Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт металлургии, машиностроения и транспорта

Работа допущена к защите
зам.директора ФизМех по дополнительному образованию,
ведущий специалист,
учебно-методическая деятельность
Хайбулова Евгения Александровна
«___»_____20__ г.

КУРСОВАЯ РАБОТА

Скрипт для работы с файлами/папками (переименование, генерация файлов)

по образовательной программе
Программирование на языке Python: базовый курс

Выполнил
слушатель
Белов Анатолий Юрьевич

Руководитель
Калюжнюк Александр Всеволодович

Санкт-Петербург
2021г.

Оглавление

Введение

Я хотел бы начать с предыстории, что именно меня заставило взяться за эту тему.

С 2011 года я преподаю информационные технологии франкоязычным студентам

из Африки. Поскольку количество студентов в нашем Университете увеличивается каждый год (и не только из Африки), то я сталкиваюсь с одной и той же проблемой: очень тяжело найти свободное время для отработок в компьютерных классах. Та же проблема есть и у системных администраторов: постоянная полная загрузка компьютеров и трудности с нахождением свободного от работы времени для обслуживания этих компьютеров. Поэтому задача автоматизации процессов обслуживания, в нашем случае – работа с файловой системой и самими файлами, приобретает важное значение. Автоматизация позволяет значительно сэкономить время для обслуживания компьютеров.

Достаточно настроить несколько скриптов, и затем их можно запускать на всех необходимых компьютерах. Это действие позволяет облегчить труд администраторов, а также увеличить их производительность труда: выигрыш по времени идет колоссальный!

Для того, чтобы создать исполняемый файл, для этого есть специальные методы упаковки скриптов, которые, как входят в комплект стандартной поставки, так и поставляются третьими разработчиками. Для того, чтобы создать набор скриптов, чтобы работать с файлами, требуется среда разработки Python, тогда, как для настройки самих компьютеров, с помощью уже созданных утилит (или утилиты) – необходимости в этой среде программирования – нет. Непосредственный процесс создания этой утилиты, на основании скриптов – выходит за рамки этой работы.

После каждого семестра в нашем Университете, системные администраторы обычно выполняют один и тот же план заданий: они удаляют ненужные

файлы от студентов, которые уже прошли обучение, и создают новые папки и файлы, предназначенные для студентов, которым еще только предстоит пройти курс обучения.

Кроме того, является хорошей практикой, полезной для всех, без исключения предприятий, - создание резервной копии нужных файлов на защищенном носителе, который может продублировать информацию с основных жестких дисков. Этот процесс тоже можно автоматизировать. Надо отметить, что есть профессиональные программы, предназначенные для копирования и работы с файлами (например Acronis), но они являются платными. Есть также возможности самих Операционных систем для создания резервных копий, так называемые Backup. Но дело в том, что написанные скрипты на языке Python можно адаптировать для нужных случаев непосредственно необходимых для данного предприятия. То, есть – скрипты на Python обладают гибкостью и в них можно учитывать особенности файлов и файловой системы для данного предприятия.

Я надеюсь, что скрипты, разработанные здесь, пригодятся системным администраторам нашего Университета в качестве примера, как можно автоматизировать их нелегкий труд и сэкономить время для чего-то еще полезного.

Постановка задач проекта

В своей работе я хотел бы рассмотреть тему взаимодействия с файлами и с файловой системой и автоматизацию этого процесса с помощью языка программирования Python.

В нашем Университете все компьютеры работают под управлением операционной системы Windows различных версий, поэтому именно для этой системы и будут рассматриваться примеры в этой курсовой работе.

Здесь будут рассмотрены учебные примеры, поскольку реальные, существующие в моем Университете, могут быть исполнены по этому же принципу, с простой заменой имен папок, файлов и их путей.

Модуль OS Python – описание и методы

Модуль OS Python предоставляет возможность установить взаимодействие между пользователем и операционной системой. Он предлагает множество полезных функций, которые используются для выполнения задач, связанных с операционной системой, и получения соответствующей информации. OS входит в состав стандартных служебных модулей Python.

