Стандартная библиотека в Python.

filecmp

Модуль filecmp предоставляет функции, которые могут использоваться для сравнивания файлов и каталогов:

– cmp(file1, file2 [, shallow])

Сравнивает файлы file1 и file2 и возвращает True, если они равны, в противном случае возвращает False. По умолчанию равными считаются файлы, для которых функция os.stat() возвращает одинаковые значения атрибутов. Если в аргументе shallow функции передается значение False, дополнительно сравнивается содержимое файлов.

– cmpfiles(dir1, dir2, common [, shallow])

Сравнивает файлы, перечисленные в списке common, в двух каталогах dir1 и dir2. Возвращает кортеж с тремя списками имен файлов (match, mismatch, errors). В списке match перечислены одинаковые файлы, в списке mismatch – отличающиеся файлы и в списке errors – файлы, сравнивание которых не может быть выполнено по каким-либо причинам. Аргумент shallow имеет тот же смысл, что и в функции cmp().

dircmp(dir1, dir2 [, ignore [, hide]]) Создает объект сравнивания каталогов, который может использоваться для выполнения различных операций сравнивания каталогов dir1 и dir2. В аргументе ignore передается список имен, которые следует исключить из операции сравнивания, и который по умолчанию содержит значения [‘RCS’,’CVS’,’tags’]. В аргументе hide передается список имен для сокрытия; он по умолчанию содержит значения [os.curdir, os.pardir] ([‘.’, ‘..’] в UNIX).

Объект сравнивания каталогов d, возвращаемый функцией dircmp(), обладает следующими методами и атрибутами:

1. d.report() Сравнивает каталоги dir1 и dir2 и выводит отчет в поток sys.stdout. d.report_partial_closure() Сравнивает каталоги dir1 и dir2, а также общие подкаталоги следующего уровня. Результаты выводятся в поток sys.stdout.
2. d.report_full_closure() Сравнивает каталоги dir1 и dir2, а также все вложенные подкаталоги, рекурсивно. Результаты выводятся в поток sys.stdout. d.left_list Список всех файлов и подкаталогов в каталоге dir1. Содержимое списка фильтруется в соответствии с содержимым списков hide и ignore.

3. d.right_list Список всех файлов и подкаталогов в каталоге dir2. Содержимое списка фильтруется в соответствии с содержимым списков hide и ignore.
4. d.common Список всех файлов и подкаталогов, найденных в обоих каталогах dir1 и dir2.
5. d.left_only Список всех файлов и подкаталогов, найденных в каталоге dir1.
6. d.right_only Список всех файлов и подкаталогов, найденных в каталоге dir2.
7. d.common_dirs Список подкаталогов, общих для каталогов dir1 и dir2.
8. d.common_files Список файлов, общих для каталогов dir1 и dir2.
9. d.common_funny Список файлов в каталогах dir1 и dir2 с одинаковыми именами, которые имеют разные типы или для которых невозможно получить информацию с помощью функции os.stat().
10. d.same_files Список файлов с идентичным содержимым в каталогах dir1 и dir2.
11. d.diff_files Список файлов с разным содержимым в каталогах dir1 и dir2.
12. d.funny_files Список файлов, присутствующих в обоих каталогах dir1 и dir2, сравнивание которых не может быть выполнено по каким-либо причинам (например, из-за отсутствия прав доступа).
13. d.subdirs Словарь, отображающий имена в d.common_dirs в дополнительные объекты dircmp.

time

Модуль time предоставляет различные функции для работы со временем. Python время измеряется количеством секунд, прошедших с начала эпохи. Начало эпохи соответствует началу исчисления времени (моменту, когда время было равно 0 секунд). В UNIX началу эпохи соответствует дата 1 января 1970 года, а в других системах может быть определено вызовом функции time.gmtime(0).

Ниже перечислены переменные, объявленные в модуле:

1. accept2dyear Логическое значение, указывающее – допускается ли использовать двузначное представление года. По умолчанию эта переменная имеет значение True, но она получит значение False, если в переменную окружения $PYTHONY2K записать непустую строку. Кроме того, значение переменной accept2dyear может изменяться вручную.

2. altzone Часовой пояс, используемый после перехода на летнее время (DST), если применимо.
3. daylight Имеет ненулевое значение, если определен часовой пояс, используемый после перехода на летнее время.
4. timezone Локальный (не летний) часовой пояс. 508 Глава 19. Службы операционной системы
5. tzname Кортеж с названиями локального часового пояса и зимнего/летнего (если определен) часового пояса.

