Министерство науки и высшего образования Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ
ГИДРОМЕТЕОРОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ

В.Ю. Третьяков

ГЕОИНФОРМАЦИОННЫЕ СИСТЕМЫ В ЭКОЛОГИИ И ПРИРОДОПОЛЬЗОВАНИИ: ПРОГРАММИРОВАНИЕ НА PYTHON В ARCGIS И QUANTUM GIS

Санкт-Петербург
РГГМУ
2022

УДК [004.43:528.4](075.8)
ББК 32.973.2:26.191я73

Т66

Рецензенты:
В.В. Дмитриев, доктор геогр. наук, профессор Санкт-Петербургского государственного университета,
А.В. Юлин, кандидат геогр. наук, доцент, зав. лаб. ледового режима и долгосрочных прогнозов Северного ледовитого океана ФГБУ Арктический и антарктический научно-исследовательский институт

Третьяков, Виктор Юрьевич.

Т66 Геоинформационные системы в экологии и природопользовании : программирование на Python в ArcGIS и Quantum GIS. – Санкт-Петербург : РГГМУ, 2022. – 112 с.

ISBN 978-5-86813-557-6

Рассмотрены возможности автоматизации обработки пространственно-координированной информации в ГИС ArcGIS и Quantum GIS с помощью компьютерных программ на языке Python. Приведены примеры ряда программ для выполнения типовых задач обработки и анализа с объяснением возможностей их трансформации для других целей.

УДК [004.43:528.4](075.8)
ББК 32.973.2:26.191я73

ISBN 978-5-86813-557-6

СОДЕРЖАНИЕ

Введение

Нам, пришедшим в науки о Земле с мечтой о романтике дальних странствий, нужно понимать, что сущность исследований прошлого уже никогда не вернётся. Теперь исследования не заключаются в открытиях новых географических объектов. Хотя они изредка случаются. Автору летом 2013 года посчастливилось обнаружить у берегов Новой Земли необозначенный на карте островок. Однако капитан ледокола на предложение объявить на весь мир об этом открытии только замахал руками: «И думать забудьте! Вы не представляете, сколько бумаг придётся оформлять!» Разумеется, в сфере геологии актуальным остаётся открытие месторождений полезных ископаемых, но здесь необходимо по возможности наиболее точное определение запасов. Теперь научно-исследовательская работа не сводится к проведению экспедиционных исследований и описанию их результатов при всей важности исследований данного типа. В экологии и природопользовании необходимо получение информации третьего рода: рекомендаций для лиц, принимающих решения в этих сферах. Решения нужно принимать часто и быстро. Всё большую долю первичной информации, которая требуется для этого специалистам в сфере экологии и природопользования, поставляют спутниковые системы и автоматизированные системы мониторинга. Обработка гигантских объёмов поступающей информации возможна только с помощью компьютерных программ. Пользователь – специалист в сфере экологии и природопользования может создавать свои компьютерные программы на алгоритмическом языке «Python».

Если в любом поисковике набрать слово «Python», то появится множество ссылок на сайты, наперебой предлагающие пройти онлайн-курсы обучения программированию на этом языке. Вам будет обещаны высокие заработки и быстрый карьерный рост, если Вы пройдёте эти курсы, разумеется, не бесплатно. Однако эти курсы в основном обучают азам программирования для решения следующих задач: автоматизации анализа социальных сетей, «выуживания» из них информации, и её скачивания путём маскировки под браузер; построения нейронных сетей. В данном учебном пособии рассматривается совершенно другой аспект программирования на языке Python: автоматизация обработки информации в среде геоинформационных систем (ГИС). Разумеется, эта информация должна быть пространственно-координированной, т.е. иметь географическую привязку. В противном случае просто нет смысла затевать автоматизацию обработки данных в среде ГИС: будет проще

её выполнить с помощью таких инструментов, как Jupyter Notebook, Pandas и т.д. Естественно, что автоматизация обработки имеет смысл только в том случае, когда данных много, и они организованы в файлы и файловые структуры, в названиях которых присутствует закономерность. Чаще всего это включение в имя файла даты и времени, которые указывают, к какому времени или временному интервалу относится информация.

Python является языком интерпретирующего типа. Это означает, что каждый оператор (строка) программного кода преобразуется в последовательность исполняемых машинных кодов, и затем выполняется. Программы на языках компилирующего типа, например, Delphi или Lazarus, все целиком преобразуются в последовательности исполняемых машинных кодов, и на выполнение запускается сразу вся такая последовательность, которая может быть записана в файл с расширением «exe». Такой файл может быть запущен на выполнение программы с помощью операционной системы и программы «Проводник». Сделать подобное с программой на языке Python невозможно. Эту программу можно запустить на выполнение только из какого-либо программного приложения. В этом пособии рассматривается ситуация, когда в качестве таких программных приложений используются геоинформационные системы ArcGIS и Quantum GIS. Программы на языках интерпретирующего типа выполняются медленнее, чем exe-файлы, но зато их отладка выполняется проще, чем у программ на языках компилирующего типа.

Особенность языка Python заключается в наличии большого количества подключаемых модулей – библиотек специализированных функций. Для пространственного анализа в ArcGIS и Quantum GIS применяются разные модули пространственных функций. Поэтому программа, работающая в среде ArcGIS, не будет работать в Quantum GIS. Программы Python для исполнения в ArcGIS более устойчивы по сравнению с программами для использования в Quantum GIS. Однако ArcGIS – это пропиетарный программный пакет с закрытым исходным кодом, его стоимость достаточно высока, хотя существует возможность его аренды на определённый срок. Quantum GIS – бесплатно распространяемая программа с открытым исходным кодом. Однако в ней разные модули разработаны разными авторами, и совместимость этих модулей может отсутствовать. Поэтому программа на Python версии для этой ГИС может вообще не работать, или работать совершенно не так, как она должна в соответствии с описаниями функций.

Обычно руководства по программированию содержат описание грамматики языка, операторов и функций, а уже в конце приводятся примеры программ на данном языке. Автор данного пособия не настолько наивен, чтобы ожидать от студента Института наук о Земле скрупулёзного изучения языка Python. Автор придерживается другого подхода: лучший способ изучить язык программирования – разобраться в работе уже созданных кем-то программ, а затем скорректировать эти программы под свои нужды. А уже потом можно писать свои программы «с нуля». Если нужно сварить борщ, то нет необходимости заканчивать кулинарный техникум, достаточно иметь под рукой хорошую поварённую книгу. Если предстоит кратковременная поездка в другую страну, то нет нужды выучивать язык этой страны или используемый в ней язык международного общения на уровне выпускника филологического факультета. Достаточно выучить по разговорнику самые необходимые слова и фразы. Так и данное учебное пособие содержит ряд «рецептов» и «необходимых фраз» для решения некоторых типов задач для автоматизированной обработки пространственно-координированной информации в среде ГИС. С едиными для всех версий языка Python правилами его грамматики и операторами читатель может познакомиться с помощью источников, список которых приведён в конце учебного пособия.

Читатель вправе спросить: «А где можно получить информацию о кодировании на языке Python функций пространственной обработки, которые не приводятся в данном пособии?» Для этого следует использовать программы-поисковики в Интернете, «Справки» соответствующих ГИС, и модели рабочих потоков. В обеих рассматриваемых ГИС существует возможность визуального программирования модели рабочего потока, в которой соединяются исходные данные, функции их обработки, промежуточные результаты, следующие функции обработки этих результатов, и конечные результаты. Графически модель может представлять собой длинную цепочку из данных и функций их обработки. В модели может быть несколько соединяющихся цепочек, т.е. она может иметь древовидную структуру. Функции – это определённые опции интерфейса ГИС. В ArcGIS отдельная модель записывается в набор инструментов – файл с расширением «tbx». В наборе может содержаться более одной модели. В Quantum GIS модель может быть записана в файл с расширением «model3». Необходимо заметить, что модель рабочего потока ArcGIS – жёсткая, т.е. для изменения адресов и названий файлов исходных данных и файлов с конечными результатами обработки необходимо

корректировать модель. Модель рабочего потока Quantum GIS более гибкая: пользователь может с помощью диалоговых окон указывать источники данных и места размещения файлов результатов обработки, и вводить их названия. В обоих случаях модели рабочих потоков могут быть записаны в файлы программного кода на языке Python. Такие файлы могут служить в качестве «полуфабрикатов» при разработке компьютерной программы автоматизированной обработки данных. При этом необходимо подчеркнуть, что далеко не всякая функция, успешно работающая в модели рабочего потока, может быть применена после её конвертации в код Python в компьютерной программе. Это происходит из-за того, что в полученном после конвертации коде файлы источников данных указываются в явном виде, с абсолютным путём к ним, с существующими именами и расширениями файлов. В программе же автоматизированной обработки необходимо в цикле формировать различные значения строковой переменной, в которой должны указываться все возможные адреса и файлы источников данных. А далеко не всякая функция позволяет применять строковую переменную в качестве указания на источник данных. Тем не менее, использование модели рабочего потока после её конвертации в программный код позволяет хотя бы узнать название функции Python, осуществляющей данную операцию пространственного анализа или работы с таблицей атрибутов, и грамматику этой функции. Необходимо заметить, что программный код на языке Python может быть записан в текстовой файл, и редактирование текста программы возможно в среде приложений «Блокнот» и «WordPad».

Возможны два варианта автоматизированной обработки пространственно-координированной информации в среде ГИС: обработка растровых и векторных данных. В пособии на конкретных реальных примерах рассматриваются оба эти варианта. Растровые данные – это матрицы спутниковых снимков. Векторные данные – это шейпфайлы, например, представляющие собой результаты векторизации спутниковых снимков или компьютерного моделирования динамики ледяного покрова. Читатель может модернизировать представленный программный код для обработки других растров и файлов, или файловых структур векторных форматов.

При разборе всех представленных в пособии образцов программного кода автор старается приводить типовые задачи в сфере наук о Земле, для решения которых может пригодиться данный отрывок программного кода или программа в целом. Разумеется,

невозможно охватить все возможности применения программирования на языке Python для автоматизации пространственного и семантического анализа, а также различных видов моделирования в этой сфере наук, т.к. она «неисчерпаема, как электрон» («Электрон так же неисчерпаем, как и атом» – цитата из работы В.И. Ленина «Материализм и эмпириокритицизм»). В этой связи автор не может отказать себе в удовольствии привести историю, рассказанную ныне покойным Виталием Георгиевичем Морачевским, который продолжительное время был деканом метеорологического факультета РГГМУ и первым исполняющим обязанности заведующего кафедрой геоэкологии факультета географии и геоэкологии СПбГУ. История произошла с его однокурсником (будущим академиком) ещё в их студенческие годы. На экзамене одним из вопросов билета была сила Кориолиса. В учебнике эта сила рассматривалась на примере мухи, ползущей по переплёту поворачиваемой форточки. Студент так и рассказывает. Экзаменатор его перебивает: «Сдалась Вам это муха! Дайте общее определение». Студент отвечает: «Сила Кориолиса – это сила, действующая на любое насекомое, ползущее по вращающейся форточке», тогда как правильное определение: «Сила Кориолиса действует на любое тело, движущееся во вращающейся системе». Так и в нашем случае приводимые примеры использования программного кода на языке Python отнюдь не исчерпывают все многообразные возможности их применения в сфере наук о Земле.

1. Обработка результатов исследований в среде ArcGIS

Арктический и антарктический научно-исследовательский институт каждую зиму выполняет работы по оперативному обеспечению судоходства танкеров в Татарском проливе. Производится распознавание спутниковых снимков с созданием шейпфайлов ледовой информации. На эти файлы накладывается сетка квадратных ячеек. Информация из центра каждой ячейки, т.е. ряд матриц значений параметров ледяного покрова экспортируется в модель его динамики на основании метеорологического прогноза. Результаты моделирования конвертируются в три шейпфайла, содержащие следующую ледовую информацию: общая сплоченность льда и частные концентрации по ряду возрастных градаций льдов, возраста льда, их толщина, формы льда (средние размеры льдин) по возрастным градациям; общая торосистость льда; интенсивность сжатий льдов. Далее начинается работа дежурного специалиста, чья задача – определить оптимальный маршрут плавания и выложить его в Интернет для судоводителей. На электронную карту (фрейм) ArcGIS добавляются три вышеупомянутые шейпфайла ледовой информации. Затем специалист с помощью специально разработанной на языке VBA (Visual Basic for Application) компьютерной программы на электронной карте рисует маршрут плавания, сначала как графический объект. Затем он в программе конвертируется в шейпфайл линейного объекта. В программе же автоматически выполняется его пересечение с тремя слоями ледовой информации с созданием нового шейпфайла линейных объектов, наследующих семантическую информацию пересекаемых полигональных объектов слоёв ледовой информации. Затем в программе выполняются расчёты скорости и затрат времени плавания в пределах каждого объекта (участка плавания), а затем – общих затрат времени на преодоление всего маршрута плавания. Этот результат выводится на экран. Обыкновенно специалист создаёт несколько вариантов маршрута плавания. При этом он не выбирает автоматически вариант с наименьшими затратами времени. Он также должен учитывать вероятность ухудшения ситуации на маршруте, возможность возникновения сжатий дрейфующих льдов, т.к. метеопрогноз, на основании которого создается прогноз ледовой ситуации, не может иметь оправдываемость, равную 100%. Шейпфайл выбранного маршрута конвертируется в файлы двух форматов. Уже эти файлы «выкладываются» в интернете. Файлы одного из этих форматов могут быть преобразованы в файлы формата реляционной базы данных. В 2020 году возникла необходимость обработки всех

рекомендованных маршрутов плаваний, начиная с 2007 г. Поскольку этих маршрутов было более 1300, за весьма ограниченное время можно было выполнить эту задачу только с помощью автоматизации обработки, т.е. с помощью набора специально разработанных автором на языке Python программ. Ниже приводятся подробные описания этих программ с исходным кодом и разъяснениями, как могут применяться универсальные алгоритмы при обработке данных в среде экологии и природопользования. Необходимо отметить следующую особенность языка Python: блоки программы, например, ряд операторов, выполняемых при выполнении определённого условия (или группы условий), выделяются единым размером сдвига начала кода от левого края текста программы. Это его отличие от, например, языка Lazarus, в котором для этого применяются операторные скобки «begin» и «end». Если в пособии ширина страницы в каких-либо случаях не позволяла абсолютно сохранить структуру текстового файла программы, то на это указывается в тексте пособия. Повторяющиеся блоки программного кода не повторяются при описании каждой программы. У приведённых ниже программ начало практически идентично. Перед строками программного кода приводятся разъяснения, какие действия этот код выполняет.

1.1. Начало программы: «печка, от которой можно начать танцевать»

Чтобы читатель не закрыл с содроганием это пособие сразу после прочтения первых страниц, автор намеренно приводит код программ таким образом, чтобы читатель имел максимальную возможность использовать представленный код для достижения своих целей. Можно привести следующую аналогию: в разговорниках приводятся фразы на иностранных языках типа: «Как я могу доехать до такого-то пункта?» Туристу остаётся только вставить в эту фразу название нужного ему пункта. Так и на основании данного пособия читатель сразу сможет создавать свой программный код.

Сначала нужно объяснить компьютеру, в какой кодировке ему следует воспринимать текст программы. Поэтому первая строка программы должна содержать именно эту информацию. Вообще в программах на языке Python символ «#» в начале строки означает, что далее идёт комментарий, нужный для разъяснения программы, и не обрабатываемый при преобразовании текста программы в исполняемый машинный код. Однако в самой первой строке программы после этого символа ставится разъяснение, в какой кодировке записан текст программы. По умолчанию это кодировка

UTF-8 символов Unicode:
-*- coding: utf-8 -*-
Недостаток этой кодировки: в ней нет символов кириллицы. Поэтому для корректной работы программы следует позаботиться о том, чтобы в полных адресах обрабатываемых и создаваемых файлов и файловых структур не встретился ни один символ кириллицы. Напоминаю, что полный адрес включает в себя как название файла с его расширением, так и всю «матрёшку» папок, в которой он находится, и наименование диска. Теоретически можно изменить в этой строке кодировку на другую, допускающую символы кириллицы, например, Windows-1251, KOI8-R и т.д. Однако попытки автора «победить русофобию» версий языка Python для ГИС не увенчались успехом. Поэтому следует придерживаться простого правила: ни папки, в которых содержатся файлы исходных данных, ни имена этих файлов, ни папки для записи результатов выполнения программы и сами имена файлов результатов не должны содержать символов кириллицы.

После указания кодировки крайне желательно поместить комментарий с кратким описанием цели данной программы, например,
программа создания слоёв событий.
Это необходимо потому, что сам автор программы скоро может забыть, для какой цели написана данная программа, не говоря уже о других пользователях. Затем следует подключить возможности операционной системы, чтобы она тоже участвовала в выполнении программы:
import os
import sys
Далее следует подключить модули функций Python для ArcGIS:
import arcpy
from arcpy import env
import arcpy.mapping as mapping
Автор не приводит описание модуля ArcPy. Пытливый читатель сможет получить исчерпывающую информацию в Интернете и Справке ArcGIS. Просто обязательно вставляйте эти строки в код своих программ, и они будут работать. Так, Али-баба из сказки «Али-баба и сорок разбойников» не стал задумываться о взаимосвязи между словами «Сим-сим, открой дверь», и появлением входа в пещеру сокровищ. Так и читатели могут сразу зайти в волшебный мир создания непосредственно программы.

В ArcGIS есть понятие рабочего пространства (WorkSpace), указывающего, где размещены файлы и/или файловые структуры. Чтобы не усложнять программу, следует придерживаться следующего

правила: все исходные данные и файл документа карты с расширением «mxd» помещаются в одной папке, при этом данная папка помещается в корневом каталоге, т.е. непосредственно на конкретном диске, флэшке, а не внутри какой-нибудь другой папки. Папка с документом карты может содержать вложенные папки, например, отдельно папки исходных данных и папки для записи результатов. В ArcGIS пользователь может работать только в каком-либо документе карты. И только из этого документа карты он может запускать на исполнение программный код Python. Поэтому в начале непосредственно исполняемого кода программы нужно поставить возвращение текущего документа карты:

```
the_mxd = mapping.MapDocument('Current')
```

Далее «вытаскиваем» адрес и название папки, в которой находится файл текущего документа карты. Как только что было указано, в этой папке должны также содержатся файлы исходных данных и будут содержаться результаты работы программы, возможно, в отдельных вложенных папках. Для этого вставляем в программу следующие строки:

```
the_mxd_FilePath = the_mxd.filePath
len_the_mxd_FilePath = len(the_mxd_FilePath)
n_slash_kon = the_mxd_FilePath.rfind('\\')
str_dir = the_mxd_FilePath[0:n_slash_kon]
```

Здесь переменная «len_the_mxd_FilePath» – это число символов полного пути к файлу документа карты и названия этого файла, а переменная n_slash_kon – номер символа косой черты (обратный слэш) «\» в строковой переменной the_mxd_FilePath пути к папке документа карты и его названия, считая слева направо. Строковая переменная str_dir содержит полный адрес папки с файлом документа карты и всеми исходными данными.

Документ карты ArcGIS является аналогом проекта Quantum GIS. Одно из существенных различий заключается в том, что документ карты может содержать более одной электронной карты (фрейма). Но рассматриваемые программы выполняют обработку данных в первом фрейме из списка фреймов документа карты. Поэтому во избежание ошибок лучше работать в документах карт, содержащих только один фрейм. Получаем список фреймов текущего документа карты:

```
frame_list = arcpy.mapping.ListDataFrames(the_mxd)
```

И в нём получаем первый фрейм из списка:

```
frame1=frame_list[0]
```

Если разрабатываемая программа будет включать операции пространственного анализа, то необходимо получение системы

координат этого фрейма:
frame1CoordSys = frame1.spatialReference
Пренебречь этой строкой кода можно только в том случае, если алгоритм программы состоит только в обработке семантических данных в таблице атрибутов. В случае же операций пространственного анализа необходимо строжайшее соблюдение условия совпадения систем координат фрейма и всех слоёв, используемых при выполнении оверлейных операций. Весь приведённый выше код можно без каких-либо изменений вставлять в свою программу. Непосредственно написание кода программы можно выполнять как в среде ArcGIS, так и в текстовых редакторах «Блокнот» и WordPad. Автор считает оптимальным создание и редактирование программы в среде редактора «Блокнот», т.к. в этом случае разработчик имеет возможности вставки в текст программы уже готовых блоков из других программ.

1.2. Вставка текста программы в окно Python ArcGIS и выполнение программы

Давайте загрузим приведённый выше код Python в ArcGIS и запустим его на выполнение. Откроем документ карты ArcGIS, на его панели инструментов в верхней части – вкладку Geoprocessing, затем в появившемся ниспадающем меню – опцию Python (рис. 1). На экране появится окно «Python». Открываем текстовой файл программы в редакторе «Блокнот», выделяем и копируем текст программы. Возвращаемся в окно «Python» в ArcGIS, нажимаем правую клавишу манипулятора «мышь». В окне «Python» появится меню (рис. 2).

Выбираем опцию «Paste». Скопированный текст программы будет вставлен в окно «Python». В нём можно редактировать программу и вообще писать её с самого начала. На рисунке 3 представлен вид окна «Python» со вставленным программным кодом. Чтобы запустить программу на исполнение, необходимо навести курсор в конец самой последней строки кода, и нажать на левую клавишу манипулятора «мышь». В этом месте окна «Python» появится мигающее изображение вертикальной черты (как на рисунке 3).

Затем следует нужно нажать на клавишу «Enter» клавиатуры компьютера: программа будет исполняться. После завершения её выполнения в окне «Python» ниже текста программы появится символ «>>>» (рис. 4).

Пользователь теперь может узнать значения переменных программы. Для этого достаточно написать ниже обозначение переменной или скопировать его из расположенного выше текста

программы, и нажать клавишу «Enter» клавиатуры. В окне появятся значения переменных (рис. 5).

Рисунок 1. Открывание окна для редактирования программы на языке Python

Вывод значений переменных очень полезен при отладке программы. Программный код, записанный в формате программы на языке Python (расширение файлов – «ру») можно вставить в окно «Python» с помощью опции «Paste» ниспадающего меню (рис. 2). Очистка окна «Python» от всего текста выполняется с помощью опции «Clear all» этого же меню.

Задание. Повторите все описанные выше действия. Система координат фрейма может быть любой, или может даже вообще отсутствовать.

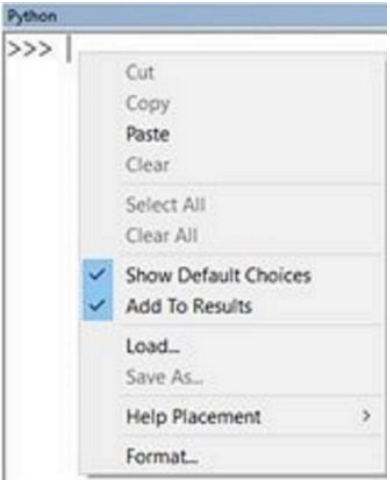


Рисунок 2. Вставка текста программы в окно «Python»

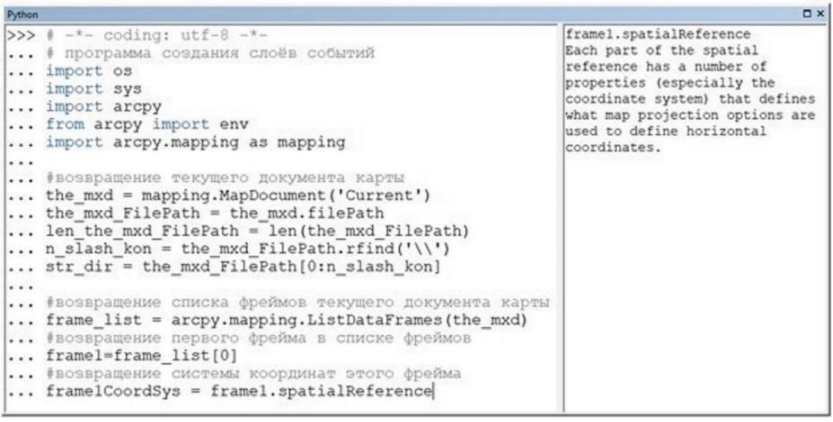


Рисунок 3. Код программы в окне «Python»

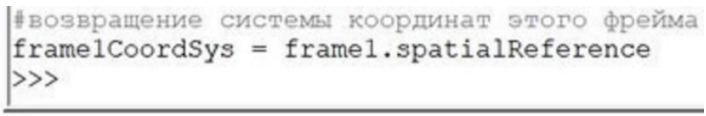


Рисунок 4. Сообщение об окончании работы программы.

```
>>> the_mxd
<MapDocument object at 0x3836aaf0[0x38045ba0]>
>>> str_dir
u'F:\\Sakhalin_2020'
>>> frame1CoordSys
<SpatialReference object at 0x3763f490[0x3783bab8]>
```

Рисунок 5. Вывод значений переменных

1.3. Программный код создания координатного слоя событий

В терминологии ГИС «слой событий» – это векторный слой, источником которого служит таблица реляционного формата. Они бывают координатными и маршрутными. В первом случае таблица должна содержать поля координат X и Y точечных объектов. Координаты могут быть записаны в единицах различных систем координат. При этом для локализации точечных объектов на электронной карте нет необходимости, чтобы карта содержала какие-либо слои: она может быть пустой. Главное, чтобы у карты была задана система координат. Другой формат слоёв событий – это маршрутные слои событий. Их источниками также служат таблицы реляционного формата, содержащие идентификаторы линейных объектов и значения расстояний до начальных узлов этих объектов вдоль самих линейных объектов. Для создания маршрутного слоя событий необходимо, чтобы электронная карта содержала слой маршрута: слой линейных объектов типа PolylineM, все вершины которых кроме координат X и Y имеют ещё и третью координату M – расстояние в линейной мере от данной вершины до начального узла линейного объекта вдоль самого этого объекта, т.е. сумму длин всех сегментов объекта от начального узла до данной вершины. Напоминаем, что сегмент – это отрезок прямой между двумя соседними вершинами линейного объекта, который представляет собой ломаную линию.

Любое научное исследование в сфере наук о Земле начинается с получения ответов на вопросы «что, где, когда и сколько». На следующем этапе рассматриваются вопросы «как, каким образом», т.е. как функционирует данная природная или антропогенезированная система. Затем встаёт вопрос «почему», т.е. почему всё происходит именно так, а не иначе. И наконец, исследование завершает поиск ответов на вопросы типа «что будет, если», для чего применяются различные виды компьютерного моделирования, включая имитационное.

Очевидно, что без ответа на вопрос «где» вся дальнейшая работа теряет смысл. Ценность полевых исследований без географической привязки результатов равна нулю. Поэтому результаты всех измерений и определений должны быть нанесены на карту. Это самый первый и наиважнейший этап обработки данных полевых исследований. Взглянув на созданную карту, исследователь уже сможет сделать какие-либо выводы и сформулировать задачи для продолжения исследований, что невозможно сделать путём просмотра таблиц результатов. Отчёты о результатах экспедиционных и вообще полевых исследований должны содержать комплекты карт с исчерпывающей информацией.

