Министерство образования и науки Российской Федерации

Санкт-Петербургский Политехнический Университет Петра Великого

—

Институт кибербезопасности и защиты информации

ЛАБОРАТОРНАЯ РАБОТА № 1

«Управление ФС/реестром посредством CLI приложения на ЯП Python»

1. по дисциплине «Цифровая культура»
2. Выполнил
3. студент гр. 4851004/10001 Веремейчук Я.Ю.

<подпись>

1. Преподаватель
2. асс. преподавателя Данилов В.Д.

<подпись>

1. Санкт-Петербург
2. 2023
3. Цель работы

Получение навыков управления файловой системой и реестром с использованием языка программирования Python.

1. Постановка задачи
2. Установить Python 3.X на систему
3. Установить IDE для разработки с помощью языка программирования Python
4. Реализовать консольное приложение на языке программирования Python, которое будет поддерживать операции управления ФС и реестром.
5. Описание решения

Для выполнения работы был использованы модули argparse, shutil, winreg и os.

Argparse модуль упрощает написание удобных интерфейсов командной строки. Программа определяет, какие аргументы ей требуются, и argparse выяснит, как их проанализировать. Argparse модуль также автоматически генерирует справочные и пользовательские сообщения. Модуль также будет выдавать ошибки, когда пользователи предоставляют программе недопустимые аргументы.

Модуль os предоставляет множество функций для работы с операционной системой, причём их поведение, как правило, не зависит от ОС, поэтому программы остаются переносимыми. Argparse написан на основе os.

Модуль shutil содержит набор функций высокого уровня для обработки файлов, групп файлов, и папок. В частности, доступные здесь функции позволяют копировать, перемещать и удалять файлы и папки.

Эти функции предоставляют доступ к API реестра Windows для Python. Вместо использования целого числа в качестве дескриптора реестра используется объект дескриптора для обеспечения правильного закрытия дескрипторов, даже если программист пренебрегает их явным закрытием.

Были заданы опциональные (необязательные) аргументы, а также функциональный (обязательный) аргумент.

Для каждой опции, которую было необходимо реализовать, был задан ключ и описание. Также для опций по работе с реестром были заданы choices, чтобы работать с разными папками реестра.

parser.add\_argument('-c', '--create\_file', help='создать файл', action='store\_true')

parser.add\_argument('-d', '--delete\_file', help='удалить файл', action='store\_true')

parser.add\_argument('-w', '--write\_file', help='записать в файл', action='store\_true')

parser.add\_argument('-r', '--read\_file', help='прочитать из файла', action='store\_true')

parser.add\_argument('-p', '--copy\_file', help='скопировать файл из одной директории в другую', action='store\_true')

parser.add\_argument('-n', '--rename\_file', help='переименовать файл', action='store\_true')

parser.add\_argument('-C', '--create\_key', choices=['HKEY\_CURRENT\_USER',

'HKEY\_LOCAL\_MACHINE',

'HKEY\_USERS',

'HKEY\_CLASSES\_ROOT',

'HKEY\_CURRENT\_CONFIG'], help='создать ключ')

parser.add\_argument('-D', '--delete\_key', choices=['HKEY\_CURRENT\_USER',

'HKEY\_LOCAL\_MACHINE',

'HKEY\_USERS',

'HKEY\_CLASSES\_ROOT',

'HKEY\_CURRENT\_CONFIG'], help='удалить ключ')

parser.add\_argument('-W', '--write\_key', choices=['HKEY\_CURRENT\_USER',

'HKEY\_LOCAL\_MACHINE',

'HKEY\_USERS',

'HKEY\_CLASSES\_ROOT',

'HKEY\_CURRENT\_CONFIG'],help='записать значение в ключ')

# ^^^ optional arguments

parser.add\_argument('fr\_name', help='имя файла или путь реестра', type=str) # <<< positional argument

Далее для каждой функции была создана отдельная процедура def.

def func\_create\_file(name):

f = open(name, 'x')

f.close()

return 0

Исполнительный код состоит из конструкции if-elif, который обрабатывает введенные аргументы и вызывает нужную функцию.

if args.create\_file:

func\_create\_file(args.fr\_name)

elif args.delete\_file:

func\_delete\_file(args.fr\_name)

1. Выводы

Были получены навыки управления файловой системой и реестром с использованием языка программирования Python.

