Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии

Департамент цифровых, робототехнических систем и электроники

**ОТЧЕТ**

**ПО ЛАБОРАТОРНОЙ РАБОТЕ №4.4**

**дисциплины «Объектно-ориентированное программирование»**

**Вариант 9**

|  |  |  |  |
| --- | --- | --- | --- |
|  | | Выполнил:  Иващенко Олег Андреевич  3 курс, группа ИВТ-б-о-22-1,  09.03.02 «Информационные и вычислительные машины», направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |
|  | | Руководитель практики:  Воронкин Роман Александрович, доцент департамента цифровых, робототехнических систем и электроники  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) | |
|  | |  | |

Отчет защищен с оценкой \_\_\_\_\_\_\_\_\_\_\_ Дата защиты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ставрополь, 2024 г.

**Тема**: «Работа с исключениями в языке Python»

**Цель**: Приобретение навыков по работе с исключениями при написании программ с помощью языка программирования Python версии 3.x.

Порядок выполнения работы:

Пример 1. Для примера 2 лабораторной работы 9 добавить возможность работы с исключениями и логгирование.

Листинг 1 - Код программы примера

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*  from dataclasses import dataclass, field  from datetime import date  import logging  import sys  from typing import List  import xml.etree.ElementTree as ET  # Класс пользовательского исключения в случае, если неверно  # введен номер года.  class IllegalYearError(Exception):  def \_\_init\_\_(self, year, message="Illegal year number"):  self.year = year  self.message = message  super(IllegalYearError, self).\_\_init\_\_(message)  def \_\_str\_\_(self):  return f"{self.year} -> {self.message}"  # Класс пользовательского исключения в случае, если введенная  # команда является недопустимой.  class UnknownCommandError(Exception):  def \_\_init\_\_(self, command, message="Unknown command"):  self.command = command  self.message = message  super(UnknownCommandError, self).\_\_init\_\_(message)  def \_\_str\_\_(self):  return f"{self.command} -> {self.message}"  @dataclass(frozen=True)  class Worker:  name: str  post: str  year: int  @dataclass  class Staff:  workers: List[Worker] = field(default\_factory=lambda: [])  def add(self, name, post, year):  # Получить текущую дату.  today = date.today()  if year < 0 or year > today.year:  raise IllegalYearError(year)  self.workers.append(  Worker(  name=name,  post=post,  year=year  )  )  self.workers.sort(key=lambda worker: worker.name)  def \_\_str\_\_(self):  # Заголовок таблицы.  table = []  line = '+-{}-+-{}-+-{}-+-{}-+'.format(  '-' \* 4,  '-' \* 30,  '-' \* 32,  '-' \* 8  )  table.append(line)  table.append(  '| {:^4} | {:^30} | {:^32} | {:^8} |'.format(  "№",  "Ф.И.О.",  "Должность",  "Год"  )  )  table.append(line)  # Вывести данные о всех сотрудниках.  for idx, worker in enumerate(self.workers, 1):  table.append(  '| {:>4} | {:<30} | {:<32} | {:>8} |'.format(  idx,  worker.name,  worker.post,  worker.year  )  )  table.append(line)  return '\n'.join(table)  def select(self, period):  # Получить текущую дату.  today = date.today()  result = []  for worker in self.workers:  if today.year - int(worker.year) >= int(period):  result.append(worker)  return result  def load(self, filename):  with open(filename, 'r', encoding='utf8') as fin:  xml = fin.read()  parser = ET.XMLParser(encoding="utf8")  tree = ET.fromstring(xml, parser=parser)  self.workers = []  for worker\_element in tree:  name, post, year = None, None, None  for element in worker\_element:  if element.tag == 'name':  name = element.text  elif element.tag == 'post':  post = element.text  elif element.tag == 'year':  year = int(element.text)  if name is not None and post is not None \  and year is not None:  self.workers.append(  Worker(  name=name,  post=post,  year=year  )  )  def save(self, filename):  root = ET.Element('workers')  for worker in self.workers:  worker\_element = ET.Element('worker')  name\_element = ET.SubElement(worker\_element, 'name')  name\_element.text = worker.name  post\_element = ET.SubElement(worker\_element, 'post')  post\_element.text = worker.post  year\_element = ET.SubElement(worker\_element, 'year')  year\_element.text = str(worker.year)  root.append(worker\_element)  tree = ET.ElementTree(root)  with open(filename, 'wb') as fout:  tree.write(fout, encoding='utf8', xml\_declaration=True)  if \_\_name\_\_ == '\_\_main\_\_':  # Выполнить настройку логгера.  logging.basicConfig(  filename='workers.log',  level=logging.INFO  )  # Список работников.  staff = Staff()  # Организовать бесконечный цикл запроса команд.  while True:  try:  # Запросить команду из терминала.  command = input(">>> ").lower()  # Выполнить действие в соответствие с командой.  if command == 'exit':  break  elif command == 'add':  # Запросить данные о работнике.  name = input("Фамилия и инициалы? ")  post = input("Должность? ")  year = int(input("Год поступления? "))  # Добавить работника.  staff.add(name, post, year)  logging.info(  f"Добавлен сотрудник: {name}, {post}, "  f"поступивший в {year} году."  )  elif command == 'list':  # Вывести список.  print(staff)  logging.info("Отображен список сотрудников.")  elif command.startswith('select '):  # Разбить команду на части для выделения номера года.  parts = command.split(maxsplit=1)  # Запросить работников.  selected = staff.select(parts[1])  # Вывести результаты запроса.  if selected:  for idx, worker in enumerate(selected, 1):  print(  '{:>4}: {}'.format(idx, worker.name)  )  logging.info(  f"Найдено {len(selected)} работников со "  f"стажем более {parts[1]} лет."  )  else:  print("Работники с заданным стажем не найдены.")  logging.warning(  f"Работники со стажем более {parts[1]} лет не найдены."  )  elif command.startswith('load '):  # Разбить команду на части для имени файла.  parts = command.split(maxsplit=1)  # Загрузить данные из файла.  staff.load(parts[1])  logging.info(f"Загружены данные из файла {parts[1]}.")  elif command.startswith('save '):  # Разбить команду на части для имени файла.  parts = command.split(maxsplit=1)  # Сохранить данные в файл.  staff.save(parts[1])  logging.info(f"Сохранены данные в файл {parts[1]}.")  elif command == 'help':  # Вывести справку о работе с программой.  print("Список команд:\n")  print("add - добавить работника;")  print("list - вывести список работников;")  print("select <стаж> - запросить работников со стажем;")  print("load <имя\_файла> - загрузить данные из файла;")  print("save <имя\_файла> - сохранить данные в файл;")  print("help - отобразить справку;")  print("exit - завершить работу с программой.")  else:  raise UnknownCommandError(command)  except Exception as exc:  logging.error(f"Ошибка: {exc}")  print(exc, file=sys.stderr) |

