МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования
«Новосибирский государственный университет экономики и управления «НИНХ»
(ФГБОУ ВО «НГУЭУ», НГУЭУ)

Кафедра информационных технологий

КУРСОВАЯ РАБОТА

Программное приложение «Каталог звезд нашей солнечной системы»

Дисциплина: Программирование
Ф.И.О студента: Ярославцев Никита Витальевич
Направление: 02.03.02 Фундаментальная информатика и информационные технологии
Направленность (профиль): Программная инженерия
Номер группы: ФИ202
Номер зачетной книжки: 220145
Проверил: Пестунов Андрей Игоревич, Кандидат физико-математических наук, Заведующий кафедрой информационных технологий

Новосибирск 2023

Задание на курсовую

Заявление о

СОДЕРЖАНИЕ

I. РЕАЛИЗАЦИЯ ПОСТАВЛЕННОЙ ЗАДАЧИ

Задача предельно проста, написать консольное приложение на языке Python. Консольное приложение или программа командной строки - это компьютерная программа, предназначенная для использования через текстовый пользовательский интерфейс, такой как текстовый терминал, интерфейс командной строки некоторых операционных систем (Unix, DOS, и т.д.) или текстовый интерфейс, входящий в состав большинства операционных систем с графическим пользовательским интерфейсом (GUI), таких как консоль Windows в Microsoft Windows, Terminal в macOS и xterm в Unix.

Для того, чтобы начать работу с программой, необходимо её запустить и ввести команду. А для этого, программа должна пригласить пользователя на ввод команд.

Листинг 1.1 – Вывод списка команд и приглашение на взаимодействие

```
if __name__ == "__main__":
  print("«Каталог звезд нашей галактики»\n\nСписок возможных команд:\n"
        "добавить - добавление звезды\n"
        "удалить - удаление выбранной звезды\n"
        "список - вывод всех звезд в виде таблицы\n"
        "сортировать - сортировка звезд по выбранному параметру\n"
        "фильтр - вывод всех звезд, у которых масса не превосходит
заданной величины\n"
        "перевод - вывод расстояния от Солнца до звезд в парсеках\n"
        "выход - выход из программы\n")
  while True:
    command = input("Введите команду: ")
    executing_the_command(command)
```

Обратимся к листингу 1.1.

Если мы захотим импортировать наши процедуры в другие проекты, то конструкция `if __name__ == "__main__":` не позволит перенести в новый код, часть повторно запрашивающую команду от пользователя. Далее с помощью `print("")` для человека единожды выводится название программы и список возможных команд с кратким пояснением их работы. Последние три строки листинга самые важные, здесь запускается вечный цикл,

приглашающий пользователя ввести команду и исполнение этой команды с помощью процедуры из листинга 1.2.

Листинг 1.2 – Выполнение команды

```
def executing_the_command():
  try:
    commands = {"выход": close_console, "добавить": add_obj,
                "удалить": del_obj, "список": list_obj,
                "сортировать": sort_obj, "фильтр": filter_obj,
                "перевод": convert_obj}
    commands[command.lower().split(" ")[0]]()
  except KeyError:
    print("Такой команды не существует!")
```

Рассмотрим в листинге 1.2 конструкцию `commands[command.lower().split(" ")[0]]()`. Данная строка обращается к словарю `commands`, по ключу-команде создает ссылку на процедуру ответственную за выполнение запрашиваемых действий и инициализируется с помощью `()`. Обратим внимание на `command.lower().split(" ")[0]`, эта цепочка методов приводит введенную команду к нижнему регистру, разделяет строку по пробелу на элементы списка и берёт самый первый элемент. Таким образом наша программа становится не чувствительной к регистру и игнорирует случайно введенные символы через пробел.

Как можно заметить часть процедуры заключена в конструкцию `try: … except KeyError: …` . Она необходима на тот случай, когда человек введет команду которая содержит синтаксические ошибки или не существует вообще. В нашей процедуре мы создали словарь, в который внесли ключи – названия команд и значения – названия процедуры. При обращении к словарю по неправильному ключу интерпретатор Python выдаст ошибку `KeyError`, на эту ошибку и реагирует наша конструкция `try: … except …:` выдавая сообщение о ошибке в запросе.

