МИНОБРНАУКИ РОССИИ

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Казанский национальный исследовательский технологический университет»**

**(ФГБОУ ВО «КНИТУ»)**

РЕФЕРАТ

на тему:

# «Модуль Os.path»

                                    Выполнил:

студент гр.103201-с82

Семенов Алексей

Казань, 2022

Оглавление

[«Модуль Os.path» 1](#_Toc122278240)

[**Общие манипуляции с путями файловой системы.** 3](#_Toc122278241)

[**os.path** 3](#_Toc122278242)

[**os.path.basename** 4](#_Toc122278243)

[**os.path.dirname** 4](#_Toc122278244)

[**os.path.exists** 5](#_Toc122278245)

[**os.path.isdir / os.path.isfile** 5](#_Toc122278246)

[**os.path.join** 6](#_Toc122278247)

[**os.path.split** 6](#_Toc122278248)

[**Подведем итоги** 7](#_Toc122278249)

**Модуль OS**

Модуль *OS* позволяет работать с файловой системой, с окружением, управлять процессами.

В этом подразделе рассматриваются лишь несколько полезных возможностей.

Разные операционные системы (ОС) имеют разные соглашения об именах путей, поэтому в стандартной библиотеке есть несколько версий модуля os.path. Модуль os автоматически подгружает нужную часть для работы с текущей ОС. Например, при запуске одних и тех же функций модуля os на Windows и Linux, разделителем пути будут считаться разные значения.

При необходимости работы на Linux с путями Windows и наоборот, можно использовать модули posixpath, ntpath вместо os.path.

## **Общие манипуляции с путями файловой системы.**

Модуль os-path реализует некоторые полезные функции для манипулирования путями файловой системы. Параметры пути могут быть переданы в виде строк или байтов. Приложениям рекомендуется представлять имена файлов в виде (Unicode) символьных строк. К сожалению, некоторые имена файлов могут не представляться в виде строк в Unix, поэтому приложения, которым необходимо поддерживать произвольные имена файлов в Unix, должны использовать байтовые объекты для представления имен путей. И наоборот, использование байтовых объектов не может представлять все имена файлов в Windows в стандартной кодировке mbcs, поэтому приложениям Windows следует использовать строковые объекты для доступа ко всем файлам.

## 

## **os.path**

Модуль os.path — это часто и широко используемая функция, применяемая, когда мы хотим обрабатывать файлы из разных областей в системе. Основная функциональность этого модуля включает слияние, нормализацию и получение имен путей в python. Поскольку эта функция полезна при применении функции к путям, параметры путей передаются либо в форме строк, либо в байтах. Недостаточно используемым приложениям необходимо было представлять имена файлов в виде строк символов Unicode. Результирующее значение функции находится в той же форме, либо путь к папке, либо сама папка / имя файла.

Подмодуль **os.path** модуля os имеет широкий ряд встроенных преимуществ. Ознакомимся со следующими функциями:

* basename
* dirname
* exists
* isdir and isfile
* join
* split

## **os.path.basename**

Функция **basename** вернет **название файла пути**. Пример:

Python

|  |  |
| --- | --- |
| 1  2  3  4 | **import** os    os.path.basename(r'C:\Python27\Tools\pynche\ChipViewer.py')  *# ChipViewer.py* |

Это очень полезная функция, особенно в тех случаях, когда нужно **использовать имя файла** для наименования того или иного связанного с работой файла, например лог-файл. Такая ситуация возникает часто при работе с файлами данных.

## 

## **os.path.dirname**

Функция **dirname** возвращает только **часть каталога пути**. Это проще понять, если мы взглянем на пример кода:

Python

|  |  |
| --- | --- |
| 1  2  3  4 | **import** os    print( os.path.dirname(r'C:\Python27\Tools\pynche\ChipViewer.py') )  *# C:\\Python27\\Tools\\pynche* |

В данном примере мы просто **возвращаем путь к каталогу**. Это также полезно, когда вам нужно сохранить другие файлы рядом с тем, который вы обрабатываете в данный момент. Как и в случае с лог-файлом, упомянутым выше.