Разумеется, во многих случаях нет необходимости в определении координат точек измерений и отбора проб. Например, если проба воды в реке Смоленке отобрана с восточного спуска к воде у северного конца Наличного моста (остров Декабристов, или Голодай, недалеко от станции метро «Приморская»), то этой информации достаточно для создания в ГИС точечного объекта, разумеется, при наличии детальной электронной карты Санкт-Петербурга. Но если исследования происходят на обширной акватории, или на суше при отсутствии отмеченных на карте ориентиров, то в этих случаях необходимо определять координаты точек отбора проб и измерений. К счастью, теперь это делается очень просто с помощью спутниковой навигаторов и даже смартфонов. Только нужно знать, в какой системе координат прибор показывает координаты. По умолчанию это международная система WGS84. Она была введена в 1995 году. Символы «84» в её названии означают, что в ней ось координат Z имеет такое же направление, как и ось вращения Земли в 1984 году. Данные мониторинга состояния вод крупных акваторий (Невская губа, Финский залив, Ладожское озеро и т.д.) относятся к станциям мониторинга, т.е. к точкам с определёнными координатами. Координаты точек мониторинга или точек отбора проб и измерений проще всего записать в таблицы Microsoft Excel. Ниже приведён пример подобной таблицы (1).

При создании координатного слоя событий «вручную» можно использовать как файлы формата Microsoft Excel, так и файлы форматов «csv» и «txt», т.е. текстовые формата реляционных таблиц с разделителями. В этом случае поля в таблицах разделяются между собой определённым символом. Лучше всего использовать формат «csv» с разделителем «точка с запятой». В Microsoft Excel есть возможность записи таблиц в этом формате: «Сохранить как», тип

файла – CSV (разделители – запятые). Несмотря на это название типа, в файле результата разделителями будут символы точки с запятой.

Таблица 1

Пример таблицы реляционного формата для записи координат станций мониторинга

Station	Shirota	Dolgota
1	59,983	30,217
2	59,967	30,217
3	60,117	28,067
4	60,117	27,383
5	59,883	30,217

Однако для автоматизированного создания координатных слоёв событий лучше подходит формат файлов СУБД dBASE реляционного формата с расширением «dbf». В более старых версиях Microsoft Excel существовала возможность записи листов в файлы этого формата. В современных версиях эта возможность отсутствует, но существуют свободные программы для конвертации:
«onlineconvertfree.com» (https://onlineconvertfree.com/ru/convert-format/xls-to-dbf/)
и «anyconv.com» (https://anyconv.com/ru/konverter-xls-v-dbf/).

Итак, у нас есть набор файлов с расширением «dbf», имена которых различаются датами. При исследованиях в сфере экологии и природопользования такая ситуация может возникнуть при обработке данных мониторинга за различные сроки. Рассмотрим код для автоматизированного создания координатных слоёв событий и их записи в шейпфайлы. Начало программы универсально и совпадает с кодом, приведённым в разделе 1.1.

Предположим, что таблицы содержат результаты мониторинга на различных точках за разные даты. Имена файлов таблиц содержат обозначение даты мониторинга в формате «гггггммдд», где «гггг» - год, «мм» - номер месяца, «дд» - номер дня, например, «20210823». Поэтому создаём списки всех возможных строковых значений лет, месяцев и дней:

```
lst_God=['2019', '2020', '2021']
lst_Mes=['01','02','03','04', '05', '06', '07', '08', '09', '10', '11','12']
lst_Den=['01','02','03','04','05','06','07','08','09','10','11','12','13','14','15', \
```

'16','17','18','19','20','21','22','23','24','25','26','27','28','29','30','31']

Обратите внимание, что длинная строка кода разбита на две с помощью символа обратный слэш – «\». Эти две строки будут восприниматься при преобразовании кода в исполняемые машинные символы как один оператор. В данном случае можно обойтись без обратного слэша, т.к. список заключён в квадратные скобки:

```
lst_Den=['01','02','03','04','05','06','07','08','09','10','11','12','13','14','15',
        '16','17','18','19','20','21','22','23','24','25','26','27','28','29','30','31']
```

Важно! Далее в тексте пособия в примерах программного кода при размещении одного оператора на двух строках знак обратного слэша опускается, чтобы читатель мог сразу скопировать его в одну строку своей программы.

Затем определяется число элементов в каждом списке:

```
n_let = len(lst_God)
n_mes = len(lst_Mes)
n_den = len(lst_Den)
```

Далее организуются три вложенные цикла просмотра списков лет, месяцев и дней. Важно! Принадлежность операторов к телу определённого цикла определяется одинаковым отступом строк кода от левого края текста программы. То есть все строки кода, относящиеся к одному циклу, должны располагаться на одинаковом расстоянии от левого края текста. Функция «range» «перебирает» все элементы списка.

```
for i_god in range (n_let):
  tek_god=lst_God[i_god]
  for i_mes in range(n_mes):
    tek_mes=lst_Mes[i_mes]
    for i_den in range (n_den):
      tek_den=lst_Den[i_den]
      str_data=tek_god+tek_mes+tek_den
```

Далее задаётся адрес и название файла данных в формате dBASE. Это предложение – текст пособия, а не программы. Если бы мы хотели сделать комментарий в программе, то его нужно было бы разместить на таком же расстоянии от левого края, как и операторы данного цикла. Заметьте, что в названиях переменных желательно указывать из тип, например, у строковых переменных в начале имени ставить префикс «str_».

```
      #адрес файла dBASE координат точек исследования
      str_dBASE_file=str_dir+'\\Result_'+str_data+'.dbf'
```

Обратите внимание, что при указании адреса файла ставится не одинарный, а двойной обратный слэш. В данном примере все

исходные файла формата dBASE с расширением «dbf» находятся непосредственно в папке документа карты. Адрес и название папки записаны в переменную «str_dir» (см. раздел 1.1.). Можно было бы поместить их в отдельную папку внутри этой папки, например, папку «dBASE_files». В этом случае оператор должен был бы иметь вид:

```
str_dBASE_file=str_dir+'\\dBASE_files\\Result_'+str_data+'.dbf'
```

Файлы данных в этом примере имеют названия типа «Result_20210823.dbf'». Можно задать любую другую конфигурацию названий путём складывания большего числа строковых значений, поменять местами дату и общую часть в названиях файлов, например, str_data+' result.dbf'. Пока неизвестно, а существует ли файл с таким названием в этом месте или нет. Необходима проверка его существования. Во избежание возникновения аварийных ситуаций при выполнении программ следует выполнять подобную проверку при любом обращении к файлам. После оператора проверки условия «if» следует программный код, исполняемый только при выполнении этого условия.

```
if arcpy.Exists(str_dBASE_file):
  # Значения координаты X берутся из поля «Dolgota»
  x_coords = 'Dolgota'
  # Значения координаты Y берутся из поля «Shirota»
  y_coords = 'Shirota'
  # Название вновь создаваемого координатного слоя событий
  out_Layer = 'Result_'+str_data
  # Создание координатного слоя событий
  arcpy.MakeXYEventLayer_management(str_dBASE_file,
  x_coords, y_coords, out_Layer, frame1CoordSys)
```

В этом операторе str_dBASE_file – адрес и название файла формата реляционной таблицы с координатами точек, x_coords и y_coords – соответственно координаты X и Y, out_Layer – название вновь создаваемого координатного слоя событий, frame1CoordSys – система координат электронной карты (фрейма), на которую добавляется вновь создаваемый слой событий. **Внимание! Очень важно!** Система координат электронной карты должна быть точно такой же, как и система, в которой записаны координаты точек в исходной таблице.

Теперь нужно задать адрес и название шейпфайла, в который будет записан слой событий. Заметьте, что задаётся только адрес и название файла с расширением «shp». Остальные компоненты шейпфайла: файлы с этим же именем, но расширениями «dbf», «shx», «prj» будут созданы автоматически.

```
str_Shape_file=str_dir+'\\Points_'+str_data+'.shp'
```

Естественно, необходима проверка: возможно, шейпфайл с этим названием и адресом уже существует? Если он существует, то выполняется удаление «предшественника».

```
if arcpy.Exists(str_Shape_file):
  #Удаление "предшественника" в случае его существования
  arcpy.Delete_management(str_Shape_file)
```

Наконец, заключительный аккорд программы: запись координатного слоя событий, который содержится только в памяти компьютера, в шейпфайл. Это нужно для того, чтобы созданные пространственные объекты можно было помещать на карты с любыми системами координат, поскольку при создании шейпфайла создаётся текстовой файл с расширением «prj», содержащий исчерпывающую информацию о системе координат пространственных объектов. Поэтому при добавлении этих объектов на любую карту пересчёт координат в случае необходимости выполняется «на лету».

```
arcpy.Project_management(out_Layer, str_Shape_file,
frame1CoordSys)
```

Внимание! Этот оператор выполняется уже вне тела условия существования «предшественника» с его удалением.

Задание. В таблице 1 приведены координаты 5 станций мониторинга на акватории Финского залива. Запишите эту таблицу в лист файла формата Microsoft Excel реляционного формата, в колонку с названием «NTOCHKI» запишите номера точек как целые числа, в колонки «Shirota» и «Dolgota» запишите числовые значения широты и долготы точек. Преобразуйте файл формата Microsoft Excel в файл формата СУБД dBASE. Создайте на основании этих данных координатный слой событий сначала «вручную» и запишите его в шейпфайл. Затем создайте модель рабочего потока, и выполните эти же действия с его помощью. Далее с помощью смартфона определите несколько последовательностей координат точек маршрутов (2-4). Если у Вас нет смартфона, то придумайте эти последовательности координат точек. Запишите координаты в таблицы Microsoft Excel и конвертируйте их в файлы формата СУБД dBASE. Модернизируйте код программы для автоматизированного создания координатных слоёв событий этих точек маршрутов и их записи в шейпфайлы. Названия исходных таблиц можно задать как последовательность номеров маршрутов: 1, 2, 3, 4 и т.д. При модернизации кода программы уже не будут нужны списки лет, месяцев и дней, а только один список номеров маршрутов (разрезов, профилей). Разумеется, и цикл нужен будет только один, по номерам маршрутов. Список

номеров маршрутов должен содержать их строковые, а не числовые значения.

1.4. Создание линейных объектов из последовательностей точек

Часто встречающаяся задача – необходимо превратить последовательность точек в объект типа Polyline. При создании карт результатов полевых исследований такая необходимость возникает, когда нужно показать на карте разрезы, профили, по которым на карте проводились исследования. Другой вариант: во время исследования обнаружен новый объект, который необходимо показать на карте, например, несанкционированную свалку. Для этого нужно определить методами спутниковой навигации координаты последовательности точек на границе свалки, которые впоследствии образуют вершины ломаной линии - границы этой свалки. Затем создаём координатный слой событий, локализующий на карте эти точки на границе, и записываем этот слой событий в шейпфайл. Приводимый ниже программный код был разработан для автоматизированного создания рекомендованных маршрутов плавания в Татарском проливе по шейпфайлам точечных объектов – поворотных точек маршрутов. В названия как исходных, так и создаваемых шейпфайлов входит дата маршрута. Поэтому в этом программном коде, как и в программе создания координатных слоёв, присутствуют три вложенных цикла по всем элементам списков лет, месяцев и дней. В случае обозначения шейпфайлов маршрутов (разрезов) их номерами, естественно, будет нужен только один цикл по этим номерам. Поэтому в приводимом ниже коде программы не приводятся циклы определения названий исходных шейпфайлов и шейпфайлов результатов. Вот оператор определения названия исходного шейпфайла, он совпадает с оператором задания шейпфайла результата в предыдущей программе создания координатных слоёв событий и их записи в шейпфайлы:

```
str_Shape_file=str_dir+'\\Points_'+str_data+'.shp'
```

При использовании в названиях исходных шейпфайлов номеров маршрутов, разрезов, профилей следует заменить в этом операторе переменную «str_data» на переменную, содержащую номер маршрута. Проверка, а существует ли этот файл:

```
if arcpy.Exists(str_Points_file):
```

Очень важное замечание! В абсолютном большинстве случаев координаты точек будут записаны в географических системах координат: международной WGS84 или российских Pulkovo42 и Pulkovo95. Электронные карты имеют проецированные декартовы системы координат. Это означает, что карта создаётся в определённой

картографической проекции со своими параметрами, и пространственные объекты переносятся с поверхности воображаемого датума (эллипсоид + параметры сдвига его центра относительно центра Земли) на поверхность карты. Единицы координат X и Y проецированной системы – линейные, в России это метры. Разумеется, создание координатного слоя событий по географическим координатам точек возможно и на электронной карте с проецированной системой координат. Но всё же надёжнее делать это на карте с такой системой координат, в которой записаны координаты точек, т.е. какой-либо географической. А вот создание линейного слоя маршрута (разреза, профиля) обязательно следует выполнять во фрейме (электронной карте) с проецированной системой координат. Поэтому нужно выполнить перепроецирование шейпфайла, т.е. запись объектов из исходного шейпфайла в новый шейпфайл с проецированной системой координат, совпадающей с системой координат фрейма. Сначала следует задать адрес и название создаваемого файла:

```
#адрес шейпфайла - результата проецирования
str_Points_pr_file=str_dir+'\\Points_pr_'+str_data+'.shp'
```

Текст этого оператора и последующих должен быть сдвинут относительно оператора проверки существования исходного шейпфайла, т.к. они образуют блок операторов, выполняемых только при выполнении проверяемого условия. Далее нужно проверить: может быть шейпфайл, который только планируется создать, уже существует? В последнем случае «предшественник» уничтожается.

```
#Проверка: возможно, шейпфайл с этим названием и адресом уже
существует?
if arcpy.Exists(str_Points_pr_file):
    #Удаление "предшественника" в случае его существования
    arcpy.Delete_management(str_Points_pr_file)
```

Запись объектов из шейпфайла с географической системой координат в новый шейпфайл с проецированной системой координат:

```
arcpy.Project_management(str_Points_file, str_Points_pr_file,
frame1CoordSys)
```

Этот оператор уже не входит в блок проверки существования шейпфайла точечных объектов в проецированной системе координат. Далее выполняется создание линейного слоя маршрута (разреза, профиля). В нём будет находиться только один объект. Нужно задать размещение и название нового шейпфайла, и создать его. Разумеется, также желательно выполнить проверку возможного существования его

«предшественника», но блок этой проверки здесь опущен. Задание места и названия шейпфайла маршрута (разреза, профиля):

```
str_route_file=str_dir+'\\Route_'+str_data+'.shp'
```

Создание шейпфайла маршрута (разреза, профиля):

```
arcpy.PointsToLine_management(str_Points_pr_file, str_route_file, "",
"NTOCHKI", "NO_CLOSE")
```

Здесь каждая точка исходного шейпфайла порождает вершину вновь создаваемого линейного объекта. Последовательность точек определяется значениями из целого числового поля «NTOCHKI». Параметр "NO_CLOSE" означает, что линию не следует замыкать.

Задание. Создайте программу автоматизированного создания маршрутов, и запустите её на исполнение. В качестве исходных шейпфайлов последовательностей точечных объектов используйте результаты выполнения задания в предыдущем разделе 1.3. Названия создаваемых шейпфайлов задайте «route_1», «route_2», «route_3».

1.5. Автоматизация оверлейных операций

Важной частью пространственного анализа в ГИС являются оверлейные операции: различные способы наложения двух и более векторных слоёв с генерацией нового векторного слоя, наследующего семантические характеристики обоих «родителей». Возможны пересечение, объединение, слияние, разность слоёв. При пересечении создаются новые объекты только в пределах области, в пределах которой существуют объекты всех участвующих в этой операции слоёв. Размерность слоя – результата пересечения определяется размерностью первого по порядку слоя с списке пересекаемых слоёв. В случае пересечения двух полигональных слоёв их порядок безразличен, он определяет только расположение полей в таблице атрибутов слоя результата. Поля из таблицы первого слоя помещаются в левой части таблицы атрибутов результата, а из таблицы второго слоя – в правой. При пересечении линейного и полигонального слоёв линейный слой обязательно должен стоять первым. Это означает, что при выполнении пересечения с помощью интерфейса ГИС, т.е. «вручную», в окне списка пересекаемых слоёв слой линейных объектов должен быть самым верхним. Возможно и пересечение слоя точек с полигональным слоем, но такой же результат можно получить с помощью функции «Присвоение по расположению», когда точечные объекты получают семантические характеристики полигональных объектов, внутри которых они находятся.

Разумеется, автоматизация выполнения оверлейных операций с помощью компьютерных программ на языке Python оправдана только в случае большого количества обрабатываемых векторных слоёв, у которых в названиях источников присутствует определённая система. То есть когда можно запрограммировать изменения этих названий. В противном случае оптимально применение моделей рабочих потоков. Естественно, что границы водосборных бассейнов, административных единиц и ландшафтных таксонов достаточно стабильны и не меняются раз в несколько дней. Частые изменения происходят в гидросфере и атмосфере. На суше часто меняется ситуация распространения зон природных и антропогенных катастроф, в частности, лесных и степных пожаров. На замерзающих акваториях часто изменяется ледовая обстановка, в водных объектах и атмосфере – зоны степеней загрязнения различными веществами. Логично присутствие в названиях шейпфайлов и баз геоданных, содержащих эту информацию, сведений о моменте времени или временном интервале, к которому она относится. Комбинируя все возможные сочетания обозначений лет, месяцев или номеров месяцев, номеров декад месяца или недель, полудекад, часов, минут, секунд, можно генерировать любые имена файлов источников данных.

При обработке результатов моделирования ледовой ситуации в Татарском проливе и её анализе по рекомендованным маршрутам плаваний возникла необходимость пересечения трёх полигональных слоёв ледовых данных. Они являлись результатами компьютерного моделирования развития ледяного покрова по данным дистанционного зондирования Земли и прогноза погоды. Шейпфайлы, в названия которых входила аббревиатура «ICEFRC», содержали информацию об общей сплоченности ледяного покрова, частных концентрациях льдов разных возрастных градаций, средних размерах льдин этих градаций, толщинах льда по возрастным градациям, наличии льда или его полном отсутствии, о типе льда: дрейфующий или припай. В шейпфайлах с аббревиатурой «DEFFRC», содержащейся в их именах, была записана степень торосистости ледяного покрова. Наконец, шейпфайлы с аббревиатурой «PRSFRC» содержали степень сжатия дрейфующих льдов. Координаты пространственных объектов этих шейпфайлов были записаны в географической системе координат WGS84.

Ниже приводится фрагмент программы пересечения этих трёх шейпфайлов. Разумеется, выполнялось пересечение шейпфайлов, относящихся к одним суткам. Для формирования названий файлов и адресов их размещения использованы несколько вложенных циклов

перебора всех возможных значений года, месяца, дня, и часа в названии шейпфайла. Таким образом были определены значения строковых переменных str_ICEFRC, str_DEFFRC, и str_PRSFRC, содержащих полные адреса и названия соответствующих шейпфайлов, а также строковых переменных str_ICEFRC_pr, str_DEFFRC_pr, и str_PRSFRC_pr с адресами и названиями шейпфайлов – результатов перепроецирования. Перед выполнением операций проводилась проверка существования исходных файлов и возможного существования файлов результатов с уничтожением «предшественников» в случае их существования. Ниже в тексте не указан сдвиг текста кода относительно левого края.
Перепроецирование шейпфайлов:

```
arcpy.Project_management(str_ICEFRC, str_ICEFRC_pr,
frame1CoordSys)
arcpy.Project_management(str_DEFFRC, str_DEFFRC_pr,
frame1CoordSys)
arcpy.Project_management(str_PRSFRC, str_PRSFRC_pr,
frame1CoordSys)
```

Задание адреса и имени шейпфайла – будущего результата пересечения трёх спроецированных шейпфайлов:

```
str_Led=str_dir+'\\Led_'+str_data+'_18.shp'
```

Окончание «_18» в названии шейпфайла означает 18 часов – продолжительность периода прогноза, т.е. шейпфайлы содержат результаты моделирования состояния ледяного покрова через 18 часов от времени создания исходного спутникового снимка. Далее выполняется проверка возможного существования этого шейпфайла, с уничтожением «предшественника» в случае его обнаружения, и само пересечение трёх слоёв ледовой информации:

```
arcpy.Intersect_analysis([str_ICEFRC_pr,str_DEFFRC_pr,
str_PRSFRC_pr],str_Led,"ALL","","INPUT")
```

В этом операторе в квадратных скобках помещены строковые переменные с адресами и названиями пересекаемых шейпфайлов, затем следует строковая переменная с адресом и названием шейпфайла – результата пересечения. Условие "ALL" параметра «join_attributes» в этой строке кода означает, что таблицу атрибутов шейпфайла результата пересечения должны быть добавлены все поля из таблиц атрибутов пересекаемых слоёв. Другие возможные значения этого параметра: «NO_FID» и «ONLY_FID». Задание первого из этих условий означает, что в таблицу атрибутов результата будут добавлены все поля исходных таблиц пересекаемых слоёв за исключением полей «FID». При условии «ONLY_FID» наоборот,

будут добавлены только эти поля. Следующий параметр «cluster_tolerance» означает минимальное расстояние, разделяющее узлы и вершины, а также расстояние, на которое они могут быть сдвинуты по осям X и Y. Его можно опустить, как в данном примере, и поставить на его место пустую строку. Завершающий параметр «output_type» определяет тип пересечения. При значении «INPUT» пространственная размерность результата пересечения определяется минимальной размерностью пересекаемых слоёв. Так, при пересечении линии и полигона результат пересечения будет иметь размерность линии. Если «LINE»: результатом пересечения будут исключительно линейные объекты. В этом случае среди пересекаемых полей не должно быть слоёв точечных объектов. «POINT»: результаты пересечения – точки. В этом случае при пересечении линейных и полигональных слоёв результат пересечения будет иметь пространственный тип «Multipoint». Далее в уже другой программе выполнялось пересечение маршрутов плавания с созданными в данной программе слоями суммарной ледовой информации. Естественно, что даты слоёв маршрута и ледовой обстановки должны были совпадать. Это также достигалось перебором в трёх вложенных циклах всех возможных значений лет, месяцев и дней, и генерацией имён шейпфайлов маршрута и ледовой обстановки с одинаковой датой. Строковая переменная с адресом и названием шейпфайла маршрута:

```
str_path=str_dir+'\\Routes\\Route_'+str_data+'.shp'
```

Строковая переменная с адресом и названием шейпфайла ледовой обстановки:

```
str_led=str_dir+'\\Led\\Led_'+str_data+'_18.shp'
```

Затем формируется строковая переменная с полным адресом и названием шейпфайла – будущего результата пересечения линейного слоя маршрута и полигонального слоя ледовой обстановки:

```
str_ice_route=str_dir+'\\Ice_routes\\Ice_route_'+str_data+'_18.shp'
```

Код выполнения пересечения:

```
arcpy.Intersect_analysis([str_path, str_led], str_ice_route,
"ALL","","INPUT")
```

Возможно, внимательный читатель заметит, что созданные маршруты были перемещены непосредственно из папки с документами карты в находящуюся внутри неё подпапку «Routes», а созданные шейпфайлы объединённой ледовой информации – в подпапку «Led». Это было выполнено вне ГИС с помощью программы «Проводник».

Задание. Во фрейме (электронной карте) с созданными слоями маршрутов (результат выполнения задания в разделе 1.4) создайте несколько полигональных слоёв с названиями «polig_1»,

«polig_2», «polig_3». Создайте и запустите на исполнение программу пересечения слоёв «route» и «polig» с одинаковыми номерами.

1.6. Удаление ненужных полей из таблицы атрибутов

Оверлейные операции приводят к «разбуханию» таблиц атрибутов их результатов, т.к. они содержат поля из таблиц атрибутов всех вовлечённых в операцию слоёв. Исключением является только операция «Merge (Слияние)», когда выполняется объединение «встык» слоёв одной пространственной размерности, например, полигональных, и с одинаковой структурой таблиц атрибутов. В остальных случаях неизбежны увеличение количества полей в таблицах атрибутов результатов и требуемой оперативной и долговременной памяти компьютера. А они «не резиновые». Поэтому в таблицах атрибутов следует удалять поля с информацией, не требующейся для проведения анализа. Ниже приводится программный код этой операции. Разумеется, сперва нужно задать значение строковой переменной, содержащей сведения об адресе и названии источника данных, в данном примере шейпфайла:

```
str_ice_route=str_dir+'\\Ice_routes\\Ice_route_'+str_data+'_18.shp'
```

Далее выполняется проверка его существования:

```
if arcpy.Exists(str_ice_route):
```

Если условие существования выполнено, то создаётся список всех полей таблицы атрибутов шейпфайла:

```
  fields = arcpy.ListFields(str_ice_route)
```

Далее выполняется цикл просмотра всех полей таблицы с получением имени каждого поля, которое записывается в переменную «tek_field_name»:

```
  for field in fields:
    tek_field_name=field.name
```

Затем производится проверка существования каждого поля, «обречённого на уничтожение»:

```
     if tek_field_name=='Id':
        arcpy.DeleteField_management(str_ice_route, ["Id"])
```

В этом операторе в квадратных скобках в явном виде указывается имя удаляемого из таблицы поля. Если же мы абсолютно уверены в существовании полей, которые необходимо удалить из атрибутивной таблицы, то получения имён всех полей можно не использовать, а сразу приступить к удалению полей:

```
if arcpy.Exists(str_ice_route):
   arcpy.DeleteField_management(str_ice_route, ["FID_Route_"])
```

```
arcpy.DeleteField_management(str_ice_route, ["FID_Led_20"])
arcpy.DeleteField_management(str_ice_route, ["FID_ICEFRC"])
```

Задание. Создайте и запустите на исполнение программу удаления какого-либо поля из шейпфайлов – результатов пересечения слоёв «route» и «polig». Можете добавить новое поле «вручную», а затем удалить его с помощью программы.

1.7. Разбиение составных объектов на простые

Полигональные объекты могут быть простыми, сложными, составными, односвязными и многосвязными. Простой полигон состоит только из одной части, внутри которой нет не принадлежащих этому полигону участков. Например, пруд в Летнем саду Санкт-Петербурга – это простой полигон. Если же внутри полигона есть не относящиеся к нему участки, то он относится к типу сложных полигонов, и называется «полигон с дыркой», хотя и состоит только из одной части. В качестве примера можно привести Ладожское озеро, внутри которого есть острова. Другой класс сложных полигонов, это составные полигоны, состоящие более чем из одной части. Примеры таких полигонов: сухопутные части Василеостровского района, состоящего непосредственно из Васильевского острова, острова Декабристов и Серного острова; Невского района, расположенного на двух берегах Невы; Петроградского района, состоящего из Петроградской Стороны, островов Аптекарского, Елагина, Каменного, Крестовского, Петровского; Сахалинской области, состоящей из острова Сахалин и Курильских островов. Также полигоны подразделяются на односвязные и многосвязные. Между любыми двумя точками односвязного полигона можно провести линию без пересечения границы полигона. Так, Ладожское озеро является односвязным полигоном, так как из любой его точки до другой его точки можно добраться по воде без необходимости перетаскивания лодки или катера по суше. А вот сухопутная часть Василеостровского района является многосвязным полигоном: чтобы перейти с Васильевского острова на остров Декабристов, придётся по мосту перейти реку Смоленку. Объекты типа Polyline в шейпфайлах также могут быть составными.

Составные объекты могут появляться в результате выполнения оверлейных операций даже в том случае, если все объекты пересекаемых слоёв были простыми. Но из-за сложной формы этих объектов их сгенерированные «потомки» могут быть составными. Например, при пересечении линейного и полигонального слоёв

линейный объект несколько раз заходил внутрь полигона и выходил из него. Или границы водосборных бассейнов и ландшафтных таксонов извилисты. В результате полигон определённого ландшафтного таксона в пределах определённого водосборного бассейна может оказаться состоящим из нескольких частей.

