**zip\_app.py и util.py**

**Структура**

1. Можно разбить на несколько **модулей**: модуль (файл), который запускает программу, и вспомогательные модули (файлы), и реализовать модель MVC.

**Содержание**

1. Мало **комментариев**. Они написаны только к файлам zip\_app.py и util.py в целом и одной из восьми функций файла zip\_app.py.
2. В комментарии к программе можно описать пункты меню. Исходя из примера выполнения программы не всегда понятно, что именно будет выполнено. Например, нет комментария к команде «**end»** и выводу **«Done**». «Done» можно написать, в том числе, и после проведения расчетов.
3. **Пункты** **меню**: Command ('loc', 'zip') с точки зрения пользователя **двусмысленны**. Местоположение и индекс будут вводить или искать?
4. **Пункты** **меню** лучше сделать **цифрами**, поскольку при вводе даже коротких слов шанс допустить ошибку выше.
5. **Вывод строк**. Можно использовать файл с инструкциями, написанными на разных языках, и подставлять их в вывод в зависимости от выбранного языка.
6. **Волшебные числа:**
7. def calculate\_distance(location1, location2): 3959.191 – радиус земли в милях.
8. def degree\_minutes\_seconds(location):

60 – минуты в градусах;

60 – секунды в минутах.

1. def zip\_by\_location(codes, location):

…

if location[0].lower() == code[3].lower() and \

location[1].lower() == code[4].lower():

zips.append(code[0])

…

Привязка к конкретной БД.

1. def location\_by\_zip(codes, zipcode):

for code in codes:

if code[0] == zipcode:

return tuple(code[1:])

return ()

Привязка к конкретной БД.

1. def process\_loc(codes):

…

print('ZIP Code {} is in {}, {}, {} county,\ncoordinates: {}'.

format(zipcode, location[2], location[3], location[4],

format\_location((location[0], location[1]))))

…

Привязка к конкретной БД.

Для всех волшебных чисел лучше создать переменные в начале функции, которым будет присвоено соответствующее значение. Если оно изменится, то в коде его нужно будет переписать только один раз.

1. **def calculate\_distance**(location1, location2):

…

del\_lat = (lat1 - lat2) / 2

del\_long = (long1 - long2) / 2

…

Повторяющийся код. Можно написать отдельную функцию и передать ей в качестве аргументов lat1, lat2 и long1, long2 соответственно.

1. **def degree\_minutes\_seconds(location):**

seconds = 60 \* seconds

Сокращенная запись: seconds \*= 60

Запись без волшебного числа:

seconds\_in\_minute = 60

seconds \*= seconds\_in\_minute

1. **def format\_location(location):**
2. …

ns = ""

if location[0] < 0:

ns = 'S'

elif location[0] > 0:

ns = 'N'

ew = ""

if location[1] < 0:

ew = 'W'

elif location[0] > 0

ew = 'E'

…

1. В США всегда 'N' и 'W'
2. Если программа универсальная и этот код необходим, то его лучше вынести в **отдельную функцию**, определяющую стороны света. (Каждая функция должна отвечать за одну конкретную задачу).
3. А если значения равны нулю? На экваторе расположено, по меньшей мере, 6 городов, и почтовых отделений с 0.0 в.д. достаточно много.
4. Названия переменных latdegree, latmin, latsecs, longdegree, longmin, longsecs лучше написать через подчеркивание.
5. **def location\_by\_zip(codes, zipcode):**

Добавить вызов функции валидации введенного индекса

1. Написать функцию валидации введенного индекса (в комментариях упомянута длина индекса).

Длина указывается в начале функции, следовательно, при использовании другой БД ее будет легко найти и поменять.

У пользователя будет возможность исправить ошибку сразу.

Также можно добавить функцию валидации вводимого пользователем названия штата (оно должно состоять из двух букв)

1. **def process\_loc(codes):**

…

zipcode = input('Enter a ZIP Code to lookup => ')

…

Можно вызвать методы strip() или replace(“ ”, “”)

1. **def process\_zip(codes):**

…

city = input('Enter a city name to lookup => ')

print(city)

city = city.strip().title()

state = input('Enter the state name to lookup => ')

print(state)

state = state.strip().upper()

…

Можно написать в одну строку:

city = input('Enter a city name to lookup => ').strip().title()

state = input('Enter the state name to lookup => ') .strip().upper()

1. **def process\_dist(codes):**
2. …

location1 = location\_by\_zip(codes, zip1) #returns tuple(code[1:])

location2 = location\_by\_zip(codes, zip2) #returns tuple(code[1:])

…

dist = calculate\_distance(location1, location2)

…

Функции calculate\_distance() передаётся кортеж со всеми данными, которые можно найти по введенному индексу, следовательно, она становится зависимой от БД. Индексы ее переменных location1 и location2, [0] и [1], должны совпадать с индексами соответствующих столбцов БД.

Возможное решение:

dist = calculate\_distance((location1[lat\_column\_index],

location1[long\_column\_index]), (location2[lat\_column\_index], location2[long\_column\_index]))

Функции в качестве аргументов передаются кортежи, содержащие только два значения: широты и долготы.

1. zip1 = input('Enter the first ZIP Code => ')

zip2 = input('Enter the second ZIP Code => ')

Можно вызвать методы strip() или replace(“ ”, “”)

+ учитывая, что мы добавили функцию валидации введенного индекса, вызовы функций location1 = location\_by\_zip(codes, zip1) и location2 = location\_by\_zip(codes, zip2) проверят введенные значения и в случае ошибки позволят пользователю ввести их еще раз

1. **utils.py**

**def read\_zip\_all():**

1. **Названия переменных:**

i, m

1. if header[idx] == "**latitude**" or header[idx] == "**longitude**" – привязка к конкретной БД
2. val = float(m[idx]) – конвертация сработает, только если значение записано через точку. В русской культуре принято использовать запятую. Возможно, стоит вызвать метод replace(“,”, “.”)

**Ошибки**

1. **zip\_app.py:**

Комментарий:

This is a program to find location by ZIP code and ZIP code by location,

as well as to calculate the distance between **to** ZIP codes. #two

1. **def format\_location(location):**

…

ew = ""

if location[1] < 0:

ew = 'W'

elif location[0] > 0 #[0]?

ew = 'E'

…

Опечатка? Значение долготы содержится под индексом [1].

**Наблюдения**

1. **def degree\_minutes\_seconds(location):**

может быть локальной функцией для def format\_location(location).