## **os.path.exists**

Функция **exists** говорит нам, существует ли файл, или нет. Все что вам нужно, это указать ему путь. Взглянем на пример:

Python

|  |  |
| --- | --- |
| 1  2  3  4  5 | **import** os    os.path.exists(r'C:\Python27\Tools\pynche\ChipViewer.py') *# True*    os.path.exists(r'C:\Python27\Tools\pynche\fake.py') *# False* |

В первом примере, мы указали **функции exists** настоящий путь, на что она указывает как **True**. Это говорит о том, что данный путь существует. Во втором примере, мы указали неправильный путь, от чего функция указывает нам на это сообщением **False**.

## **os.path.isdir / os.path.isfile**

Методы **isdir** и **isfile** тесно связаны с **методом exists**, так как они также тестируют присутствие или отсутствие файлов или папок на тех или иных путях. Однако, **isdir** проверяет только **пути к папкам**, а **isfile**, соответственно, к файлам. Если вам нужно проверить путь, и не важно, папка это или файл, проще будет воспользоваться **методом exists**. В любом случае, взглянем на пару примеров:

Python

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **import** os    os.path.isfile(r'C:\Python27\Tools\pynche\ChipViewer.py') *# True*    os.path.isdir(r'C:\Python27\Tools\pynche\ChipViewer.py') *# False*    os.path.isdir(r'C:\Python27\Tools\pynche') *# True*    os.path.isfile(r'C:\Python27\Tools\pynche') *# False* |

Уделите особое внимание данным примерам. В первом мы **указали путь к файлу** и проверили, является ли этот путь в действительности файлом. Затем, во втором примере, мы проделали то же самое, но в контексте папки. Вы можете лично ознакомиться с результатами. После этих двух примеров, мы немного изменили условия, указав путь к папке для обеих функций. Эти примеры наглядно демонстрируют то, как эти функции работают.

## **os.path.join**

**Метод join** позволяет вам совместить несколько путей при помощи присвоенного разделителя. К примеру, в Windows, в роли разделителя выступает бэкслэш (косая черта, указывающая назад), однако в Linux функция разделителя присвоена косой черте, указывающей вперед (forward slash). Как это работает:

Python

|  |  |
| --- | --- |
| 1  2  3  4 | **import** os    print( os.path.join(r'C:\Python27\Tools\pynche', 'ChipViewer.py') )  *# C:\\Python27\\Tools\\pynche\\ChipViewer.py* |

В данном примере мы **совместили путь каталога и файла вместе**, для получения рабочего пути. Обратите внимание на то, что **метод join** не указывает на то, какой результат в итоге вышел.

## **os.path.split**

Метод **split** разъединяет путь на кортеж, который содержит и файл и каталог. Взглянем на пример:

Python

|  |  |
| --- | --- |
| 1  2  3  4 | **import** os    print( os.path.split(r'C:\Python27\Tools\pynche\ChipViewer.py') )  *# ('C:\\Python27\\Tools\\pynche', 'ChipViewer.py')* |

В данном примере показано, что происходит, когда мы указываем путь к файлу. Теперь взглянем на то, что происходит, если в конце пути **нет названия файла**:

Python

|  |  |
| --- | --- |
| 1  2  3  4 | **import** os    print( os.path.split(r'C:\Python27\Tools\pynche') )  *# (‘C:\Python27\Tools’, ‘pynche’)* |

Как видите, данная функция **берет путь и разъединяет его** таким образом, что подпапка стала вторым элементом кортежа с остальной частью пути в первом элементе. Напоследок, взглянем на бытовой случай **использования split**:

Python

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | **import** os    dirname, fname = os.path.split(r'C:\Python27\Tools\pynche\ChipViewer.py')  print(dirname)  *# C:\\Python27\\Tools\\pynche*    print(fname)  *# ChipViewer.py* |

В данном примере указано, как сделать множественное назначение. Когда вы разъединяете путь, он становится кортежем, состоящим из двух частей. После того, как мы опробовали две переменные с левой части, первый элемент кортежа назначен к первой переменной, а второй элемент к второй переменной соответственно

## **Подведем итоги**

С данного момента **модуль os** можно смело назвать старым добрым знакомым. В данном разделе мы научились:

1. Работать со значениями среды;
2. Менять каталоги и находить тот, который в данный момент является рабочим;
3. Создавать и удалять папки и файлы;
4. Переименовывать папки и файлы;
5. Открывать файл в соответствующей ему программе;
6. Работать с путями.

И это **не полный список** того, что можно делать при помощи **модуля os**.