В каких-то случаях наличие составных полигонов и линий не влияет на результаты пространственного анализа. Например, когда нужно узнать общую площадь верховых болот в бассейне реки Оредеж в Ленинградской области, то не важно, из какого числа частей состоит их полигон. Или, когда нужно рассчитать общую продолжительность плавания судна по Северному Морскому пути, можно не беспокоиться, из какого количества частей состоят линейные объекты участков маршрута плавания в однородных ледовых условиях. Но когда нужно моделировать конкретное плавание судна при разработке оперативных рекомендаций, то здесь необходимо отдельно рассматривать каждый простой объект маршрута плавания с однородными ледовыми условиями. Также и с первым примером: если планируется создание особо охраняемой территории для сохранения какого-либо эндемичного вида, то здесь уже необходимо учитывать площади отдельных болот, т.к. на слишком маленькой площади вид может не сохраниться.

При обработке результатов моделирования ледовой обстановки в Татарском проливе и анализе ледовой ситуации вдоль рекомендованных маршрутов возникла необходимость разбиения составных линейных объектов, получившихся в результате пересечения рекомендованных маршрутов плаваний с соответствующими полигональными слоями ледовой обстановки, на простые объекты. Ниже приводится часть кода разработанной для выполнения этой операции компьютерной программы. Здесь опущена часть программы, отвечающая за формирование имён шейпфайлов результатов пересечения. Адрес и название шейпфайла маршрута с ледовой информацией:

```
str_ice_route=str_dir+'\\Ice_routes\\Ice_route_'+str_data+'_18.shp'
```

Проверка существования этого шейпфайла:

```
if arcpy.Exists(str_ice_route):
  #Название и адрес шейпфайла результата
  str_ice_route_single=str_dir+'\\Single_ice_routes\\Single_ice_route_'
  +str_data+'_18.shp'
  if arcpy.Exists(str_ice_route_single):
    #Удаление "предшественника" в случае его существования
    arcpy.Delete_management(str_ice_route_single)
```

«Расчленение» составного объекта на простые:

```
arcpy.MultipartToSinglepart_management(str_ice_route,
str_ice_route_single)
```

Каждая отдельная часть составного объекта превратилась в простой объект в новом шейпфайле.

Задание. Создайте программу разбиения составных объектов на простые. Если таковые есть в шейпфайлах – результатах пересечения слоёв «route» и «polig», то используйте их. Если нет, то создайте извилистые объекты типа Polyline, выполните их пересечения со слоями «polig», и программным способом разбейте составные объекты в результатах этих пересечений на простые объекты.

1.8. Объединение объектов по значениям полей таблицы атрибутов

Часто бывает необходимым и выполнение противоположной операции: объединение пространственных объектов векторного слоя, имеющих одинаковые значения какого-либо поля таблицы атрибутов, или группы полей, или даже всех семантических полей таблицы. Например, нам нужно определить суммарные площади всех ландшафтных таксонов в пределах какой-либо административной единицы, или водосборного бассейна. Другой пример: нужно получить суммарную площадь льдов с разными характеристиками в пределах какой-либо акватории или буферной зоны маршрута плавания. Конечно, можно выполнить расчёты отдельно для каждого объекта, и затем сложить результаты. Но в случае автоматизированной обработки данных такой подход не является оптимальным. Логичнее сначала объединить все объекты, а уже потом рассчитывать площади результатов объединения. Также объединение необходимо при визуализации объектов по значениям какого-нибудь поля. Если мы предварительно не объединим объекты с одинаковыми значениями этого поля, то на карте останутся ненужные границы между объектами с одинаковой символикой, что снизит восприятие карты.

Перед выполнением объединения крайне желательно выполнить удаление возможных пространственных дубликатов, т.е. когда несколько объектов одного и того же слоя пространственно совпадают, следует оставить только один из них:

```
arcpy.DeleteIdentical_management(str_led_bufer, "Shape", "", "0")
```

Здесь «str_led_bufer» – строковая переменная, которая содержит адрес и название шейпфайла или другой файловой структуры, у которой нужно выполнить удаление пространственных дубликатов; "Shape" – поле таблицы атрибутов, по значениям которого нужно удалить

дубликаты. Фактически это означает сравнение пространственных объектов, выявление дубликатов и удаление в каждой группе дубликатов всех объектов кроме первого по порядку в таблице атрибутов. Следующая пустая строковая переменная «xy_tolerance» – это линейная единица, в которой должно рассчитываться расстояние между вершинами двух сравниваемых объектов. По умолчанию имеет пустое значение. Завершающее значение в скобках – "0". Это «z_tolerance» – допустимое различие высот вершин. Имеет смысл только для пространственных типов PolygonZ, PolylineZ, PointZ, вершины объектов которых имеют третью координату Z. Заметьте, что при удалении дубликатов редактируется исходный шейпфайл, а новый не создаётся.

Пример кода объединения пространственных объектов:

arcpy.Dissolve_management(str_led_bufer,str_diss_buf_led,"S;S_1;F_1_1; A_1_1;F_1_2; A_1_2;F_1_3;A_1_3;S_2;F_2_1;A_2_1;F_2_2;A_2_2; F_2_3;A_2_3;S_3;F_3_1;A_3_1;F_3_2;A_3_2;F_3_3;A_3_3; Dlina_nm;Area_km2", "", "MULTI_PART", "DISSOLVE_LINES")

Здесь «str_led_bufer» – строковая переменная с адресом и именем шейпфайла источника данных для объединения пространственных объектов, а «str_diss_buf_led» строковая переменная с адресом и именем результата объединения, который только будет создан. Разумеется, перед выполнением объединения значение этой переменной уже должно быть определено. Далее в двойных кавычках следует список полей, по значениям которых будет выполняться объединение пространственных объектов. Выполняется объединение объектов с одинаковыми значениями всех полей. Если на месте списка полей поставить пустую строку (""), то будет выполнено объединение абсолютно всех пространственных объектов. За списком полей в коде операции стоит значение параметра «statistics_fields», т.е. способа задания атрибутов объединяемых объектов. В случае, когда объединяются объекты с абсолютно одинаковыми значениями полей, по умолчанию ставится пустая строка, как в примере. Следующий параметр – «multi_part», указывающий на разрешение или запрещение многосвязных объектов (разрешение: «MULTI_PART», запрещение: «SINGLE_PART»). Завершающий параметр кода: «unsplit_lines», который определяет правило объединения линейных объектов. По умолчанию задаётся значение «DISSOLVE_LINES», дозволяющее объединение линий только в простые объекты. Другое возможное значение этого параметра – «UNSPLIT_LINES». Здесь объединение выполняется только тогда, когда две линии имеют общую вершину конца.

Задание. Создайте программу объединения объектов по значениям полей таблицы атрибутов. В качестве шейпфайла можете взять любой из уже созданных шейпфайлов, и добавить в его таблицу атрибутов новые поля.

1.9. Объединение векторных слоёв

При обработке и анализе данных в среде ГИС может понадобиться объединить слои. Например, эти слои созданы векторизацией соседних листов карты, или соседних спутниковых снимков. В идеальном варианте между этими соседними слоями не должно быть зазоров, а также наложений друг на друга. Если они будут, то наложения можно будет потом удалить с помощью удаления дубликатов, исправления геометрии и объединения объектов. **Очень важное требование**: таблицы атрибутов всех объединяемых слоёв (шейпфайлов или других файловых структур) должны быть абсолютно одинаковы! В противном случае в таблице атрибутов результата появятся поля, унаследованные от одного из «родителей», а в записях, произошедших от других «родителей», в этих полях будут пустые значения или значения «0» в числовых полях. Эта операция может быть полезна для временного анализа каких-либо величин. Допустим, необходимо выполнить анализ межгодовой изменчивости суммарных за год значений площадей лесных пожаров в каком-либо регионе, или суммарной протяжённости участков Северного морского пути с тяжёлыми ледовыми условиями плавания в первую декаду мая за ряд лет. В этом случае цель объединения слоёв заключается в получении таблицы атрибутов результата, которая будет содержать поле со значениями площадей лесных пожаров или протяжённостей маршрутов плавания в тяжёлых ледовых условиях за ряд лет. Таблицы атрибутов слоёв за отдельные годы здесь должны содержать только по одной записи, т.е. каждый слой отдельного года должен состоять только из одного пространственного объекта. Т.к. и число отдельных шейпфайлов при пространственном объединении соседей может быть значительным, и ряд лет может быть длинным, следует создать список строковых переменных, содержащих адреса и названия объединяемых шейпфайлов. Например, требуется выполнить анализ межгодовой изменчивости приведённой протяжённости Северного морского пути в старых льдах в первую декаду мая. Термин «приведённая протяжённость» означает, что общая протяжённость участка маршрута с наличием старых льдов умножается на частную концентрацию этих льдов в баллах (по 10-балльной российской шкале), и затем делится на

10. Затем все такие значения суммируются. В результате мы получаем условную (приведённую) протяжённость маршрута в одних старых льдах, т.е. льдах, которые пережили хотя бы одно летнее таяние.

Создаётся пустой список строковых переменных с адресами и названиями объединяемых шейпфайлов: lst_05_1_priv=[]. Также создается список строковых переменных – обозначений лет, т.к. номер года входит в названия шейпфайлов:

```
lst_God=['2011','2012','2013','2014','2015','2016','2017','2018','2019','2020','
2021']
```

Определяется число элементов в списке лет: n_let = len(lst_God).

Затем в список в цикле добавляются строковые переменные адресов и названий объединяемых шейпфайлов:

```
for i_god in range (n_let):
  tek_05_1=tek_god+"_05_1"
  #Заполнение списка шейпфайлов приведённых протяжённостей
  priv_05_1=str_dir+"\\Priv\\priv_"+tek_05_1+".shp"
  if arcpy.Exists(priv_05_1):
    lst_05_1_priv.append(priv_05_1)
```

В этом фрагменте кода «tek_05_1» – строковая переменная, которая содержит обозначение данного года и первой декады мая; «str_dir» – строковая переменная с путём к директории (папке), в которой находится папка «Priv» с шейпфайлами со значениями приведённой протяжённости за разные годы; «priv_05_1» – строковая переменная, содержащая путь и название к шейпфайлу с данными за конкретный год. Значения переменных «tek_05_1» и «priv_05_1» изменяются при каждом проходе цикла. После проверки существования шейпфайла строковое значение адреса и названия шейпфайла добавляется в список «lst_05_1_priv». Вне данного цикла в строковую переменную «tab_05_1» задаётся имя и адрес создаваемого шейпфайла, куда впоследствии будут добавлены все шейпфайлы из списка: tab_05_1=str_dir+"\\Tab\\tab_05_1.shp".

Наконец, при условии непустого списка priv_05_1 выполняется объединение шейпфайлов:

```
if len(lst_05_1_priv)>0:
  arcpy.Merge_management(lst_05_1_priv,tab_05_1)
```

Здесь функция «len» определяет длину списка. Синтаксис функции «Merge» («объединение») таков: в скобках на первом месте нужно указать список объединяемых шейпфайлов. Если их немного, то этот список можно указать непосредственно в этом операторе в квадратных скобках. На втором месте стоит строковая переменная с адресом и названием шейпфайла результата объединения. Третий,

необязательный параметр, который может стоять в скобках – «field_mappings», тип данных Field Mappings. Это набор полей и их содержания, выбранный для добавления в таблицу атрибутов результата. Создание такого объекта:
fieldMappings = arcpy.FieldMappings()
Затем в него добавляются таблицы из всех объединяемых шейпфайлов:
fieldMappings.addTable(<строковая переменная адреса и имени шейпфайла>)
Если объединяются поля с одинаковыми структурами таблиц атрибутов, этот параметр можно не указывать.

Задание. Создайте программу объединения векторных слоёв. Можно создать шейпфайл с одним пространственным объектом, записать его в несколько других шейпфайлов, и объединить их. При этом значения хотя бы одного поля в их таблицах атрибутов должны различаться.

1.10. Преобразование пространственных объектов типа Polyline в объекты типа PolylineM

Различие объектов типов Polyline и PolylineM заключается в следующем. В шейпфайле формата Polyline каждая вершина (вертекс по международной терминологии) имеет две пространственные координаты X и Y, однозначно определяющие его положение на плоскости электронной карты. Напоминаем, что линейные объекты состоят из последовательностей Line-объектов, т.е. отрезков прямых линий (сегментов), а вершины – это точки начала и конца каждого такого объекта. Разумеется, точка конца данного сегмента является точкой начала следующего сегмента, и координаты этой точки записываются в файле с расширением «shp» или другой файловой структуре только один раз. А в шейпфайле формата PolylineM к координатам X и Y добавляется третья координата M, представляющая собой суммарное расстояние вдоль маршрута от самого его начала до данной вершины в единицах карты, т.е. в тех же единицах, в которых измеряются значения координат X и Y на картах в прямоугольных системах координат. В России в большинстве случаев это метры. Заниматься же измерениями протяжённости линейных объектов на электронной карте в географической системе координат нет смысла, т.к. у такой карты единицы – градусы, а линейные размеры градуса широты и долготы сильно различаются для всех регионов Земли за исключением экваториальной области.

При мониторинге экологического состояния водотоков местоположение створов мониторинга часто указывается не с помощью географических или прямоугольных координат, а по расстоянию вниз или вверх по течению от какого-либо объекта, например, моста, места впадения притока и т.д. Аналогично для локализации мест аварий на автомобильных и железных дорогах указывается расстояние от начала дороги. Для описания интенсивности движения на автомобильных дорогах, и, соответственно, интенсивности загрязнения окружающей среды выхлопными газами дороги разбиваются на участки с относительно одинаковым уровнем движения и загрязнения. Эти участки локализуются указанием расстояний от начала и конца каждого такого участка до начала дороги. При моделировании распространения загрязняющих веществ по водотокам и оценке их экологического состояния они также подразделяются на относительно однородные участки, локализация которых производится так же, как и участков дорог – указанием расстояний от начала и конца участка до начала водотока (истока). Во всех этих случаях необходимо, чтобы линейные объекты, на которых выполняется локализация створов, аварий, участков дорог и водотоков, имели тип PolylineM.

При анализе плаваний в Татарском проливе в шейпфайлах результатов пересечений, т.е. шейпфайлах маршрутов с ледовой информацией образовывались составные линейные объекты, состоящие более чем из одной части. Было выполнено разбиение составных объектов на простые с созданием новых шейпфайлов простых линейных объектов с ледовыми характеристиками. Однако порядок расположения этих объектов в таблицах атрибутов слоёв (шейпфайлов) был совершенно произволен, он не соответствовал порядку движения судна по маршруту. Порядок расположения объектов в таблице атрибутов слоя – результата пересечения маршрута плавания и полигонального слоя ледовой информации определялся порядком расположения объектов в таблице атрибутов последнего. А этот порядок мог быть каким угодно. Поэтому возникла необходимость перестановки объектов слоя маршрута с ледовой информацией «по порядку»: от начальной точки маршрута до конечной. Для этого были выполнены следующие действия. Сначала шейпфайлы маршрутов плаваний с простыми пространственными объектами формата Polyline, но с произвольным расположением объектов были преобразованы в шейпфайлы пространственных объектов формата PolylineM. Ниже приводится фрагмент программного кода, выполняющий это преобразование:

```
#Адрес и название шейпфайла маршрута с ледовой информацией с простыми объектами
str_ice_route_single=str_dir+'\\Single_ice_routes\\Single_ice_route_'
+str_data+'_18.shp'
Проверка, а существует ли этот шейпфайл?
if arcpy.Exists(str_ice_route_single):
  #Адрес шейпфайла маршрута формата PolylineM
  str_ice_Mroute=str_dir+'\\Mroutes\\Ice_Mroute_'+str_data+'_18.shp'
Создание шейпфайла формата PolylineM:
  arcpy.CreateRoutes_lr(str_ice_route_single, "Id", str_ice_Mroute,
  "LENGTH", "", "", "UPPER_LEFT", "1", "0", "IGNORE", "INDEX")
```

В этом операторе переменная «str_ice_route_single» содержит адрес и название шейпфайла (или класса пространственных объектов базы геоданных), из которого будет создан шейпфайл (класс объектов базы геоданных) формата PolylineM; "Id" – название поля в таблице атрибутов исходного шейпфайла, которое содержит уникальный идентификатор каждого маршрута; «str_ice_Mroute» - переменная, содержащая адрес и название класса пространственных объектов, который будет создан (шейпфайл или класс объектов базы геоданных). Следующий параметр может принимать значения "LENGTH", "ONE_FIELD", "TWO_FIELDS". По умолчанию устанавливается значение "LENGTH". Это означает, что геометрические протяжённости исходных объектов будут суммироваться. При задании значения "ONE_FIELD" числовые величины, хранящиеся в одном поле, будут использованы для их суммирования. То есть при задании параметра "LENGTH" протяженность будет рассчитываться в единицах карты, для России это в большинстве случаев метры. Это не всегда бывает удобно, т.к. длины водотоков и дорог измеряются в километрах, а морских маршрутов плаваний в морских милях. Поэтому можно сначала создать в таблице атрибутов исходного шейпфайла (класса объектов) новое числовое поле, выполнить расчёты протяжённостей объектов в необходимых единицах длины, и записать результаты в это поле. При задании параметра "ONE_FIELD" значения третьей координаты M в вершинах (вертексах) линейных объектов будут записаны в этой единице. В случае задания параметра "TWO_FIELDS" для расчёта значений координаты M будут использоваться значения из двух числовых полей, в которых записаны расстояния от начала линейного объекта до начала участка на этом объекте, и от начала линейного объекта до конца участка на этом объекте. Следующий параметр – «from_measure_field». Это имя поля, в котором записаны протяжённости объектов в какой-либо числовой

мере. Его необходимо указывать только в тех случаях, когда источниками протяжённостей объектов заданы значения одного поля ("ONE_FIELD"), или двух полей ("TWO_FIELDS") таблицы атрибутов. При задании значения "LENGTH" на месте этого параметра ставится пустое строковое значение "", как в данном примере. Далее нужно задать значение параметра «to_measure_field» – имя поля, в котором записано второе значение протяжённости. Этот параметр указывается только в случае задания значения "TWO_FIELDS". В остальных случаях задаётся пустое строковое значение "". Следующий задаваемый параметр – «coordinate_priority», показывающий, где находится начало линейного объекта относительно углов прямоугольной электронной карты. Возможные значения: "UPPER_LEFT" (верхний левый), "LOWER_LEFT" (нижний левый), "UPPER_RIGHT" (верхний правый), "LOWER_RIGHT" (нижний правый). В приводимом примере задано значение "UPPER_LEFT", поскольку рекомендуемые маршруты плаваний начинались на рейде порта Де-Кастри на западном берегу Татарского пролива, в его северной части, а заканчивались на чистой воде в южной части пролива. Затем устанавливается значение параметра «measure_factor» – множителя, на который нужно умножать значения протяжённостей. По умолчанию задаётся равным 1. Следующий параметр «measure_offset» – величина, добавляемая к рассчитанным протяжённостям перед определением значений координаты M. По умолчанию – 0. Далее следует указать значение параметра «ignore_gaps», показывающего, следует ли игнорировать возможные разрывы линейного объекта. Это параметр применяется только в случаях задания в качестве источников протяжённостей "LENGTH" или "ONE_FIELD". По умолчанию задаётся значение "IGNORE", т.е. протяжённости разрывов не учитываются. Если задать " NO_IGNORE", то протяжённости разрывов будут учитываться при расчёте протяжённости объектов. При этом протяжённость разрыва будет определяться по прямой, соединяющей точки на концах разрыва. Значение завершающего параметра «build_index» указывает, нужно ли создавать в таблице атрибутов результата операции поле идентификатора созданных объектов. По умолчанию задаётся значение "INDEX" – создать. Если поле идентификатора создавать не нужно, то ставится " NO_INDEX".

В результате выполнения оператора будет создан один линейный объект формата PolylineM с двумя семантическими полями в таблице атрибутов: «FID» и «ID».

Задание. Создайте программу преобразования слоёв пространственных объектов типа Polyline в слои объектов типа

PolylineM и протестируйте её.

1.11. Превращение вершин линий и границ полигонов в точечные объекты

Ранее, в разделе 1.4 было рассмотрено создание линейных объектов из каких-либо последовательностей точек. Возможна необходимость и противоположного действия: превращения вершин объектов Polyline и PolylineM в точечные объекты. Потребность в этом может возникнуть, например, при выделении водосборных бассейнов. Для этого нужно создать цифровую модель рельефа, т.е. матрицу высот. Допустим, есть векторные объекты типа Polyline – горизонтали, т.е. линии равных высот. Но горизонтали при интерполяции использовать невозможно: нужны точки. Для этого можно из вершин горизонталей сделать точки. Другой близкий пример: интерполяция глубин озера. Предположим, есть достаточное количество точек измерения глубины. Но нужно ещё добавить точки с нулевой глубиной на урезе воды. Здесь также пригодятся превращённые в точки вершины береговой линии. Заметим, что эти проблемы можно решить и другим способом: превратить горизонтали или береговую черту (для водоёма это тоже горизонталь) в объекты типа PolylineM, и расставить вдоль них точки как объекты маршрутного слоя событий с любой необходимой частотой, чтобы точки глубин на акватории и на береговой линии были расположены по возможности равномерно.

При анализе плаваний в Татарском проливе генерация точек на основе вершин линейных объектов была выполнена как первый этап создания Line-объектов – отрезков прямых. Оператор создания класса точечных объектов из вершин линий таков:

```
#Задание места и названия создаваемого шейпфайла точек из вертексов маршрута
str_Vert_Points=str_dir+'\\VertPoints\V_Points_'+str_data+'_18.shp'
arcpy.FeatureVerticesToPoints_management(str_ice_Mroute,
str_Vert_Points, "ALL")
```

Здесь переменная «str_ice_Mroute» содержит адрес и название шейпфайла маршрута типа PolylineM, «str_Vert_Points» – адрес и название создаваемого класса объектов точки (в данном случае это шейпфайл), последний параметр в скобках «point_location» устанавливает, где будут создаваться точки. По умолчанию задаётся "ALL": каждая вершина порождает точку. Другие возможные значения: «MID» – точки создаются на серединах сегментов, «START» – точки из начальных вершин объектов, «END» – точки из конечных

вершин объектов, «BOTH_ENDS» точки из начальных и конечных вершин объектов, «DANGLE» – точки создаются из всех начальных и конечных точек сегментов линии, если они не являются местами соединения с другими линейными объектами. Эта опция не может применяться в случае превращения в точки вершин на границах полигонов.

Задание. Напишите программу создания шейпфайлов точечных объектов из вершин линейных объектов и протестируйте её.

1.12. Цепочка действий по упорядочиванию последовательности линейных объектов

Представим себе следующую задачу: необходимо расставить по порядку объекты типа Polyline слоя, полученного в результате пересечения исходного слоя типа Polyline и слоя полигонов. Это может быть пересечение планируемого маршрута плавания судна или каравана судов под ледокольной проводкой и слоя ледовой обстановки (возможно, результата моделирования динамики льда в соответствии с метеорологическим прогнозом), или пересечение проектируемого линейного объекта (железной или автомобильной дороги, трубопровода, туннеля, линии электропередач, кабельной линии, линии связи, линии метрополитена или водного канала) с полигональными слоями ландшафтных, биогеографических, почвенных таксонов, со слоями электронной геологической карты и т.п. Однако в атрибутивной таблице результата пересечения, т.е. слоя линейных объектов порядок записей, и соответственно порядок самих объектов может быть совершенно произволен. Если мы видим на электронной карте состоящий из множества объектов маршрут из пункта А в пункт Б, то совершенно необязательно, чтобы начинающийся в пункте А объект был первым, самым верхним в таблице атрибутов. Порядок может быть самый произвольный, например, сначала идёт седьмой по пути маршрута объект, потом двадцатый, потом первый, потом восьмой и т.д. Очевидно, что в случае планирования морской транспортной операции только для коротких маршрутов можно пренебречь действительным порядком участков маршрута и рассчитать время прохождения судном (караваном судов) каждого участка, а затем их просуммировать и получить общие затраты времени на прохождение маршрута. Это возможно только тогда, когда ожидаемое время прохождения маршрута не превышает времени метеорологического прогноза, на основании которого выполнялся модельный расчёт изменения

состояния ледяного покрова. Понятно, что наибольшей оправдываемостью обладают краткосрочные прогнозы на доли суток и сутки. С увеличением срока прогноза его оправдываемость существенно снижается. Поэтому линейные объекты вдоль протяжённого маршрута, например, Северного морского пути необходимо расставить по порядку их следования от пункта выхода судна, например, Сабетты. Оперативная поддержка морской транспортной операции должна выполняться итерационно: сначала по долгосрочному прогнозу ледовой обстановки разрабатывается генеральный маршрут плавания, причём его самый первый участок строится по краткосрочному прогнозу метеорологической обстановки, и он должен быть тщательно выверен. Ко времени окончания прохождения судном этого первого участка маршрута на основании следующего краткосрочного прогноза метеорологической обстановки должны быть определены ожидаемые изменения ледовой обстановки на следующем участке маршрута, и построен уточнённый маршрут плавания на следующем отрезке пути, и так далее до окончания прохождения всего пути плавания в условиях ледяного покрова. Для морских операций вне ледяного покрова аналогичный алгоритм оперативной поддержки объясняется необходимостью последовательного учёта краткосрочных прогнозов интенсивности и направления ветра, параметров волнения.

Возведение линейных объектов редко начинается сразу по всему их протяжению. При создании туннеля и прокладке подводного трубопровода или кабеля это просто невозможно. Обычно работы планируют и выполняют начиная с одного или обоих концов создаваемого линейного объекта. Так, например, в «Золотом телёнке» описан завершающий этап строительства Туркестано-Сибирской железной дороги. Работы проводились с помощью двух поездов, двигавшихся навстречу друг другу по вновь прокладываемым рельсам. Поэтому во многих случаях для планирования работ необходимо расставить по порядку объекты создаваемого линейного сооружения с атрибутами пересекаемых им полигональных слоёв (геологического строения, ландшафтов, растительности, почв, характеристик грунтов и т.д.). Разумеется, в ряде случаев порядок следования объектов в таблице атрибутов не важен. Например, если необходимо определить суммарный ущерб для окружающей среды от всего сооружения в целом за время его строительства и/или во время эксплуатации. Но если требуется рассчитать ущерб для окружающей среды по отдельным временным этапам строительства, то здесь также придётся расставить участки сооружения по времени их возведения, т.е. по

порядку вдоль всего линейного слоя.

Итак, сформулируем задачу. Необходимо создать слой объектов типа Polyline или PolylineM, в таблице атрибутов которого соответствующие пространственным объектам записи должны следовать строго по порядку расположения пространственных объектов вдоль линии от её начального узла до конечного. При этом объекты линейного слоя должны обладать семантическими характеристиками пересечённых ими полигональных слоёв (слоя).

Исходный слой линейных объектов типа Polyline имеет произвольный порядок расположения объектов в таблице атрибутов. Объекты могут быть как односвязными, так и многосвязными: нет необходимости их разбиения на односвязные. Его следует конвертировать в слой из единственного объекта типа PolylineM. Для этого в качестве поля идентификаторов маршрутов (Route Identifier Field) следует указать поля с одинаковыми значениями во всех записях таблицы атрибутов. В приведённом ниже фрагменте программного кода это "Id":

```
arcpy.CreateRoutes_lr(str_ice_route_single, "Id", str_ice_Mroute,
"LENGTH", "", "", "UPPER_LEFT", "1", "0", "IGNORE", "INDEX")
```

(Смотрите раздел 1.8).

В результате выполнения этого фрагмента кода будет создан слой с одним объектом пространственного типа PolylineM. Внимательный читатель может спросить: «Зачем же мы перед этим выполнили пересечение слоя маршрута с полигональным слоем (или даже несколькими слоями), если все унаследованные созданным в результате этого пересечения линейным слоем семантические характеристики полигонального слоя исчезли при его конвертации в объект типа PolylineM?» Да, они исчезли, зато сохранились вершины на местах пересечения маршрутом границ полигональных объектов, и это даст нам потом возможность создать упорядоченную последовательность линейных объектов.

