]

CL.append(changes[0])

CL.append(changes[1])

CL.append(changes[2])

Cell\_info = [CL[2], A2[3], A2[4]]

return Cell\_info

class A1(Cell):

CL = ['A', '1', 'WHITE', 'PAWN', '♙']

changes = []

print(A1(changes))

Вроде бы, даже работает. Но он не выводит то, что должен выводить. Он будто выводит функцию, а не её значение.

Возможно, здесь нужен \_\_repr\_\_? Или str()?