Листинг 1.3 – Процедура вывода списка команд и закрытия приложения

```
def close_console():
  """Закрытие программы"""
  print("Программа закрыта!")
  exit(1)

def help_commands():
  """Выдача всех команд"""
  print("«Каталог звезд нашей галактики»\n\nСписок возможных команд:\n"
        "добавить - добавление звезды\n"
        "удалить - удаление выбранной звезды\n"
        "список - вывод всех звезд в виде таблицы\n"
        "сортировать - сортировка звезд по выбранному параметру\n"
        "фильтр - вывод всех звезд, у которых масса не превосходит
заданной величины\n"
        "перевод - вывод расстояния от Солнца до звезд в парсеках\n"
        "помощь - вывод возможных команд\nвыход - выход из программы\n")
```

В листинге 1.3 приведены две процедуры. `help_commands()` выводит список существующих команд, на тот случай если надо проверить правильность написания команд или вспомнить их. А `close_console()` запускает процесс закрытия программы.

Листинг 1.4 – Добавление нового объекта

```
star_dict = {"Сириус": ("Большой Пёс", 9, 2),
             "Поллукс": ("Близнецы", 34, 2),
             "Альнитак": ("Орион", 817, 4),
             "Антарес": ("Скорпион", 550, 13),
             "Проксима Центавра": ("Альфа Центавра", 4, 7)}

def add_obj():
  while True:
    if len(star_dict) < 12:
      while True:
        name_star = input("Введите название звезды: ")
        if name_star not in [i * " " for i in range(10)]:
          break
        else:
          print("Название не может быть пустым!")
      while True:
        name_constellation = input("Введите название созвездия: ")
        if name_constellation not in [i * " " for i in range(10)]:
          break
        else:
          print("Название не может быть пустым!")
      while True:
        try:
          distance = int(input("Введите расстояние в световых годах
(округлите до целого числа): "))
          break
        except ValueError:
          print("Данные не являются целым числом. Попробуйте ещё раз.")
```

```
      while True:
        try:
          weight = int(input("Введите массу в массах Солнца (округлите
до целого числа): "))
          break
        except ValueError:
          print("Данные не являются целым числом. Попробуйте ещё раз.")
      print(f"Будет создан новый объект:\nЗвезда:
{name_star}\nСозвездие: {name_constellation}\n"
          f"Расстояние от Солнца (в св. годах): {distance}\nМасса (в
массах Солнца): {weight}")
      while True:
        confirmation = input("Верны ли введеные данные? (д/н)")
        if confirmation == "д":
          star_dict[name_star] = (name_constellation, distance, weight)
          print(f"Новый объект добавлен!")
          break
        elif confirmation != ["д", "н"]:
          print(f"Неизвестный ответ. Попробуйте ещё раз.")
      break
    else:
      print("Невозможно добавить новый объект!\nУдалите один объект,
чтобы добавить новый.")
      break
```

Пользователь должен иметь возможность добавлять новые объекты в каталог приложения. Реализация этого процесса представлена в листинге 1.4. Процедура `add_obj()` представляет собой несколько вечных циклов, заключенных в ещё один цикл. Каждый цикл ожидает от пользователя некоторую информацию о добавляемом объекте (название звезды, название созвездия, расстояние и масса). При корректном вводе данных, осуществляется выход из одного и запуск следующего цикла. Последний цикл выводит введенные данные на проверку и ожидает подтверждения. Для хранения объектов каталога, в глобальной области создан словарь звезд, ключом является название самой звезды, а значением – множество данных об этой звезде. Перед началом запроса данных о новом объекте, проходит проверка на количество имеющихся звезд. Максимально возможное число объектов – 12.

Листинг 1.5 – Удаление объекта

```
def del_obj():
  key = list(star_dict.keys())
  print("Звезды которые можно удалить:  ")
  print(f"--|{key[0]}|", end="--")
  for i in range(1, len(key)-1):
    print(f"|{key[i]}|", end="--")
```

```
  print(f"|{key[-1]}|", end="--\n")
  while True:
    deleted = input("Выберите какую звезду будем удалять: ")
    try:
      del star_dict[deleted]
      print("Звезда успешно удалена")
      break
    except KeyError:
      print("Такой звезды не существует. Попробуйте ещё раз")
```

Также в программе имеется возможность удалить какую-либо имеющуюся звезду. Реализация процедуры представлена в листинге 1.5.

Функция `del_obj()` выводит список ключей (названий звезд), и запрашивает один ключ для удаления объекта.

