

Módulo 4 (Intermedio): Biblioteca os

Introducción al módulo os

En esta sección, aprenderás sobre un módulo llamado **os**, que te permite **interactuar con tu sistema operativo usando Python**.

Proporciona funciones que están disponibles en sistemas Unix y/o Windows. Si estás familiarizado con la consola de comandos, verás que algunas funciones dan los mismos resultados que los comandos disponibles en los sistemas operativos.

Un buen ejemplo de esto es la función *mkdir*, que te permite crear un directorio como el comando *mkdir* en Unix y Windows. Si no conoces este comando, no te preocupes.

Pronto tendrás la oportunidad de aprender las funciones del módulo *os*, para realizar operaciones en archivos y directorios junto con los comandos correspondientes.

Además de las operaciones de archivos y directorios, el módulo *os* te permite:

- Obtener información sobre el sistema operativo.
- Manejar procesos.
- Operar en streams de E/S usando descriptores de archivos.

En un momento, verás cómo obtener información básica sobre tu sistema operativo, aunque la administración de procesos y el trabajo con descriptores de archivos no se discutirán aquí, porque estos son temas más avanzados que requieren conocimiento de los mecanismos del sistema operativo.

Obtener información sobre el sistema operativo

Antes de crear tu primera estructura de directorios, verás cómo puedes obtener información sobre el sistema operativo actual. Esto es realmente fácil porque el módulo *os* proporciona una función llamada *uname*, que devuelve un objeto que contiene los siguientes atributos:

- **systemname**: almacena el nombre del sistema operativo.
- **nodename**: almacena el nombre de la máquina en la red.
- **release**: almacena el release (actualización) del sistema operativo.
- **version**: almacena la versión del sistema operativo.
- **machine**: almacena el identificador de hardware, por ejemplo, *x86_64*.

Veamos cómo es en la práctica:

```
import os
print(os.uname())
```

Resultado:

```
posix.uname_result(sysname='Linux', nodename='192d19f04766', release='4.4.0-164-generic', version='#192-Ubuntu SMP Fri Sep 13 12:02:50 UTC 2019', machine='x86_64')
```

Como puedes ver, la función *uname* devuelve un objeto que contiene información sobre el sistema operativo. El código anterior se ejecutó en Ubuntu 16.04.6 LTS, así que no te sorprendas si obtienes un resultado diferente, porque depende de tu sistema operativo.

Desafortunadamente, la función `uname` solo funciona en algunos sistemas Unix. Si usas Windows, puede usar la función `uname` en el módulo `platform`, que devuelve un resultado similar.

El módulo `os` te permite distinguir rápidamente el sistema operativo mediante el atributo `name`, que soporta uno de los siguientes nombres:

- **posix**: obtendrás este nombre si usas Unix.
- **nt**: obtendrás este nombre si usas Windows.
- **java**: obtendrás este nombre si tu código está escrito en Jython.

Para Ubuntu 16.04.6 LTS, el atributo `name` devuelve el nombre `posix`:

```
import os
print(os.name)
```

Resultado:

```
posix
```

NOTA: En los sistemas Unix, hay un comando llamado `uname` que devuelve la misma información (si lo ejecutas con la opción `-a`) que la función `uname`.

Creando directorios en Python

El módulo `os` proporciona una función llamada `mkdir`, la cual, como el comando `mkdir` en Unix y Windows, te permite crear un directorio. La función `mkdir` requiere una ruta que puede ser relativa o absoluta. Recordemos cómo se ven ambas rutas en la práctica:

- **my_first_directory**: esta es una ruta relativa que creará el directorio `my_first_directory` en el directorio de trabajo actual.
- **./my_first_directory**: esta es una ruta relativa que apunta explícitamente al directorio de trabajo actual. Tiene el mismo efecto que la ruta anterior.
- **../my_first_directory**: esta es una ruta relativa que creará el directorio `my_first_directory` en el directorio superior del directorio de trabajo actual.
- **/python/my_first_directory**: esta es una ruta absoluta que creará el directorio `my_first_directory`, que a su vez está en el directorio raíz de python.

Observa el código en el editor. Muestra un ejemplo de cómo crear el directorio `my_first_directory` usando una ruta relativa. Esta es la variante más simple de la ruta relativa, que consiste en pasar solo el nombre del directorio.

Si pruebas tu código aquí, generará el directorio recién creado `['my_first_directory']` (y todo el contenido del catálogo de trabajo actual).

La función `mkdir` crea un directorio en la ruta especificada. Ten en cuenta que ejecutar el programa dos veces generará un `FileExistsError`.