Для большинства последующих примеров, приведенных в этом разделе, нам потребуется модуль os, поэтому его надо импортировать в начале каждого сценария и при каждом перезапуске IDLE (это интегрированная среда разработки и обучения). В противном случае будет выведено сообщение об ошибке NameError: name 'os ' is not defined.

Абсолютные и относительные пути доступа

Существуют два способа определения пути доступа к файлу:

абсолютный путь, который всегда начинается с имени корневой папки;

относительный путь, который задается относительно текущего рабочего каталога программы.

Существуют также папки, обозначаемые одним (.) или двумя (..) символами точки. Это не реальные папки, а специальные имена, которые могут использоваться при задании путей. Одиночная точка является сокращенным обозначением, имеющим смысл “данная папка”. Двойная точка имеет смысл “родительская папка”.

Обработка абсолютных и относительных путей

os.path()

Модуль os.path предоставляет функции для возврата абсолютного пути по заданному относительному пути и проверки того, является ли путь абсолютным.

Вызов os.path.abspath(path) возвращает строку абсолютного пути аргумента. Это простой способ преобразования относительного пути в абсолютный.

Вызов os.path.isabs(path) возвращает значение True, если аргумент — абсолютный путь, и False, если аргумент — относительный путь.

Вызов os.path.relpath (path, start) возвращает строку относительного пути от start к path. Если путь start не предоставлен, то в качестве него используется текущий рабочий каталог.

Пример:

```
import os
os.path.abspath('.')
```

Выход:

```
'C:\\Python34'
```

Поскольку в момент вызова функции os.path.abspath() текущим рабочим каталогом был C:\Python34, папка . \ представляет абсолютный путь 'C:\\Python34'.

os.path.dirname()

os.path.dirname(path) – эта функция возвращает строку, содержащую всю часть пути, которая предшествует последней косой черте в аргументе path.

Пример:

```
import os
path = 'C:\\Windows\\System32\\calc.exe'
os.path.dirname (path)
```

Выход:

```
'C:\\Windows\\System32'
```

os.getcwd()

os.getcwd() - Функция возвращает текущий рабочий каталог(CWD) файла.

Каждая программа, которая выполняется на компьютере, имеет текущий рабочий каталог (current working directory — cwd). Предполагается, что любые имена файлов или пути, которые не начинаются с указания корневой папки, заданы относительно текущего рабочего каталога.

Пример:

```
import os
print(os.getcwd())
```

Выход текущего каталога, откуда был запущен файл:

```
C:\Users\Python\Desktop\Module\OS
```

Для получения значения текущего рабочего каталога в виде строки используется функция

os. get.cwd(), а для его изменения — функция os. chdir().

os.chdir()

Модуль os предоставляет функцию chdir() для изменения текущего рабочего каталога.

```
import os
os.chdir("C:\\")
```

Выход:

```
C:\
```

При попытке перейти в несуществующий каталог Python выведет сообщение об ошибке.

Проверка существования пути

Многие функции Python завершаются аварийно с выдачей сообщения об ошибке в случае, если предоставленный им путь не существует. Модуль os.path представляет функции, позволяющие проверить, существует ли заданный путь и соответствует ли он файлу или папке.

Вызов os.path.exists(path) возвращает значение True, если файл (или папка), на который ссылается аргумент, существует, и значение False, если он не существует.

Вызов os.path.isfile(path) возвращает значение True, если заданный аргументом путь существует и является файлом, и значение False в противном случае.

Вызов os.path,isdir(path) возвращает значение True, если заданный аргументом путь существует и является папкой; иначе — False.