Далее выполняем превращение вершин объекта PolylineM в точки слоя типа PointM (см. раздел 1.9). Таблица атрибутов созданного слоя будет обладать замечательным свойством: связанные с записями точки будут расположены по порядку движения от узла начала линии к конечному узлу. Также в таблице атрибутов появится поле «FID», в котором все записи будут пронумерованы, начиная с 0. Однако извлечь данные из этого поля в программе на языке Python невозможно. Поэтому придётся добавить в таблицу атрибутов новое поле самим и заполнить его номерами записей (строк) таблицы.

1.12.1. Добавление нового поля в таблицу атрибутов

В разделе 1.6 был рассмотрен алгоритм автоматизированного удаления из таблиц атрибутов лишних, уже ненужных полей для экономии памяти компьютера. Возможна и противоположная потребность автоматизации добавления новых полей в таблицы атрибутов шейпфайлов или других используемых файловых структур. Может возникнуть необходимость нумерации записей (строк) в таблице атрибутов, выполнения картометрических операций и записи их результатов в новые поля таблицы. В нашем примере рассмотрим добавление поля «Npoints» целых числовых значений в таблицу атрибутов объектов типа PointM, расположенных вдоль какого-либо маршрута:

```
arcpy.AddField_management(str_Vert_Points, "Npoints", "SHORT", "", "", "", "", "NULLABLE", "NON_REQUIRED", "")
```

В приведённом фрагменте программного кода «str_Vert_Points» – строковая переменная, содержащая название шейпфайла (включающее полный путь к нему), в таблицу атрибутов которого будет добавлено новое поле «Npoints»; "SHORT" – тип поля, в данном случае это короткое целое число. Далее в примере кода расположены четыре пустые строковые значения. Эти переменные – опциальные, т.е. их не обязательно заполнять. Первое из них – возможное число цифр в поле. Второе – число знаков после запятой. Разумеется, его можно заполнять только в том случае, когда тип поля «FLOAT» или «DOUBLE». Третье – длина поля, заполняется только в случае добавление поля типа «TEXT». Четвёртое – «Field Alias», т.е. альтернативное название создаваемого поля. Далее стоит "NULLABLE", что означает допуск пустых значений поля. Альтернативный вариант, т.е. запрет этого – "NON_NULLABLE". Затем в примере кода расположено значение "NON_REQUIRED", что разрешает отсутствие значений в добавляемом поле. Естественно, что значения "NULLABLE" и "NON_REQUIRED" должны устанавливаться совместно. Противоположное значение " REQUIRED" требует обязательного заполнения поля. Завершающий параметр кода, в данном примере принимающий значение пустой строки – это «Field Domain», т.е. диапазон возможных значений поля или список его возможных значений. Этот параметр можно задавать только в случае добавления нового поля в таблицу атрибутов класса пространственных объектов базы геоданных. Возможные типы добавляемых объектов: «TEXT» – текстовый, «FLOAT» – число с плавающей запятой, «DOUBLE» – число двойной точности с плавающей запятой, «SHORT» – короткое

целое число, «LONG» – длинное целое число, «BLOB» – двоичный большой объект, может содержать изображения, аудио- и видеоинформацию и т.д., «RASTER» – растровые изображения, «GUID» – статистически уникальный 128-битный идентификатор.

1.12.2. Заполнение поля таблицы атрибутов

Добавленное в таблицу атрибутов поле нужно заполнить информацией. Также может возникнуть необходимость переопределения значений уже существующего поля. Для добавления значений в какое-либо поле таблицы нужно просмотреть все записи (строки) этой таблицы и в каждой из них занести значение в столбец данного поля. Перед этим необходимо создать объект типа курсор. Заметим, что курсор невозможно создать «вручную» с помощью интерфейсов приложений ArcMap или ArcCatalog, или с помощью модели рабочего потока. Ниже приводится код создания курсора:

```
cursor = arcpy.UpdateCursor(str_Vert_Points)
```

В этом примере «str_Vert_Points» – строковая переменная, содержащая полный адрес размещения и название шейпфайла, таблица атрибутов которого должна просмотрена. Затем в программе помещается цикл просмотра таблицы:

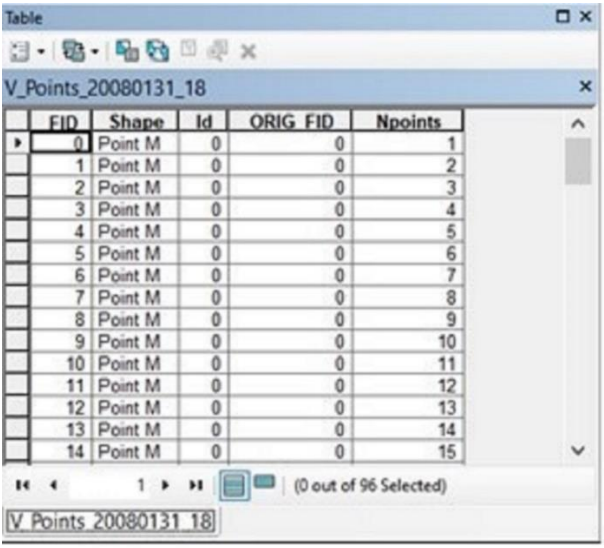
```
n=0
for row in cursor:
    n=n+1
    row.setValue("Npoints", n)
    cursor.updateRow(row)
```

В данном примере выполняется заполнение поля «Npoints» номерами записей (строк таблицы), начиная с 1. Поэтому перед началом цикла стоит оператор присвоения числовой переменной n значения 0. Разумеется, в других случаях этот оператор не нужен. Сам цикл задаётся оператором «for row in cursor:», который означает: «Просмотреть каждый ряд (строку) таблицы, для которой создан курсор cursor». В первом операторе внутри цикла значение переменной n увеличивается на единицу. Естественно, что этот оператор необходим только в данном случае. Второй оператор «row.setValue("Npoints", n)» представляет собой команду «Записать в данной строке просматриваемой таблицы величину из переменной n в поле "Npoints"». При просмотре таблицы атрибутов возможна и противоположная операция: получение значения из какого-нибудь поля таблицы и его запись в переменную. Синтаксис этого оператора: value=row.getValue("<имя поля>"). Здесь «value» – переменная, куда

будет записано значение из поля в данной строке. Разумеется, обозначение переменной может быть произвольным. Следующий оператор в нашем примере «cursor.updateRow(row)» означает команду «Записать изменения в данной строке». Этот оператор должен быть завершающим оператором цикла просмотра и изменения таблицы атрибутов. Сразу за ним должен стоять оператор удаления курсора:

del cursor, row

В нашем примере после добавления в таблицу атрибутов слоя точек типа PointM поля «Npoints» и его заполнения номерами строк таблицы, т.е. номерами точек вдоль маршрута таблица атрибутов приобрела вид, представленный на рисунке 6.



FID	Shape	Id	ORIG_FID	Npoints
0	Point M	0	0	1
1	Point M	0	0	2
2	Point M	0	0	3
3	Point M	0	0	4
4	Point M	0	0	5
5	Point M	0	0	6
6	Point M	0	0	7
7	Point M	0	0	8
8	Point M	0	0	9
9	Point M	0	0	10
10	Point M	0	0	11
11	Point M	0	0	12
12	Point M	0	0	13
13	Point M	0	0	14
14	Point M	0	0	15

Рисунок 6. Таблица атрибутов после добавления поля «Npoints» и его заполнения

1.12.3. Пространственное соединение двух слоёв

Вспомним цель: нужно создать слой линейных объектов, в котором расположение соответствующих им записей в таблице атрибутов упорядочено. Сейчас у нас есть слой простых объектов типа Polyline маршрута плавания, в таблице атрибутов которого записана ледовая информация, но порядок записей в таблице не соответствует

порядку расположения соответствующих записям объектов вдоль маршрута. И ещё есть слой точек типа PointM с их упорядоченным расположением вдоль маршрута. Следовательно, нужно добавить в таблицу атрибутов поля точек недостающую информацию из таблицы атрибутов слоя линейных объектов. Здесь мы должны выполнить часто встречающуюся при работе в среде ГИС и систем управления базами данных (СУБД) операцию соединения таблиц, когда в одну таблицу добавляются данные из другой таблицы. В ГИС это может быть выполнено двумя путями: соединением по атрибуту или соединением по расположению. В первом случае в обеих таблицах атрибутов должны быть поля с одинаковой информацией, по которым устанавливается взаимосвязь между таблицами. Эта взаимосвязь может быть типов «один к одному», «один ко многим», «многие к одному». В первой таблице есть поле с какой-то информацией, и во второй – с этой же информацией. Устанавливается связь между записями двух таблиц, имеющими одинаковые значения в этих полях. Затем в каждую запись первой таблицы заносится информация из связанной с этой записью записи второй таблицы. Такой вариант соединения возможен не только для таблиц атрибутов шейпфайлов и других файловых структур, содержащих пространственную информацию, но и вообще для таблиц реляционного формата. Но в нашем примере соединение по атрибуту невозможно, т.к. в таблицах атрибутов обоих слоёв нет полей с одинаковой информацией. Поэтому нужно создать соединение по расположению, когда связи устанавливаются между расположенными в одном и том же месте пространственными объектами. В примере это означает, что устанавливаются связи между линейными объектами и касающимися их точками. Ниже приведён пример программного кода пространственного присоединения:

```
arcpy.SpatialJoin_analysis(str_ice_route_single, str_Vert_Points,
str_join_route)
```

Синтаксис этого оператора понятен: «str_ice_route_single» – строковая переменная с адресом размещения и названием шейпфайла простых объектов типа Polyline маршрута плавания с ледовыми характеристиками; «str_Vert_Points» – строковая переменная с адресом размещения и названием шейпфайла объектов типа PointM с их упорядоченным расположением вдоль маршрута; «str_join_route» строковая переменная с адресом размещения и названием шейпфайла результата соединения. В случае такого кода в таблице атрибутов результата соединения будут записаны все поля таблиц атрибутов обоих соединяемых шейпфайлов. При этом в таблице слева будут

расположены поля из таблицы атрибутов первого указанного в скобках шейпфайла (в примере это «str_ice_route_single»), а справа – второго («str_Vert_Points»). Если в таблицах атрибутов соединяемых шейпфайлов есть совпадения имён полей, в к именам таких полей из таблицы атрибутов второго шейпфайла будут добавлены символы «_1». Например, в таблице атрибутов результата будут поля «Id» и «Id_1». Такое добавление абсолютно всех полей приводит к наличию в таблице атрибутов результата соединения ненужных полей, что увеличивает затраты памяти компьютера и тормозит его работу. Эти поля можно удалить уже рассмотренным способом (см. раздел 1.6). Другой подход: указать в программном коде все поля, которые нужно добавить в таблицу атрибутов результата:

```
arcpy.SpatialJoin_analysis(str_ice_route_single, \
str_Vert_Points, str_join_route, "JOIN_ONE_TO_ONE","KEEP_ALL", \
"CT \"CT\" true true false 2 Text 0 0 ,First,#,str_ice_route_single,CT,-1,-1; \
CA \"CA\" true true false 2 Text 0 0 ,First,#,str_ice_route_single,CA,-1,-1;\
CB \"CB\" true true false 2 Text 0 0 ,First,#,str_ice_route_single,CB,-1,-1;\
CC \"CC\" true true false 2 Text 0 0 ,First,#,str_ice_route_single,CC,-1,-1;\
SA \"SA\" true true false 2 Text 0 0 ,First,#,str_ice_route_single,SA,-1,-1;\
SB \"SB\" true true false 2 Text 0 0 ,First,#,str_ice_route_single,SB,-1,-1;\
SC \"SC\" true true false 2 Text 0 0 ,First,#,str_ice_route_single,SC,-1,-1;\
CN \"CN\" true true false 2 Text 0 0 ,First,#,str_ice_route_single,CN,-1,-1;\
CD \"CD\" true true false 2 Text 0 0 ,First,#,str_ice_route_single,CD,-1,-1;\
FA \"FA\" true true false 2 Text 0 0 ,First,#,str_ice_route_single,FA,-1,-1;\
FB \"FB\" true true false 2 Text 0 0 ,First,#,str_ice_route_single,FB,-1,-1;\
FC \"FC\" true true false 2 Text 0 0 ,First,#,str_ice_route_single,FC,-1,-1;\
FP \"FP\" true true false 2 Text 0 0 ,First,#,str_ice_route_single,FP,-1,-1;\
FS \"FS\" true true false 2 Text 0 0 ,First,#,str_ice_route_single,FS,-1,-1;\
POLY_TYPE \"POLY_TYPE\" true true false 1 Text 0 0  ,First,#,\
str_ice_route_single,POLY_TYPE,-1,-1;\
C \"C\" true true false 8 Float 1 7 ,First,#,str_ice_route_single,C,-1,-1;\
Ca_thk \"Ca_thk\" true true false 8 Float 1 7 , \
First,#,str_ice_route_single,Ca_thk,-1,-1;\
Ca_mid \"Ca_mid\" true true false 8 Float 1 7 , \
First,#,str_ice_route_single,Ca_mid,-1,-1;\
Ca_thn \"Ca_thn\" true true false 8 Float 1 7 , \
First,#,str_ice_route_single,Ca_thn,-1,-1;\
Ca_yng \"Ca_yng\" true true false 8 Float 1 7 , \
First,#,str_ice_route_single,Ca_yng,-1,-1;\
Fld_thk \"Fld_thk\" true true false 3 Text 0 0 , \
First,#,str_ice_route_single,Fld_thk,-1,-1;\
```

```
Fld_mid \"Fld_mid\" true true false 3 Text 0 0 , \
First,#,str_ice_route_single,Fld_mid,-1,-1;\
Fld_thn \"Fld_thn\" true true false 3 Text 0 0 , \
First,#,str_ice_route_single,Fld_thn,-1,-1;\
Fld_yng \"Fld_yng\" true true false 3 Text 0 0 , \
First,#,str_ice_route_single,Fld_yng,-1,-1;\
Tk_thk \"Tk_thk\" true true false 12 Double 2 11 , \
First,#,str_ice_route_single,Tk_thk,-1,-1;\
Tk_mid \"Tk_mid\" true true false 12 Double 2 11 , \
First,#,str_ice_route_single,Tk_mid,-1,-1;\
Tk_thn \"Tk_thn\" true true false 12 Double 2 11 , \
First,#,str_ice_route_single,Tk_thn,-1,-1;\
Tk_yng \"Tk_yng\" true true false 12 Double 2 11 , \
First,#,str_ice_route_single,Tk_yng,-1,-1;\
Fast \"Fast\" true true false 10 Double 0 10 , \
First,#,str_ice_route_single,Fast,-1,-1;\
Age \"Age\" true true false 2 Text 0 0 ,First,#, \
str_ice_route_single,Age,-1,-1;\
HUM_FLD \"HUM_FLD\" true true false 8 Float 1 7 , \
First,#,str_ice_route_single,HUM_FLD,-1,-1;\
COMPACT_FL \"COMPACT_FL\" true true false 8 Float 1 7 , \
First,#,str_ice_route_single,COMPACT_FL,-1,-1;\
Npoints \"Npoints\" true true false 4 Short 0 4 , \
First,#,str_Vert_Points,Npoints,-1,-1", "INTERSECT", "", "")
```

Как видно, при большом числе полей такой подход требует написания довольно громоздкого кода. Его грамматика такова: сначала идут строковые переменные с адресами и названиями файлов источников данных для соединения, затем строковая переменная с адресом и названием файла результата, как в уже приведённом примере кода; параметр «тип операции», в примере "JOIN_ONE_TO_ONE" – задание типа соединения «один к одному», альтернативный вариант – "JOIN_ONE_TO_MANY", т.е. соединение типа «один ко многим»; параметр «тип присоединения», в примере "KEEP_ALL" – т.е. в таблицу атрибутов результата добавляются все записи обеих соединяемых таблиц, даже те, для которых нет связей с записями в другой таблице. В последнем случае в полях, добавленных из этой другой таблицы ставятся пустые значения или нули в числовых полях. Такой тип присоединения задаётся по умолчанию. Альтернативный тип присоединения "KEEP_COMMON" задаёт условие добавления в таблицу атрибутов результата только тех записей исходных таблиц, которые имеют между собой связи. Далее в коде следуют параметры

добавляемых полей. Сначала указывается имя добавляемого поля (именно так, дважды, с косыми чертами и символами двойных кавычек), затем параметры field_mapping, устанавливающие правила добавления полей из исходных таблиц в таблицу результата. Поля могут добавляться, удаляться, переименовываться, их характеристики могут изменяться. В этом же блоке задаются тип и размеры полей в таблице атрибутов результата, и указывается источник данных, т.е. строковая переменная с адресом и названием шейпфайла или другой файловой структуры. По окончании этого блока задаётся значение параметра match_option – критерия для выбора соединяемых объектов. Этих критериев 17, в нашем примере стоит "INTERSECT", т.е. пресечение объектов. Завершающие параметры: это радиус поиска (search_radius) и расстояние между соединяемыми объектами (distance_field_name). В тех случаях, когда объекты должны пересекаться или касаться друг друга, в коде ставятся пустые строковые значения, как в примере. Чтобы не ошибиться при создании кода, лучше создать модель рабочего потока, в интерфейсе задания её параметров отметить все добавляемые в таблицу атрибутов результата поля исходных слоёв, и сохранить модель в файл программы на языке Python с расширением «ру». Далее открыть этот файл в приложении ArcMap (Geoprocessing, Python), скопировать программный код и вставить его в создаваемую программу. Разумеется, этот код нужно отредактировать: заменить в нём обозначения исходных слоёв и слоя результата на строковые переменные, содержащие адреса и названия соответствующих шейпфайлов или иных файловых структур.

В таблице атрибутов шейпфайла результата соединения теперь у каждого линейного объекта есть номер пространственно совпавшей с ним точки. В общем случае с линейным объектом могли совпасть не одна, а несколько точек. Но поскольку в качестве типа соединения было задано правило «один к одному», то в таблицу атрибутов результата добавлялся номер той из нескольких точек, которая расположена в таблице атрибутов слоя точек ранее, т.е. выше других. Чтобы расставить по порядку линейные объекты, теперь остаётся только отсортировать его таблицу атрибутов по номерам точек.

1.12.4. Сортировка таблицы атрибутов по значениям слоя

Пример программного кода сортировки таблицы атрибутов:

```
arcpy.Sort_management(str_join_route, str_sort_route, "Npoints
ASCENDING", "UR")
```

Здесь «str_join_route» – адрес и название шейпфайла, таблицу атрибутов которого необходимо отсортировать; «str_sort_route» – адрес и название шейпфайла с результатом сортировки; "Npoints ASCENDING" – имя поля, по значениям которого будет выполняться сортировка таблицы (Npoints), и тип сортировки, в примере это сортировка по увеличению значения поля (ASCENDING). Сортировка таблицы по уменьшению значений: DESCENDING. Завершающий параметр кода: метод пространственной сортировки (Spatial Sort Method). Этот параметр учитывается, если поле «Shape» выбрано в качестве одного из полей, по значениям которого выполняется сортировка. По умолчанию значение параметра – "UR", что означает начало сортировки с объекта, расположенного в правом верхнем углу карты. Если поле «Shape» не участвует в сортировке таблицы атрибутов, то можно оставить это значение параметра: выбор нисколько не влияет на сортировку. Другие возможные значения параметра: "UL" – начало сортировки с объекта, расположенного в верхнем левом углу карты; "LR" – с расположенного в нижнем правом углу карты; "LL" – с нижнего левого угла карты; "PEANO" – сортировка использует специальный алгоритм пространственного заполнения кривой. Это последовательный поиск ближайших объектов, начиная от первого объекта.

Поскольку на соединение с линейным объектом могло претендовать более одной точки, а для каждого линейного объекта было присоединено только одно значение поля "Npoints", то в отсортированной таблице атрибутов записи расположены строго по возрастанию поля "Npoints", но сам числовой ряд значений этого поля имеет пробелы. Чтобы пронумеровать все участки маршрута, следует добавить в таблицу атрибутов целочисленное поле (например, "N_zone"), и заполнить его номерами линейных объектов точно таким же способом, как было заполнено номерами точек поле "Npoints" таблицы атрибутов слоя точек (1.11.2).

Задание. Напишите программу расстановки объектов слоя полилиний по порядку.

1.13. Картометрические операции

При выполнении НИР в сфере экологии и природопользования часто необходимо выполнить расчёты протяжённостей и площадей пространственных объектов. При планировании морской транспортной операции нужно рассчитать протяжённости участков маршрута плавания с различными условиями.

При проектировании сооружения линейных объектов (трубопроводов, дорог, линий электропередач и т.д.) следует определить протяжённости участков с разными геологическими и геоморфологическим условиями. При оценке ожидаемого ущерба окружающей среде необходимы определения площадей антропогенного воздействия на разные ландшафтные таксоны, участки с различной растительностью и т.д. При планировании особо охраняемых территорий и акваторий следует учитывать, что любой вид для своего устойчивого сохранения должен иметь определённую численность, и соответствующую этой численности площадь местообитания. Это требует выполнения расчётов площадей отдельных участков с определёнными ландшафтными и биогеографическими характеристиками. Ниже приведён пример программного кода определения протяжённостей участков маршрута плавания.

Сначала в таблицу атрибутов добавляется новое поле «Length_nm», в которое будут записаны протяжённости участков плавания в морских милях:

```
arcpy.AddField_management(str_sort_route, "Length_nm", "FLOAT", "6", "2", "", "", "NULLABLE", "NON_REQUIRED", "")
```

Здесь «str_sort_route» – строковая переменная, содержащая адрес и название шейпфайла, в таблицу атрибутов которого будет добавлено поле "Length_nm"; "FLOAT" – тип поля: числовое с плавающей запятой (точкой); "6" – общее число позиций этого поля; "2" – число дробных разрядов. Т.е. в данном примере предусмотрены три разряда до точки и два после. Далее в примере кода следуют две пустые строковые переменные. Первый из этих параметров («field_length») – максимальное число символов, которое может быть помещено в поле. Этот параметр может применяться только для полей типов TEXT и BLOB. Следующая пустая строка зарезервирована для параметра «field_alias» – альтернативного имени поля. Этот параметр может применяться только в базах геоданных и покрытиях. Далее идёт параметр «field_is_nullable», разрешающий или запрещающий пустые значения поля. В этом примере значение "NULLABLE" разрешает пустые значения поля. Следующий параметр «field_is_required» определяет, является ли создаваемое поле требующимся для данной таблицы. Этот параметр поддерживается для полей баз геоданных, но и при добавлении поля в таблицу атрибутов шейпфайла его значение следует задать. В примере значение параметра "NON_REQUIRED" означает, что это поле не является требующимся. Данное значение устанавливается по умолчанию. Наконец, завершающая пустая строка

в примере кода – это «field_domain», т.е. имя домена, устанавливающего ограничения для возможных значений поля. Домены применяются только в базах геоданных.
Затем выполняется сам расчёт протяжённостей линейных объектов:
arcpy.CalculateField_management(str_sort_route,"Length_nm",
"!shape.length@nauticalmiles!", "PYTHON_9.3")
Здесь "Length_nm" – имя поля таблицы атрибутов, в которое будут записаны результаты расчётов, "!shape.length@nauticalmiles!" – картометрическая операция, означающая расчёты протяжённостей пространственных объектов, ссылки на которые записаны в поле «Shape» таблицы атрибутов, в морских милях. "PYTHON_9.3" – применяемая в ArcGIS версия языка Python. Разумеется, возможны расчёты протяжённостей линейных объектов в других единицах, например, в километрах или метрах: "!shape.length@kilometers!" или "!shape.length@meters!". Также возможны расчёты протяжённостей в миллиметрах, сантиметрах, дюймах, футах, ярдах и милях США. При расчётах площадей объектов прежде следует добавить в таблицу атрибутов поле для записи результатов «Zona_km2» – площади в квадратных километрах:
arcpy.AddField_management(str_diss_buf_led, "Zona_km2", "FLOAT",
"10", "3", "", "", "NULLABLE", "NON_REQUIRED", "")
Затем выполняется сам расчёт:
arcpy.CalculateField_management(str_diss_buf_led, "Zona_km2",
"!shape.area@squarekilometers!", "PYTHON_9.3")
Как видно, различие с расчётом протяжённости заключается только в обозначении картометрической операции. Необходимо заметить, что расчёты при картометрических операциях выполняются сразу для всех объектов, и здесь не нужно организовывать цикл просмотра всех записей таблицы атрибутов.

1.14. Математические операции

А вот в случае выполнения математических операций со значениями в полях таблицы атрибутов без выполнения цикла просмотра всех её записей (строк) не обойтись. Рассмотрим следующий пример. Необходимо выполнить анализ средней протяжённости плавания по стандартному маршруту Северного морского пути в пределах ледовых зон с различными условиями. Для этого сначала создаётся буферная зона вокруг маршрута плавания шириной 20 км, т.к. 10 км – это предельное расстояние от судна, в пределах которого судоводитель с мостика может визуально оценивать

ледовую ситуацию, и с плоскими концами зоны. Слой буферной зоны состоит из одного полигонального объекта. Затем выполняется расчёт площади буферной зоны в квадратных километрах, и его результат заносится в поле «Area_km2». Далее с помощью нескольких вложенных циклов по элементам списков лет, месяцев, декад и половин декад определяются строковые переменные адресов и названий шейпфайлов ледовой информации. В цикле же выполняются операции пересечения слоёв буферной зоны маршрута плавания и ледовой информации. Затем в таблицу атрибутов шейпфайла результата пересечения добавляется числовое поле «Zona_km2», и выполняется расчёт площади каждого объекта, т.е. каждой зоны с уникальными ледовыми характеристиками в пределах буферной зоны маршрута (см. раздел 1.13). Далее в таблицу атрибутов добавляется числовое поля «Dolya», предназначенное для записи отношение площади каждого объекта ко всей площади буферной зоны:

```
arcpy.AddField_management(str_diss_buf_led, "Dolya", "FLOAT", "7",
"5", "", "", "NULLABLE", "NON_REQUIRED", "")
```

Также в таблицу атрибутов добавляется числовое поле «Usl_dlina», предназначенное для последующей записи средней длины маршрута плавания в каждом объекте (ледовой зоне) зоне с учётом его площади:

```
arcpy.AddField_management(str_diss_buf_led, "Usl_dlina", "FLOAT",
"10", "4", "", "", "NULLABLE", "NON_REQUIRED", "")
```

Следующий этап: создание объекта типа «курсор» для просмотра атрибутивной таблицы слоя ледовой обстановки в буферной зоне:

```
cursor = arcpy.UpdateCursor(str_diss_buf_led)
```

Затем выполняется цикл просмотра всех записей таблицы:

```
for row in cursor:
  Bufer_area=row.getValue("Area_km2")
  Zona_area=row.getValue("Zona_km2")
  otn=Zona_area/Bufer_area
  row.setValue("Dolya",otn)
  Dl_put=row.getValue("Dlina_nm")
  usl_dl=Dl_put*otn
  row.setValue("Usl_dlina",usl_dl)
  cursor.updateRow(row) # запись изменений в таблицу атрибутов
```

«Bufer_area» и «Zona_area» – это переменные, в которых сохраняются величины, содержащиеся соответственно в полях «Area_km2» и «Zona_km2» в данной записи (строке) таблицы атрибутов. В переменную «otn» сохраняется значение отношения площади данной ледовой зоны (данного полигонального объекта) к общей площади всей буферной зоны маршрута плавания, которое затем записывается в

поле «Dolya». В переменную «Dl_put» из поля «Dlina_nm» сохраняется длина всего маршрута в морских милях. Сначала это поле было создано в таблице атрибутов линейного слоя маршрута, состоявшего из одного объекта типа «Polyline». Затем вокруг этого объекта был создан новый, состоящий из одного объекта, полигональный слой буферной зоны маршрута, в таблицу атрибутов которого поле «Dlina_nm» попало «по наследству». Потом в слое буфера было создано поле «Area_km2», куда была записана рассчитанная площадь буферной зоны в квадратных километрах. После пересечения слоя буфера со слоем ледовой информации поля «Dlina_nm» и «Area_km2» попадают уже в таблицу атрибутов слоя однородных ледовых зон в пределах буфера. В этой таблице в общем случае уже более одной записи, и в каждой из них в полях «Dlina_nm» и «Area_km2» содержатся одни и те же значения, рассчитанные для маршрута и буферной зоны. Средняя возможная протяжённость пути плавания в каждой ледовой зоне рассчитывается как произведение переменных «Dl_put» и «otn», которое сохраняется в переменную «usl_dl», и затем записывается в поле «Usl_dlina». В конце каждого «прохода» цикла записываются изменения в данной записи таблицы атрибутов (оператор cursor.updateRow(row)). После окончания цикла курсор удаляется: «del cursor, row».