Листинг 1.6 – Вывод всех объектов в виде таблицы

```
try:
  from prettytable import PrettyTable
except ModuleNotFoundError:
  import subprocess
  import sys
  package = 'PrettyTable'
  print(f"Внимание! Будет установлен недостающий модуль \"{package}\"")
  subprocess.check_call([sys.executable, '-m', 'pip', 'install',
package])

th = ["№", "Название", "Созвездие", "Расстояние от Солнца (в св.г.)",
"Масса (в массах Солнца)"]

def list_obj(sort_key: list = None):
  columns = len(th)
  table = PrettyTable(th)
  td_data = []
  number = 1
  if sort_key is None:
    key = star_dict
  else:
    key = sort_key
  for i in key:
    td_data.append(number)
    td_data.append(i)
    td_data.append(star_dict[i][0])
    td_data.append(star_dict[i][1])
    td_data.append(star_dict[i][2])
    number += 1
  while td_data:
    table.add_row(td_data[:columns])
    td_data = td_data[columns:]
  print(table)
```

Следующая команда которая может быть выполнена приложением – вывод всех объектов в виде таблицы (листинг 1.6).

В данной ситуации два варианта реализации. Или продумывать систему контроля отступов в выводимой таблице, для правильного и визуально удобного чтения таблицы, или воспользоваться существующими библиотеками. В листинге 1.6 была использована библиотека `PrettyTable`[1]. Чтобы мы могли воспользоваться данной библиотекой, её надо импортировать. Воспользуемся конструкцией `try: … except ModuleNotFoundError:` на тот случай, если данный модуль не был установлен ранее (в случае возникновения ошибки, нужная библиотека установится перед запуском приложения, заранее предупредив пользователя).

В глобальной области переменных создаем список из названий колонок (в последствии он пригодится и в других процедурах). В самой процедуре подсчитываем количество колонок, создаем таблицу на основе списка колонок. Преобразуем словарь `star_dict` в список `td_data`. И построчно будем добавлять данные из списка в созданную таблицу. Завершающим шагом выводим таблицу пользователю.

Листинг 1.7 – Выбор параметров сортировки

```
def sort_obj():
  while True:
    try:
      par_sort = input("Выберите параметр сортировки
                        (название/созвездие/расстояние/масса): ")
      dict_sort = {"название": name_sort,
                   "созвездие": constellation_sort,
                   "расстояние": distance_sort, "масса": weight_sort}
      dict_sort[par_sort.split(" ")[0]]()
      break
    except KeyError:
      print("Нет такого параметра!")
```

Листинг 1.7 по своему функционалу повторяет код из листинга 1.2, тот же принцип создания ссылки, реализованный через обращение к словарю.

Листинг 1.8 – Сортировка по выбранному параметру

```
def sorting_selection():
  """Выбор сортировки"""
  while True:
    par_sort = input("Сортировать по возрастанию или убыванию? (+/-): ")
    if par_sort in ["+", "-"]:
      break
    else:
      print("Неизвестный параметр! Укажите \"+\" - возрастание или \"-\"
- убывание.")
  if par_sort == "+":
    par_sort = False
  else:
    par_sort = True
  return par_sort

def weight_sort():
  """Сортировка списка по массе"""
  par_sort = sorting_selection
  keys = list(star_dict.keys())
  part_star_dict = {}
  for i in keys:
    part_star_dict[i] = star_dict[i][2]
  sorted_dict = {}
  sorted_keys = sorted(part_star_dict, key=part_star_dict.get,
reverse=par_sort())
  for w in sorted_keys:
    sorted_dict[w] = part_star_dict[w]
  list_obj(list(sorted_dict.keys()))

def distance_sort():
  """Сортировка списка по расстоянию"""
  par_sort = sorting_selection
  keys = list(star_dict.keys())
  part_star_dict = {}
  for i in keys:
    part_star_dict[i] = star_dict[i][1]
  sorted_dict = {}
  sorted_keys = sorted(part_star_dict, key=part_star_dict.get,
reverse=par_sort())
  for w in sorted_keys:
    sorted_dict[w] = part_star_dict[w]
  list_obj(list(sorted_dict.keys()))

def constellation_sort():
  """Сортировка списка по созвездию"""
  par_sort = sorting_selection
  keys = list(star_dict.keys())
  part_star_dict = {}
  for i in keys:
    part_star_dict[i] = star_dict[i][0]
  sorted_dict = {}
  sorted_keys = sorted(part_star_dict, key=part_star_dict.get,
reverse=par_sort())
  for w in sorted_keys:
    sorted_dict[w] = part_star_dict[w]
  list_obj(list(sorted_dict.keys()))
```

```
def name_sort():
  name = list(star_dict.keys())
  par_sort = sorting_selection
  name.sort(reverse=par_sort())
  list_obj(name)
```

В листинге 1.8 представлено 5 функций. Процедуры `name_sort()`, `weight_sort()`, `distance_sort()`, `constellation_sort()` имеют схожую структуру. `name_sort()` отличается своей простотой, т.к. сортировка осуществляется по названиям ключей. Остальные 3 процедуры сложнее. В каждой из них создается новый временный словарь который хранит общий для всех словарей ключ (название объекта) и значение по которому проходит сортировка. Далее ключи сортируются по значению. Программа может сортировать значения по убыванию или возрастанию. Для этого вызывается функция `sorting_selection()`, которая уточняет этот параметр у человека и возвращает `True/False`.