Вот примеры:

```
import os
os.path.exists('C:\\Windows')
```

Выход:

```
True
```

```
import os
os.path.isdir('C:\\Windows\\System32')
```

Выход:

```
True
```

Чтобы определить, подключен ли в данный момент к компьютеру DVD или флеш-диск, можно воспользоваться функцией os.path.exists(). Например, если бы я хотел проверить, подключен ли флеш-диск D:\ на моем Windows-компьютере, то я мог бы это сделать с помощью следующей команды.

os.path.exists('D:\\')

Выход:

False

Значит, система не нашла флешку.

os.mkdir()

Функция os.mkdir() используется для создания нового каталога. Рассмотрим следующий пример.

```
import os
os.mkdir("C:\\newdir")
```

Он создаст новый каталог по пути на диске C с именем newdir.

listdir()

Выводим на экран содержимое каталога с помощью функции listdir()

Теперь получим список всех файлов, содержащихся в этом каталоге (которых пока нет):

```
os.listdir("C:\\newdir")
```

Выход:

[]

Далее создадим подкаталог:

```
os.mkdir("C:\\newdir\\test")
os.listdir("C:\\newdir")
```

Выход:

['test']

os.rmdir()

Функция rmdir() удаляет указанный каталог с абсолютным или относительным путем. Здесь есть одна особенность: каталог должен быть

пустым. Каким образом удалить каталог, который содержит файлы или другие каталоги, это будет рассмотрено позднее.

Прежде всего, нужно изменить текущий рабочий каталог и удалить папку.

Пример:

Допустим наш текущий рабочий каталог находится в поддиректории директории, которая расположена на диске C (в результате применения комманды os.getcwd() это можно выяснить).

```
import os
# для удаления каталога и использования команды os.rmdir("c:\\newdir"), нужно изменить текущий рабочий каталог
os.chdir("..")
os.rmdir("newdir")
```

Здесь надо пояснить, что после символа #, все, что находится по правую сторону на этой же строке, интерпретатор воспринимает как комментарий, и никак не реагирует на содержимое этой строки. Эта информация является служебной и предназначена для людей.

Следующий момент: две точки внутри команды os.chdir("..") означает переход (смена) на один уровень вверх относительно текущего рабочего каталога. В этом случае применен относительный путь.

Переименование файлов в Python

rename()

Функция rename() используется для переименования файлов в Python. Пользователь может переименовать файл, если у него есть право на изменение. Для того, чтобы это проверить, можно вызвать контекстное меню на файле, выбрать опцию Свойства и там посмотреть во вкладке Безопасность, кто имеет и какие права для этого файла. Для использования функции, сперва нужно импортировать модуль os.

Синтаксис следующий.

```
import os
os.rename(src,dest)
```

Где,

src = файл, который нужно переименовать

dest = новое имя файла

Пример:

```
import os
# переименование xyz.txt в abc.txt
os.rename("xyz.txt","abc.txt")
```

Проверяем существование файла с помощью функции exists()

Для того чтобы убедиться, что файл или каталог действительно существуют, можно воспользоваться функцией exists(), передав ей относительное или абсолютное имя файла, как показано здесь:

```
import os
os.path.exists("abc.txt")
```

Выход:

True

Копируем файлы с помощью функции copy()

Функция copy() находится в другом модуле, shutil.

Модуль shutil

Этот встроенный модуль shutil в Python предоставляет множество функций для выполнения высокоуровневых операций с файлами и коллекциями файлов, который поставляется в процессе установки языка программирования Python.

Модуль shutil (от англ, “shell utilities” — утилиты командной оболочки) содержит функции, позволяющие копировать, перемещать, переименовывать и удалять файлы с помощью программ на Python. Прежде чем использовать эти утилиты, необходимо выполнить инструкцию import shutil.

shutil.copy()

shutil.copy(исходный_путь, путь__назначения) – функция приведет к копированию файла, расположение которого определяется путем исходный_путь, в папку, определяемую путем путь_назначения. (Параметры исходный__путь и путь_назначения являются строками.) Если аргумент путь_назначения — это имя файла, то оно будет использовано в качестве нового имени скопированного файла. Эта функция возвращает строку, содержащую путь к скопированному файлу.

В этом примере файл abcd.txt копируется в файл efgh.txt:

```
import shutil
shutil.copy(" abcd.txt", "efgh.txt")
```

shutil.move()

shutil.move() - Функция копирует файл, а затем удаляет оригинал.