Задание. Напишите программу вычисления значений числового поля таблицы атрибутов. Например, можете рассчитать площади всех полигональных объектов слоя, и затем определить, какую долю от общей площади всех объектов слоя занимает площадь каждого объекта.

1.15. Разбиение линейных объектов на их вершинах

Описанную в разделе 1.13 методику расчётов протяжённостей линейных объектов и площадей полигональных можно применять к любым объектам: простым, сложным, составным, односвязным и многосвязным. Заметим, что даже простой объект типа Polyline чаще всего представляет из себя ломаную линию, состоящую из последовательности Line-объектов, т.е. прямых отрезков, называемых также сегментами. Когда расчёты протяжённостей выполняются, например, для определения затрат времени при планировании морских транспортных операций, или для определения трудозатрат, расхода материалов и также затрат времени при проектировании сооружения линейных объектов (дорог, трубопроводов, линий электропередач, линий связи т.д.), то «кривизна» этих объектов не имеет значения.

Когда же судоводитель должен вести судно, а прораб – руководить производством работ, то им нужно дать координаты отрезков прямых, чтобы судоводитель мог по азимуту вести судно, а прораб – наметить линию для укладки трубопровода, например. Естественно, эти отрезки прямых не могут быть слишком длинными. Желательно, чтобы их длина не превышала 15-20 км, т.е. расстояние, в пределах которого поверхность Земли можно считать почти плоской. Поэтому возникает необходимость «расщепления» объектов типа Polyline на отдельные сегменты. Эти сегменты будут новыми объектами типа Polyline, но состоящими каждый только из одного Line-объекта, унаследовавшего семантику своего «родителя».

Рассмотрим пример: нужно разбить на вершинах отсортированные по порядку расположения на маршруте объекты типа Polyline (см. раздел 1.12). Пусть в строковой переменной «str_sort_route» записаны адрес и название шейпфайла объектов типа Polyline, расположенных по порядку вдоль маршрута. Проверяем существование этого шейпфайла, затем задаём место и название шейпфайла результата:

```
str_line_route=str_dir+'\\Line_Objects\line_objects_'+str_data+'_18.shp'
```

Разбиение объектов типа Polyline на вершинах:

```
arcpy.SplitLine_management(str_sort_route, str_line_route)
```

Создание объекта типа «курсор» для просмотра и редактирования атрибутивной таблицы:

```
cursor = arcpy.UpdateCursor(str_line_route)
```

Обнуление переменной для счётчика: «n=0».

Цикл просмотра таблицы с заполнением поля N_zone:

```
for row in cursor:
  n=n+1
  row.setValue("N_zone", n)
  cursor.updateRow(row)
del cursor, row  #Удаление курсора
```

Добавление в таблицу атрибутов числового поля «Length_nm»:

```
arcpy.AddField_management(str_line_route, "Length_nm", "FLOAT", "9",
"5", "", "", "NULLABLE", "NON_REQUIRED", "")
```

Расчёт протяжённости сегментов маршрута:

```
arcpy.CalculateField_management (str_line_route, "Length_nm",
"!shape.length@nauticalmiles!", "PYTHON_9.3")
```

Задание. Напишите программу разбиения объектов Polyline на вершинах и расчёта протяжённостей вновь созданных объектов.

1.16. Получение координат вершин

Теперь маршрут состоит из прямолинейных объектов. Но судоводителю и прорабу нужны географические координаты точек изменения курса (поворота маршрута) и изменения азимута линии строящегося сооружения. Сначала нужно создать фрейм (электронную карту) с географической системой координат. Скорее всего, на судне стоит международная спутниковая навигационная система WGS84. У прораба геодезические приборы могут быть универсальны: выдавать географические координаты в системах WGS84 и ГЛОНАС. В этот проект добавляем шейпфайл прямолинейных объектов, или много таких шейпфайлов в случае их автоматизированной обработки в цикле. Естественно, необходимо сохранить этот слои в новые шейпфайлы с записью координат в системе координат фрейма. Для упрощения автоматизации обработки лучше создать новый документ карты, а в нём – единственный фрейм. Универсальное начало кода программы с получением системы координат фрейма при таком условии подробно рассмотрено в разделе 1.1. Система координат фрейма сохранена в переменной «frame1CoordSys». В строковой переменной «str_line_route» сохраняется адрес и название исходного шейпфайла прямолинейных объектов. В случае автоматизированной обработки большого количества шейпфайлов значение этой переменной может изменяться с помощью циклов. Пример алгоритма подобного изменения подробно рассмотрен в разделе 1.3. В данном случае формирование значения переменной «str_line_route» может выполняться с помощью следующего оператора:

```
str_line_route=str_dir+'\\Line_Objects\line_objects_'+str_data+'_18.shp'
```

Выполняется проверка существования этого шейпфайла, а затем задаётся значение строковой переменной с адресом и названием шейпфайла результата, т.е. шейпфайла упорядоченного маршрута из прямолинейных объектов в географической системе координат, например, WGS84:

```
str_wgs84_route=str_dir+'\\WGS84_ice_routes\wgs84_route_'+str_data
+'_18.shp'
```

Далее выполняется перепроецирование исходного шейпфайла в новый с записью координат его вершин в географической системе координат:

```
arcpy.Project_management(str_line_route, str_wgs84_route,
frame1CoordSys)
```

Теперь в таблицу атрибутов вновь созданного шейпфайла нужно добавить числовые поля для записи географических координат начальных и конечных точек каждого прямолинейного объекта.

Добавление в таблицу атрибутов числового поля «Lat_first» – широты начальной точки объекта в десятичных градусах:

```
arcpy.AddField_management(str_wgs84_route, "Lat_first", "FLOAT", "8", "5", "", "", "NULLABLE", "NON_REQUIRED", "")
```

Добавление в таблицу атрибутов числового поля «Long_first» – долготы начальной точки объекта в десятичных градусах:

```
arcpy.AddField_management(str_wgs84_route, "Long_first", "FLOAT", "9", "5", "", "", "NULLABLE", "NON_REQUIRED", "")
```

Добавление в таблицу атрибутов числового поля «Lat_last» – широты конечной точки объекта в десятичных градусах:

```
arcpy.AddField_management(str_wgs84_route, "Lat_last", "FLOAT", "8", "5", "", "", "NULLABLE", "NON_REQUIRED", "")
```

Добавление в таблицу атрибутов числового поля «Long_last» – долготы конечной точки объекта в десятичных градусах:

```
arcpy.AddField_management(str_wgs84_route, "Long_last", "FLOAT", "9", "5", "", "", "NULLABLE", "NON_REQUIRED", "")
```

Теперь нужно выполнить расчёты и заполнить их результатами вновь созданные поля таблицы атрибутов. Нужно создать объект типа «курсор» для просмотра и редактирования таблицы атрибутов, затем в цикле определить координаты точек начала и конца каждого объекта, и записать результаты в таблицу атрибутов:

```
cursor = arcpy.UpdateCursor(str_wgs84_route)
for row in cursor:
    the_polyline = row.getValue("Shape")
    first_X = the_polyline.firstPoint.X
    first_Y = the_polyline.firstPoint.Y
    last_X = the_polyline.lastPoint.X
    last_Y = the_polyline.lastPoint.Y
    row.setValue("Lat_first", first_Y)
    row.setValue("Long_first", first_X)
    row.setValue("Lat_last", last_Y)
    row.setValue("Long_last", last_X)
    cursor.updateRow(row)
del cursor, row  #Удаление курсора
```

В результате создаётся новый шейпфайл прямолинейных отрезков маршрута с записью в таблице атрибутов географических координат начальных и конечных точек всех отрезков.

Задание. Напишите программу получения и записи в таблице атрибутов координат точек начала и конца прямолинейных объектов.

1.17. Запись таблицы атрибутов во внешний файл формата Microsoft Excel

Полученные координаты точек начала и конца прямолинейных отрезков маршрута нужно представить потребителям. Естественно, что информация должна доводиться не в формате шейпфайлов ESRI, а в каком-либо другом широко распространённом формате. Наиболее предпочтительным представляется формат таблиц Microsoft Excel, которые далее можно сохранять в файлах текстового формата «CSV», занимающих очень небольшие объёмы памяти. При оперативной поддержке морских транспортных операций эти файлы можно выкладывать в Интернете, и потребители смогут их скачивать с помощью спутниковой связи, которая сейчас доступна практически по всему Земному шару. В случае отсутствия спутниковой связи координаты можно передавать с помощью радиограмм.

В программе следует создать строковую переменную для записи адреса и названия вновь создаваемого файла формата Microsoft Excel, и заполнить её информацией, например: «str_excel=str_dir+'\\Excel_files\ice_route_'+str_data+'_18.xls'».
Затем выполняется код записи атрибутивной таблицы шейпфайла в файл таблицы Microsoft Excel:
arcpy.TableToExcel_conversion(str_wgs84_route, str_excel, "NAME", "CODE")
В этом операторе «str_wgs84_route» – строковая переменная, содержащая адрес и название шейпфайла; «str_excel» – строковая переменная, содержащая адрес и название файла формата Microsoft Excel. Следующий параметр данного оператора – «Use_field_alias_as_column_header», значение которого определяет названия столбцов в таблице Microsoft Excel. Возможные значения: «NAME» – в качестве заголовков столбцов будут использованы имена полей таблицы атрибутов, оно применяется по умолчанию; «ALIAS» – используются псевдонимы полей таблицы. Завершающим параметром кода является «Use_domain_and_subtype_description». Его значение определяет способ преобразования значений из полей таблицы атрибутов в значения в таблице Microsoft Excel. По умолчанию применяется значение «CODE»: все данные сохраняются в тех же форматах, в каких они находятся в таблице атрибутов. Другое возможное значение «DESCRIPTION» применяется при использовании подтипов данных, определённых с помощью доменов, т.е. в базах геоданных. Это означает использование описаний подтипов данных в доменах при экспорте значений в таблицу Microsoft Excel.

Задание. Напишите программу записи в таблицы атрибутов шейпфайла в файл формата Microsoft Excel.

1.18. Импорт шейпфайлов в классы объектов набора классов файловой базы геоданных

Базы геоданных имеют ряд преимуществ по сравнению с шейпфайлами. Они позволяют задавать топологические правила пространственных взаимодействий между объектами одного и того же векторного слоя, и между объектами разных слоёв, например, запрет наложений полигонов одного слоя друг на друга и требование примыкания соседних полигонов друг к другу, т.е. запрет наличия «зазоров» между соседними полигонами. Возможно задание для наборов классов пространственных доменов, т.е. диапазонов возможных значений координат объектов, а также семантических доменов для значений атрибутов. Это могут быть как диапазоны допустимых значений для числовых полей, так и списки значений для числовых и текстовых полей. Вообще база геоданных – это «очень удобный чемодан с колёсиками и выдвигающейся ручкой» для хранения любых данных: векторных нетопологических и топологических, растровых. Все слои электронной карты можно сохранить в одной базе геоданных, что очень удобно для переноса данных с носителя на носитель и их пересылки по Интернету. При импорте шейпфайла в базу геоданных он преобразуется в класс объектов. В свою очередь классы объектов могут объединяться в набор классов пространственных объектов. Все классы объектов одного набора должны иметь одинаковую систему координат.

Рассмотрим пример программного кода для импорта шейпфайла в базу геоданных. Пусть в строковой переменной «str_wgs84_route» содержится адрес и название шейпфайла, который планируется импортировать в класс пространственных объектов файловой базы геоданных:

```
str_wgs84_route=str_dir+'\\WGS84_ice_routes\wgs84_route_'+str_data+'_18.shp'
```

В программе нужно задать строковую переменную с параметрами базы геоданных и набора классов пространственных объектов, куда в качестве класса объектов будет добавлен шейпфайл:

```
str_spatial_object=str_dir+'\\Tatar_Strait_ice_routes.gdb\\Ice_Routes'
```

В этом операторе «str_dir» – адрес и название папки, в которой находится база геоданных Tatar_Strait_ice_routes.gdb. «Ice_Routes» – набор классов пространственных объектов, в который будет

импортирован шейпфайл. База геоданных и набор классов пространственных объектов уже должны существовать перед выполнением оператора импорта. База геоданных и набор классов при этом могут быть пустыми, т.е. не содержать никаких данных. Код импорта шейпфайла в базу геоданных:

```
arcpy.FeatureClassToGeodatabase_conversion([str_wgs84_route],
str_spatial_object)
```

В этой строке кода в квадратных скобках нужно указать одну строковую переменную с адресом и названием импортируемого шейпфайла, или список таких переменных, разделённых символом запятой.

Задание. Напишите программу импорта шейпфайла в класс объектов файловой базы геоданных.

1.19. Расчёт индекса мутности NDTI

Часто при выполнении исследований в сфере экологии и природопользования очень информативно применение индексов, рассчитываемых по данным дистанционного зондирования Земли, например, вегетационных индексов и индекса мутности. Последний по данным отражённого излучения водной поверхности позволяет определить различия мутности воды, а соответственно, и определить степень однородности водного объекта. Для расчётов индексов можно использовать снимки Sentinel-2 в файлах формата «jp2», которые уже содержат всю информацию о системе координат снимка (американская система UTM, для Санкт-Петербурга и Ленинградской области зоны 35 и 36) и месте расположения территории и/или акватории снимка на поверхности Земли. Снимки можно бесплатно скачивать с сайта https://scihub.copernicus.eu/dhus/#/home, их разрешение от 10 до 60 м. Данные разных диапазонов спектра хранятся в отдельных файлах. Для расчётов индекса мутности NDTI необходимы снимки диапазонов 3 и 4, т.е. в зелёном и красном диапазонах видимой части спектра. Для расчётов вегетационного индекса NDVI нужны снимки в красной и ближней инфракрасной (NIR – диапазон 8) областях спектра.

Рассмотрим листинг программы расчёта индекса мутности NDTI. В её начале необходимо указать кодировку и подключить модули и библиотеки:

```
# -*- coding: utf-8 -*-
# программа расчёта индекса мутности NDTI
import os
import sys
```

```
import arcpy
import math
from arcpy import env
import arcpy.mapping as mapping
from arcpy.sa import *
arcpy.CheckOutExtension("spatial")
```
Обратите внимание на необходимость подключения библиотеки «math» математических функций и «env» приложений окружения. Функция «CheckOutExtension» возвращает лицензию Менеджеру лицензий и другим приложениям, которые могут быть задействованы. Параметр «spatial» обеспечивает возможность применения модуля «Spatial Analyst». Далее выполняется возвращение текущего документа карты:
```
the_mxd = mapping.MapDocument('Current')
the_mxd_FilePath = the_mxd.filePath
len_the_mxd_FilePath = len(the_mxd_FilePath)
n_slash_kon = the_mxd_FilePath.rfind('\\')
str_dir = the_mxd_FilePath[0:n_slash_kon]
```
Строковая переменная «str_dir» содержит адрес и название папки, в которой находится файл документа карты, исходные файлы снимков, и будут находиться файлы результатов обработки. Следующий этап – возвращение списка фреймов текущего документа карты:
```
frame_list = arcpy.mapping.ListDataFrames(the_mxd)
```
Возвращение первого фрейма в списке фреймов. Желательно, чтобы документ карты содержал только один фрейм.
```
frame1=frame_list[0]
```
Возвращение системы координат этого фрейма
```
frame1CoordSys = frame1.spatialReference
```
Т.к. в названия файлов снимков входят обозначения дат их производства, то создаём списки обозначений лет, месяцев и дней:
```
lst_God=['2018', '2019', '2020', '2021']
lst_Mes=['04','05','06','07','08','09','10','11']
lst_Den=['01','02','03','04','05','06','07','08','09','10','11','12','13','14','15','16',
'17','18','19','20','21','22','23','24','25','26','27','28','29','30','31']
```
Определение числа элементов в каждом списке:
```
n_let = len(lst_God)
n_mes = len(lst_Mes)
n_den = len(lst_Den)
```
Шейпфайл с полигоном акватории, в пределах которой будет рассчитан индекс мутности, должен быть помещён в каталог с файлом документа карты. Задаём адрес и название шейпфайла акватории. Для

другой акватории это место в программе нужно изменить.

```
str_Water_area_file=str_dir+'\\Neva_stvor_161_3.shp'
```

Далее в программе следуют три вложенные цикла для генерации всех возможных названий файлов снимков:

```
for i_god in range (n_let):
  tek_god=lst_God[i_god]
  for i_mes in range(n_mes):
    tek_mes=lst_Mes[i_mes]
    for i_den in range (n_den):
      tek_den=lst_Den[i_den]
      str_data=tek_god+tek_mes+tek_den
```

Задание адреса и названия файла с изображением снимка в красном диапазоне:

```
      str_B04_image=str_dir+'\\T36VUM_'+str_data
      +'T092039_B04_10m.jp2'
```

Задание адреса и названия файла с изображением снимка в зелёном диапазоне:

```
      str_B03_image=str_dir+'\\T36VUM_'+str_data
      +'T092039_B03_10m.jp2'
```

Т.к. эти файлы могут и не существовать (снимки делаются не каждый день), то необходима проверка их существования:

```
      if arcpy.Exists(str_B04_image):
        if arcpy.Exists(str_B03_image):
```

Задание адреса и названия файла – результата обрезки изображения снимка в красном диапазоне:

```
          str_Red=str_dir+'\\'+str_data+'_Red.tif'
```

Формат растрового файла, который будет создан: GeoTIFF, т.е. он уже будет содержать всю информацию о системе координат и месте расположения созданного растра. Проверка возможного существования «предшественника» в этом же месте и с этим же названием, и его удаление в этом случае:

```
          if arcpy.Exists(str_Red):
            arcpy.Delete_management(str_Red)
```

Вырезка растра из снимка в красном диапазоне:

```
          arcpy.gp.ExtractByMask_sa(str_B04_image, str_Water_area_file,
          str_Red)
```

Задание адреса и названия файла – результата обрезки изображения снимка в зелёном диапазоне:

```
          str_Green=str_dir+'\\'+str_data+'_Green.tif'
```

Проверка возможного существования «предшественника» и его удаление:

```
if arcpy.Exists(str_Green):
  arcpy.Delete_management(str_Green)
```

Вырезка растра из снимка в зелёном диапазоне:

```
arcpy.gp.ExtractByMask_sa(str_B03_image, str_Water_area_file,
str_Green)
```

Задание адреса и названия файла – результата расчёта индекса мутности NDTI:

```
str_NDTI=str_dir+'\\NDTItif\\'+str_data+'_NDTI.tif'
```

Проверка возможного существования «предшественника» и его удаление:

```
if arcpy.Exists(str_NDTI):
  arcpy.Delete_management(str_NDTI)
```

Создание растров:

```
RedRaster=Raster(str_Red)
GreenRaster=Raster(str_Green)
NDTIRaster=(RedRaster-
GreenRaster)*1.0/((RedRaster+GreenRaster)*1.0)
```

Обратите внимание! И числитель, и знаменатель дроби необходимо умножить на 1.0. Это вызвано тем, что растры RedRaster и GreenRaster содержат целочисленные значения, а NDTIRaster должен содержать вещественные числа с дробной частью. Если не сделать этого умножения, то NDTIRaster будет содержать только числа -1, 0 и 1.

Сохранение результата в файл:

```
NDTIRaster.save(str_NDTI)
```

Задание адреса и названия шейпфайла, в который будет преобразован растр индекса мутности. Центры ячеек растра преобразуются в точечные объекты.

```
str_NDTI_shp=str_dir+'\\NDTIshp\\'+str_data+'_NDTI.shp'
```

Проверка возможного существования «предшественника» и его удаление:

```
if arcpy.Exists(str_NDTI_shp):
  arcpy.Delete_management(str_NDTI_shp)
```

Конвертация растра в шейпфайл:

```
arcpy.RasterToPoint_conversion(NDTIRaster, str_NDTI_shp,
"Value")
```

Параметр "Value" в этом операторе означает, что в таблицу атрибутов создаваемого шейпфайла будут записаны значения ячеек растра (матрицы). Они будут записаны в поле "GRID_CODE". Добавляем в таблицу атрибутов поля прямоугольных координат точек X и Y:

```
arcpy.AddXY_management(str_NDTI_shp)
```

Имена этих полей: "POINT_X" и "POINT_Y". Разумеется, можно

оставить и такие поля, но лучше мы перепишем из них значения в новые поля с заданными нами форматами. Добавление поля Value:

```
arcpy.AddField_management(str_NDTI_shp, "Value", "FLOAT",
"10", "6", "", "", "NULLABLE", "NON_REQUIRED", "")
```

Запись значений из поля «GRID_CODE» в поле «Value»:

```
arcpy.CalculateField_management(str_NDTI_shp, "Value",
"!GRID_CODE!", "PYTHON_9.3", "")
```

Удаление поля GRID_CODE:

```
arcpy.DeleteField_management(str_NDTI_shp, "GRID_CODE")
```

Добавление поля X:

```
arcpy.AddField_management(str_NDTI_shp, "X", "FLOAT",
"10", "1", "", "", "NULLABLE", "NON_REQUIRED", "")
```

Запись значений поля X:

```
arcpy.CalculateField_management(str_NDTI_shp, "X",
"!POINT_X!", "PYTHON_9.3", "")
```

Удаление поля POINT_X:

```
arcpy.DeleteField_management(str_NDTI_shp, "POINT_X")
```

Добавление поля Y:

```
arcpy.AddField_management(str_NDTI_shp, "Y", "FLOAT",
"10", "1", "", "", "NULLABLE", "NON_REQUIRED", "")
```

Запись значений поля Y:

```
arcpy.CalculateField_management(str_NDTI_shp, "Y",
"!POINT_Y!","PYTHON_9.3", "")
```

Удаление поля POINT_Y:

```
arcpy.DeleteField_management(str_NDTI_shp, "POINT_Y")
```

Данная программа может быть легко модернизирована для расчётов любых индексов. Для этого необходимо только заменить блок задания исходных растров спутниковых снимков и расчёт значения индекса. В результате работы программы создаётся большое количество число шейпфайлов точечных объектов, атрибутивные таблицы которых содержат значения индекса в каждой точке и прямоугольные координаты точек в системе координат карты. Эти точки находятся в пределах определённой акватории или территории. Разумеется, каждый шейпфайл характеризует ситуацию на определённую дату производства космических снимков. Экспорт таблиц атрибутов во внешние файлы позволяет выполнять анализ временных различий с помощью статистических критериев однородности. Для оценки различий по значениям индексов для разных частей акватории (территории) за одну и ту же дату можно выделять точки, находящиеся в разных частях, и экспортировать их в отдельные файлы. Затем выполняется проверку наличия или

отсутствия статистически значимых различий между значениями индекса в разных частях акватории или территории. Разумеется, можно выполнять и анализ изменения индекса на одной и той же акватории.

1.20. Пересчёт координат точек

Предположим, что нужно определить степень однородности створа мониторинга, расположенного ниже впадения в реку её притока. Для этого можно проанализировать изменение индекса мутности вдоль створа. Сначала нужно создать буферную зону створа, и с помощью её полигона выбрать точки со значениями индекса мутности, находящиеся в непосредственной близости от линии створа. Затем нужно перенести начало осей координат в точку начала створа, а далее повернуть оси координат таким образом, чтобы одна из осей совпала с линией створа, выполнить расчёты координат точек в новой системе координат. Теперь координаты точек по совпадающей с линией створа оси будут равны расстояниям от точек до берега, на котором расположено начало линии створа.

Рассмотрим программу пересчёта координат. Её начало практически такое же, как и у предыдущей программы, только отсутствует функция «CheckOutExtension», т.к. нет необходимости в подключении модуля «Spatial Analyst». Также выполняется возвращение системы координат фрейма, и создаются списки обозначений лет, месяцев, дней. Т.к. программа предназначена для пересчётов координат точек, расположенных в пределах буферной зоны только одного створа, то значения строковых переменных адресов и названий шейпфайлов створа и его буферной зоны задаются только однажды:

```
str_stvor=str_dir+'\\stv_161_3.shp '
str_Bufer=str_dir+'\\buf_stvor_161_3.shp'
```

При смене створа и/или буферной зоны в тексте программы следует сделать соответствующие изменения.

Далее выполняется определение прямоугольных координат точек начала и конца линии створа. Несмотря на то, что створ – это один линейный объект, состоящий из одного сегмента, для этого необходимо создать курсор и выполнить просмотр таблицы атрибутов:

```
cursor = arcpy.UpdateCursor(str_stvor)
for row in cursor:
  the_stvor = row.getValue("Shape")
  first_X = the_stvor.firstPoint.X
```

```
  first_Y = the_stvor.firstPoint.Y
  last_X = the_stvor.lastPoint.X
  last_Y = the_stvor.lastPoint.Y
del cursor, row  #Удаление курсора
```
В данном примере рассмотрим случай, когда линия створа расположена в первом квадранте, т.е. имеет направление в секторе от направления на север до направления на восток, и начинается «внизу слева».
```
if (first_X<last_X) and (first_Y<last_Y):
  Xn=first_X
  Yn=first_Y
  Xk=last_X
  Yk=last_Y
if (first_X>last_X) and (first_Y>last_Y):
  Xk=first_X
  Yk=first_Y
  Xn=last_X
  Yn=last_Y
```
Здесь Xn и Yn – координаты начала линии створа, а Xk и Yk – её конца.
Затем выполняется расчёт синуса и косинуса угла между направлением горизонтальной оси X и направлением линии створа:
```
delta_X=abs(Xk-Xn)
delta_Y=abs(Yk-Yn)
hypotenuza=math.sqrt(delta_X**2+delta_Y**2)
sin_alfa=delta_Y/hypotenuza
cos_alfa=delta_X/hypotenuza
```
Примечание: знак «**» – возведение в степень.
Затем организуются вложенные циклы по спискам лет, месяцев и дней:
```
for i_god in range (n_let):
  tek_god=lst_God[i_god]
  for i_mes in range(n_mes):
    tek_mes=lst_Mes[i_mes]
    for i_den in range (n_den):
      tek_den=lst_Den[i_den]
      str_data=tek_god+tek_mes+tek_den
```
Задаётся адрес и название шейпфайла точек со значениями индекса мутности в таблице атрибутов:
```
      str_NDTI_shp=str_dir+'\\NDTIshp\\'+str_data+'_NDTI.shp'
```
Адрес и название шейпфайла с точками в пределах буферной зоны створа:

```
str_NDTI_buf=str_dir+'\\Buf\\'+str_data+'_NDTI_buf.shp'
```

«Обрезка» слоя точек по границе буферной зоны:

```
arcpy.Clip_analysis(str_NDTI_shp, str_Bufer, str_NDTI_buf, "")
```

Добавление в таблицу атрибутов результата обрезки полей Xnov и Ynov:

```
arcpy.AddField_management(str_NDTI_buf, "Xnov", "FLOAT",
"10", "1", "", "", "NULLABLE", "NON_REQUIRED", "")
arcpy.AddField_management(str_NDTI_buf, "Ynov", "FLOAT",
"10", "1", "", "", "NULLABLE", "NON_REQUIRED", "")
```

Создание объекта типа «курсор» для просмотра атрибутивной таблицы слоя точек:

```
cursor = arcpy.UpdateCursor(str_NDTI_buf)
```

Сам просмотр таблицы атрибутов:

```
for row in cursor:
  X = row.getValue("X")
  Y = row.getValue("Y")
```

Перенос начала осей:

```
  X0 = X-Xn
  Y0 = Y-Yn
  X_new = X0*cos_alfa+Y0*sin_alfa
  Y_new = Y0*cos_alfa-X0*sin_alfa
  row.setValue("Xnov",X_new)
  row.setValue("Ynov",Y_new)
  cursor.updateRow(row)
```

Удаление курсора:

```
del cursor, row
```

В полях «Xnov» и «Ynov» теперь записаны координаты точек в новой системе, в которой ось «X» совпадает с линией створа, а её начало расположено на берегу в начальной точке створа. Затем создаётся строковая переменная с адресом и названием файла Microsoft Excel, куда будет экспортирована таблица атрибутов:

```
str_excel=str_dir+'\\Excel_files\\'+str_data+'_NDTI_buf.xls'
```

Завершающий этап программы – экспорт таблицы атрибутов в файл Microsoft Excel:

```
arcpy.TableToExcel_conversion(str_NDTI_buf, str_excel, "NAME",
"CODE")
```

Данные из файла Microsoft Excel могут быть использованы для определения степени однородности значений индекса мутности в буферной зоне створа. Таблицу Microsoft Excel следует отсортировать по увеличению расстояния от начала створа (от берега), и потом для выявления возможной неоднородности применить метод интегральных

кривых (накопленных сумм).

