04 Python - Files & zip-a-Dee-Doo-Dah

Important Functions

1. getcwd():

- Returns the current working directory as a string.
- No arguments are passed to the function.
- Example:

```
import os

print(os.getcwd())  # output: C:\Users\Username\Documents
```

2. chdir():

- Changes the current working directory to the given path.
- The path parameter specifies the directory to be set as the current working directory.
- Example:

```
import os

os.chdir('C:/Users/Username/Desktop')  # change directory to Desktop
print(os.getcwd())  # output: C:\Users\Username\Desktop
```

3. exists():

- Checks if a file or directory exists at the given path.
- The path parameter specifies the path to the file or directory.
- Example:

```
import os

print(os.path.exists('C:/Users/Username/Documents'))  # output: True
print(os.path.exists('C:/Users/Username/FakeDirectory'))  # output: False
```

4. getsize():

- Returns the size of a file in bytes.
- The path parameter specifies the path to the file.
- Example:

```
import os

print(os.path.getsize('C:/Users/Username/Documents/example.txt'))  # output: 1234
```

5. listdir():

- Returns a list containing the names of the files and directories in the given directory.
- The path parameter specifies the directory to list the contents of.
- Example:

```
import os

print(os.listdir('C:/Users/Username/Documents'))  # output: ['file1.txt', 'file2.txt', 'folder1', 'folder2']
```

6. mkdir():

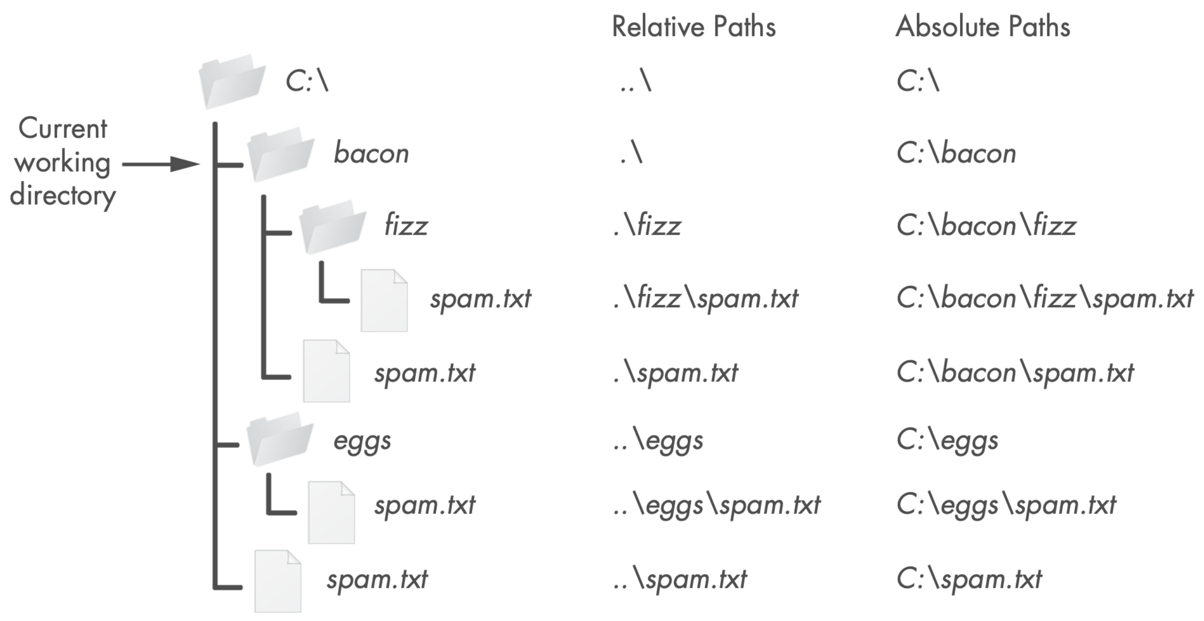
- Creates a new directory at the given path.
- The path parameter specifies the path to the directory to be created.
- Example:

```
import os
```

```
os.mkdir('C:/Users/Username/Desktop/NewFolder')  # create new folder
print(os.path.exists('C:/Users/Username/Desktop/NewFolder'))  # output: True
```

Absolute and Relative path

1. Absolute paths always begin with the root folder and provide the full path to the file, while
2.
3. relative paths are relative to the program's current working directory.
4. The dot (.) folder is a shorthand for the current directory, and the dot-dot (..) folder refers to the parent directory.
5. Using absolute paths ensures that the file can always be found, but relative paths can be useful for easily accessing files in the same directory or in a subdirectory.