В модуле time также определены следующие функции:

1. asctime([tuple]) Преобразует кортеж, представляющий время, возвращаемое функцией gmtime() или localtime(), в строку вида 'Mon Jul 12 14:45:23 1999'. При вызове без аргумента возвращает представление текущего времени.
2. clock() Возвращает текущее процессорное время в секундах в виде числа с плавающей точкой.
3. ctime([secs]) Преобразует время secs, выраженное в секундах от начала эпохи, в строку, представляющую локальное время. Вызов ctime(secs) эквивалентен вызоэквивалентен вызову asctime(localtime(secs)). При вызове без аргумента или со значением None в аргументе возвращает представление текущего времени. gmtime([secs]) Принимает время secs, выраженное в секундах от начала эпохи по Гринвичскому времени (Coordinated Universal Time, UTC), и возвращает объект типа struct_time, обладающий следующими атрибутами:

Атрибут	Значение
tm_year	Четырехзначный год, например 1998
tm_mon	1-12
tm_mday	1-31
tm_hour	0-23
tm_min	0-59
tm_sec	0-61
tm_wday	0-6 (0 = понедельник)
tm_yday	1-366
tm_isdst	-1, 0, 1

4. localtime([secs]) Возвращает объект типа struct_time, как и функция gmtime(), информация в котором соответствует локальному часовому поясу. При вызове без аргумента или когда в аргументе secs передается значение None, используется текущее время.
5. mktime(tuple) Принимает объект типа struct_time или кортеж, представляющий время для локального часового пояса (в том же формате, в каком оно возвращается функцией localtime()), и

возвращает число с плавающей точкой, представляющее количество секунд, прошедших от начала эпохи. Если в аргументе tuple передается недопустимое значение, возбуждает исключение OverflowError.

6. sleep(secs) Приостанавливает работу текущего процесса на secs секунд. В аргументе secs передается число с плавающей точкой.
7. time() Возвращает текущее время по Гринвичу (Coordinated Universal Time, UTC) в виде количества секунд, прошедших от начала эпохи.

Точность представления времени в функциях может оказаться хуже, чем можно было бы предположить, исходя из единиц представления, с которыми они работают. Например, операционная система может обновлять внутренние часы всего 50-100 раз в секунду.

sched

Модуль sched предоставляет класс scheduler, который реализует планируемые события.

sched.scheduler(timefunc = time.monotonic, delayfunc = time.sleep)

Определяет интерфейс к планируемым событиям. Принимает две функции:

– timefunc, возвращающую время в любых единицах. По умолчанию используется функция time.monotonic, а если она недоступна, то time.time

– delayfunc, определяющую время задержки. Время задержки определяется передаваемым аргументом.

```
>>> import sched, time
>>> s = sched.scheduler(time.time, time.sleep)
>>> def print_time(a='default'):
...     print("From print_time", time.time(), a)
...
>>> def print_some_times():
...     print(time.time())
...     s.enter(10, 1, print_time)
...     s.enter(5, 2, print_time, argument=('positional',))
...     s.enter(5, 1, print_time, kwargs={'a': 'keyword'})
...     s.run()
...     print(time.time())
...
>>> print_some_times()
930343690.257
From print_time 930343695.274 positional
From print_time 930343695.275 keyword
From print_time 930343700.273 default
930343700.276
```

Экземпляры класса scheduler имеют следующие функции и атрибуты:

– scheduler.enterabs(*time*, *priority*, *action*, *argument=()*, *kwargs={}*)

Добавляет новую задачу в план. Аргумент *time* должен быть числовым значением, совместимым с результатом *timefunc*. События, планируемые на одно и то же время *time* будут выполнены согласно их приоритетам *priority*. Чем меньше значение, тем выше приоритет.

– action(*argument, **kwargs), выполняет некоторое действие события. *argument* это последовательность, содержащая именованные аргументы для *action.kwargs*, который является словарём, содежащим аргументы для метода *action*.

– scheduler.enter(*delay*, *priority*, *action*, *argument=()*, *kwargs={}*)

Добавляет на исполнение новое событие с задержкой *delay*.

– scheduler.cancel(*event*)

Удаляет событие с очереди на выполнение. Если событие *event* не присутствует в очереди, то будет возбуждено исключение ValueError.

– scheduler.empty()

Возвращает True, если очередь на выполнение пуста

– scheduler.run(*blocking=True*)

Запускает все запланированные события. Будет ожидания новые события в течении времени, заданного delayfunc.

Если *blocking* = False выполняет событие по истечению активного (если есть) и возвращает время завершения следующего запланированного вызова события в очереди (если есть).