1.21. Анализ таблицы атрибутов

При исследованиях в сфере экологии и природопользования встречается необходимость переклассификации таблицы атрибутов, т.е. заполнения нового поля на основании обработки значений других полей этой таблицы. Например, в таблице атрибутов слоя точек мониторинга представлены концентрации загрязняющих веществ, нормированные по значениям ПДК (предельно допустимых концентраций). Необходимо определить для каждой точки «самое вредное» загрязняющее вещество, у которого наивысшее значение отношения концентрации к величине ПДК. Или есть данные о частном проективном покрытии различных видов растительности, или о их коэффициентах встречаемости, и нужно определить доминантов. Рассмотрим пример программы, в которой определяется доминирующая возрастная градация морских льдов. Начало программы «традиционное»:

```
import os
import sys
import arcpy
import arcpy.mapping as mapping
```

Возвращение текущего документа карты:

```
the_mxd = mapping.MapDocument('Current')
the_mxd_FilePath = the_mxd.filePath
len_the_mxd_FilePath = len(the_mxd_FilePath)
n_slash1 = the_mxd_FilePath.find('\\')
n_slash2 = the_mxd_FilePath.find('\\',n_slash1+1,len_the_mxd_FilePath)
str_dir = the_mxd_FilePath[0:n_slash2]
```

Возвращение списка фреймов текущего документа карты:

```
frame_list = arcpy.mapping.ListDataFrames(the_mxd)
```

Возвращение первого фрейма в списке фреймов (в документе карты должен быть только один фрейм):

```
frame1=frame_list[0]
```

Возвращение списка слоёв:

```
LayersList=arcpy.mapping.ListLayers(the_mxd,"",frame1)
led_layer=LayersList[0]
led_layer_DatasetName=led_layer.datasetName
str_led_layer_DatasetName=str(led_layer_DatasetName) #название слоя
led_layer_workspacePath=led_layer.workspacePath
#название папки источника слоя
```

str_led_layer_workspacePath=str(led_layer_workspacePath)
len_workspacePath=len(str_led_layer_workspacePath)
Создание строковой переменной с адресом и названием шейпфайла, таблица атрибутов которого будет анализироваться:
str_led_layer=str_led_layer_workspacePath+"\\"
+str_led_layer_DatasetName+".shp"
Необходимо добавить в таблицу атрибутов ряд числовых полей, но сперва нужно выяснить, нет ли их уже в таблице. Поэтому создаём список полей таблицы атрибутов шейпфайла:
fields = arcpy.ListFields(str_led_layer),
и ряд логических переменных для последующей проверки существования полей:
field_exist_C=False
field_exist_C_old=False
field_exist_C_second_year=False
field_exist_C_residual=False
field_exist_C_thick=False
field_exist_C_medium=False
field_exist_C_thin=False
field_exist_C_gr_white=False
field_exist_C_grey=False
field_exist_C_new=False
field_exist_C_blin=False
field_exist_nevyazka=False
field_exist_Preobl=False
Необходимо добавить в таблицу и заполнить значениями следующие числовые поля:
«C» – общая сплоченность (концентрация) морских льдов; частные концентрации льдов: «C_old» – старых, «C_second» – двухлетних, «C_residual» – остаточных, «C_thick» – толстых однолетних, «C_medium» – однолетних средней толщины, «C_thin» – тонких однолетних, «C_gr_white» – серо-белых, «C_grey» – серых, «C_new» – начальных, «C_blin» – блинчатых. В новое числовое поле «nevyazka» должна записываться разность общей сплоченности льда и суммы частных концентраций. В новое текстовое поле «Preobl» будет записываться обозначение преобладающей возрастной градации льдов.
Цикл по всем полям таблицы атрибутов:
```
for field in fields:
  tek_field_name=field.name
  if tek_field_name=='Preobl':
     field_exist_Preobl=True
```

```
  if tek_field_name=='C':
    field_exist_C=True
  if tek_field_name=='C_old':
    field_exist_C_old=True
  if tek_field_name=='C_second':
    field_exist_C_second_year=True
  if tek_field_name=='C_residual':
    field_exist_C_residual=True
  if tek_field_name=='C_thick':
    field_exist_C_thick=True
  if tek_field_name=='C_medium':
    field_exist_C_medium=True
  if tek_field_name=='C_thin':
    field_exist_C_thin=True
  if tek_field_name=='C_gr_white':
    field_exist_C_gr_white=True
  if tek_field_name=='C_grey':
    field_exist_C_grey=True
  if tek_field_name=='C_new':
    field_exist_C_new=True
  if tek_field_name=='C_blin':
    field_exist_C_blin=True
  if tek_field_name=='nevyazka':
    field_exist_nevyazka=True
```

При обнаружении поля в таблице изменяется значение соответствующей логической переменной.

В случае их отсутствия в таблице выполняется добавление новых полей:

```
if field_exist_C==False:
  arcpy.AddField_management(str_led_layer, "C", "FLOAT", "4", "1", "",
  "", "NULLABLE", "NON_REQUIRED", "")
if field_exist_C_old==False:
  arcpy.AddField_management(str_led_layer, "C_old", "FLOAT", "6", "3",
  "", "", "NULLABLE", "NON_REQUIRED", "")
if field_exist_C_second_year==False:
  arcpy.AddField_management(str_led_layer, "C_second", "FLOAT", "6",
  "3", "", "", "NULLABLE","NON_REQUIRED", "")
if field_exist_C_residual==False:
  arcpy.AddField_management(str_led_layer, "C_residual", "FLOAT", "6",
  "3", "", "", "NULLABLE","NON_REQUIRED", "")
if field_exist_C_thick==False:
```

```
  arcpy.AddField_management(str_led_layer, "C_thick", "FLOAT", "6",
  "3", "", "", "NULLABLE","NON_REQUIRED", "")
if field_exist_C_medium==False:
  arcpy.AddField_management(str_led_layer, "C_medium", "FLOAT", "6",
  "3", "", "", "NULLABLE","NON_REQUIRED", "")
if field_exist_C_thin==False:
  arcpy.AddField_management(str_led_layer, "C_thin", "FLOAT", "6", "3",
  "", "", "NULLABLE","NON_REQUIRED", "")
if field_exist_C_gr_white==False:
  arcpy.AddField_management(str_led_layer, "C_gr_white", "FLOAT",
  "6", "3", "", "", "NULLABLE","NON_REQUIRED", "")
if field_exist_C_grey==False:
  arcpy.AddField_management(str_led_layer, "C_grey", "FLOAT", "6",
  "3", "", "", "NULLABLE", "NON_REQUIRED", "")
if field_exist_C_new==False:
  arcpy.AddField_management(str_led_layer, "C_new", "FLOAT", "6",
  "3", "", "", "NULLABLE", "NON_REQUIRED", "")
if field_exist_C_blin==False:
  arcpy.AddField_management(str_led_layer, "C_blin", "FLOAT", "6", "3",
  "", "", "NULLABLE", "NON_REQUIRED", "")
if field_exist_nevyazka==False:
  arcpy.AddField_management(str_led_layer, "nevyazka", "FLOAT", "5",
  "3", "", "", "NULLABLE", "NON_REQUIRED", "")
if field_exist_Preobl==False:
  arcpy.AddField_management(str_led_layer,"Preobl","TEXT","","","25",
  "","NULLABLE", "NON_REQUIRED","")
```

В таблице атрибутов есть текстовое поле «S», содержащее значения общей сплоченности льда, и три текстовых поля частных концентраций льдов («S_1», «S_2», «S_3»), в которых могут быть записаны значения частных концентраций трёх групп возрастных градаций льда. В каждой группе может содержаться максимум три возрастные градации льда. Поэтому в программе создаются три числовые переменные для хранения значений концентраций льда:

```
C1=0
C2=0
C3=0
```

Создание объекта типа «курсор» для просмотра и редактирования атрибутивной таблицы:

```
cursor = arcpy.UpdateCursor(str_led_layer)
```

Далее следует цикл просмотра таблицы с заполнением поля «Preobl». Этот цикл очень длинный, и в него входит много коротких

операторов. Код на языке Python нельзя размещать в несколько колонок, но ниже в особо оговоренных случаях для экономии места будет сделано именно так.

Начало цикла:

```
for row in cursor:
  S_value = row.getValue("S")
  S1_value = row.getValue("S_1")
```

Ниже код программы помещён в две колонки.

```
#чистая вода
  if (S_value=="free"):
    C=0
  if (S_value=="free"):
    C1=0
  if (S_value=="free"):
    C2=0
  if (S_value=="free"):
    C3=0
  #припай
  if S_value=="fast":
    C=10
    C1=10
  #льдина
  if (S_value=="floe")
  and (S1_value=="10"):
    C=10
  if S_value=="0-1":
    C=0.5
  if S_value=="1":
    C=1
  if S_value=="1-2":
    C=1.5
  if S_value=="2":
    C=2
  if S_value=="1-3":
    C=2
  if S_value=="2-3":
    C=2.5
  if S_value=="3":
    C=3
  if S_value=="3-4":
    C=3.5
  if S_value=="4-6":
    C=5
  if S_value=="5-6":
    C=5.5
  if S_value=="6":
    C=6
  if S_value=="6-7":
    C=6.5
  if S_value=="7":
    C=7
  if S_value=="7-8":
    C=7.5
  if S_value=="8":
    C=8
  if S_value=="8-9":
    C=8.5
  if S_value=="9":
    C=9
  if S_value=="9-10":
    C=9.5
  if S_value=="10":
    C=10
  if S1_value=="":
    C1=0
  if S1_value=="0-1":
    C1=0.5
  if S1_value=="1":
    C1=1
  if S1_value=="1-2":
    C1=1.5
  if S1_value=="2":
    C1=2
```

```
if S_value=="4":
  C=4
if S_value=="4-5":
  C=4.5
if S_value=="5":
  C=5
if S1_value=="3-4":
  C1=3.5
if S1_value=="4":
  C1=4
if S1_value=="4-5":
  C1=4.5
if S1_value=="5":
  C1=5
if S1_value=="4-6":
  C1=5
if S1_value=="5-6":
  C1=5.5
if S1_value=="6":
  C1=6
if S1_value=="6-7":
  C1=6.5
if S1_value=="7":
  C1=7
if S1_value=="7-8":
  C1=7.5
if S1_value=="8":
  C1=8
if S1_value=="8-9":
  C1=8.5
if S1_value=="9":
  C1=9
if S1_value=="9-10":
  C1=9.5
if S1_value=="10":
  C1=10
S2_value = row.getValue("S_2")
if S2_value=="":
  C2=0
if S2_value=="0-1":
  C2=0.5
if S1_value=="1-3":
  C1=2
if S1_value=="2-3":
  C1=2.5
if S1_value=="3":
  C1=3
if S2_value=="1-3":
  C2=2
if S2_value=="2-3":
  C2=2.5
if S2_value=="3":
  C2=3
if S2_value=="3-4":
  C2=3.5
if S2_value=="4":
  C2=4
if S2_value=="4-5":
  C2=4.5
if S2_value=="5":
  C2=5
if S2_value=="4-6":
  C2=5
if S2_value=="5-6":
  C2=5.5
if S2_value=="6":
  C2=6
if S2_value=="6-7":
  C2=6.5
if S2_value=="7":
  C2=7
if S2_value=="7-8":
  C2=7.5
if S2_value=="8":
  C2=8
if S2_value=="8-9":
  C2=8.5
if S2_value=="9":
  C2=9
if S2_value=="9-10":
  C2=9.5
if S2_value=="10":
```

```
  if S2_value=="1":
    C2=1
  if S2_value=="1-2":
    C2=1.5
  if S2_value=="2":
    C2=2
  if S3_value=="1":
    C3=1
  if S3_value=="1-2":
    C3=1.5
  if S3_value=="2":
    C3=2
  if S3_value=="1-3":
    C3=2
  if S3_value=="2-3":
    C3=2.5
  if S3_value=="3":
    C3=3
  if S3_value=="3-4":
    C3=3.5
  if S3_value=="4":
    C3=4
  if S3_value=="4-5":
    C3=4.5
  if S3_value=="5":
    C3=5
  if S3_value=="4-6":
    C3=5
    C2=10
 S3_value = row.getValue("S_3")
 if S3_value=="":
    C3=0
 if S3_value=="0-1":
    C3=0.5
  if S3_value=="5-6":
    C3=5.5
  if S3_value=="6":
    C3=6
  if S3_value=="6-7":
    C3=6.5
  if S3_value=="7":
    C3=7
  if S3_value=="7-8":
    C3=7.5
 if S3_value=="8":
    C3=8
  if S3_value=="8-9":
    C3=8.5
  if S3_value=="9":
    C3=9
  if S3_value=="9-10":
    C3=9.5
 if S3_value=="10":
    C3=10
```

Если льдов нет совсем, то в поле «Preobl» записывается значение «чистая вода».

```
  if (S_value=="free"):
    row.setValue("Preobl", "чистая вода")
  if (C>0):  # есть лёд!
```

Если лёд есть, то в каждой группе градаций льда может быть до трёх возрастных градаций. Общую частную концентрацию данной группы градаций делим поровну между всеми присутствующими льдами: «Всем сёстрам по серьгам!» Создаём и обнуляем числовые переменные всех возможных частных концентраций льда:

```
    C_1_1=0
    C_1_2=0
```

```
C_1_3=0
C_2_1=0
C_2_2=0
C_2_3=0
C_3_1=0
C_3_2=0
C_3_3=0
```

Создаём и обнуляем числовые переменные числа возрастных градаций льда в каждой группе:

```
n_led_1=0
n_led_2=0
n_led_3=0
```

Получаем значения форм льда (размеров льдин) по всем возможным градациям:

```
F_1_1_value=row.getValue("F_1_1")
F_1_2_value=row.getValue("F_1_2")
F_1_3_value=row.getValue("F_1_3")
F_2_1_value=row.getValue("F_2_1")
F_2_2_value=row.getValue("F_2_2")
F_2_3_value=row.getValue("F_2_3")
F_3_1_value=row.getValue("F_3_1")
F_3_2_value=row.getValue("F_3_2")
F_3_3_value=row.getValue("F_3_3")
```

Получаем обозначения всех возрастных градаций льда:

```
A_1_1_value=row.getValue("A_1_1")
A_1_2_value=row.getValue("A_1_2")
A_1_3_value=row.getValue("A_1_3")
A_2_1_value=row.getValue("A_2_1")
A_2_2_value=row.getValue("A_2_2")
A_2_3_value=row.getValue("A_2_3")
A_3_1_value=row.getValue("A_3_1")
A_3_2_value=row.getValue("A_3_2")
A_3_3_value=row.getValue("A_3_3")
```

Исправление возможных ошибок, если самая возрастная градация льда в группе отсутствует, а символ его формы есть. Логический оператор «==» означает «строго равно», а оператор «!=» – «не равно».

```
if (A_1_1_value==" ") and (F_1_1_value!=" "):
  if (A_1_2_value!=" ") and (F_1_2_value==" "):
     A_1_1_value_old=A_1_1_value
     A_1_2_value_old=A_1_2_value
     A_1_1_value=A_1_2_value_old
```

```
        A_1_2_value=A_1_1_value_old
    if (A_1_2_value==" ") and (F_1_2_value!=" "):
      if (A_1_3_value!=" ") and (F_1_3_value==" "):
        A_1_2_value_old=A_1_2_value
        A_1_3_value_old=A_1_3_value
        A_1_2_value=A_1_3_value_old
        A_1_3_value=A_1_2_value_old
    if (A_2_1_value==" ") and (F_2_1_value!=" "):
      if (A_2_2_value!=" ") and (F_2_2_value==" "):
        A_2_1_value_old=A_2_1_value
        A_2_2_value_old=A_2_2_value
        A_2_1_value=A_2_2_value_old
        A_2_2_value=A_2_1_value_old
    if (A_2_2_value==" ") and (F_2_2_value!=" "):
      if (A_2_3_value!=" ") and (F_2_3_value==" "):
        A_2_2_value_old=A_2_2_value
        A_2_3_value_old=A_2_3_value
        A_2_2_value=A_2_3_value_old
        A_2_3_value=A_2_2_value_old
    if (A_3_1_value==" ") and (F_3_1_value!=" "):
      if (A_3_2_value!=" ") and (F_3_2_value==" "):
        A_3_1_value_old=A_3_1_value
        A_3_2_value_old=A_3_2_value
        A_3_1_value=A_3_2_value_old
        A_3_2_value=A_3_1_value_old
    if (A_3_2_value==" ") and (F_3_2_value!=" "):
      if (A_3_3_value!=" ") and (F_3_3_value==" "):
        A_3_2_value_old=A_3_2_value
        A_3_3_value_old=A_3_3_value
        A_3_2_value=A_3_3_value_old
        A_3_3_value=A_3_2_value_old
```

Определение числа возрастных градаций льда в каждой группе градаций и задание числовых значений частных концентраций:

```
    if C1>0:
      if A_1_1_value!=" ":
        n_led_1=n_led_1+1
      if A_1_2_value!=" ":
        n_led_1=n_led_1+1
      if A_1_3_value!=" ":
        n_led_1=n_led_1+1
      if n_led_1==1:
```

```
        C_1_1=C1
      if n_led_1==2:
        C_1_1=C1/2.0
        C_1_2=C1/2.0
      if n_led_1==3:
        C_1_1=C1/3.0
        C_1_2=C1/3.0
        C_1_3=C1/3.0
    if C2>0:
      if A_2_1_value!=" ":
        n_led_2=n_led_2+1
      if A_2_2_value!=" ":
        n_led_2=n_led_2+1
      if A_2_3_value!=" ":
        n_led_2=n_led_2+1
      if n_led_2==1:
        C_2_1=C2
      if n_led_2==2:
        C_2_1=C2/2.0
        C_2_2=C2/2.0
      if n_led_2==3:
        C_2_1=C2/3.0
        C_2_2=C2/3.0
        C_2_3=C2/3.0
    if C3>0:
      if A_3_1_value!=" ":
        n_led_3=n_led_3+1
      if A_3_2_value!=" ":
        n_led_3=n_led_3+1
      if A_3_3_value!=" ":
        n_led_3=n_led_3+1
      if n_led_3==1:
        C_3_1=C3
      if n_led_3==2:
        C_3_1=C3/2.0
        C_3_2=C3/2.0
      if n_led_3==3:
        C_3_1=C3/3.0
        C_3_2=C3/3.0
        C_3_3=C3/3.0
```

Создание и обнуление числовых переменных частных концентраций

льда всех возрастных градаций:

```
C_old=0          #старый лёд
C_second=0       #двухлетний лёд
C_residual=0     #остаточный лёд
C_thick=0        #толстый лёд
C_medium=0       #средний лёд
C_thin=0         #тонкий лёд
C_grey_white=0   #серо-белый лёд
C_grey=0         #серый лёд
C_new=0          #начальный лёд
C_blin=0         #блинчатый лёд
```

Определение частных концентраций возрастных градаций:

```
#старый лёд
if (A_1_1_value=="10"):
  C_old=C_old+C_1_1
if (A_1_2_value=="10"):
  C_old=C_old+C_1_2
if (A_1_3_value=="10"):
  C_old=C_old+C_1_3
if (A_2_1_value=="10"):
  C_old=C_old+C_2_1
if (A_2_2_value=="10"):
  C_old=C_old+C_2_2
if (A_2_3_value=="10"):
  C_old=C_old+C_2_3
if (A_3_1_value=="10"):
  C_old=C_old+C_3_1
if (A_3_2_value=="10"):
  C_old=C_old+C_3_2
if (A_3_3_value=="10"):
  C_old=C_old+C_3_3
#двухлетний лёд
if (A_1_1_value=="12"):
  C_second=C_second+C_1_1
if (A_1_2_value=="12"):
  C_second=C_second+C_1_2
if (A_1_3_value=="12"):
  C_second=C_second+C_1_3
if (A_2_1_value=="12"):
  C_second=C_second+C_2_1
if (A_2_2_value=="12"):
```

```
  C_second=C_second+C_2_2
if (A_2_3_value=="12"):
  C_second=C_second+C_2_3
if (A_3_1_value=="12"):
  C_second=C_second+C_3_1
if (A_3_2_value=="12"):
  C_second=C_second+C_3_2
if (A_3_3_value=="12"):
  C_second=C_second+C_3_3
#остаточный лёд
if (A_1_1_value=="11"):
  C_residual=C_residual+C_1_1
if (A_1_2_value=="11"):
  C_residual=C_residual+C_1_2
if (A_1_3_value=="11"):
  C_residual=C_residual+C_1_3
if (A_2_1_value=="11"):
  C_residual=C_residual+C_2_1
if (A_2_2_value=="11"):
  C_residual=C_residual+C_2_2
if (A_2_3_value=="11"):
  C_residual=C_residual+C_2_3
if (A_3_1_value=="11"):
  C_residual=C_residual+C_3_1
if (A_3_2_value=="11"):
  C_residual=C_residual+C_3_2
if (A_3_3_value=="11"):
  C_residual=C_residual+C_3_3
#толстый однолетний лёд
if (A_1_1_value=="9"):
  C_thick=C_thick+C_1_1
if (A_1_2_value=="9"):
  C_thick=C_thick+C_1_2
if (A_1_3_value=="9"):
  C_thick=C_thick+C_1_3
if (A_2_1_value=="9"):
  C_thick=C_thick+C_2_1
if (A_2_2_value=="9"):
  C_thick=C_thick+C_2_2
if (A_2_3_value=="9"):
  C_thick=C_thick+C_2_3
```

```
if (A_3_1_value=="9"):
  C_thick=C_thick+C_3_1
if (A_3_2_value=="9"):
  C_thick=C_thick+C_3_2
if (A_3_3_value=="9"):
  C_thick=C_thick+C_3_3
#однолетний лёд средней толщины
if (A_1_1_value=="8"):
  C_medium=C_medium+C_1_1
if (A_1_2_value=="8"):
  C_medium=C_medium+C_1_2
if (A_1_3_value=="8"):
  C_medium=C_medium+C_1_3
if (A_2_1_value=="8"):
  C_medium=C_medium+C_2_1
if (A_2_2_value=="8"):
  C_medium=C_medium+C_2_2
if (A_2_3_value=="8"):
  C_medium=C_medium+C_2_3
if (A_3_1_value=="8"):
  C_medium=C_medium+C_3_1
if (A_3_2_value=="8"):
  C_medium=C_medium+C_3_2
if (A_3_3_value=="8"):
  C_medium=C_medium+C_3_3
#тонкий однолетний лёд
if (A_1_1_value=="7"):
  C_thin=C_thin+C_1_1
if (A_1_2_value=="7"):
  C_thin=C_thin+C_1_2
if (A_1_3_value=="7"):
  C_thin=C_thin+C_1_3
if (A_2_1_value=="7"):
  C_thin=C_thin+C_2_1
if (A_2_2_value=="7"):
  C_thin=C_thin+C_2_2
if (A_2_3_value=="7"):
  C_thin=C_thin+C_2_3
if (A_3_1_value=="7"):
  C_thin=C_thin+C_3_1
if (A_3_2_value=="7"):
```

```
  C_thin=C_thin+C_3_2
if (A_3_3_value=="7"):
  C_thin=C_thin+C_3_3
#серо-белый лёд
if (A_1_1_value=="5"):
  C_grey_white=C_grey_white+C_1_1
if (A_1_2_value=="5"):
  C_grey_white=C_grey_white+C_1_2
if (A_1_3_value=="5"):
  C_grey_white=C_grey_white+C_1_3
if (A_2_1_value=="5"):
  C_grey_white=C_grey_white+C_2_1
if (A_2_2_value=="5"):
  C_grey_white=C_grey_white+C_2_2
if (A_2_3_value=="5"):
  C_grey_white=C_grey_white+C_2_3
if (A_3_1_value=="5"):
  C_grey_white=C_grey_white+C_3_1
if (A_3_2_value=="5"):
  C_grey_white=C_grey_white+C_3_2
if (A_3_3_value=="5"):
  C_grey_white=C_grey_white+C_3_3
#серый лёд
if (A_1_1_value=="4"):
  C_grey=C_grey+C_1_1
if (A_1_2_value=="4"):
  C_grey=C_grey+C_1_2
if (A_1_3_value=="4"):
  C_grey=C_grey+C_1_3
if (A_2_1_value=="4"):
  C_grey=C_grey+C_2_1
if (A_2_2_value=="4"):
  C_grey=C_grey+C_2_2
if (A_2_3_value=="4"):
  C_grey=C_grey+C_2_3
if (A_3_1_value=="4"):
  C_grey=C_grey+C_3_1
if (A_3_2_value=="4"):
  C_grey=C_grey+C_3_2
if (A_3_3_value=="4"):
  C_grey=C_grey+C_3_3
```

```
    #начальный лёд
    if (A_1_1_value=="2"):
      C_new=C_new+C_1_1
    if (A_1_2_value=="2"):
      C_new=C_new+C_1_2
    if (A_1_3_value=="2"):
      C_new=C_new+C_1_3
    if (A_2_1_value=="2"):
      C_new=C_new+C_2_1
    if (A_2_2_value=="2"):
      C_new=C_new+C_2_2
    if (A_2_3_value=="2"):
      C_new=C_new+C_2_3
    if (A_3_1_value=="2"):
      C_new=C_new+C_3_1
    if (A_3_2_value=="2"):
      C_new=C_new+C_3_2
    if (A_3_3_value=="2"):
      C_new=C_new+C_3_3
    #блинчатый лёд
    if (A_1_1_value=="1"):
      C_blin=C_blin+C_1_1
    if (A_1_2_value=="1"):
      C_blin=C_blin+C_1_2
    if (A_1_3_value=="1"):
      C_blin=C_blin+C_1_3
    if (A_2_1_value=="1"):
      C_blin=C_blin+C_2_1
    if (A_2_2_value=="1"):
      C_blin=C_blin+C_2_2
    if (A_2_3_value=="1"):
      C_blin=C_blin+C_2_3
    if (A_3_1_value=="1"):
      C_blin=C_blin+C_3_1
    if (A_3_2_value=="1"):
      C_blin=C_blin+C_3_2
    if (A_3_3_value=="1"):
      C_blin=C_blin+C_3_3
```