6. `os.path.abspath(path)` returns the absolute path of the argument. This is helpful in converting relative paths to absolute paths.
   Example:

```
>>> os.path.abspath('.')
'C:\\Users\\username\\Desktop'
```

2. `os.path.isabs(path)` returns True if the argument is an absolute path and False if it is a relative path.
   Example:

```
>>> os.path.isabs('.')
False
>>> os.path.isabs('C:\\Users\\username\\Desktop')
True
```

3. `os.path.relpath(path, start)` returns a string of a relative path from the start path to path. If start is not provided, the current working directory is used as the start path.
   Example:

```
>>> os.path.relpath('C:\\Users\\username\\Documents', 'C:\\')
'Users\\username\\Documents'
```

4. `os.path.join(path1, path2, ...)` joins one or more path components intelligently using the appropriate separator for the current operating system. This is useful for creating file paths that work across multiple platforms.
   Example:

```
>>> os.path.join('C:\\', 'Users', 'username', 'Documents')
'C:\\Users\\username\\Documents'
```

Reading

For the `read()` method:

1. The `read()` method of a file object reads the entire contents of the file and returns it as a string.

2. If no size is specified in the method call, then the entire contents of the file will be read.
3. The file pointer is placed at the end of the file after the contents have been read.

Example:

```
>>> file = open('example.txt')
>>> content = file.read()
>>> print(content)
This is an example file.
It contains some text.
```

Output:

```
This is an example file.
It contains some text.
```

For the `readlines()` method:

1. The `readlines()` method of a file object reads the contents of the file and returns it as a list of strings.
2. Each string in the list corresponds to a line of text in the file.
3. The newline character at the end of each line is included in the strings.

Example:

```
>>> file = open('example.txt')
>>> lines = file.readlines()
>>> print(lines)
['This is an example file.\n', 'It contains some text.\n']
```

Output:

```
['This is an example file.\n', 'It contains some text.\n']
```

Shelve and Shelf

```
import shelve

# Create a shelf file and store some data in it
shelf_file = shelve.open('mydata')
shelf_file['fruits'] = ['apple', 'banana', 'orange']
shelf_file['prices'] = [1.99, 0.99, 1.50]
shelf_file.close()

# Open the shelf file and retrieve the data
shelf_file = shelve.open('mydata')
fruits = shelf_file['fruits']
prices = shelf_file['prices']
shelf_file.close()

# Print the retrieved data
print(fruits)  # ['apple', 'banana', 'orange']
print(prices)  # [1.99, 0.99, 1.5]
```

1. The shelve module allows you to save variables in your Python programs to binary shelf files on the hard drive.
2. The shelve.open() function is used to create and open a shelf file, which can store data as key-value pairs.
3. You can write data to the shelf file using dictionary-like syntax, where the key is a string and the value can be any picklable object.
4. Once you're done with the shelf file, you should close it using the close() method to ensure that any changes are written to disk and resources are freed up.
5. In this example, we create a shelf file named 'mydata' and store 2 lists in it: `['apple', 'banana', 'orange']` and `[1.99, 0.99, 1.50]`. We then close the file and reopen it, and retrieve the data using the same keys. Finally, we print the retrieved data as comments.

Shutil module

shutil.copy:

1. The shutil.copy() function is used to copy a single file from one location to another.
2. The function takes two arguments, the source file path and the destination file path.

3. If the destination file already exists, it will be overwritten by the source file.
4. The function returns the path to the newly copied file.

Example:
Suppose we have a file named "source.txt" in the directory "C:\Users\John" and we want to copy it to the directory "C:\Users\Public". Here's how we can use shutil.copy() to do that:

```
import shutil

# Path to the source file
src_file = r"C:\Users\John\source.txt"

# Path to the destination directory
dest_dir = r"C:\Users\Public"

# Copy the file to the destination directory
shutil.copy(src_file, dest_dir)

print("File copied successfully.")
```

shutil.copytree:

1. The shutil.copytree() function is used to recursively copy an entire directory tree (i.e., a directory and all of its contents) from one location to another.
2. The function takes two arguments, the source directory path and the destination directory path.
3. If the destination directory already exists, it will raise an error.
4. The function returns the path to the newly copied directory.