Как *action*, так и *delayfunc* могутвозбудить исключение. Как бы то ни было, scheduler будет продолжать работу, а также будет передавать исключение далее. Если же исключение было возбуждено *action*, событие не будет дальше вызывать метод run().

Если время на выполнения последовательности событий больше, чем время, необходимое на поступление нового события то планировщик scheduler будет отставать. Никакие события не будут исключены из очереди; ответственность за исключение неактуальных событий лежит на вызывающем коде.

– scheduler.queue

Read-only атрибут возвращающий список предстоящих событий в порядке их выполнения. Каждое событие отображено именованным кортежем со следующими полями: time, priority, action, argument, kwargs.

getpass

Модуль getpass предоставляет две функции:

– getpass.getpass(*prompt='Password: '*, *stream=None*). Приглашение на ввод пароля без эхо. В качестве приглашения используется строка *prompt*. Приглашение выводится в поток stream.

Если вывод без эха невозможен, то функция выводит предупреждение в *stream*, и читает из sys.stdin используя функцию с эхом GetPassWarning.

– getpass.getpass() возвращает логин пользователя. Функция проверяет переменные среды LONGNAME, USER, LNAME и USERNAME в указанном порядке и возвращает значение первого не равного пустой строке. Если таковых нет, то логин берётся из базы данных, если система поддерживает модуль pwd, иначе выбрасывается исключение.

getopt

Модуль getopt это парсер команды командной строки. API модуля разработано с елью быть похожим на getopt() из C. Этот модуль используется с целью разбора аргументов командной строки в sys.argv. Поддерживаются такие же конвенции, как в функции Unix getopt(), например, специальное значение аргументов вида «-»(ключи) и «--» (встроенные команды).

Функции модуля:

– getopt.getopt(args, shortopts, longopts = [])

Разбирает команду и опции (параметры) командной строки. Args – это список аргументов, который будет распознан без ссылок на текущую программу. Обычно, это означает что shortopts это строка опций, которя требуется скрипту. Аргументы указываются с последующим знаком ‘:’. Longopts, если присутствует, должен быть списком строк с именами длинных опций. Символы – не должны быть включены в строку-имя. Аргументы указываются с последующим знаком ‘=’. Необязательные аргументы не поддерживаются. Чтобы исключить аргументы с короткими именами, shortopts должен быть пустой строкой. Длинные операции распознаются, пока они являются префиксом лишь для одной строки из longopts, в противном случае будет возбуждено исключение GetoptError. Например, пусть longopts = [‘foo’, ’frob’], тогда опция –fo будет распознана, так как она является префиксом строки ‘foo’. В то же время опция ‘--f’ распознана не будет, так она является префиксом как ‘foo’, так и ’frob’.

Функция возвращает два значения, первое – список из кортежей (option, value), второе – это список аргументов, оставшихся после опции, по которой было произведено разделение. Каждая опция из пары «опция – значение» имеет строку, представляющую опцию, с возможным префиксом из одного или двух дефисов и аргумент. Если аргумента нет, то строка option пустая. Порядок опций в списке совпадает с порядком появления их в команде.

– getopt.gnu_getopt(args, shortopts, longopts=[]) работает аналогично предыдущей функции, кроме того, что по умолчанию используется режим сканирования GNU, что означает, среди прочего, что обязательные и необязательные опции могут быть перемешаны. Если первый символ строки опций ‘+’ или если переменная среды POSIXLY_CORRECT установлена, то разбор опций будет остановлен как только будет встречена обязательная опция.

Параметры в Unix.

```
>>> import getopt
>>> args = '-a -b -cfoo -d bar a1 a2'.split()
>>> args
['-a', '-b', '-cfoo', '-d', 'bar', 'a1', 'a2']
>>> optlist, args = getopt.getopt(args, 'abc:d:')
>>> optlist
[('-a', ''), ('-b', ''), ('-c', 'foo'), ('-d', 'bar')]
>>> args
['a1', 'a2']
```

Длинные опции

```
>>> s = '--condition=foo --testing --output-file abc.def -x a1 a2'
>>> args = s.split()
>>> args
['--condition=foo', '--testing', '--output-file', 'abc.def', '-x', 'a1', 'a2']
>>> optlist, args = getopt.getopt(args, 'x', [
...     'condition=', 'output-file=', 'testing'])
>>> optlist
[('--condition', 'foo'), ('--testing', ''), ('--output-file', 'abc.def'), ('-x', '')]
>>> args
['a1', 'a2']
```

