Что сделано:

1. Инициализирован: class Angles, class Polygon, class DB (DataBase), class Point
   1. Angles для хранения углов, для операций над углами (перевод из DD в DMS и обратно, контроль углов, которые измерялись на пунктах) и между ними, складывание, вычитание
   2. Polygon для хранения всех измерений, полученных на пунктах полигона. Внутри выполняются вычисления теории, исправленные значения и получение всей таблицы координат
   3. DB для чтения исходных данных с локального компа (заранее подготовленные данные), либо данных, которые будут передаваться через API, либо документом, либо через скрапинг указанных данных на вебе.
      1. В этот класс должен попадать JSON уже. Если что-то другое будет, надо будет проверять, что получает экземпляр класса и на основании проверки принимать решение, как читать, что проскрапить
   4. Point хранит в себе все данные, которые находятся на уровень ниже в нормальной человеческой таблице вычисления координат, пока так:
      1. ID Для различия пунктов, для потенциальной сортировки по этому атрибуту
      2. Угол направление (от предыдущей точки на следующую)
      3. Дистанцию до следующей точки
      4. Возможно Дир. Угол. Т.к. это не мастхев поле может его назначать просто через сеттер???
   5. BearingAngle:
      1. Переопределил несколько методов. Класс для объектов дир. Углов, главное, что б был пересчёт, если больше 360 или меньше нуля.
      2. Обратить внимание на раздел «ВОЗМОЖНО»

Процедура вычисления таблицы координат

1. Указать измеренные горизонтальные углы и горизонтальное проложение на пунктах
2. Указать исходные координаты (координаты первого «твёрдого» пункта и последнего)
3. Указать какой дирекционный угол между ними
4. Посчитать практическую и теоретическую сумму углов
5. Посчитать невязку горизонтальных углов
   1. Посчитать теоретическую невязку горизонтальных углов (Зависит от выбранного прибора, по-моему, двойная точность прибора или что-то такое)
      1. Добавить условие на проверки полученной невязки с теорией, если выполняется, значит делать – иначе вернуть какую-нибудь фигню

Вызов функции изменит углы в fixed\_angles, если невязка получилась допустимой. Если она получится недопустимой... Вся прога рухнет RAISE, надо подумать, как обойти это, и как по-другому передавать эту ошибку

1. Посчитать невязку, раскидать поправки в углы ✔️
2. Сверить исправленные горизонтальные углы с теорией ✔️
   1. Если исправленные углы получились неправильно, то стоит вызывать raise или нет? И вопрос с многострочным raise. Выглядит не очень, как поправить?
   2. И понять, нужно ли в таких случаях вызывать raise? Что тогда на фронт в таком случае будет передаваться?
   3. [!]Типа есть вариант выполнять все расчёты, но писать все ошибки в отдельную переменную и если эти ошибки будут, то передать на фронт, что б там большими буквами все ошибки отобразились и уведомили, что вычисления неправильные…
3. Передать дирекционный угол на последующие линии✔️
   1. Проверить на то, что б последний дир. Угол в списке равнялся исходному, иначе беда.
4. Посчитать приращение координат✔️
   1. У меня есть список углов на точках, список дир. Углов с этих точек на последующие пункты.
   2. Но при расчёте конечных координат мне надо проходиться циклом по предпоследний элемент, наверное. Или проходиться по последний элемент, а потом так же делать проверку на исходные координаты и на вычисленные.
5. Посчитать все координаты✔️
   1. Сравнить с исходниками.

ВСЁ СДЕЛАНО, НО ЕСТЬ ВОПРОСЫ В ТОМ ПЛАНЕ, ЧТО НЕ СХОДИТСЯ С ТЕМ, ЧТО Я СЧИТАЛ ДО ЭТОГО. У МЕНЯ 6Й

Возврат вычисленных значений.

Попытка через класс Point

Мне надо будет пройтись по всем точкам кроме последней и собрать прям все все данные, т.к. в последней точке у меня не будет вычисления приращений и вычисления координат следующей точки, т.к. это исходная и это не требуется, если дир. Угол сошёлся, то всё норм.