На основе отсортированных ключей создается итоговый словарь, который подобен глобальному `star_dict`, отличием является порядок данных.

При выполнении этих процедур нам также необходимо выводить таблицу. Для оптимизации программы мы будем вызывать процедуру `list_obj()` (листинг 1.6), и передавать ей новый словарь данных.

Листинг 1.9 – Изменение данных одной графы и вывод списка

```
def convert_obj():
  """Вывод расстояния до всех звезд в парсеках вместо световых
лет"""
  th1 = ["№", "Название", "Созвездие", "Расстояние от Солнца (в
парсеках)", "Масса (в массах Солнца)"]
  columns = len(th1)
  table = PrettyTable(th1)
  td_data = []
  number = 1
  for i in star_dict:
    td_data.append(number)
    td_data.append(i)
    td_data.append(star_dict[i][0])
    td_data.append(round(star_dict[i][1] * 0.306, 2))
    td_data.append(star_dict[i][2])
    number += 1
```

```
  while td_data:
    table.add_row(td_data[:columns])
    td_data = td_data[columns:]
  print(table)
```

Одна из возможностей программы, это вывод таблицы данных, но с изменением единиц измерения расстояния от Солнца до объекта. Принцип работы идентичен работе процедуры `list_obj()` (листинг 1.6). Единственное отличие, в момент создания списка, значение расстояния умножается на 0,306. Это коэффициент различия одного светового года к парсеку.

Листинг 1.10 – Вывод объектов подходящих по условие

```
def filter_obj():
  """Вывод всех звезд, у которых масса не превосходит заданной
величины"""
  columns = len(th)
  table = PrettyTable(th)
  td_data = []
  number = 1
  while True:
    try:
      weight_user = int(input("Задайте фильтр по массе (целое число):
"))
      break
    except ValueError:
      print("Данные не являются целым числом. Попробуйте ещё раз.")
  for i in star_dict:
    if star_dict[i][2] <= weight_user:
      td_data.append(number)
      td_data.append(i)
      td_data.append(star_dict[i][0])
      td_data.append(star_dict[i][1])
      td_data.append(star_dict[i][2])
      number += 1
  while td_data:
    table.add_row(td_data[:columns])
    td_data = td_data[columns:]
  print(table)
```

Последняя функция консольного приложения, фильтрация имеющихся данных (листинг 1.10). Суть работы схожа с кодом из листинга 1.6. Для работы дополнительно запрашивается параметр для фильтрации, и запись в список данных происходит, когда заданный параметр больше или равен значению в у объекта в каталоге.

II. СПИСОК ТЕСТОВЫХ СЛУЧАЕВ ДЛЯ РУЧНОГО ТЕСТИРОВАНИЯ ПРОГРАММЫ

№	**Описание тестового случая (выполняемые действия)**	**Ожидаемый результат**
1	Запуск приложения	Вывод названия каталога и списка существующих команд
2	Ввод несуществующей команды	Сообщение о неправильности запроса
3	Ввод команды «Добавить»	Поочередный запрос данных об объекте (название, наз. созвездия, расстояние от солнца (в св.г.), масса объекта (в массах Солнца)). Вывод введенных данных на проверку. Сообщение о невозможности добавления объекта в случае, если это 13-й объект
4	Ввод команды «Удалить»	Вывод возможных объектов для удаления
5	Ввод команды «Сортировать»	Запрос параметров сортировки (название объекта, название созвездия, расстояние, масса; для каждого параметра по убыванию или возрастанию). Вывод списка отсортированного по выбранному параметру
6	Ввод команды «Фильтр»	Запрос параметра фильтрации (масса). вывод всех звезд, у которых масса не превосходит заданной величины
7	Ввод команды «Перевод»	Замена световых лет на парсеки. Вывод списка объектов с изменённой колонкой «Расстояние от Солнца»
8	Ввод команды «Список»	Вывод всех объектов в каталоге
9	Ввод команды «Помощь»	Вывод списка возможных команд
10	Ввод команды «Выход»	Сообщение о закрытии программы. Выход из программы

III. СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕАТУРЫ

1. Описание проекта PrettyTable. — Текст : электронный // pypi : [сайт]. — URL: https://pypi.org/project/prettytable/ (дата обращения: 02.04.2023).