Добавим несколько записей в нашу условную таблицу при помоги команды add, запускающего метод добавления записи (фамилия и инициалы, должность и год поступления работника). Процесс представлен на рисунке 1.1.

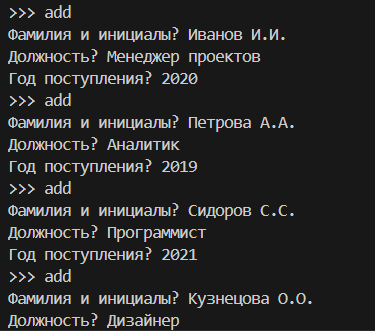


Рисунок 1.1 – Добавление новых записей

Выведем нашу таблицу для проверки корректности добавленных данных при помощи команды list (рисунок 1.2).

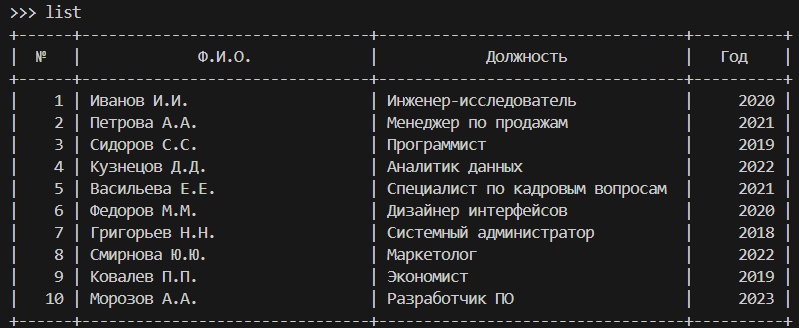


Рисунок 1.2 – Вывод существующих записей

Введём ещё 5 новый записей и выведем текущие данные в терминал (рисунок 1.3).

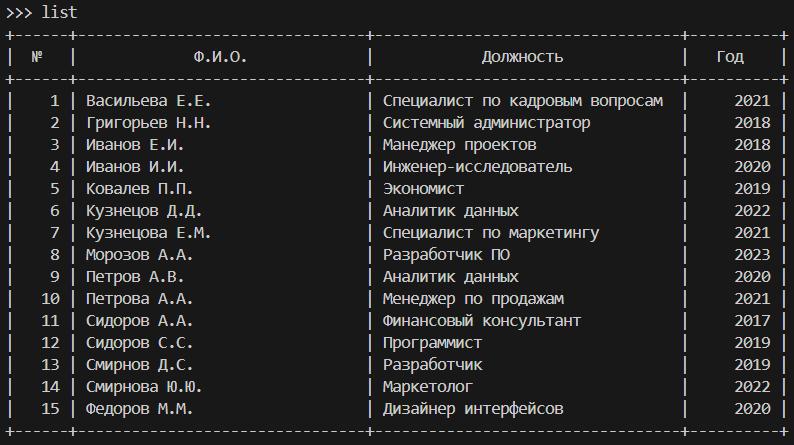


Рисунок 1.3 – Вывод общего списка сотрудников

Сохраним имеющиеся данные в папке data в файле example\_data.json (рисунок 1.4).

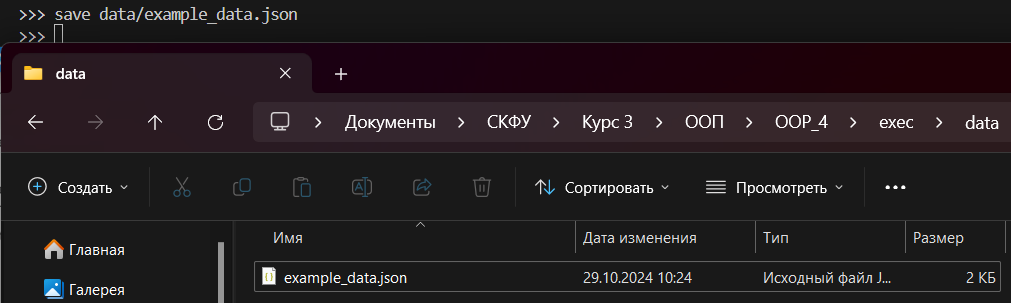


Рисунок 1.4 – Сохранение данных

Теперь можно завершить выполнение программы и посмотреть логи, созданные программой. В них записаны все действия оператора в течение всего выполнения программы (рисунок 1.5).