Вызов функции shutil .move [исходный^путь, путь_назначения) перемещает файл или папку из расположения исходный_луть в расположение путь_назначения и возвращает строку, представляющую абсолютный путь к новому расположению.

Если параметр путь_назначения представляет папку, исходный файл перемещается в папку назначения и сохраняет свое текущее имя

```
import shutil
shutil.move("abcd.txt", "efgh.txt")
```

Если в папке назначения уже существует файл efgh.txt, то он будет заменен. Поскольку таким образом можно очень легко случайно потерять нужный файл, то надо должным проявлять определенную осторожность, когда используется функцию move ().

shutil.copytree()

В то время как функция shutil.copy() копирует одиночный файл, функция shutil.copytree() копирует папку вместе со всеми папками и файлами, которые в ней содержатся. В результате вызова shutil.copytree(исходный_ путь, путь_назначения) папка, находящаяся в расположении исходный_ путь, копируется вместе со всеми находящимися в ней файлами и подпапками в расположение путь_назначения. Параметры исходный_путь и путь_назначения являются строками. Функция возвращает строку, представляющую путь к копии папки.

Пример:

```
import shutil, os
os.chdir ('C:\\')
shutil.copytree('C:\\newdir', 'C:\\newdir_backup')
```

Выход:

```
'C:\\newdir_backup'
```

В результате вызова shutil.copytree() создается новая папка newdir_backup с таким же содержимым, как и содержимое исходной папки newdir. Теперь у нас есть резервная копия папки newdir. Процедура – очень полезная для сохранения данных.

Безвозвратное удаление файлов и папок

Если для удаления одиночного файла или одиночной пустой папки можно воспользоваться функциями, содержащимися в модуле os, то для удаления папки вместе со всем ее содержимым следует использовать модуль shutil.

Вызов os.unlink (путь) удаляет файл, расположенный по указанному пути.

Вызов os. mdir (путь) удаляет папку, расположенную по указанному пути. Эта папка должна быть пуста, т.е. не должна содержать никаких других папок и файлов.

shutil.rmtree()

Функция shutil.rmtree (путь) удаляет папку, расположенную по указанному пути, вместе со всеми содержащимися в ней другими папками и файлами.

Используя эти функции, надо соблюдать осторожность, поскольку удаление идет окончательное, минуя корзину!

send2trash

Сохранение резервных копий удаленных файлов и папок с помощью модуля send2trash.

Поскольку встроенная функция Python shutil.rmtree() необратимо удаляет файлы и папки, ее использование связано с определенным риском. Намного лучший способ удаления файлов и папок предлагает модуль send2trash от независимых разработчиков. Этот модуль можно установить, выполнив команду pip install send2trash в окне терминала.

Модуль send2trash намного безопаснее в использовании, чем обычные функции Python, выполняющие операцию удаления, поскольку он отправляет удаляемые файлы и папки в корзину компьютера или в специальную корзину, а не удаляет их безвозвратно. Если из-за ошибок в программе будут удалены файлы, которые не собирались удалять, но при этом использовался

модуль send2trash, то впоследствии будет возможность восстановить их из специальной корзины.

Однако, несмотря на то, что отправка файлов в специальную корзину оставляет возможность их последующего восстановления, размер свободного дискового пространства при этом не увеличивается, как в случае безвозвратного удаления файлов. Если возникает потребность в освобождении дискового пространства, то используются для удаления файлов и папок функции модулей os и shutil. Функция send2trash () может лишь отправлять файлы в корзину, но не извлекать их из нее.

Резюме

Даже для опытного пользователя, современные файловые менеджеры упрощают работу с небольшим количеством файлов. Но иногда возникают задачи, для выполнения которых с помощью проводника потребуется несколько часов работы.

Модули os и shutil предлагают функции, позволяющие осуществлять копирование, перемещение, переименование и удаление файлов. В следующем разделе, я хотел бы привести код для генерации каталогов в количестве 10 штук.

Генерация каталогов с помощью программных средств языка Python.

