Федеральное государственное автономное образовательное учреждение высшего образования

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

«Языки системного программирования: Работа с файловой системой в языке высокого уровня Python»

Проверил:
Сентерев Ю.А. _____
«_____» _____ 201_г.

Выполнил:
Студент группы P3255
Кабардинов Д. В. _____

Оценка _____

Санкт-Петербург

2019

Задание:
Составить программу которая создает резервные копии важных файлов

Цель работы:

- Рассмотреть какими возможностями для работы с файловой системой обладает язык Python на примере модулей os и pathlib
- На основании полученных знаний попытаться ответить на вопрос о корректности отнесения языка Python к языкам системного программирования

Задачи:

1. Составить программу, создающую резервные копии файлов с использованим модуля *pathlib*
2. Составить программу, с тем же функционалом, что в задаче 1, но используя модуль *os*

Ход выполнения работы

1. Решим поставленную задачу с использованием модуля *pathlib*:

```
from argparse import ArgumentParser
from pathlib import Path
from datetime import datetime
import tarfile

parser = ArgumentParser()
parser.add_argument('src', help='''Source directory. \
        If it is not specified, current directory \
        will be used as source.''', nargs='?', default='.')
parser.add_argument('dest', help='''destination directory -\
         all important files from source directory will be placed here''',
         nargs='?', default=None)
args = parser.parse_args()
srcPath = Path(args.src)
# get list of important files
files = [x for x in srcPath.glob('**/important*') if not x.is_dir()]
# create archive in the source directory
archiveName = datetime.now().strftime('%Y-%m-%d') + '.tar.gz'
archivePath = str(srcPath.joinpath(archiveName))
tar = tarfile.open(archivePath, 'w:gz')
for file in files:
        tar.add(str(file), file.name, False)
tar.close()
# create destination directory if it doesn't exist already
dest = args.dest
backupDir = Path(dest) if dest else Path.home() / 'backup'
if not backupDir.exists():
        backupDir.mkdir(0o777, parents=True)
destPath = backupDir / archiveName
# move archive to destination directory
archive = Path(archivePath)
archive.rename(destPath)
```

Программа, код которой приведен выше, работает следующим образом. Из аргументов командой строки считываются путь директории, из которой необходимо произвести резервное копирование файлов, а также путь, по которому будет сохранён архив со всеми важными файлами, найденными в этой директории и всех её поддиректориях. Программа выбирает файлы, которые необходимо скопировать, инспектируя названия файлов: имена файлов, подлежащих резервному копированию должны иметь префикс *important*. Например, имея следующую структуру директорий:

```
test
├── 1.txt
├── important
│   └── important_something.py
└── important_document.txt
```

и передав программе путь к директории tes в качестве первого аргумента, резервные копии будут созданы для файлов *important_something.py* и *important_document.txt*. Эти файлы будут упакованы в архив tar и сжаты с помощью gzip. Название архива будет сгенерировано из текущей даты (например 2019-04-13.tar.gz). Если при вызове программы вторым аргументом командной строки будет передан путь, архив будет сохранён по этому пути. Если же второй аргумент опущен, в домашней директории пользователя будет создана папка *backup* и архив будет сохранён в неё. Если же программа вызвана без аргументов, в качестве директории-источника будет использована текущая директория.

Таким образом задача 1 решена. Модуль pathlib предоставляет удобный интерфейс для работы с файловой системой.

2. Перепишем программу, на этот раз используя модуль os

```
from argparse import ArgumentParser
from datetime import datetime
import tarfile
import os
from glob import glob
parser = ArgumentParser()
parser.add_argument('src', help='''Source directory. If it is not \
        specified current directory \
        will be used as source.''', nargs='?', default=os.getcwd())
parser.add_argument('dest', help='''destination directory -\
         all important files from source directory will be placed here''',
         nargs='?', default=None)
args = parser.parse_args()
srcPath = args.src
# get list of important files
files = glob(srcPath + '/**/important*', recursive=True)
files = filter(lambda file: os.path.isfile(file), files)
# create archive in the source directory
archiveName = datetime.now().strftime('%Y-%m-%d') + '.tar.gz'
archivePath = str(os.path.join(archiveName))
tar = tarfile.open(archivePath, 'w:gz')
for file in files:
```

```
        tar.add(file, os.path.basename(file), False)
tar.close()
# create destination directory if it doesn't exist already
dest = args.dest
backupDir = dest if dest else os.path.join(os.path.expanduser('~'), 'backup')
if not os.path.exists(backupDir):
        os.makedirs(backupDir)
destPath = os.path.join(backupDir, archiveName)
# move archive to destination directory
os.rename(archivePath, destPath)
```

Программа работает идентично исходному варианту, и не слишком отличается от него по структуре. Как видно, оба модуля хорошо справляются с задачей и предоставляют все необходимые функции, которые могут потребоваться при работе с файлами, как то: запись, чтение, удаление, копирование и перемещение файлов. Следует также ометить, что работа с файловой системой является вообще говоря платформо-зависимой - В Windows есть ряд ограничений, накладываемых на названия файлов. В POSIX-совместимых системах разделителями фрагментов пути является прямой слэш (/), а в Windows - обратный, и тд. При этом оба рассмотренных модуля предоставляют интерфейс, абстрагирующий данную специфику в разлиных операционных системах, что конечно же оказывается весьма удобным.

Выводы:
Нам удалось выполнить поставленную задачу двумя способами. В результате можно сделать вывод, что язык программирования Python предоставляет все возможности для полноценной работы с файлами, и это одна из причин почему его можно отнести к языкам системного программирования. Таким образом, ответ на поставленный вопрос получен, цель работы достигнута.

Список используемой литературы:

1. `https://tinyurl.com/y3aj6pku`
   - Linked Lists in Detail with Python Examples: Single Linked Lists
2. `https://docs.python.org/3.5/tutorial/index.html` - The Python Tutorial
3. `https://docs.python.org/3/library/pathlib.html` - pathlib documentation
4. `https://docs.python.org/3/library/os.html#module-os` - os documentation
5. `https://docs.python.org/3/library/argparse.html` - argparse documentation
6. `https://docs.python.org/3/library/tarfile.html#module-tarfile` -
   tarfile documentation