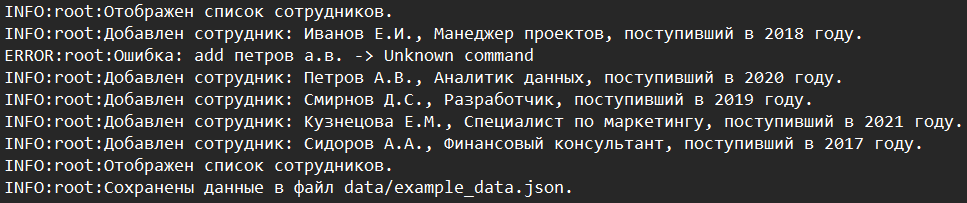


Рисунок 1.5 – Логи программы

После завершения работы запустим программу вновь. Так как «память» программы обнулилась после её предыдущего завершения, нужно загрузить данные из файла, который был сохранён в предыдущий сеанс. Сделаем это при помощи команды load, указав путь до файла. Вновь выведем данные (рисунок 1.6).

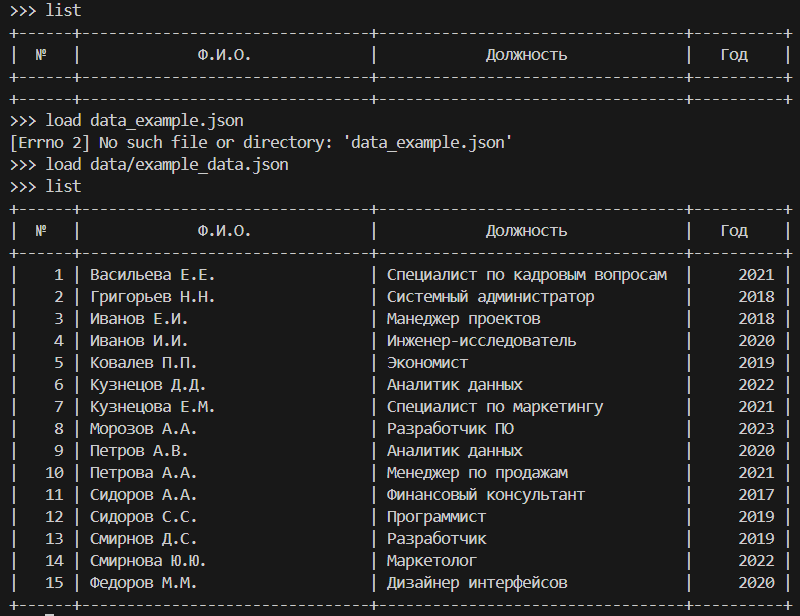


Рисунок 1.6 – Загрузка данных из файла и вывод

При помощи команды select выведем всех сотрудников, стаж которых будет удовлетворять указанному значению (рисунок 1.7).

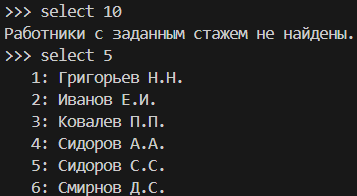


Рисунок 1.7 – Вывод сотрудников по стажу

После завершения работы вновь обратимся к логам, чтобы удостовериться, что каждое действие оператора записывается в текстовый файл (рисунок 1.8).

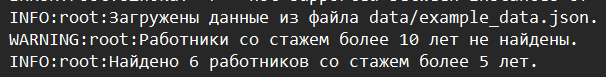


Рисунок 1.8 – Новые записи в логах

Задание 1. Написать программу, которая запрашивает ввод двух значений. Если хотя бы одно из них не является числом, то должна выполняться конкатенация, т.е. соединение строк. В остальных случаях введённые числа суммируются.

Листинг 2 – Код программы задания 1

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*  '''  Написать программу, которая запрашивает ввод двух значений.  Если хотя бы одно из них не является числом, то должна выполняться  конкатенация, т.е. соединение, строк. В остальных случаях введённые  числа суммируются.  '''  if \_\_name\_\_ == '\_\_main\_\_':  # Вводим два значения a и b  a = input('Введите первое значение: ')  b = input('Введите второе значение: ')  try:  # Случай, если оба значения оказались числами  c = int(a) + int(b)  print('Оба значения оказались числами. Выполнение блока try...')  print(f'Результат суммирования: {a} + {b} = {c}')  except Exception:  # Случай, если как минимум одно из значений не число  c = str(a) + str(b)  print('Одно из значений оказалось не числом.',  'Выполнение блока except...')  print(f'Результат конкатенации: {c}') |

В программе, код которой представлен в листинге 2, у нас по ходу выполнения может быть 2 случая: либо пользователь ввёл два числа, и они в результате выполнения суммируются, выполнив блок try (т.е. нормальное выполнение программы, без возникновения каких-либо исключения), либо пользователь ввёл как минимум в одном из значений не число, и программа соединит два полученных значения как строку (т.е. в процессе выполнения блока try возникла ошибка/исключение, и программа выполняет блока except). Оба варианта завершения программы представлены на рисунках 2.1 и 2.2.

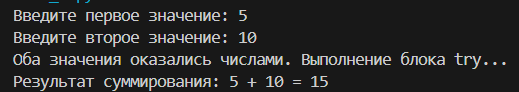


Рисунок 2.1 – Нормальное выполнение программы при вводе чисел

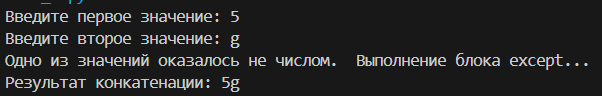


Рисунок 2.2 – Выполнение участка кода, выполняющегося в случае возникновения ошибки/исключения

Задание 2. Написать программу, которая будет генерировать матрицу из случайных целых чисел. Пользователь может указать число строк и столбцов, а также диапазон чисел. Произвести обработку ошибок ввода пользователя.