Esto significa que no podemos crear un directorio si ya existe. Además del argumento de la ruta, la función `mkdir` puede tomar opcionalmente el argumento `mode`, que especifica los permisos del directorio. Sin embargo, en algunos sistemas, el argumento `mode` se ignora.

Para cambiar los permisos del directorio, recomendamos la función `chmod`, que funciona de manera similar al comando `chmod` en sistemas Unix. Puedes encontrar más información al respecto en la documentación.

En el ejemplo anterior, se usa otra función proporcionada por el módulo `os` llamada `listdir`. La función `listdir` devuelve una lista que contiene los nombres de los archivos y directorios que se encuentran en la ruta pasada

como argumento.

Si no se le pasa ningún argumento, se utilizará el directorio de trabajo actual (como en el ejemplo anterior). Es importante que el resultado de la función `listdir` omita las entradas `'.'` y `'..'`, que se muestran, por ejemplo, cuando se usa el comando `ls -a` en sistemas Unix.

NOTA: Tanto en Windows como en Unix, hay un comando llamado `mkdir`, que requiere una ruta de directorio. El equivalente del código anterior que crea el directorio `my_first_directory` es el comando `mkdir my_first_directory`.

Creación recursiva de directorios

La función `mkdir` es muy útil, pero ¿qué sucede si necesitas crear **otro directorio dentro del directorio** que acabas de crear? Por supuesto, puedes ir al directorio creado y crear otro directorio dentro de él, pero afortunadamente el módulo `os` proporciona una función llamada *makedirs*, que facilita esta tarea.

La función *makedirs* permite la creación recursiva de directorios, lo que significa que se crearán todos los directorios de la ruta.

```
import os

os.makedirs("my_first_directory/my_second_directory")
os.chdir("my_first_directory")
print(os.listdir())
```

El código debería producir el siguiente resultado:

```
['my_second_directory']
```

El código crea dos directorios. El primero de ellos se crea en el directorio de trabajo actual, mientras que el segundo en el directorio `my_first_directory`.

No tienes que ir al directorio `my_first_directory` para crear el directorio `my_second_directory`, porque la función *makedirs* hace esto por ti. En el ejemplo anterior, vamos al directorio `my_first_directory` para mostrar que el comando *makedirs* crea el subdirectorio `my_second_directory`.

Para moverte entre directorios, puedes usar una función llamada `chdir`, que cambia el directorio de trabajo actual a la ruta especificada. Como argumento, toma cualquier ruta relativa o absoluta. En nuestro ejemplo, le pasamos el nombre del primer directorio.

NOTA: El equivalente de la función *makedirs* en sistemas Unix es el comando `mkdir` con el indicador `-p`, mientras que en Windows, simplemente el comando `mkdir` con la ruta:

- Sistemas tipo Unix:

```
mkdir -p my_first_directory/my_second_directory
```

- Windows:

```
mkdir my_first_directory/my_second_directory
```

¿Dónde estoy ahora?

Ya sabes cómo crear directorios y cómo moverte entre ellos. A veces, cuando tienes una estructura de

directorio muy grande en la que navegas, es posible que no sepas en qué directorio estás trabajando actualmente.

Como probablemente habrás adivinado, el módulo `os` proporciona una función que devuelve información sobre el directorio de trabajo actual. Se llama *getcwd*.

```
import os

os.makedirs("my_first_directory/my_second_directory")
os.chdir("my_first_directory")
print(os.getcwd())
os.chdir("my_second_directory")
print(os.getcwd())
```

Resultado:

```
../my_first_directory
../my_first_directory/my_second_directory
```

En el ejemplo, creamos el directorio `my_first_directory` y el directorio `my_second_directory` dentro de él. En el siguiente paso, cambiamos el directorio de trabajo actual al directorio `my_first_directory` y luego mostramos el directorio de trabajo actual (primera línea del resultado).

A continuación, vamos al directorio `my_second_directory` y nuevamente mostramos el directorio de trabajo actual (segunda línea del resultado). Como puedes ver, la función *getcwd* devuelve la ruta absoluta a los directorios.

NOTA: En sistemas tipo Unix, el equivalente de la función *getcwd* es el comando *pwd*, que imprime el nombre del directorio de trabajo actual.

Eliminando directorios en Python

El módulo `os` también te permite eliminar directorios. Te da la opción de borrar un solo directorio o un directorio con sus subdirectorios. Para eliminar un solo directorio, puedes usar una función llamada *rmdir*, que toma la ruta como argumento.

```
import os

os.mkdir("my_first_directory")
print(os.listdir())
os.rmdir("my_first_directory")
print(os.listdir())
```

El ejemplo anterior es realmente simple. Primero, se crea el directorio `my_first_directory` y luego se elimina usando la función *rmdir*. La función *listdir* se utiliza como prueba de que el directorio se ha eliminado correctamente. En este caso, devuelve una lista vacía. Al eliminar un directorio, asegúrate de que exista y esté vacío; de lo contrario, se generará una excepción.

