Working with Directories Metadata

In Python, you can work with directory metadata using the `os` module to retrieve information about directories, such as their contents, permissions, creation time, and modification time. Here's an overview of working with directories and their metadata, along with examples:

1. **Listing Directory Contents:**
   You can use the `os.listdir()` function to get a list of files and directories in a given path.

   ```python
   import os

   directory_path = '/path/to/directory'

   try:
       contents = os.listdir(directory_path)
       print("Directory contents:", contents)
   except FileNotFoundError:
       print("Directory not found.")
   ```

   This example lists the contents of the specified directory.

2. **Getting Directory Information:**
   The `os.stat()` function provides detailed information about a file or directory, including permissions, size, and timestamps.

   ```python
   import os
   from datetime import datetime

   directory_path = '/path/to/directory'

   try:
       dir_info = os.stat(directory_path)
       print("Directory Permissions:", oct(dir_info.st_mode)[-3:])
       print("Size:", dir_info.st_size, "bytes")
       print("Last Modified:", datetime.fromtimestamp(dir_info.st_mtime))
   except FileNotFoundError:
       print("Directory not found.")
   ```

   This example retrieves and prints information about the specified directory, including permissions, size, and last modification time.

3. **Iterating Over Directories Recursively:**
   You can use the `os.walk()` function to iterate over a directory and its subdirectories.

```python
import os

directory_path = '/path/to/directory'

try:
    for root, dirs, files in os.walk(directory_path):
        print(f"Current directory: {root}")
        print("Subdirectories:", dirs)
        print("Files:", files)
except FileNotFoundError:
    print("Directory not found.")
```

This example recursively iterates over the specified directory, printing information about each subdirectory and its files.

4. **Creating a Directory:**
   You can use the `os.mkdir()` function to create a new directory.

```python
import os

new_directory_path = '/path/to/new_directory'

try:
    os.mkdir(new_directory_path)
    print(f"Directory '{new_directory_path}' created successfully.")
except FileExistsError:
    print(f"Directory '{new_directory_path}' already exists.")
```

This example creates a new directory, and if it already exists, a `FileExistsError` is caught.

5. **Removing a Directory:**
   You can use the `os.rmdir()` function to remove an empty directory, and `shutil.rmtree()` to remove a directory and its contents.

```python
import os
import shutil

directory_to_remove = '/path/to/directory_to_remove'

try:
    os.rmdir(directory_to_remove)  # Removes empty directory
    print(f"Directory '{directory_to_remove}' removed successfully.")
except FileNotFoundError:
```

```
    print(f"Directory '{directory_to_remove}' not found.")
except OSError as e:
    print(f"Error: {e}")

# To remove a directory and its contents
try:
    shutil.rmtree(directory_to_remove)
    print(f"Directory '{directory_to_remove}' and its contents removed successfully.")
except FileNotFoundError:
    print(f"Directory '{directory_to_remove}' not found.")
except OSError as e:
    print(f"Error: {e}")
```

This example demonstrates removing an empty directory using `os.rmdir()` and removing a directory and its contents using `shutil.rmtree()`.

Working with directories and their metadata in Python allows you to perform various operations, from listing contents to creating and removing directories. Always handle exceptions to ensure that your code behaves gracefully, especially when dealing with file and directory operations.