Example:
Suppose we have a directory named "my_folder" in the directory "C:\Users\John" and we want to copy it to the directory "C:\Users\Public". Here's how we can use shutil.copytree() to do that:

```
import shutil

# Path to the source directory
src_dir = r"C:\Users\John\my_folder"

# Path to the destination directory
dest_dir = r"C:\Users\Public"

# Copy the directory to the destination directory
shutil.copytree(src_dir, dest_dir)

print("Directory copied successfully.")
```

shutil.move:

1. The shutil.move() function is used to move a file or directory (i.e., a folder and all of its contents) from one location to another, or to rename a file or directory.
2. The function takes two arguments, the source path and the destination path.
3. If the destination path already exists and is a file, it will be overwritten by the source file. If the destination path already exists and is a directory, the source file will be moved into that directory.
4. The function returns the path to the newly moved or renamed file or directory.

Example (Renaming a file):

1. The `shutil.move()` method can be used to rename a file by moving it from the original path to a new path with a different name.
2. The method takes two arguments: the source file path and the destination file path.
3. If the destination file path already exists, the method will overwrite it with the source file. If the destination is a directory, the source file will be moved into that directory with the new name.

```
import shutil

# Path to the source file
src_file = r"C:\Users\John\old_name.txt"

# Path to the destination file
dest_file = r"C:\Users\John\new_name.txt"

# Rename the file
shutil.move(src_file, dest_file)

print("File renamed successfully.")
```

Example (Moving a directory):
Suppose we have a directory named "my_folder" in the directory "C:\Users\John" and we want to move it to the directory "C:\Users\Public". Here's how we can use shutil.move() to do that:

```
import shutil

# Path to the source directory
src_dir = r"C:\Users\John\my_folder"

# Path to the destination directory
```

Deleting Files

- Use os.unlink(path) to delete a file at the specified path.
- Use os.rmdir(path) to delete an empty folder at the specified path.
- Use shutil.rmtree(path) to delete a folder at the specified path, including all files and subfolders it contains.
- The function `send2trash()` is used to send a file or folder to the operating system's trash or recycle bin, instead of permanently deleting it. This provides an added layer of protection, in case the user needs to recover the file or folder later.

Everything About Zip

```
import zipfile

# Creating a zip file and adding files to it
with zipfile.ZipFile('myarchive.zip', 'w') as myzip:
    myzip.write('file1.txt')
    myzip.write('file2.txt')

# Extracting all files from the zip file
with zipfile.ZipFile('myarchive.zip', 'r') as myzip:
    myzip.extractall('extracted_files')

# Compressing a folder and adding it to a zip file
with zipfile.ZipFile('myarchive.zip', 'w', compression=zipfile.ZIP_DEFLATED) as myzip:
    myzip.write('myfolder', arcname='myfolder')

# Reading contents of a zip file
with zipfile.ZipFile('myarchive.zip', 'r') as myzip:
    print(myzip.namelist())  # prints list of files/folders in the archive

# Writing a file to a zip file
with zipfile.ZipFile('myarchive.zip', 'a') as myzip:
    myzip.write('newfile.txt')
```

This code demonstrates the usage of the `zipfile` module in Python to create, compress, extract, and read a zip file.

1. Creating a zip file and adding files to it:

   The `ZipFile()` function is used to create a new zip file, and the `write()` method is used to add files to the zip file.

2. Extracting all files from the zip file:

   The `extractall()` method of `ZipFile()` is used to extract all the files from the zip file.

3. Compressing a folder and adding it to a zip file:

   The `write()` method is used to compress a folder and add it to the zip file. The `arcname` parameter is used to specify the name of the folder in the zip file.

4. Reading contents of a zip file:

   The `namelist()` method of `ZipFile()` is used to get a list of all the files/folders in the zip archive.

5. Writing a file to a zip file:

   The `write()` method of `ZipFile()` is used to add a new file to the existing zip file. The file is added in append mode using the 'a' mode.

Note: The code assumes that the files and folder mentioned in the code are present in the same directory as the Python script.

VTU File creation program

```python
# Create a folder named PYTHON
import os
if not os.path.exists('PYTHON'):
    os.mkdir('PYTHON')

# Create file1.txt and write "VTU" to it
with open('PYTHON/file1.txt', 'w') as file1:
    file1.write("VTU")

# Create file2.txt and write "UNIVERSITY" to it
with open('PYTHON/file2.txt', 'w') as file2:
    file2.write("UNIVERSITY")

# Create file3.txt and write the contents of file1.txt and file2.txt to it
with open('PYTHON/file3.txt', 'w') as file3:
    with open('PYTHON/file1.txt', 'r') as file1:
        file3.write(file1.read())
    with open('PYTHON/file2.txt', 'r') as file2:
        file3.write(file2.read())
```