Para eliminar un directorio y sus subdirectorios, puedes utilizar la función *removedirs*, que requiere que se especifique una ruta que contenga todos los directorios que deben eliminarse:

```
import os

os.makedirs("my_first_directory/my_second_directory")
```

```
os.removedirs("my_first_directory/my_second_directory")
print(os.listdir())
```

Al igual que con la función *rmdir*, si uno de los directorios no existe o no está vacío, se generará una excepción.

NOTA: Tanto en Windows como en Unix, hay un comando llamado *rmdir*, que, al igual que la función *rmdir*, elimina directorios. Además, ambos sistemas tienen comandos para eliminar un directorio y su contenido. En Unix, este es el comando *rm* con el indicador *-r*.

La función `system()`

Todas las funciones presentadas en esta parte del curso pueden ser reemplazadas por una función llamada **system**, que ejecuta un comando que se le pasa como una cadena.

La función `system` está disponible tanto en Windows como en Unix. Dependiendo del sistema, devuelve un resultado diferente.

En Windows, devuelve el valor devuelto por el shell después de ejecutar el comando dado, mientras que en Unix, devuelve el estado de salida del proceso.

```
import os

returned_value = os.system("mkdir my_first_directory")
print(returned_value)
```

Resultado:

```
0
```

El ejemplo anterior funcionará tanto en Windows como en Unix. En nuestro caso, recibimos el estado de salida 0, que indica éxito en los sistemas Unix.

Esto significa que se ha creado el directorio `my_first_directory`. Como parte del ejercicio, intenta enumerar el contenido del directorio donde se creó el directorio `my_first_directory`.

Puntos Claves

1. La función *uname* devuelve un objeto que contiene información sobre el sistema operativo actual. El objeto tiene los siguientes atributos:

- `systemname` (almacena el nombre del sistema operativo)
- `nodename` (almacena el nombre de la máquina en la red)
- `release` (almacena el release (actualización) del sistema operativo)
- `version` (almacena la versión del sistema operativo)
- `machine` (almacena el identificador de hardware, por ejemplo, `x86_64`)

2. El atributo *name* disponible en el módulo `os` te permite distinguir el sistema operativo. Devuelve uno de los siguientes tres valores:

- `posix` (obtendrás este nombre si usas Unix)
- `nt` (obtendrás este nombre si usas Windows)
- `java` (obtendrá este nombre si tu código está escrito en algo como Jython)

3. La función `mkdir` crea un directorio en la ruta pasada como argumento. La ruta puede ser relativa o absoluta, por ejemplo:

```
import os

os.mkdir("hello") # la ruta relativa
os.mkdir("/home/python/hello") # la ruta absoluta
```

Nota: Si el directorio existe, una excepción `FileExistsError` será generada. Además de la función `mkdir`, el módulo `os` proporciona la función `makedirs`, que te permite crear recursivamente todos los directorios en una ruta.

4. El resultado de la función `listdir()` es una lista que contiene los nombres de los archivos y directorios que se encuentran en la ruta pasada como argumento.

Es importante recordar que la función `listdir()` omite las entradas `'.'` y `'..'`, que se muestran, por ejemplo, cuando se utiliza el comando `ls -a` en sistemas Unix. Si no se pasa la ruta, el resultado se devolverá para el directorio de trabajo actual.

5. Para moverte entre directorios, puedes usar una función llamada `chdir()`, que cambia el directorio de trabajo actual a la ruta especificada. Como argumento, toma cualquier ruta relativa o absoluta.

Si deseas averiguar cuál es el directorio de trabajo actual, puedes usar la función `getcwd()`, que devuelve la ruta actual.

6. Para eliminar un directorio, puedes usar la función `rmdir()`, pero para eliminar un directorio y sus subdirectorios, emplea la función `removedirs()`.

7. Tanto en Unix como en Windows, puedes usar la función `system`, que ejecuta el comando que se le pasa como cadena, por ejemplo:

```
import os

returned_value = os.system("mkdir hello")
```

La función `system` en Windows devuelve el valor devuelto por el shell después de ejecutar el comando dado, mientras que en Unix devuelve el estado de salida del proceso.

From:
<https://matebcn.eu/> - miguel angel torres egea

Permanent link:
<https://matebcn.eu/info:cursos:netacad:python:pe2m4:biblioteca:os>

Last update: **07/07/2022 16:55**