1. Контрольные вопросы
2. Компилируемый язык — это такой язык, что программа, будучи скомпилированной, содержит инструкции целевой машины; этот машинный код непонятен людям. Примеры истинных компилируемых языков: C, C++, Erlang, Haskell, Rust и Go. Преимущества: Программы, скомпилированные в машинный код, как правило, работают быстрее, чем интерпретируемый код. Это связано с тем, что процесс перевода кода во время выполнения увеличивает накладные расходы и может привести к замедлению работы программы в целом. Недостатки: Дополнительное время, необходимое для завершения всего этапа компиляции перед тестированием и зависимость сгенерированного двоичного кода от платформы. Интерпретируемый же язык — это такой, в котором инструкции не исполняются целевой машиной, а считываются и исполняются другой программой (которая обычно написана на языке целевой машины). Примеры популярных интерпретируемых языков: Basic, PHP, Ruby, Python и JavaScript. Достоинства: интерпретируемые языки, как правило, более гибкие и часто предлагают такие функции, как динамическая типизация и меньший размер программы. Кроме того, поскольку интерпретаторы сами выполняют исходный программный код, сам код не зависит от платформы. Недостатки: большое время выполнения кода.
3. Input(), а также с помощью модулей sys, getopt и optparse.
4. Конструкция with open(‘test.txt’, ‘r’) as fd принудительно вызывает функцию закрытия файла. Для fd = open(‘test.txt’, ‘r’) необходимо вызывать функцию fd.close().
5. В языке программирования Python подключение пакетов и модулей осуществляется с помощью import. Это позволяет распределять код по логическим "узлам" приложения(модели данных, обработчики, и тп.), что позволяет получить менее нагруженные кодом файлы.   
   методы установки пакетов Python:

* Еeasy\_install
* pip

Чтобы использовать [easy\_install](https://python-scripts.com/how-to-install-modules-python), вам сначала нужно установить setuptools. Чтобы сделать это, необходимо скачать tar или zip файл с PyPI, и извлечь его в системе.

Если нужно начать установку пакета при помощи easy\_install, нужно сделать следующее:

easy\_install package\_name

easy\_install попытается скачать пакет с PyPI, скомпилировать его (если нужно) и установить его. Если зайти в свою директорию site-packages, можно найти файл под названием easy-install.pth, который содержит доступ ко всем пакетам, установленным через easy\_install. Python использует этот файл, чтобы помочь в импорте модуля или пакета. Можно указать easy\_install на установку через URL или через путь на вашем компьютере. easy\_install также может выполнить установку прямиком из файла tar. Также можно использовать easy\_install для обновления пакета, воспользовавшись функцией upgrade (или–U). И наконец, можно использовать easy\_install для установки файла egg файлов.

Чтобы установить pip, необходимо зайти в PyPI, но вместо того, чтобы скачать пакет и запустить его скрипт setup.py, можно скачать скрипт под названием get-pip.py. После этого необходимо выполнить:

python get-pip.py

ПРИЛОЖЕНИЕ А

Листинг программы «Управление ФС/реестром посредством CLI приложения на ЯП Python»

import argparse

import os

import shutil

import winreg

parser = argparse.ArgumentParser()

parser.add\_argument('-c', '--create\_file', help='создать файл', action='store\_true')

parser.add\_argument('-d', '--delete\_file', help='удалить файл', action='store\_true')

parser.add\_argument('-w', '--write\_file', help='записать в файл', action='store\_true')

parser.add\_argument('-r', '--read\_file', help='прочитать из файла', action='store\_true')

parser.add\_argument('-p', '--copy\_file', help='скопировать файл из одной директории в другую', action='store\_true')

parser.add\_argument('-n', '--rename\_file', help='переименовать файл', action='store\_true')

parser.add\_argument('-C', '--create\_key', choices=['HKEY\_CURRENT\_USER',

'HKEY\_LOCAL\_MACHINE',

'HKEY\_USERS',

'HKEY\_CLASSES\_ROOT',

'HKEY\_CURRENT\_CONFIG'], help='создать ключ')

parser.add\_argument('-D', '--delete\_key', choices=['HKEY\_CURRENT\_USER',

'HKEY\_LOCAL\_MACHINE',

'HKEY\_USERS',

'HKEY\_CLASSES\_ROOT',

'HKEY\_CURRENT\_CONFIG'], help='удалить ключ')

parser.add\_argument('-W', '--write\_key', choices=['HKEY\_CURRENT\_USER',

'HKEY\_LOCAL\_MACHINE',

'HKEY\_USERS',

'HKEY\_CLASSES\_ROOT',

'HKEY\_CURRENT\_CONFIG'],help='записать значение в ключ')