Листинг 3 – Код программы задания 2

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*  '''  Написать программу, которая будет генерировать матрицу из случайных целых  чисел. Пользователь может указать число строк и столбцов, а также диапазон  чисел. Произвести обработку ошибок ввода пользователя.  '''  import random  class MinValueBiggest(Exception):  '''  Пустое исключение, вызываемое в случае, если минимальное значение  чисел для случайной генерации оказалось больше максимального  '''  pass  if \_\_name\_\_ == '\_\_main\_\_':  '''  Основное тело программы  '''  try:  matrix\_width = int(input("Введите ширину матрицы: "))  matrix\_height = int(input("Введите высоту матрицы: "))  min\_value = int(input("Введите минимальное значение диапазона: "))  max\_value = int(input("Введите максимальное значение диапазона: "))  if min\_value > max\_value:  raise MinValueBiggest # Вызов исключения  matrix = [[0] \* matrix\_width for \_ in range(matrix\_height)]  for i in range(matrix\_height): # Заполнение матрицы  for j in range(matrix\_width):  matrix[i][j] = random.randint(min\_value, max\_value)  for i in range(matrix\_height): # Вывод матрицы  temp\_str = str()  j = 0  for j in range(matrix\_width):  if j < matrix\_width - 1: # Проверка на последнее значение  temp\_str += str(matrix[i][j]) + " "  else:  temp\_str += str(matrix[i][j])  j += 1  print(f"|{temp\_str}|")  print('Завершение работы')  except ValueError: # Ошибка при неправильном указании чисел  print('Ошибка: Ошибка при вводе чисел')  except MinValueBiggest: # Ошибка при указании min больше max  print("Ошибка: Минимальное значение больше максимального") |

Проведём несколько тестов программы, работающей в нормальном режиме, без намеренного вызова ошибок (рисунок 3.1 и 3.2).

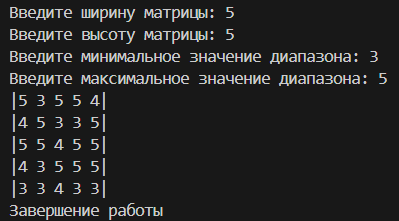


Рисунок 3.1 – Результат работы при правильной работе программы

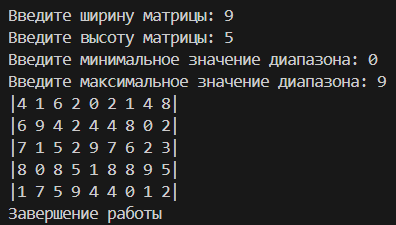


Рисунок 3.2 – Результат работы при правильной работе программы

Укажем одно из числовых значений не числом. В результате программа завершит работу, вызвав исключение ValueError, указав это в терминале (рисунок 3.3).



Рисунок 3.3 – Результат работы исключения

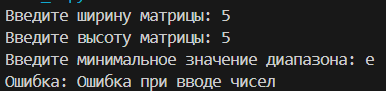


Рисунок 3.4 – Результат работы исключения

Ещё одно место, где практически возможно совершить ошибку – указание минимального значения диапазона чисел больше, чем максимальное значение. Это исключение также подходит ValueError, но для пользователя это будет не явный признак конкретно этой совершённой ошибки. Для этого в программа добавляется собственное исключение MinValueBiggest, которое далее по коду вызывается после проверки того, какое из значений больше. Если минимальное значение оказывается больше максимального, с помощью оператора raise вызывается исключение. Результат срабатывания исключения представлен на рисунке 3.5.

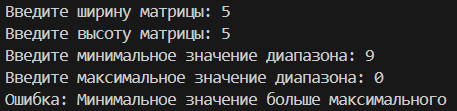


Рисунок 3.5 – Вызов пользовательского исключения

Индивидуальное задание 1. Для индивидуального задания 1 лабораторной работы 2.19 добавить возможность работы с исключениями и логгирование.

Одно из самых банальным мест, где пользователь может допустить ошибку – добавление пользователя. Так как данная процедура требует ввода множества данных, шанс ошибки тут наибольший. Для имени и фамилии пользователя ограничения не нужно делать, но, например, для номера телефона стоит сделать проверку на правильность введённого значения. Проверка будет наиболее банальной – если указанный номер – число, то программа пропустит его. Но если в нём указаны какие-то дополнительные символы, то система вызовет исключение и запись не будет добавлена (рисунок 4.1).

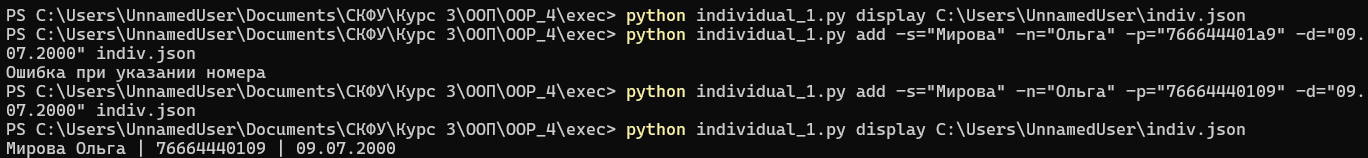


Рисунок 4.1 – Попытка добавления работника с неправильным номером

Аналогично нужно реализовать исключение для даты. В ней не должно быть ничего, кроме как чисел указания дня, месяца и года, и разделительных символов – в нашем случае это точки. Результат представлен на рисунке 4.2.