Пример использования

```
import getopt, sys

def main():
    try:
        opts, args = getopt.getopt(sys.argv[1:], "ho:v", ["help", "output="])
    except getopt.GetoptError as err:
        # print help information and exit:
        print(err)  # will print something like "option -a not recognized"
        usage()
        sys.exit(2)
    output = None
    verbose = False
    for o, a in opts:
        if o == "-v":
            verbose = True
        elif o in ("-h", "--help"):
            usage()
            sys.exit()
        elif o in ("-o", "--output"):
            output = a
        else:
            assert False, "unhandled option"
    # ...

if __name__ == "__main__":
    main()
```

Tempfile

Используется для создания временных файлов и директорий.

```
>>> import tempfile

# create a temporary file and write some data to it
>>> fp = tempfile.TemporaryFile()
>>> fp.write(b'Hello world!')
# read data from file
>>> fp.seek(0)
>>> fp.read()
b'Hello world!'
# close the file, it will be removed
>>> fp.close()

# create a temporary file using a context manager
>>> with tempfile.TemporaryFile() as fp:
...     fp.write(b'Hello world!')
...     fp.seek(0)
...     fp.read()
b'Hello world!'
>>>
# file is now closed and removed

# create a temporary directory using the context manager
>>> with tempfile.TemporaryDirectory() as tmpdirname:
...     print('created temporary directory', tmpdirname)
>>>
# directory and contents have been removed
```

shutil

Модуль shutil используется для выполнения таких операций высокого уровня над файлами, как копирование, удаление и переименование. Функции из этого модуля должны применяться только к обычным файлам и каталогам. В частности, они не могут работать со специальными файлами, такими как именованные каналы, блочные устройства и так далее. Кроме того, эти функции не всегда корректно обрабатывают некоторые дополнительные типы метаданных (например, ответвления ресурсов, коды создателей и другие).

Функции, представленные в модуле:

1. copy(src,dst) Копирует файл src в файл или каталог dst, с сохранением прав доступа. Значения аргументов src и dst должны быть строками.
2. copy2(src, dst) Действует аналогично функции copy(), но дополнительно копирует время последнего обращения и время последнего изменения.
3. copyfile(src, dst) Копирует содержимое файла src в файл dst. Значения аргументов src и dst должны быть строками.
4. dst. copystat(src, dst) Копирует биты разрешений, время последнего обращения и время последнего изменения из файла

src в файл dst. Содержимое файла dst, владелец и группа остаются без изменений.

5. copytree(src, dst, symlinks [,ignore]]) Рекурсивно копирует дерево каталогов с корнем в каталоге src. Каталог назначения dst не должен существовать (он будет создан). Копирование файлов выполняется с помощью функции copy2(). Если в аргументе symlinks передается истинное значение, символические ссылки в исходном дереве каталогов будут представлены символическими ссылками в новом дереве. Если в аргументе symlinks передается ложное значение или он опущен, в новое дерево каталогов будет скопировано содержимое файлов, на которые указывают символические ссылки. В необязательном аргументе ignore передается функция, которая будет использоваться для фильтрации файлов. Эта функция должна принимать имя каталога и список его содержимого и возвращать список имен файлов, которые не должны копироваться. Если в процессе копирования будут происходить какие-либо ошибки, они будут собраны все вместе и в конце будет возбуждено исключение Error. В качестве аргумента исключению будет передан список кортежей (srcname, dstname, exception), по одному для каждой возникшей ошибки.
6. ignore_patterns(pattern1, pattern2, ...) Создает функцию, которая может использоваться для исключения из операции имен, соответствующих шаблонам pattern1, pattern2 и так далее. Возвращаемая функция принимает два аргумента. В первом она принимает имя каталога, а во втором – список содержимого этого каталога. В качестве результата она возвращает список имен файлов, которые должны быть исключены из операции. Обычно эта функция используется, как значение аргумента ignore при вызове функции copytree(). Однако она также может использоваться в операциях с привлечением функции os.walk().
7. move(src, dst) Перемещает файл или каталог src в dst. Если src – это каталог и он перемещается в другую файловую систему, выполняется рекурсивное копирование его содержимого.
8. rmtree(path [, ignore_errors [, onerror]]) Удаляет дерево каталогов целиком. Если в аргументе ignore_errors передается истинное значение, ошибки, возникающие при удалении, будут игнорироваться. В противном случае для обработки ошибок будет вызываться функция, переданная в аргументе onerror. Эта функция должна принимать три аргумента (func, path и excinfo), где func – функция, которая вызвала ошибку (os.remove() или os.rmdir()), path – путь, который был передан функции, и excinfo –

информация об исключении, полученная вызовом функции sys.exc_info(). Если аргумент onerror опущен, появление ошибок будет вызывать исключение.

nmap

Используется для работы с сканнером портов nmap.