Округление значений:

```
    C = C * 1000
    C_old = C_old * 1000
```

C_second = C_second*1000
C_residual = C_residual * 1000
C_thick = C_thick * 1000
C_medium = C_medium * 1000
C_thin = C_thin * 1000
C_grey_white = C_grey_white * 1000
C_grey = C_grey * 1000
C_new = C_new * 1000
C_blin = C_blin * 1000
C=round(C)
C_old=round(C_old)
C_second=round(C_second)
C_residual=round(C_residual)
C_thick=round(C_thick)
C_medium=round(C_medium)
C_thin=round(C_thin)
C_grey_white=round(C_grey_white)
C_grey=round(C_grey)
C_new=round(C_new)
C_blin=round(C_blin)
C=C/1000.0
C_old=C_old/1000.0
C_second=C_second/1000.0
C_residual=C_residual/1000.0
C_thick=C_thick/1000.0
C_medium=C_medium/1000.0
C_thin=C_thin/1000.0
C_grey_white=C_grey_white/1000.0
C_grey=C_grey/1000.0
C_new=C_new/1000.0
C_blin=C_blin/1000.0

Определение невязки:

nev=C-C_old-C_second-C_residual-C_thick-C_medium-C_thin-C_grey_white-C_grey-C_new-C_blin

Запись значений в таблицу атрибутов:

row.setValue("C", C)
row.setValue("C_old", C_old)
row.setValue("C_second",C_second)
row.setValue("C_residual", C_residual)
row.setValue("C_thick", C_thick)
row.setValue("C_medium", C_medium)

```
    row.setValue("C_thin", C_thin)
    row.setValue("C_gr_white", C_grey_white)
    row.setValue("C_grey", C_grey)
    row.setValue("C_new", C_new)
    row.setValue("C_blin",C_blin)
    row.setValue("nevyazka", nev)
```

Определение и запись в таблицу атрибутов доминирующей возрастной градации льда:

```
    if (C_old>C_second) or (C_old == C_second):
      if (C_old>C_residual) or (C_old == C_residual):
        if (C_old>C_thick) or (C_old == C_thick):
          if (C_old>C_medium) or (C_old == C_medium):
            if (C_old>C_thin) or (C_old == C_thin):
              if (C_old>C_grey_white) or (C_old == C_grey_white):
                if (C_old>C_grey) or (C_old == C_grey):
                  if (C_old>C_new) or (C_old == C_new):
                    if (C_old>C_blin) or (C_old == C_blin):
                      row.setValue("Preobl", "старый")
    if (C_second>C_old):
      if (C_second>C_residual) or (C_second==C_residual):
        if (C_second>C_thick) or (C_second==C_thick):
          if (C_second>C_medium) or (C_second==C_medium):
            if (C_second>C_thin) or (C_second==C_thin):
              if (C_second>C_grey_white) or (C_second==C_grey_white):
                if (C_second>C_grey) or (C_second==C_grey):
                  if (C_second>C_new) or (C_second==C_new):
                    if (C_second>C_blin) or (C_second==C_blin):
                      row.setValue("Preobl", "двухлетний")
    if (C_residual>C_old):
      if (C_residual>C_second):
        if (C_residual>C_thick) or (C_residual == C_thick):
          if (C_residual>C_medium) or (C_residual == C_medium):
            if (C_residual>C_thin) or (C_residual == C_thin):
              if (C_residual>C_grey_white) or (C_residual ==
              C_grey_white):
                if (C_residual>C_grey) or (C_residual == C_grey):
                  if (C_residual>C_new) or (C_residual == C_new):
                    if (C_residual>C_blin) or (C_residual == C_blin):
                      row.setValue("Preobl", "остаточный")
    if (C_thick>C_old):
      if (C_thick>C_second):
```

```
    if (C_thick>C_residual):
      if (C_thick>C_medium) or (C_thick == C_medium):
        if (C_thick>C_thin) or (C_thick == C_thin):
          if (C_thick>C_grey_white) or (C_thick == C_grey_white):
            if (C_thick>C_grey) or (C_thick == C_grey):
              if (C_thick>C_new) or (C_thick == C_new):
                if (C_thick>C_blin) or (C_thick == C_blin):
                  row.setValue("Preobl", "толстый")
if (C_medium>C_old):
  if (C_medium>C_second):
    if (C_medium>C_residual):
      if (C_medium>C_thick):
        if (C_medium>C_thin) or (C_medium == C_thin):
          if (C_medium>C_grey_white) or (C_medium ==
          C_grey_white):
            if (C_medium>C_grey) or (C_medium == C_grey):
              if (C_medium>C_new) or (C_medium == C_new):
                if (C_medium>C_blin) or (C_medium == C_blin):
                  row.setValue("Preobl", "средний")
if (C_thin>C_old):
  if (C_thin>C_second):
    if (C_thin>C_residual):
      if (C_thin>C_thick):
        if (C_thin>C_medium):
          if (C_thin>C_grey_white) or (C_thin == C_grey_white):
            if (C_thin>C_grey) or (C_thin == C_grey):
              if (C_thin>C_new) or (C_thin == C_new):
                if (C_thin>C_blin) or (C_thin == C_blin):
                  row.setValue("Preobl", "тонкий")
if (C_grey_white>C_old):
  if (C_grey_white>C_second):
    if (C_grey_white>C_residual):
      if (C_grey_white>C_thick):
        if (C_grey_white>C_medium):
          if (C_grey_white>C_thin):
            if (C_grey_white>C_grey) or (C_grey_white == C_grey):
              if (C_grey_white>C_new) or (C_grey_white == C_new):
                if (C_grey_white>C_blin) or (C_grey_white == C_blin):
                  row.setValue("Preobl", "серо-белый")
if (C_grey>C_old):
  if (C_grey>C_second):
```

```
        if (C_grey>C_residual):
          if (C_grey>C_thick):
            if (C_grey>C_medium):
              if (C_grey>C_thin):
                if (C_grey>C_grey_white):
                  if (C_grey>C_new) or (C_grey == C_new):
                    if (C_grey>C_blin) or (C_grey == C_blin):
                      row.setValue("Preobl", "серый")
    if (C_new>C_old):
      if (C_new>C_second):
        if (C_new>C_residual):
          if (C_new>C_thick):
            if (C_new>C_medium):
              if (C_new>C_thin):
                if (C_new>C_grey_white):
                  if (C_new>C_grey):
                    if (C_new>C_blin) or (C_new == C_blin):
                      row.setValue("Preobl", "начальный")
    if (C_blin>C_old):
      if (C_blin>C_second):
        if (C_blin>C_residual):
          if (C_blin>C_thick):
            if (C_blin>C_medium):
              if (C_blin>C_thin):
                if (C_blin>C_grey_white):
                  if (C_blin>C_grey):
                    if (C_blin>C_new):
                      row.setValue("Preobl", "блинчатый")
    cursor.updateRow(row)
```

Завершение цикла просмотра и редактирования записей атрибутивной таблицы и удаление курсора:

```
del cursor, row
```

1.22. Применение диалоговых окон и подпрограмм

Во всех уже рассмотренных компьютерных программах указание адресов и названий файлов как источников исходных данных, так и результатов работы программ выполняется непосредственно в самом программном коде. Логика здесь такая: «Зачем создавать интерфейс пользователя, когда есть интерфейсы самой ArcGIS? Если пользователь сам указывает с помощью диалоговых окон источники

данных и задаёт расположения и названия файлов с результатами работы программы, то это требует времени и снижает скорость работы программы». Однако в некоторых случаях применение в программе диалоговых окон оказывается вполне оправданным. Рассмотрим такой случай на примере программы для расчёта ледовитости акватории. Ледовитость – это отношение площади какой-либо акватории, занятой льдами с любой общей сплоченностью, к площади данной акватории.

Новым в блоке подключения модулей и библиотек является подключение библиотеки Tkinter, с помощью которой происходит создание графических объектов (виджетов), в данном случае диалоговых окон:

```
# -*- coding: utf-8 -*-
import os
import sys
import arcpy
import arcpy.mapping as mapping
from Tkinter import *
```

В операционной системе приложение пользователя заключается в главное окно, в котором находятся остальные графические объекты (виджеты). Объект – окно верхнего уровня создается путём обращения к классу «Tk» модуля Tkinter. Необходимо создать переменную, связанную с этим окном. Обычно её название – «root»:

```
root=Tk()
```

Далее указываем название диалогового окна:

```
root.title("Выбор слоёв ледовых карт для объединения")
```

И свойства диалогового окна:

```
root.resizable(True, True)
```

В этом операторе заданы два логических значения «True», которые позволяют пользователю изменять размеры окна как по горизонтали, так и по вертикали. Затем в данной программе следует текст трёх подпрограмм. Каждая подпрограмма начинается с зарезервированного слова «def». Затем следует название подпрограммы, а в скобках – параметры:

```
#Подпрограмма выбора слоёв ледовых карт
def vibor_ice(event,lst_nsellayers):
   lst_nsellayers=the_listbox.curselection()
#Подпрограмма выбора слоя акваторий для расчётов ледовитости
def vibor_akvator(event,lst_nselakvator):
   lst_nselakvator=the_listbox.curselection()
#Подпрограмма выбора поля идентификаторов акваторий
```

```
def vibor_id(event,lst_fields):
   lst_fields=the_listbox.curselection()
```
Во всех этих подпрограммах первый параметр в скобках – «event», который означает, что работа подпрограммы начинается при каком-либо внешнем событии. Возможны три типа внешних событий: нажатие на клавишу манипулятора «мышь», нажатие на клавишу клавиатуры, изменение другого графического объекта. Вторые параметры в скобках – это объекты, которые будут созданы или изменены в результате работы подпрограммы. В каждой из подпрограмм только по одному оператору, в результате работы которого создаётся список выбранных элементов списка в диалоговом окне. Метод «curselection» создаёт кортеж индексов выбранных элементов экземпляра Listbox(), т.е. выведенного на экран списка. Кортеж – это неизменяемая последовательность элементов. Далее расположен уже «традиционный» блок операторов:

```
#возвращение текущего документа карты
the_mxd = mapping.MapDocument('Current')
#возвращение списка фреймов текущего документа карты
frame_list = arcpy.mapping.ListDataFrames(the_mxd)
#возвращение первого фрейма в списке фреймов
frame1=frame_list[0]
#возвращение системы координат этого фрейма
frame1CoordSys = frame1.spatialReference
#возвращение списка всех слоёв первого фрейма
SloyList=arcpy.mapping.ListLayers(the_mxd,"*",frame1)
```

А вот здесь появляется «новшество» – создаётся виджет «список слоёв», который потом появится на экране:

```
the_listbox=Listbox(root,height=20,width=80,selectmode=EXTENDED)
```

Listbox – это графический объект (виджет), представляющий собой список, из элементов которого может выбрать один или несколько. Свойство «selectmode» при значении SINGLE позволяет выбрать один элемент из списка, а при значении EXTENDED – неограниченное их количество. В виджет Listbox добавляются все элементы списка слоёв фрейма:

```
for i in SloyList:
   the_listbox.insert(END,i)
```

Параметр «END» означает, что каждый элемент списка добавляется в конец списка в виджете. Далее выполняется упаковщик «pack»:

```
the_listbox.pack()
```

Упаковщиком называется специальный механизм, размещающий виджеты на окне. В модуле Tkinter существуют три упаковщика: pack,

place, grid. При этом в одном виджете возможно использование только одного типа упаковки. Самым «интеллектуальным» из них является pack(). Создание виджета кнопки:
but_ice = Button(root)
Создание надписи на кнопке:
but_ice["text"]="Слои характеристик льда выбраны"
Далее следует указание, что после выполнения связанно с кнопкой команды главное окно будет закрыто:
but_ice["command"]=root.quit

Пока на экране никаких виджетов не появляется. Это всё ещё предварительные действия по созданию объектов. Чтобы приступить к выбору слоёв фрейма на экране компьютера, необходимо применить метод «bind». Этот метод привязывает некоторое событие к какому-либо действию, например, нажатию клавиши «мыши», нажатию клавиши клавиатуры и т.п. Метод имеет два обязательных аргумента: название события и функцию, вызываемую при наступлении события. Есть ещё третий необязательный аргумент: строковое значение "+". При его наличии данное привязываемое событие добавляется к уже существующим привязкам. Если этот аргумент отсутствует, то данная привязка является единственной, и все предыдущие привязки данного события к виджету отменяются. Применение метода «bind» в данной программе:
but_ice.bind("<Button-1>",vibor_ice)
Аргумент "<Button-1>" означает, что событием является нажатие левой клавиши «мыши», а аргумент «vibor_ice» приводит к работе данной подпрограммы. Кнопка размещается на окне:
but_ice.pack()
Tkinter – это событийно-ориентированная библиотека. В таких приложениях есть главный цикл обработки событий. В Tkinter он запускается с помощью метода mainloop. Для выхода из интерпретатора и завершения цикла обработки событий применяется метод quit. Только после оператора с применением метода mainloop на экране появится окно с кнопкой (рис. 7):
root.mainloop()
Несмотря на то, что в подпрограмме уже сделан выбор слоёв, в тексте программы нужно поставить оператор записи номеров выбранных слоёв в список:
lst_nsellayers=the_listbox.curselection()
Определение числа выбранных слоёв:
kolsellayers=len(lst_nsellayers)
Теперь окно можно закрыть:

root.destroy()
Затем создаётся пустой список будущих названий слоёв ледовых карт в прямоугольной системе координат фрейма:
IcelayersList=[]

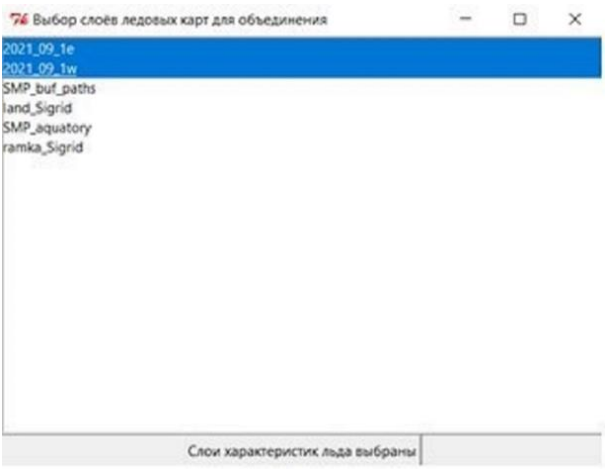


Рисунок 7. Вид созданного диалогового окна

Цикл просмотра списка номеров выбранных слоёв ледовых карт:
```
for i in range(kolsellayers):
    a=lst_nsellayers[i]
    str_a=str(a)
    n=int(str_a)
    sel_layer=SloyList[n]
    sel_layer_DatasetName=sel_layer.datasetName
```
Занесение в строковую переменную названия текущего слоя:
```
    str_sel_layer=str(sel_layer_DatasetName)
```
Определение временного интервала, к которому относится ледовая карта:
```
    dlina_stroki=len(str_sel_layer)
    str_data=str_sel_layer[0:dlina_stroki-1]
```
Папка расположения шейпфайлов определяется по первому слою списка:
```
    if i==0:
      #Запоминание даты первой карты списка
      str_data0=str_data
```

```
    sel_layer_workspacePath=sel_layer.workspacePath
    str_sel_layer_workspacePath=str(sel_layer_workspacePath)
```
Строковая переменная, в которую заносится адрес и название шейпфайла – источника первого слоя в списке:
```
  str_tek_layer=str_sel_layer_workspacePath+"\\"+str_sel_layer
  +".shp"
```
Задание адреса и названия спроецированного шейпфайла ледовой информации:
```
  str_tek_layer_pr=str_sel_layer_workspacePath+"\\"
  +str_sel_layer+"_pr.shp"
```
Добавление в список названия спроецированного шейпфайла ледовой информации:
```
  IcelayersList.append(str_tek_layer_pr)
```
Проверка: возможно, шейпфайл с этим названием и адресом уже существует?
```
  if arcpy.Exists(str_tek_layer_pr):
    #Удаление "предшественника" в случае его существования
    arcpy.Delete_management(str_tek_layer_pr)
```
Перепроецирование исходного слоя ледовой информации
```
  arcpy.Project_management(str_tek_layer, str_tek_layer_pr,
  frame1CoordSys)
```
Цикл просмотра списка номеров выбранных слоёв ледовых карт закончен. Задаётся имя шейпфайла – результата объединения:
```
str_merge='map_'+str_data0+'.shp'
```
Затем задаётся имя и адрес шейпфайла результата объединения:
```
str_merge_layer=str_sel_layer_workspacePath+"\\"+str_merge
```
Проверка: возможно, шейпфайл с этим названием и адресом уже существует?
```
if arcpy.Exists(str_merge_layer):
  #Удаление "предшественника" в случае его существования
  arcpy.Delete_management(str_merge_layer)
```
Слияние выбранных слоёв фрейма:
```
arcpy.Merge_management(IcelayersList,str_merge_layer)
```
Удаление возможных пространственных дубликатов:
```
arcpy.DeleteIdentical_management(str_merge_layer, "Shape", "", "0")
```
Задание имени шейпфайла, который будет содержать только объекты со льдом:
```
str_led='led_'+str_data0+'.shp'
```
Задание имени и адреса шейпфайла без объектов чистой воды:
```
str_led_layer=str_sel_layer_workspacePath+"\\"+str_led
```
Проверка: возможно, шейпфайл с этим названием и адресом уже

существует?
```
if arcpy.Exists(str_led_layer):
  #Удаление "предшественника" в случае его существования
  arcpy.Delete_management(str_led_layer)
```
Задание имени шейпфайла, содержащего объединение объектов со льдом:
```
str_sum_led='sum_led_'+str_data0+'.shp'
```
Задание имени и адреса шейпфайла объединения всех участков со льдом:
```
str_sum_led_layer=str_sel_layer_workspacePath+"\\"+str_sum_led
```
Проверка: возможно, шейпфайл с этим названием и адресом уже существует?
```
if arcpy.Exists(str_sum_led_layer):
  #Удаление "предшественника" в случае его существования
  arcpy.Delete_management(str_sum_led_layer)
```
Запись в новый шейпфайл объектов, удовлетворяющих условию наличия льда:
```
arcpy.Select_analysis(str_merge_layer, str_led_layer, "\"S\" <> 'free'")
```
В этот шейпфайл записаны все пространственные объекты объединённого шейпфайла ледовых карт, в которых присутствует лёд любой сплоченности. Выяснение числа строк в таблице атрибутов этого шейпфайла:
```
n_rows=arcpy.GetCount_management(str_led_layer)
```
Конвертация числа строк таблицы в строковую переменную:
```
str_nrows=str(n_rows)
```
Проверка, а есть ли вообще лёд, и в случае его наличия объединение всех пространственных объектов со льдом:
```
if str_nrows!='0':
  #Объединение всех объектов слоя
  # Process: Слияние по атрибуту
  arcpy.Dissolve_management(str_led_layer, str_sum_led_layer, "",
  "", "MULTI_PART","DISSOLVE_LINES")
```
Затем также с помощью модуля Tkinter выполняется выбор пользователем слоя акваторий, для которых будет рассчитана ледовитость. Этот слой может содержать и только одну акваторию. Здесь выбирается только один слой, поэтому при выборе в качестве значения параметра selectmode задаётся SINGLE. Алгоритм выбора слоя такой же, как при выборе слоёв ледовых карт:
```
root=Tk()
root.title("Выбор слоя акваторий")
root.resizable(True, True)
```

```
the_listbox=Listbox(root,height=20,width=80,selectmode=SINGLE)
for i in SloyList:
  the_listbox.insert(END,i)
the_listbox.pack()
but_akvator = Button(root)
but_akvator["text"]="Слой акваторий выбран"
but_akvator["command"]=root.quit
but_akvator.bind("<Button-1>",vibor_akvator)
but_akvator.pack()
root.mainloop()
```
Ссписок, состоящий из одного номера выбранного слоя акваторий:
```
lst_nselakvator=the_listbox.curselection()
root.destroy()
```
Определение названия выбранного слоя, адреса и названия шейпфайла его источника:
```
a=lst_nselakvator[0]
str_a=str(a)
n=int(str_a)
sel_akvator=SloyList[n]
sel_akvator_DatasetName=sel_akvator.datasetName
str_sel_akvator=str(sel_akvator_DatasetName)
sel_akvator_workspacePath=sel_akvator.workspacePath
str_sel_akvator_workspacePath=str(sel_akvator_workspacePath)
str_tek_akvator=str_sel_akvator_workspacePath+"\\"
+str_sel_akvator+".shp"
```
Задание адреса и названия спроецированного шейпфайла акваторий:
```
str_tek_akvator_pr=str_sel_akvator_workspacePath+"\\"
+str_sel_akvator+"_pr.shp"
```
Проверка: возможно, шейпфайл с этим названием и адресом уже существует?
```
if arcpy.Exists(str_tek_akvator_pr):
  #Удаление "предшественника" в случае его существования
  arcpy.Delete_management(str_tek_akvator_pr)
```
Перепроецирование исходного шейпфайла акваторий:
```
arcpy.Project_management(str_tek_akvator, str_tek_akvator_pr,
frame1CoordSys)
```
Далее выполняется выбор поля, содержащего идентификаторы акваторий. Программа позволяет рассчитать ледовитость сразу ряда акваторий. Сначала создаётся список полей таблицы атрибутов шейпфайла акваторий:
```
fields = arcpy.ListFields(str_tek_akvator_pr)
```

```
NameFieldList=[]
for field in fields:
  tek_field_name=field.name
  NameFieldList.append(tek_field_name)
```
Затем с помощью модуля Tkinter пользователь выбирает поле с идентификаторами акваторий:
```
root=Tk()
root.title("Выбор поля идентификаторов акваторий")
the_listbox=Listbox(root,height=20,width=80,selectmode=SINGLE)
for i in NameFieldList:
  the_listbox.insert(END,i)
the_listbox.pack()
but_id_field = Button(root)
but_id_field["text"]="Поле идентификаторов акваторий выбрано"
but_id_field["command"]=root.quit
but_id_field.bind("<Button-1>",vibor_id)
but_id_field.pack()
root.mainloop()
```
Создание списка, состоящего из одного номера выбранного поля идентификаторов акваторий:
```
lst_nselid=the_listbox.curselection()
root.destroy()
```
Затем следует несколько «изощрённое вытаскивание» названия этого поля в строковую переменную:
```
sel_field=lst_nselid[0]
str_sel_field=str(sel_field)
n=int(str_sel_field)
sel_Name=NameFieldList[n]
str_sel_Name=str(sel_Name)
```
Теперь в строковой переменной «str_sel_Name» содержится имя поля идентификаторов акваторий. В таблицу атрибутов нужно добавить поле «Akva_km2» для последующей записи площади каждой акватории, разумеется, если это поле ещё не существует. Для проверки этой возможности создаём логическую переменную «field_exist_Akva_area», и присваиваем её значение:
```
field_exist_Akva_area=False
```
Просматриваем в цикле все поля таблицы атрибутов, нет ли уже поля с таким названием:
```
for field in fields:
  tek_field_name=field.name
```
Если встретилось поле с этим названием, изменяем значение

логической переменной:

```
  if tek_field_name=='Akva_km2':
    field_exist_Akva_area=True
```

Если поля с названием «Akva_km2» в таблице нет, то создаём поле с этим названием:

```
if field_exist_Akva_area==False:
  arcpy.AddField_management(str_tek_akvator_pr, "Akva_km2",
  "FLOAT", "12", "1", "", "", "NULLABLE", "NON_REQUIRED",
  "")
```

Выполняем расчёты площадей акваторий:

```
arcpy.CalculateField_management(str_tek_akvator_pr, "Akva_km2",
"!shape.area@squarekilometers!", "PYTHON_9.3")
```

На следующем этапе в программе выполняется оверлейная операция пересечения слоя акваторий со слоем объектов ледовитости. Сначала создаём строковую переменную и заносим в неё имя шейпфайла для создаваемого слоя пересечения:

```
str_ledovit='ledovit_'+str_data0+'.shp'
```

А затем строковую переменную с адресом и именем этого шейпфайла:

```
str_ledovit_layer=str_sel_layer_workspacePath+"\\"+str_ledovit
```

Проверка: не существует ли уже шейпфайл с таким именем и адресом?

```
if arcpy.Exists(str_ledovit_layer):
  arcpy.Delete_management(str_ledovit_layer)
```

Выполнение самого пересечения:

```
arcpy.Intersect_analysis([str_tek_akvator_pr,str_sum_led_layer],
str_ledovit_layer,"ALL","","INPUT")
```

Удаление пространственных дубликатов:

```
arcpy.DeleteIdentical_management(str_ledovit_layer, "Shape", "", "0")
```

Задание имени шейпфайла, в который будет записан результат объединения объектов с одинаковыми идентификаторами акваторий:

```
str_diss_ledovit='diss_ledovit_'+str_data0+'.shp'
```

Задание его адреса и имени:

```
str_diss_ledovit_layer=str_sel_layer_workspacePath+"\\"
+str_diss_ledovit
```

Проверка: возможно, шейпфайл с этим названием и адресом уже существует?

```
if arcpy.Exists(str_diss_ledovit_layer):
  #Удаление "предшественника" в случае его существования
  arcpy.Delete_management(str_diss_ledovit_layer)
```

Создание строковой переменной, содержащей поле идентификатора акваторий и поле «Akva_km2»:

```
str_diss_fields=str_sel_Name+';Akva_km2'
```

Объединение объектов с одинаковыми идентификаторами акваторий и одинаковыми величинами площадей акваторий:

```
arcpy.Dissolve_management(str_ledovit_layer,str_diss_ledovit_layer,
str_diss_fields, "","MULTI_PART", "DISSOLVE_LINES")
```

Добавление в таблицу атрибутов поля «Led_km2» (площадь зон с наличием льда):

```
arcpy.AddField_management(str_diss_ledovit_layer, "Led_km2",
"FLOAT", "12", "1", "", "","NULLABLE", "NON_REQUIRED", "")
```

Расчёт значений этого поля:

```
arcpy.CalculateField_management(str_diss_ledovit_layer,
"Led_km2","!shape.area@squarekilometers!", "PYTHON_9.3")
```

Добавление в таблицу атрибутов поля «Ledovit» для записи ледовитости каждой акватории:

```
arcpy.AddField_management(str_diss_ledovit_layer, "Ledovit", "FLOAT",
"5", "1", "", "", "NULLABLE", "NON_REQUIRED", "")
```

Создание объекта типа «курсор» для просмотра атрибутивной таблицы:

```
cursor = arcpy.UpdateCursor(str_diss_ledovit_layer)
```

Цикл просмотра таблицы с расчётами ледовитости каждой акватории:

```
for row in cursor:
  Pl_akva = row.getValue("Akva_km2")
  Pl_led = row.getValue("Led_km2")
  Ledovit=Pl_led/Pl_akva*100
  row.setValue("Ledovit", Ledovit)
  cursor.updateRow(row)
# Завершение цикла просмотра атрибутивной таблицы
del cursor, row  #Удаление курсора
```

Задание. Создайте программу с диалоговым окном выбора слоя фрейма.