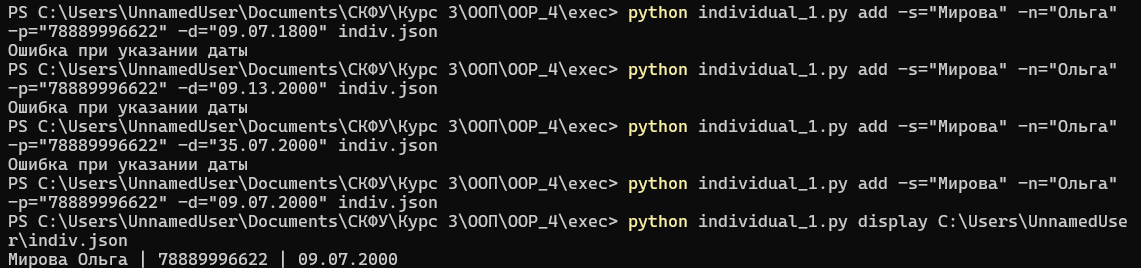


Рисунок 4.2 – Вызов ошибки при указании неправильной даты (или формата даты)

Для сохранения действий, выполненных пользователем при работе программы, необходимо добавить запись этих действий в текстовый файл – лог. Он позволит проверить выполненные действия, а также более ясно определить, что вызвало ошибку при выполнении программы. Пример лога представлен на рисунке 4.3.