# ^^^ optional arguments

parser.add\_argument('fr\_name', help='имя файла или путь реестра', type=str) # <<< positional argument

args = parser.parse\_args()

def func\_create\_file(name):

f = open(name, 'x')

f.close()

return 0

def func\_delete\_file(name):

os.remove(name)

return 0

def func\_write\_file(name):

f = open(name, 'w')

print("Напишите ваш текст:")

user\_str = input()

f.write(user\_str)

return 0

def func\_read\_file(name):

f = open(name, 'r')

file\_str = f.read()

print(file\_str)

f.close()

return 0

def func\_copy\_file(name):

print("Напишите место для копирования файла:")

user\_str = input()

shutil.copy(name, user\_str)

return 0

def func\_rename\_file(name):

print("Напишите новое имя файла:")

user\_str = input()

os.rename(name, user\_str)

return 0

def func\_create\_key(name):

if args.create\_key == 'HKEY\_CURRENT\_USER':

path = winreg.HKEY\_CURRENT\_USER

elif args.create\_key == 'HKEY\_LOCAL\_MACHINE':

path = winreg.HKEY\_LOCAL\_MACHINE

elif args.create\_key == 'HKEY\_USERS':

path = winreg.HKEY\_USERS

elif args.create\_key == 'HKEY\_CLASSES\_ROOT':

path = winreg.HKEY\_CLASSES\_ROOT

elif args.create\_key == 'HKEY\_CURRENT\_CONFIG':

path = winreg.HKEY\_CURRENT\_CONFIG

software = winreg.OpenKeyEx(path, name)

if software:

print("Назовите новую папку:")

user\_str = input()

new\_key = winreg.CreateKey(software, user\_str)

print("Назовите новый ключ:")

user\_name = input()

print("Укажите новое значение:")

user\_value = input()

winreg.SetValueEx(new\_key, user\_name, 0, winreg.REG\_SZ, user\_value)

if new\_key:

winreg.CloseKey(new\_key)

return 0

def func\_delete\_key(name):

if args.delete\_key == 'HKEY\_CURRENT\_USER':

path = winreg.HKEY\_CURRENT\_USER

elif args.delete\_key == 'HKEY\_LOCAL\_MACHINE':

path = winreg.HKEY\_LOCAL\_MACHINE

elif args.delete\_key == 'HKEY\_USERS':

path = winreg.HKEY\_USERS

elif args.delete\_key == 'HKEY\_CLASSES\_ROOT':

path = winreg.HKEY\_CLASSES\_ROOT

elif args.delete\_key == 'HKEY\_CURRENT\_CONFIG':

path = winreg.HKEY\_CURRENT\_CONFIG

software = winreg.OpenKeyEx(path, name)

if software:

winreg.DeleteKey(path, name)

return 0

def func\_write\_key(name):

if args.write\_key == 'HKEY\_CURRENT\_USER':

path = winreg.HKEY\_CURRENT\_USER

elif args.write\_key == 'HKEY\_LOCAL\_MACHINE':

path = winreg.HKEY\_LOCAL\_MACHINE

elif args.write\_key == 'HKEY\_USERS':

path = winreg.HKEY\_USERS

elif args.write\_key == 'HKEY\_CLASSES\_ROOT':

path = winreg.HKEY\_CLASSES\_ROOT

elif args.write\_key == 'HKEY\_CURRENT\_CONFIG':

path = winreg.HKEY\_CURRENT\_CONFIG

software = winreg.OpenKeyEx(path, name)

if software:

print("Укажите новое значение:")

user\_value = input()

winreg.SetValue(path, name, winreg.REG\_SZ, user\_value)

winreg.CloseKey(software)

return 0

print(args)

if args.create\_file:

func\_create\_file(args.fr\_name)

elif args.delete\_file:

func\_delete\_file(args.fr\_name)

elif args.write\_file:

func\_write\_file(args.fr\_name)

elif args.read\_file:

func\_read\_file(args.fr\_name)

elif args.copy\_file:

func\_copy\_file(args.fr\_name)

elif args.rename\_file:

func\_rename\_file(args.fr\_name)

elif args.create\_key:

func\_create\_key(args.fr\_name)

elif args.delete\_key:

func\_delete\_key(args.fr\_name)

elif args.write\_key:

func\_write\_key(args.fr\_name)