В качестве примера, показывающего преимущество программирования при выполнении рутинных и многочисленных операций, перед обычными ручными методами, которые предоставляют файловые менеджеры, я хотел бы продемонстрировать код, с помощью которого можно сгенерировать в автоматическом режиме серию каталогов (в количестве 10 штук).

Задача заключается в том, чтобы создать скрипт, который создает 10 папок в каталоге Temp с именами ID1, ID2, ID3, …, ID10. Папка Temp выбрана не случайно, поскольку в ней хранятся всякие промежуточные файлы, которые участвовали в установке программ на компьютер. И совершенно безболезненно можно (лучше сказать - нужно) удалить все содержимое папки Temp. Кстати, при очистке системы, сама система, в целях освобождения жесткого диска, предлагает удалить все содержимое этой папки.

Пример скрипта:

```
import os
path = 'C:\\Temp'
for i in range (1,11):
        os.chdir(path)
        Newfolder =  'ID' +  str(i)
        os.makedirs(Newfolder)
```

В первой строке – импортируем соответствующий модуль, далее во второй – присваиваем переменной path строковое значение 'C:\\Temp'. Это - путь, характеризующий расположение папки Temp. Третья строка, это начало цикла for. Внутри скобок находятся диапазоны: начальный номер элемента цикла и через запятую – конечный. В конце строки с ключевым словом for, обязательно должно стоять двоеточие. Следующая строка, с обязательным отступом в 4 пробела или отступа в один знак табуляции (должно быть что-то одно), это – начало цикла.

os.chdir(path) – функция смены рабочей директории на ту, что указана в скобках (т е – это аргументы) в виде переменной path. Эту переменную мы раньше определили, как строковую, содержащую путь до папки Temp. Значит, после выполнения этой инструкции, наша рабочая директория будет Temp.

Следующая строка цикла – Newfolder = 'ID'+ str(i)

Здесь происходит в правой части сложение строковых переменных (правильнее сказать конкатенация) – просто присоединение выражения ID с

порядковым номером цикла, который автоматически превращается в строковое значение. Затем, значение правой части присваиваем к левой.
Последняя строка цикла os.makedirs(Newfolder).
Здесь происходит внутри цикла создание новой папки с именем, которое содержит в себе переменная Newfolder. Как известно из предыдущих инструкций, эта переменная – строковая и меняется с каждым циклом на одну единицу. Цикл будет осуществлен 10 раз, то есть пока не будет выбран весь диапазон от минимального значения до максимального.

Выводы по работе.

В этой работе были рассмотрены программные методы работы с файловой системой и самими файлами. Работоспособность всех рассматриваемых примеров была успешно протестирована на различных версиях ОС Windows.

- Эти методы являются эффективными при работе с большим количеством файлов и папок: или на одном компьютере надо произвести множество операций, или же на множестве других компьютерах.
- Скрипты обладают определенной гибкостью: а именно, есть возможностью настройки на те данные, что есть на данном предприятии.
- Отработав соответствующие скрипты один раз, и убедившись, что они работают корректно, можно сберечь массу времени для пользователей, которые осуществляют операции по работе с файлами и папками.
- Скрипты можно упаковать в исполняемый файл, что значительно упрощает работу с ними и не требует загрузки на компьютеры, где они будут применяться, программной среды, в которой они были созданы.
- Кроме того, эти разработанные скрипты позволяют предохранить от возможных ошибок, которые свойственны человеку.

К недостаткам этих методов можно отнести то, что при несущественном количестве операций, нет смысла разрабатывать эти скрипты – быстрее осуществить все тоже самое традиционными методами средствами файловых менеджеров. Но можно написать эти скрипты в исключительно учебных целях.

Я надеюсь, что этой работой, я смогу внести свой вклад в автоматизацию процессов настройки компьютеров для студентов в своем Университете, где я работаю с 2011 года.

Список используемой литературы.

- Билл Любанович - Простой Python. Современный стиль программирования-Питер (2016)
- Эл Свейгарт - Автоматизация рутинных задач с помощью Python. Практическое руководство для начинающих-Вильямс (2017)
- Марк Лутц Изучаем Python (Том 1, 5-е издание) (2019)