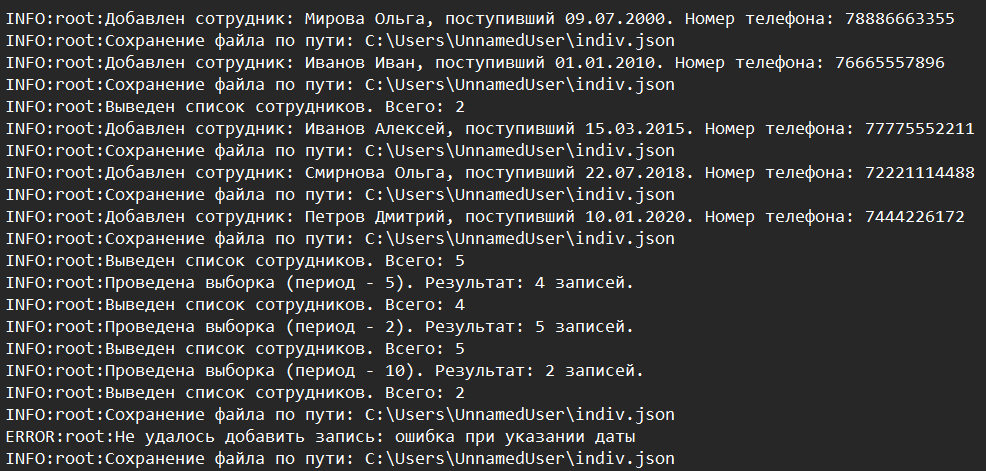


Рисунок 4.3 – Лог программы

Листинг 4 – Код программы индивидуального задания 1

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*-  '''  Выполнить индивидуальное задание 1 лабораторной работы 2.19,  добавив возможность работы с исключениями и логгирование.  '''  import json  from datetime import datetime  import argparse  import os.path  import pathlib  import logging  class DateError(Exception):  pass  def print\_help():  """  Функция вывода доступных пользователю команд.  Аргументы:  None  Возвращает:  None  """  print("list - вывод всех добавленных записей")  print("add - добавление новых записей")  print("find - найти запись по фамилии")  print("exit - завершение работы программы")  def add\_worker(workers, surname, name, phone, date):  """  Функция добавления новой записи в список.  Аргументы:  workers (list) - Список, в который будет добавляться запись  surname (str) - Фамилия добавляемого сотрудника  name (str) - Имя добавляемого сотрудника  phone (int) - Номер телефона добавляемого сотрудника  date (datetime) - Дата нанятия добавляемого сотрудника  Возвращает:  workers (list) - Обновлённый список сотрудников  """  try:  phone = int(phone)  if not isinstance(phone, int): # Проверка номера на правильность  raise ValueError  temp\_data = str(date).split('.')  for i in range(len(temp\_data)):  temp\_data[i] = int(temp\_data[i])  if not isinstance(temp\_data[i], int):  raise DateError  if (temp\_data[0] < 0 or temp\_data[0] > 31):  raise DateError  if (temp\_data[1] < 0 or temp\_data[1] > 12):  raise DateError  if (temp\_data[2] < 1900 or temp\_data[2] > 2024):  raise DateError  workers.append(  {  "surname": surname,  'name': name,  'phone': phone,  'date': date  }  )  logging.info(  f"Добавлен сотрудник: {surname} {name}, "  f"поступивший {date}. Номер телефона: {phone}"  )  except ValueError:  print("Ошибка при указании номера")  logging.error("Не удалось добавить запись:"  "ошибка при указании номера телефона")  except DateError:  print("Ошибка при указании даты")  logging.error("Не удалось добавить запись: ошибка при указании даты")  return workers  def print\_list(list):  """  Функция выводит на экран список всех существующих записей.  Аргументы:  list (list) - Список всех сотрудников (записей) для вывода  Возвращает:  None  """  count = 0  for member in list:  print(f"{member['surname']} {member['name']} | "  f"{member['phone']} | {member['date']}")  count += 1  logging.info(f"Выведен список сотрудников. Всего: {count}")  def find\_member(workers, period):  """  Функция для вывода на экран всех записей, чей стаж работы больше  или равен указанному.  Аргументы:  workers (list) - Список сотрудников для поиска  period (int) - Количество лет стажа  Возвращает:  members (list) - Сипсок сотрудников, чей стаж больше  или равен указанному  """  count = 0  members = []  for member in workers:  year = datetime.strptime(member['date'], "%d.%m.%Y").year  if datetime.now().year - period >= year:  members.append(member)  count += 1  logging.info(f"Проведена выборка (период - {period}). "  f"Результат: {count} записей.")  if count == 0:  print("Записи не найдены")  else:  return members  def get\_home\_path(filename):  """  Получение полного пути к файлу, расположенному в домашнем каталоге  пользователя (пример: C:/Users/<имя\_пользователя>/<имя\_файла>).  Аргументы:  filename (str) - Имя файла    Возвращает:  Полный путь к файлу (str)  """  home\_dir = pathlib.Path.home()  return home\_dir / filename  def save\_file(filename, workers):  """  Сохранение списка сотрудников в файл.  Аргументы:  filename (str) - Имя сохраняемого файла  workers (list) - Список сохраняемых сотрудников (записей)  Возвращает:  None  """  try:  with open(get\_home\_path(filename), "w", encoding="utf-8") as file:  logging.info(f"Сохранение файла по пути: {get\_home\_path(filename)}")  json.dump(workers, file, ensure\_ascii=False, indent=4)  except Exception as e:  logging.error(f"Ошибка при сохранении файла: {e}")  def load\_file(filename):  """  Загрузка данных о сотрудниках из указанного JSON-файла.  Аргументы:  filename (str) - Имя загружаемого файла  Возвращает:  file (list) - список записей из файла  Пустой список - если произошла ошибка с при чтении файла  """  try:  with open(get\_home\_path(filename), "r", encoding="utf-8") as file:  return json.load(file)  except json.JSONDecodeError as e:  logging.error(f"Ошибка загрузки JSON: {e}")  print("Файл поврежден или имеет некорректный формат.")  return []  except FileNotFoundError:  logging.warning("Файл не найден. Создан пустой список.")  return []  def parse\_datetime(value):  """  Преобразование указанного значения в формат даты.  Аргументы:  value (str) - указанная дата в текстовом формате  Возвращает:  Дата (datetime) - Преобразованное значение value в формат datetime  Текст 'Error' - В случае ошибки преобразования  """  try:  return datetime.strptime(value, "%d.%m.%Y")  except ValueError:  print("Error")  def main(command\_line=None):  file\_parser = argparse.ArgumentParser(add\_help=False)  file\_parser.add\_argument(  "filename",  action="store",  help="The data file name"  )  parser = argparse.ArgumentParser("workers")  parser.add\_argument(  "--version",  action="version",  version="%(prog)s 0.1.0"  )  subparsers = parser.add\_subparsers(dest="command")  add = subparsers.add\_parser(  "add",  parents=[file\_parser],  help="Add a new worker"  )  add.add\_argument(  "-s",  "--surname",  action="store",  required=True,  help="The worker's surname"  )  add.add\_argument(  "-n",  "--name",  action="store",  required=True,  help="The worker's name"  )  add.add\_argument(  "-p",  "--phone",  action="store",  help="The worker's phone"  )  add.add\_argument(  "-d",  "--date",  action="store",  required=True,  help="The date of hiring"  )  \_ = subparsers.add\_parser(  "display",  parents=[file\_parser],  help="Display all workers"  )  select = subparsers.add\_parser(  "select",  parents=[file\_parser],  help="Select the workers"  )  select.add\_argument(  "-p",  "--period",  action="store",  type=int,  required=True,  help="The required period"  )  args = parser.parse\_args(command\_line)  is\_dirty = False  try:  workers = load\_file(args.filename)  except FileExistsError:  workers = []  if args.command == "add":  workers = add\_worker(  workers,  args.surname,  args.name,  args.phone,  args.date  )  is\_dirty = True  elif args.command == "display":  print\_list(workers)  elif args.command == "select":  selected = find\_member(workers, args.period)  print\_list(selected)  if is\_dirty:  save\_file(args.filename, workers)  if \_\_name\_\_ == "\_\_main\_\_":  """  Основная программа  """  logging.basicConfig(  filename='individual\_1.log',  level=logging.INFO  )  main() |

Индивидуальное задание 2. Изучить возможности модуля logging. Добавить для предыдущего задания вывод в файлы лога даты и времени выполнения пользовательской команды с точностью до миллисекунды.

Добавление даты и времени в модуле logging происходит при настройке базовой конфигурации этого модуля. В нашем случае в блоке основной программы мы уже имеем строки конфигурации модуля, в которых указываются имя файла, куда записываются логи, а также минимальный уровень логирования, который будет записываться в файл (в нашем случае минимальный уровень логирования – INFO, что значит, что все уровни, кроме DEBUG (так как по уровню он ниже), будут записываться в файл). Для того, чтобы добавить дату и время в логи, нам необходимо добавить ещё одну конфигурацию модуля – format, в котором мы изменим вывод информации, добавив к сообщению дату и время. Происходит это при помощи строки:

format=('[%(asctime)s] %(message)s')

Этой строкой мы изменяем формат выводимых сообщений. %(asctime)s значит, что в сообщении будет выводиться дата и время, при этом время будет выводиться с точностью до миллисекунды. %(message)s – есть само выводимое сообщение. Как итог, при выводе списка сотрудников или при отборе по стажу мы получаем записи в логе, продемонстрированные на рисунке 5.1.

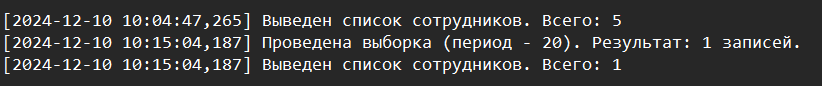


Рисунок 5.1 – Записи в файле лога

Эти записи были выведены в соответствии с введёнными командами пользователем, указанные на рисунке 5.2.