2. Обработка результатов исследований в среде Quantum GIS

Применяемая в Quantum GIS версия языка Python существенно отличается от версии, применяемой в среде ArcGIS, т.к. здесь применяются другие библиотеки функций. Поэтому отлаженная и великолепно работающая в ArcGIS программа в Quantum GIS работать не будет. Для выполнения таких же операций будет проще написать новую программу. Разумеется, синтаксис операций присвоения значений переменным, математических операций, организации циклов останется неизменным. Рассмотрим особенности версии языка для применения в среде Quantum GIS на примере программы для перепроецирования шейпфайла.

2.1. Перепроецирование шейпфайла

Как, наверное, уже ожидает читатель, получивший опыт программирования на версии языка Python для ArcGIS, в начале программы нужно подключить библиотеки и модули. Вы совершенно правы! Начинаем именно с этого:

```
from qgis.core import QgsProcessing
from qgis.core import QgsProcessingAlgorithm
from qgis.core import QgsProcessingMultiStepFeedback
from qgis.core import (QgsCoordinateReferenceSystem,
QgsCoordinateTransform,QgsProject,QgsVectorLayer)
import processing
import os
import ogr
import osr
```

Здесь автор собрал все когда-либо понадобившиеся подключения модулей и библиотек по принципу «Кашу маслом не испортишь!». Далее нужно выяснить систему координат электронной карты, т.к. алгоритм действий прежний: загружаем шейпфайл и сохраняем его в новый шейпфайл с записью координат объектов уже в системе координат данной карты. Поскольку в проекте Quantum GIS может быть только одна электронная карта, система координат электронной карты является системой координат проекта, и параметры системы координат можно узнать в свойствах проекта. Помним, что в ArcGIS систему координат фрейма можно получить с помощью четырёх строк кода: получить текущий документ карты, получить список фреймов этого документа, в списке получить нужный фрейм по его номеру в списке, и затем получить систему координат этого фрейма. Всё изящно

и логично. В версии языка Python для Quantum GIS тоже есть своя логика, но получение системы координат выглядит более громоздко. Сначала нужно получить WKT (Well Known Text) системы координат:

```
wkt = 'PROJCRS["Stereographic_North_Pole",BASEGEOGCRS["WGS
    84", ' \ 'DATUM["World Geodetic System 1984",' \
    'ELLIPSOID["WGS 84", \
    6378137,298.257223563,LENGTHUNIT["metre",1]], ' \
    'ID["EPSG",6326]],PRIMEM["Greenwich",0, ' \
    'ANGLEUNIT["Degree",0.0174532925199433]], \
    CONVERSION["unnamed", ' \
    'METHOD["Polar Stereographic (variant B)",ID["EPSG",9829]], ' \
    'PARAMETER["Latitude of standard parallel",90, ' \
    'ANGLEUNIT["Degree",0.0174532925199433], ' \
    'ID["EPSG",8832]],PARAMETER["Longitude of origin", ' \
    '120,ANGLEUNIT["Degree",0.0174532925199433], ' \
    'ID["EPSG",8833]],PARAMETER["False easting",0, ' \
    'LENGTHUNIT["metre",1],ID["EPSG",8806]], ' \
    'PARAMETER["False northing",0,LENGTHUNIT["metre",1], ' \
    'ID["EPSG",8807]]],CS[Cartesian,2],AXIS["(E)",south, ' \
    'MERIDIAN[90,ANGLEUNIT["degree",0.0174532925199433, ' \
    'ID["EPSG",9122]]],ORDER[1],LENGTHUNIT["metre",1, ' \
    'ID["EPSG",9001]]],AXIS["(N)",south,MERIDIAN[180, ' \
    'ANGLEUNIT["degree",0.0174532925199433,ID["EPSG",9122]]], ' \
    'ORDER[2],LENGTHUNIT["metre",1,ID["EPSG",9001]]]]'
```

Не падайте в обморок, дорогой читатель! Вам не придётся разрабатывать подобный код. Всего-навсего нужно открыть «Проект» \ «Свойства» \ «Система координат». На экране появится окно «Свойства проекта – Система координат». В его нижней части находятся два прямоугольника. В правом расположена карта с выделением той части Земли, для изображения которой применяется данная система координат, а в левом – название системы координат и её описание в форматах WKT и Proj4 (рис. 8). Скопируйте отсюда WKT и вставьте в текст программы. Только не забывайте ставить символ переноса строки – обратный слэш (косую черту). На основании WKT определяется система координат проекта:

```
crs = QgsCoordinateReferenceSystem(wkt)
```

Эта программа также разработана для автоматизации выполнения пространственных операций с шейпфайлами. Поэтому исходные шейпфайлы для их перепроецирования будут «подаваться» с использованием вложенных циклов, с помощью которых определяются адрес и название исходного шейпфайла, которое

содержит год, месяц и декаду или полудекаду, к которым относятся данные.

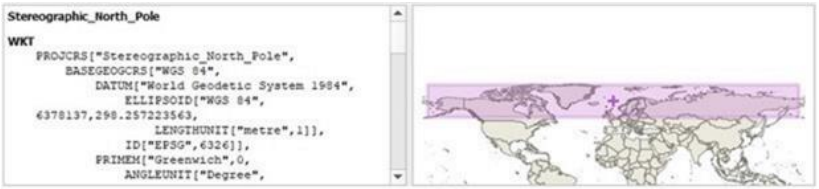


Рисунок 8. Описание системы координат в формате WKT в Свойствах проекта

Для отладки программы лучше, если списки лет, месяцев, декад и полудекад содержат только по одному элементу:

```
lst_God=['2000']
lst_Mes=['05']
lst_Dec=['3a']
```

Определение числа элементов в списках:

```
n_let = len(lst_God)
n_mes = len(lst_Mes)
n_dec = len(lst_Dec)
```

Видно, что по синтаксису эти операторы ничем не отличаются от операторов версии языка Python для ArcGIS. Организация циклов в программе также не имеет отличий от Python для ArcGIS:

```
for i_god in range (n_let):
  tek_god=lst_God[i_god]
  for i_mes in range(n_mes):
    tek_mes=lst_Mes[i_mes]
    for i_dec in range (n_dec):
      tek_dec=lst_Dec[i_dec]
```

Задание адреса и названия исходного шейпфайла:

```
      str_led_karta='F:\\KLVC_VSM_1998_2000\\'+tek_god+'_'
     +tek_mes+'_'+tek_dec+'_KLVC_VSM.shp'
```

А дальше начинаются кардинальные различия версий языка Python. Определение существования этого шейпфайла:

```
      if os.path.exists(str_led_karta):
```

В случае его существования выполняется загрузка объекта типа «Векторный слой»:

```
        ledlayer = QgsVectorLayer(str_led_karta, "tek_led_karta", "ogr")
```

В этом операторе «str_led_karta» – строковая переменная, содержащая

адрес и название источника данных; "tek_led_karta" – название создаваемого по этим данным векторного слоя; "ogr" – название провайдера. В данном случае провайдером является OGR –свободная библиотека для работы с векторными данными. Её нужно указывать в случаях использования данных шейпфайлов. В случае использования этого провайдера в качестве идентификатора источника данных используется путь к файлу с его названием. В этой программе реализована возможность задания в цикле разных источников данных, поэтому адрес и название файла заранее записывается в строковую переменную. Возможно использование ряда других провайдеров: баз данных PostGIS, файлов формата CSV или других текстовых форматов с разделителями, файлов формата GPX (треки, маршруты, путевые точки спутниковых навигаторов), баз данных SpatiaLite, WKB-геометрий баз данных MySQL.

Далее в программном коде следует задать значение строковой переменной с адресом и названием шейпфайла результата:

```
str_led_karta_pr='F:\\2021_Obrab_route_SabBer\\Pr_led_karta\\'
+tek_god+'_'+tek_mes+'_'+tek_dec+'.shp'
```

Наконец, оператор перепроецирования:

```
QgsVectorFileWriter.writeAsVectorFormat(ledlayer,
str_led_karta_pr, "CP1250", crs, "ESRI Shapefile")
```

Параметр «ledlayer» – это векторный слой, который будет спроецирован в шейпфайл, адрес и название которого хранятся во втором параметре оператора – строковой переменной «str_led_karta_pr». Третий параметр оператора определяет кодировку текстовых полей в таблице атрибутов результата перепроецирования. Правильное указание кодировки необходимо в случае использования символов кириллицы. В данном примере задано значение кодировки "CP1250". Если есть символы кириллицы, нужно поставить "CP1251". Четвёртый параметр – система координат, в которой будут записаны координаты объектов результата перепроецирования. Пятый параметр "ESRI Shapefile" – формат результата.

Задание. Напишите программу перепроецирования шейпфайлов.

2.2. Вставка текста программы в окно Редактора Python для Quantum GIS и выполнение программы

Итак, текст программы в редакторе «Блокнот» написан. Теперь на «Панели инструментов» интерфейса Quantum GIS наводим курсор на опцию «Модули», и нажимаем левую клавишу

манипулятора «мышь». Появится ниспадающее меню, в котором наводим курсор на опцию «Консоль Python» (рис. 9), и снова нажимаем на левую клавишу «мыши».

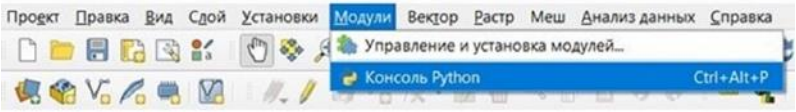


Рисунок 9. Вызов консоли Python

Ниже электронной карты появится окно консоли Python, состоящее из двух окон. Правое окно – Редактор. Вставляем в него текст программы. Для этого навести курсор внутрь редактора, и нажать правую клавишу манипулятора «мышь». Появится контекстное меню, в котором следует выбрать опцию «Вставить». Текст программы появится в окне Редактора. Наводим курсор на зелёный треугольник в Инструментах Редактора (подсказка при наведении курсора – «Выполнить сценарий», и нажимаем левую клавишу «мыши» (рис. 10).

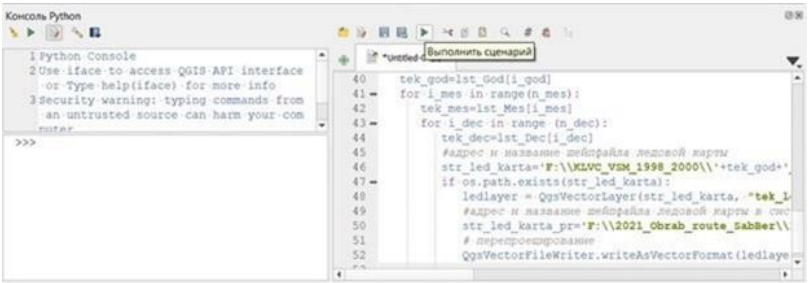


Рисунок 10. Запуск программы на выполнение

Программа выполняется. Особенность Quantum GIS: в отличие от ArcGIS слой результата перепроецирования не появится на карте автоматически. Это даже хорошо, т.к. не расходуется оперативная память компьютера.

Задание. Запустите на исполнение программу перепроецирования шейпфайлов.

2.3. Пересечение векторных слоёв

Рассмотрим программу для решения задачи пересечения большого количества слоёв ледовой информации с буферной зоной стандартного маршрута плаваний. Буфер маршрута состоит из одного полигонального объекта и записан в шейпфайл «buf_20km.shp», который уже добавлен в качестве слоя данного проекта, в котором будет выполняться эта программа. Имя этого слоя – «'buf_20km'». Система координат шейпфайла буферной зоны маршрута такая же, как у самого проекта. Имена шейпфайлов ледовой информации содержат символы года, месяца, декады или полудекады, т.е. интервала времени, к которому относится информация. Система координат шейпфайлов ледовой информации также не отличается от системы координат проекта. В начале программы подключаем модули и библиотеки:

```
import sys
from osgeo import ogr
```

Затем создаём источник данных «data», т.е. открываем каталог (папку) в качестве источника данных с возможностью записи в него:

```
data = ogr.Open(r'F:\Proba_intersection', 1)
```

В данном операторе «r'F:\Proba_intersection'» – указание расположения каталога (обратите внимание, что перед записью места каталога перед символом апострофа «'» нужно поставить символ «r»; «1» – разрешение на чтение и запись в каталог, «0» – только чтение.

Естественно, что необходимо проверить существование данного каталога:

```
if data is None:
  sys.exit('Could not open folder.')
```

Если каталог не существует, на экране появится сообщение: «'Could not open folder (Невозможно открыть этот каталог)». В случае такого сообщения пользователь просто прекращает работу программы.

Если такое сообщение не появилось, то в программе выполняется загрузка слоя буферной зоны стандартного маршрута плаваний:

```
lyr_bufer = data.GetLayer('buf_20km')
```

Получение единственного объекта слоя буферной зоны:

```
obj_buf = lyr_bufer.GetFeature(0)
```

Извлечение геометрии этого объекта:

```
geom_buf = obj_buf.geometry().Clone()
```

Теперь организуется цикл загрузки шейпфайлов ледовой информацией. Перед циклами необходимо создать списки строковых переменных обозначений лет, месяцев, декад и полудекад:

```
lst_God=['2000']
```

```
lst_Mes=['04']
lst_Dec=['3b']
```
В данном примере программы каждый из списков состоит только из одного элемента. Определение числа элементов в списках:
```
n_let = len(lst_God)
n_mes = len(lst_Mes)
n_dec = len(lst_Dec)
```
После этого организуются вложенные циклы получения элементов этих списков:
```
for i_god in range (n_let):
  tek_god=lst_God[i_god]
  for i_mes in range(n_mes):
    tek_mes=lst_Mes[i_mes]
    for i_dec in range (n_dec):
      tek_dec=lst_Dec[i_dec]
```
Здесь tek_god, tek_god, и tek_dec – текущие значения (т.е. при данном «обороте» циклов) строковых переменных обозначений года, месяца, декады или полудекады. В переменную «tek_interval» сохраняется текущее обозначение интервала времени:
```
tek_interval = tek_god+'_'+tek_mes+'_'+tek_dec
```
Эта переменная совпадает с именем текущего шейпфайла ледовой информации без расширения «shp». Помним о сдвиге операторов! Все операторы внутри цикла должны иметь одинаковый сдвиг от левого края. Определение адреса и названия текущего шейпфайла ледовой информации:
```
str_led_karta='D:\\Proba_intersection\\'+tek_interval+'.shp'
```
Определение существования этого шейпфайла и продолжение работы при этом условии:
```
if os.path.exists(str_led_karta):
```
Создание названия слоя результата пересечения слоя ледовой информации и буферной зоны (сдвиг операторов увеличивается, т.к. они выполняются только при существовании шейпфайла!):
```
tek_led_bufer = 'led_buf_'+tek_interval
```
Создаётся слой пересечения, система координат шейпфайла результата будет взята от слоя буферной зоны, пространственный тип результата – полигональные объекты:
```
layer_intersection = data.CreateLayer(tek_led_bufer,
lyr_bufer.GetSpatialRef(), ogr.wkbPolygon)
```
Загрузка текущего слоя ледовой информации:
```
layer_led = data.GetLayer(tek_interval)
```
Затем выполняется добавление атрибутов (семантических полей) слою

результата пересечения. Атрибуты берутся от слоя ледовой информации:

```
layer_intersection.CreateFields(layer_led.schema)
```

Получение описания слоя пересечения, которое содержит количество и тип полей и т.д.:

```
out_defn = layer_intersection.GetLayerDefn()
```

Далее в программе выполняется непосредственно пересечение пространственных объектов двух слоёв. Если в версии языка Python для ArcGIS это действие программируется с помощью одного оператора, то здесь необходимо организовать пересечение каждого объекта одного слоя с каждым объектом второго слоя с помощью двух вложенных операторов цикла. Поскольку слой буферной зоны маршрута состоит только из одного полигона, в данном случае нет необходимости организовывать второй вложенных цикл, и можно обойтись одним циклом извлечения пространственных объектов из слоя ледовой информации:

```
for i in range(layer_led.GetFeatureCount()):
```

Здесь «GetFeatureCount()» – функция, которая возвращает количество пространственных объектов слоя. Т.е. организован цикл по всем пространственным объектам слоя «layer_led».

Получение текущего пространственного объекта слоя ледовой информации (операторы ещё сдвинуты!):

```
obj_led = layer_led.GetFeature(i)
```

Получение геометрии этого объекта:

```
obj_led_geom = obj_led.geometry().Clone()
```

Получение её пересечения с геометрией буферной зоны как полигона:

```
intersect = obj_led_geom.Intersection(geom_buf)
```

Создание пустого объект со всеми нужными свойствами, т.е. полями таблицы атрибутов:

```
out_feat = ogr.Feature(out_defn)
```

Запись в пустой объект полигона – результата пересечения:

```
out_feat.SetGeometry(intersect)
```

Цикл по всем атрибутам текущего объекта слоя ледовой информации:

```
for j in range(obj_led.GetFieldCount()):
```

Здесь функция «GetFieldCount» возвращает число полей в таблице атрибутов.

```
value = obj_led.GetField(j)
```

Функция «GetField(j)» возвращает значение текущего слоя.

Запись в созданный объект пересечения значение атрибута слоя ледовой информации:

```
out_feat.SetField(j, value)
```

По окончании цикла просмотра всех атрибутов объекта слоя ледовой информации выполняется запись объекта пересечения в слой результатов пересечения:

```
layer_intersection.CreateFeature(out_feat)
```

После выхода из всех вложенных циклов загрузки шейпфайлов ледовой информации закрываем источник данных:

```
del data
```

Программа завершена.

Задание. Напишите программу пересечения шейпфайлов и запустите её на исполнение.

2.4. Пересчёт координат

В разделе 1.20 приведена программа пересчёта координат точек при переносе точки начала осей системы координат и повороте её осей. Рассмотрим решение этой проблемы средствами Python в Quantum GIS. Сначала подключаем модули и библиотеки:

```
from qgis.core import QgsProcessing
from qgis.core import QgsProcessingAlgorithm
from qgis.core import QgsProcessingMultiStepFeedback
from qgis.core import (QgsCoordinateReferenceSystem,
QgsCoordinateTransform,QgsProject,QgsVectorLayer)
import processing
import os
import ogr
import osr
import math
from PyQt5.QtCore import QVariant
```

Затем создаём списки лет, месяцев, дней:

```
lst_God=['2018', '2019', '2020', '2021']
lst_Mes=['04', '05', '06', '07', '08', '09', '10', '11']
lst_Den=['01','02','03','04','05','06','07','08','09','10','11','12',13','14','15','16',
'17','18','19','20','21','22','23','24','25','26','27','28','29',30','31']
```

Определение числа элементов в каждом списке:

```
n_let = len(lst_God)
n_mes = len(lst_Mes)
n_den = len(lst_Den)
```

Циклы по всем элементам списков лет, месяцев и дней:

```
for i_god in range (n_let):
  tek_god=lst_God[i_god]
  for i_mes in range(n_mes):
```

```
    tek_mes=lst_Mes[i_mes]
    for i_den in range (n_den):
      tek_den=lst_Den[i_den]
```

Создание строковой переменной с адресом и названием текущего шейпфайла точек в буферной зоне створа мониторинга:

```
        str_points='F:\\NDTI_2021\\pointsNDTI_'+tek_god+'_'+tek_mes
                    +'_'+tek_den+'.shp'
```

Разумеется, расположение файлов задано для конкретного случая. При их другом расположении необходимо изменить данный оператор. Т.к. исходные снимки есть не на каждый день, выполняется проверка существования шейпфайла:

```
        if os.path.exists(str_points):
```

Загрузка объекта типа «Векторный слой»:

```
            pointslayer = QgsVectorLayer(str_points, "tek_points", "ogr")
```

Создаём объект типа «Провайдер данных» из векторного слоя:

```
            provider = pointslayer.dataProvider()
```

Создаём объект типа «Пространственный объект»:

```
            feat = QgsFeature()
```

Получаем список всех атрибутивных полей провайдера данных:

```
            allAttrs = provider.attributeIndexes()
```

Получаем индексы полей провайдера по их именам:

```
            fld_x = provider.fieldNameIndex("x")
            fld_y = provider.fieldNameIndex("y")
            fld_X0 = provider.fieldNameIndex("X0")
            fld_Y0 = provider.fieldNameIndex("Y0")
```

Т.е. эти поля уже должны существовать в таблице атрибутов, причём в числовых полях «x» и «y» уже должны быть записаны значения координат в старой системе, а числовые поля «X0» и «Y0» должны быть созданы, но значения там пока «No Data».

Задаём значения координат точки начала створа в исходной системе координат:

```
            xn = 697000.58
            yn = 6638624.63
```

Разумеется, для другого створа и при другой системе координат будет нужно задать другие значения. Аналогично задаём координаты конечной точки створа:

```
            xk = 697259.32
            yk = 6638882.18
```

Расчёт параметров поворота осей координат:

```
            delta_x = xk-xn
            delta_y = yk-yn
```

```
delta_x2 = delta_x*delta_x
delta_y2 = delta_y*delta_y
summa_kvadratov = delta_x2+delta_y2
dlina_stvora = math.sqrt(summa_kvadratov)
sin_alfa = delta_y/dlina_stvora
cos_alfa = delta_x/dlina_stvora
```

Напоминаем, что в данном примере направление створа – с юго-запада на северо-восток, и начальная точка створа расположена на юго-западе.

Добавление в таблицу атрибутов новых полей «Xnov» и «Ynov» для записи координат точек в новой системе координат, начало которой совпадает с начальной точкой створа, а ось «X» совпадает с линией створа:

```
provider.addAttributes([QgsField("Xnov",QVariant.Double)])
provider.addAttributes([QgsField("Ynov",QVariant.Double)])
pointslayer.updateFields()
```

Получение индексов этих новых полей:

```
fld_Xnov = provider.fieldNameIndex("Xnov")
fld_Ynov = provider.fieldNameIndex("Ynov")
```

Возвращение объекта типа «Пространственные объекты»:

```
features = pointslayer.getFeatures()
```

Узнаём список доступных действий над векторным слоем с помощью функции capabilities():

```
caps = pointslayer.dataProvider().capabilities()
```

Цикл по всем пространственным объектам

```
for feature in features:
```

Получаем атрибуты конкретного объекта (точки):

```
  attrs = feature.attributes()
```

Получаем координаты точки в старой системе:

```
  tek_x = attrs[fld_x]
  tek_y = attrs[fld_y]
```

Параллельный перенос начала координат в точку начала створа:

```
  x_0 = tek_x - xn
  y_0 = tek_y – yn
```

Определение идентификатора данного объекта:

```
  id=feature.id()
```

Записываем значения в поля «X0» и «Y0»:

```
  attr_X0={fld_X0:x_0}
  if caps & provider.changeAttributeValues({id:attr_X0})
  attr_Y0={fld_Y0:y_0}
  if caps & provider.changeAttributeValues({id:attr_Y0})
```

Расчёт координат точки в новой системе координат:

```
x_nov = x_0*cos_alfa + y_0*sin_alfa
y_nov = y_0*cos_alfa - x_0*sin_alfa
attr_Xnov={fld_Xnov:x_nov}
attr_Ynov={fld_Ynov:y_nov}
if caps & provider.changeAttributeValues({id:attr_Xnov})
if caps & provider.changeAttributeValues({id:attr_Ynov})
```

Задание. Напишите программу для универсального переноса и поворота оси координат для совмещения одной из осей с линией створа мониторинга.

Заключение

Данное пособие представляет собой набор алгоритмов и программ, которые читатель может использовать для обработки больших объёмов пространственно-координированной информации, записанной в файловые структуры, имена которых изменяются по определённым правилам. Эти имена могут включать параметры даты производства космического снимка, даты мониторинга и т.п. Забираться в «дебри» программирования на языке Python имеет смысл только в том случае, когда есть действительно большое количество файлов, требующих обработки. Это могут быть выложенные в Интернете спутниковые снимки различных форматов, данные дистанционного зондирования Земли, а также результаты интерполяции результатов мониторинга различных параметров. Данные могут быть представлены как в растровом, так и в векторном форматах. Например, на сайте Арктического и антарктического НИИ представлены ледовые карты и другие материалы в формате шейпфайлов. Если требуется выполнить анализ двух-трёх электронных карт, то необходимость автоматизации обработки отсутствует, и здесь вполне можно обойтись «ручной обработкой» с помощью интерфейса ГИС. Для обработки одного-двух десятков слоёв файлов можно примененять модели рабочих потоков.

Автор считает своим долгом выразить глубокую признательность людям, благодаря которым в конечном счёте и было написано данное учебное пособие. Это профессор СПбГУ Василий Васильевич Дмитриев, заведующий лабораторией моделирования и диагностики геосистем Института наук о Земле СПбГУ, в которой в начале 90-х годов автор начал разработку баз данных и ГИС, реализованных на языке Clipper; одногруппник автора по учёбе в

Ленинградском государственном университете им. А.А. Жданова в 1976-81 гг. Александр Викторович Юлин, заведующий лабораторией ледового режима и долгосрочных прогнозов Северного ледовитого океана ФГБУ Арктический и антарктический научно-исследовательский институт, пригласивший автора в 1997 г. на работу в ААНИИ в качестве внешнего совместителя. Именно здесь автор вплотную занялся разработкой ГИС-приложений для обработки и моделирования пространственно-координированной информации в среде ArcGIS. Благодаря профессору РГГМУ Владиславу Аркадьевичу Шелутко автор начал читать в 1998 г. курс ГИС на экологическом факультете РГГМУ. Благодаря профессору СПбГУ Владиславу Николаевичу Мовчану автор с 2001 г. читает ряд курсов, связанных с геоинформационными системами и компьютерными технологиями, на факультете географии и геоэкологии СПбГУ, а с 2014 г., после объединения последнего с другими структурами, в Институте наук о Земле СПбГУ.

Рекомендованная литература

1. Гуриков С.Р. Основы алгоритмизации и программирования на Python: учеб. пособие / М.: ФОРУМ: ИНФРА-М, 2018. 343 с. http://znanium.com/catalog/product/924699

2. Седер Наоми. Python. Экспресс-курс. 3-е изд. — СПб: Питер, 2019. — 480 с.: ил. — (Серия «Библиотека программиста»). ISBN 978-5-4461-0908-1. https://www.rulit.me/data/programs/resources/pdf/Seder_Python-Ekspress-kurs_RuLit_Me_605938.pdf

3. Федоров Д. Ю. Программирование на языке высокого уровня Python. Учебное пособие для СПО // М.: Изд-во Юрайт, 2019. 126 с. https://proxy.library.spbu.ru:2183/viewer/programmirovanie-na-yazyke-vysokogo-urovnya-python-431294#page/2

4. Chris Garrard. Geoprocessing with Python. MANNING SHELTER ISLAND. ISBN: 9781617292149. https://manning-content.s3.amazonaws.com/download/4/6383ad4-7afc-4a5c-ba0e-59afc8457ebb/Garrard-AppsOnline.pdf

5. Joel Lawhead. QGIS Python Programming Cookbook. BIRMINGHAM – MUMBAI. 2015. ISBN 978-1-78398-498-5. https://www.programmer-books.com/wp-content/uploads/2019/05/QGIS-Python-Programming-Cookbook.pdf

Учебное издание

Третьяков Виктор Юрьевич,
доцент, кандидат географических наук

ГЕОИНФОРМАЦИОННЫЕ СИСТЕМЫ В ЭКОЛОГИИ И ПРИРОДОПОЛЬЗОВАНИИ: ПРОГРАММИРОВАНИЕ НА PYTHON В ARCGIS И QUANTUM GIS

Печатается в авторской редакции

Подписано в печать 12.04.2022. Формат 60×90 1/16.

Гарнитура Times New Roman. Печать цифровая.

Усл. печ. л. 7. Тираж 10 экз. Заказ № 1215.

РГГМУ, 192007, Санкт-Петербург, Воронежская ул., д. 79.