В итоге получается исправляли углы мы как для замкнутого полигона, а всё остальное делали как для разомкнутого, т.к. теоретическая невязка не равна нулю, а равна разнице между исходными пунктами. Мы так решали ведомость на 4м курсе О\_о

А если будет равна нулю, т.е. полигон будет около нормальным, где внутри полигона будет находиться только один пункт, то возможно придётся новые расчёты делать. Этот момент надо будет с нуля изучать, смотреть, как Женя решала и она мне кидала ещё методичку.

Попытка через класс не удалась, т.к. не придумал, как обработать последнюю точку в которой не все данные будут находиться.

# Надо бы

Написать класс для взаимодействия с АПИ

Там должно быть что-то типа как класс DB, методы на получение и на отдачу данных, их буду использовать внутри класса Полигон, также как с БД

Либо просто модифицировать модуль БД

Класс Angle1:

Добавить методы для того, чтобы можно было отнимать и прибавлять углы, например. Прибавление есть ОТНИМАНИЕ НАДО БЫ, что б не писать + -

Переписать ситуацию с тем, что у меня в словарике хранится весь список углов, просто сделать Айдишники внутри класса Угла, и сортировать на основании этого атрибута. НАДО БУДЕТ ПРОТЕСТИТЬ. Т.к. надо будет сортировать сначала по величине угла, а потом по величине айдишника.

«Т.е. я сначала исправляю, используя только DD, потом при исправлении я делаю пересчёт DMS и потом при проверке я проверяю через новый DD (надо это через сеттер автоматизировать, если изменяю что-то в угле, то надо делать пересчёт DMS и после сразу пересчёт DD, что б при сравнениях/вычислениях не использовать метод экземпляра, а использовать атрибут DD у экземпляра.»

Придумать то, чтобы при получении суммы углов, например с исходными данными №3 не было такого, что принтуется угол 2339°59'60", хотя угол 2340 в DD такой пишется. Как-то накидать проперти для гет и сета, и чтобы шла проверка, если больше какой-то величины получатся – то надо закинуть в следующий разряд.

Призадуматься, для получения суммы периметра нормально использовать этот же класс angle1? Либо надо наследование какое-то делать, сначала угол, а потом класс угол+длина линии, чтобы в первом классе можно было считать только величину угла, и она может быть бесконечна. НО при использовании класса угол+длина линии, там должно быть ограничение в 360 градусов, иначе надо делать – или + 360 градусов.

# ВОЗМОЖНО

Класс Polygon

должен представлять из себя связанный список (одно или двусвязный, без разницы

Что б точно не проебаться в том плане, какой угол за каким следует, что б каждый пункт знал, какой за ним или до него следует. Нужно ли?

Что б случайно при какой-нибудь сортировке не напутать.

Надо сделать отдельный класс для хранения всех объектов, которые находятся в строке…. Иначе у меня есть класс angle, я туда добавил длину стороны… Но разве туда надо пихать дирекционный угол, приращения координат, координаты?

Надо сделать отдельный класс поинт… Возможно это будет связный список и будет всё зоебись

Поменять логику классов Angle and Bearing Angle:

В ЦЕЛОМ УГОЛ НЕ МОЖЕТ БЫТЬ БОЛЬШЕ 360 градусов, лучше все проверки из BearingAngle прокинуть в Angle, а для периметральных вычислений углов, где суммы углов в полигоне, просто сделать отдельный класс PerimAngle, где в \_\_init\_\_ допустить, что б DD был больше 360 градусов , просто надо вспомнить наследование, мне надо переписать инит, но чуть-чуть, ну или полностью. Тогда метод \_\_add\_\_ надо будет тоже поменять местами.

Сделать отдельный класс для проверки, Валидации всех переданных данных. Сериализация или что-то типа такого. Это где-то перед запуском DB and Polygon

# На тестирование или на подумать

# Подумать:

Над тем, в какой момент и в каком месте производится чтение исходных данных? Надо, наверное, не в полигоне это делать, а вне его инициализации создавать объект DB, сохранять в переменную полученный результат, потом прокидывать в полигон

Придумать метод в DB, через который отдавать на фронт. Типа у меня будет объект полигона, я просто в метод прокидываю, а в методе просто формируется JSON и возвращается и всё.

Также тут должны будут быть методы на то, что б отдавать на фронт список исходных данных с локального компьютера, типа нейминг (количество их) что б можно было сформировать выпадашку на фронте и получать для тестирования.