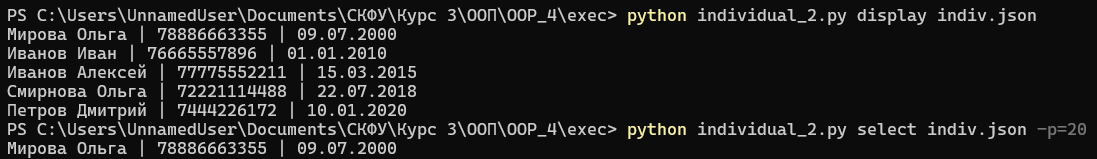


Рисунок 5.2 – Введённые команды

Листинг 5 – Код программы индивидуального задания 2

|  |
| --- |
| #!/usr/bin/env python3  # -\*- coding: utf-8 -\*-  '''  К предыдущему заданию (файл individual\_1.py) добавить вывод в лог даты  и времени записей с точностью до миллисекунду.  '''  import json  from datetime import datetime  import argparse  import os.path  import pathlib  import logging  class DateError(Exception):  pass  def print\_help():  """  Функция вывода доступных пользователю команд.  Аргументы:  None  Возвращает:  None  """  print("list - вывод всех добавленных записей")  print("add - добавление новых записей")  print("find - найти запись по фамилии")  print("exit - завершение работы программы")  def add\_worker(workers, surname, name, phone, date):  """  Функция добавления новой записи в список.  Аргументы:  workers (list) - Список, в который будет добавляться запись  surname (str) - Фамилия добавляемого сотрудника  name (str) - Имя добавляемого сотрудника  phone (int) - Номер телефона добавляемого сотрудника  date (datetime) - Дата нанятия добавляемого сотрудника  Возвращает:  workers (list) - Обновлённый список сотрудников  """  try:  phone = int(phone)  if not isinstance(phone, int): # Проверка номера на правильность  raise ValueError  temp\_data = str(date).split('.')  for i in range(len(temp\_data)):  temp\_data[i] = int(temp\_data[i])  if not isinstance(temp\_data[i], int):  raise DateError  if (temp\_data[0] < 0 or temp\_data[0] > 31):  raise DateError  if (temp\_data[1] < 0 or temp\_data[1] > 12):  raise DateError  if (temp\_data[2] < 1900 or temp\_data[2] > 2024):  raise DateError  workers.append(  {  "surname": surname,  'name': name,  'phone': phone,  'date': date  }  )  logging.info(  f"Добавлен сотрудник: {surname} {name}, "  f"поступивший {date}. Номер телефона: {phone}"  )  except ValueError:  print("Ошибка при указании номера")  logging.error("Не удалось добавить запись:"  "ошибка при указании номера телефона")  except DateError:  print("Ошибка при указании даты")  logging.error("Не удалось добавить запись: ошибка при указании даты")  return workers  def print\_list(list):  """  Функция выводит на экран список всех существующих записей.  Аргументы:  list (list) - Список всех сотрудников (записей) для вывода  Возвращает:  None  """  count = 0  for member in list:  print(f"{member['surname']} {member['name']} | "  f"{member['phone']} | {member['date']}")  count += 1  logging.info(f"Выведен список сотрудников. Всего: {count}")  def find\_member(workers, period):  """  Функция для вывода на экран всех записей, чей стаж работы больше  или равен указанному.  Аргументы:  workers (list) - Список сотрудников для поиска  period (int) - Количество лет стажа  Возвращает:  members (list) - Сипсок сотрудников, чей стаж больше  или равен указанному  """  count = 0  members = []  for member in workers:  year = datetime.strptime(member['date'], "%d.%m.%Y").year  if datetime.now().year - period >= year:  members.append(member)  count += 1  logging.info(f"Проведена выборка (период - {period}). "  f"Результат: {count} записей.")  if count == 0:  print("Записи не найдены")  else:  return members  def get\_home\_path(filename):  """  Получение полного пути к файлу, расположенному в домашнем каталоге  пользователя (пример: C:/Users/<имя\_пользователя>/<имя\_файла>).  Аргументы:  filename (str) - Имя файла    Возвращает:  Полный путь к файлу (str)  """  home\_dir = pathlib.Path.home()  return home\_dir / filename  def save\_file(filename, workers):  """  Сохранение списка сотрудников в файл.  Аргументы:  filename (str) - Имя сохраняемого файла  workers (list) - Список сохраняемых сотрудников (записей)  Возвращает:  None  """  try:  with open(get\_home\_path(filename), "w", encoding="utf-8") as file:  logging.info(f"Сохранение файла по пути: {get\_home\_path(filename)}")  json.dump(workers, file, ensure\_ascii=False, indent=4)  except Exception as e:  logging.error(f"Ошибка при сохранении файла: {e}")  def load\_file(filename):  """  Загрузка данных о сотрудниках из указанного JSON-файла.  Аргументы:  filename (str) - Имя загружаемого файла  Возвращает:  file (list) - список записей из файла  Пустой список - если произошла ошибка с при чтении файла  """  try:  with open(get\_home\_path(filename), "r", encoding="utf-8") as file:  return json.load(file)  except json.JSONDecodeError as e:  logging.error(f"Ошибка загрузки JSON: {e}")  print("Файл поврежден или имеет некорректный формат.")  return []  except FileNotFoundError:  logging.warning("Файл не найден. Создан пустой список.")  return []  def parse\_datetime(value):  """  Преобразование указанного значения в формат даты.  Аргументы:  value (str) - указанная дата в текстовом формате  Возвращает:  Дата (datetime) - Преобразованное значение value в формат datetime  Текст 'Error' - В случае ошибки преобразования  """  try:  return datetime.strptime(value, "%d.%m.%Y")  except ValueError:  print("Error")  def main(command\_line=None):  file\_parser = argparse.ArgumentParser(add\_help=False)  file\_parser.add\_argument(  "filename",  action="store",  help="The data file name"  )  parser = argparse.ArgumentParser("workers")  parser.add\_argument(  "--version",  action="version",  version="%(prog)s 0.1.0"  )  subparsers = parser.add\_subparsers(dest="command")  add = subparsers.add\_parser(  "add",  parents=[file\_parser],  help="Add a new worker"  )  add.add\_argument(  "-s",  "--surname",  action="store",  required=True,  help="The worker's surname"  )  add.add\_argument(  "-n",  "--name",  action="store",  required=True,  help="The worker's name"  )  add.add\_argument(  "-p",  "--phone",  action="store",  help="The worker's phone"  )  add.add\_argument(  "-d",  "--date",  action="store",  required=True,  help="The date of hiring"  )  \_ = subparsers.add\_parser(  "display",  parents=[file\_parser],  help="Display all workers"  )  select = subparsers.add\_parser(  "select",  parents=[file\_parser],  help="Select the workers"  )  select.add\_argument(  "-p",  "--period",  action="store",  type=int,  required=True,  help="The required period"  )  args = parser.parse\_args(command\_line)  is\_dirty = False  try:  workers = load\_file(args.filename)  except FileExistsError:  workers = []  if args.command == "add":  workers = add\_worker(  workers,  args.surname,  args.name,  args.phone,  args.date  )  is\_dirty = True  elif args.command == "display":  print\_list(workers)  elif args.command == "select":  selected = find\_member(workers, args.period)  print\_list(selected)  if is\_dirty:  save\_file(args.filename, workers)  if \_\_name\_\_ == "\_\_main\_\_":  """  Основная программа  """  logging.basicConfig(  filename='individual\_2.log',  level=logging.INFO,  format=('[%(asctime)s] %(message)s')  )  main() |

Ответы на контрольные вопросы:

1. Какие существуют виды ошибок в языке программирования Python?

В Python выделяют два различных вида ошибок: синтаксические ошибки и исключения.

Синтаксические ошибки возникают в случае, если программа написана с нарушением требований Python к синтаксису. Определяются они в процессе парсинга программы.

Исключения возникают в случае, если синтаксически программа корректна, но в процессе выполнения возникает ошибка (деление на ноль и т.д.).

1. Как осуществляется обработка исключений в языке программирования Python?

Обработка исключений нужна для того, чтобы приложение не завершилось аварийно каждый раз, когда возникает исключение. Для этого блок кода, в котором возможно появление исключительной ситуации необходимо поместить внутрь синтаксической конструкции try … except. Пример:

*print(“start”)*

*try:*

*val = int(input(“input number: “))*

*tmp = 10 / val*

*print(tmp)*

*except Exception as e:*

*print(“Error! “ + str(e))*

*print(“stop”)*

1. Для чего нужны блоки finally и else при обработке исключений?

Для выполнения определённого программного кода при выходе из блока try/except используется оператор finally.

Независимо от того, возникнет исключение или нет, код в блоке finally всё равно будет выполнен.

Если необходимо выполнить какой-то программный код, в случае если в процессе выполнения блока try не возникло исключений, то можно использовать оператор else.

1. Как осуществляется генерация исключений в языке Python?

Для принудительной генерации исключения используется инструкция raise. Пример:

*try:*

*raise Exeption(“Some exception”)*

*except Exception as e:*

*print(“Exception exception “ + str(e))*

Таким образом можно «вручную» вызывать исключения при необходимости.

1. Как создаются классы пользовательский исключений в языке Python?

В Python можно создавать собственные исключения. Такая практика позволяет увеличить гибкость процесса обработки ошибок в рамках той предметной области, для которой написана программа.

Для реализации собственного типа исключения необходимо создать класс, являющийся наследником от одного из классов исключений. Пример:

*class NegValException(Exception)*

*pass*

*try:*

*val = int(input(“input positive number: “))*

*if val < 0:*

*raise NegValException(“Neg val: “ + str(val))*

*print(val + 10)*

*except NegValException as e:*

*print(e)*

1. Каково назначение модуля logging?

С помощью logging на Python можно записывать в лог исключения. Обычно лог пишется в файл. Уровень INFO указывает, что сообщения уровней ниже не будут отображаться в логе. Python позволяет в try-except получить текст ошибки.

1. Какие уровни логгирования поддерживаются модулем logging? Приведите примеры, в которых могут быть использованы сообщения с этим уровнем журналирования.

В модуле logging присутствуют следующие уровни логирования:

|  |  |  |
| --- | --- | --- |
| Уровень | Числовое значение | Применение |
| CRITICAL | 50 | Серьёзная ошибка, обозначающая, что программа не может продолжить своё выполнение |
| ERROR | 40 | Из-за серьёзных проблем программа не может выполнять некоторые функции |
| WARNING | 30 | Индикация какого-либо неожидаемого события или проблемы в ближайшем будущем (к примеру «на диске мало места»). Программа продолжает свою работу |
| INFO | 20 | Индикация штатного выполнения программы (например, действие пользователя или инициализация каких-либо модулей) |
| DEBUG | 10 | Детальная информация, нужная для диагностики программы |
| NOTSET | 0 | Не указанный явно уровень логирования, который будет наследовать уровень логирования от своего родительского логгера |

**Выводы**: В процессе выполнения лабораторной работы были приобретены навыки по работе с исключениями при написании программ с помощью языка программирования Python версии 3.x, приобретены навыки по работе с логированием, а также проработаны примеры, выполнены общие и индивидуальные задания.

Ссылка на репозиторий GitHub:

[IUnnamedUserI/OOP\_4: Объектно-ориентированное программирование. Лабораторная работа №4](https://github.com/IUnnamedUserI/OOP_4)