Below are comprehensive revision notes for the class on Regular Expressions (Regex) and Reading Files in Python, structured to ensure a clear and thorough understanding of the discussed topics.

Revision Notes: Regular Expressions and Reading Files in Python

Introduction to Reading Files in Python

Using the `open()` Function

- To read files in Python, you use the `open()` function.
- The syntax is `open(filename, mode)`, where the mode can be:
  - `'r'`: Read mode. Default mode if not specified.
  - `'w'`: Write mode for adding new content.
- Example: To read a file named `data.txt`, use `open('data.txt', 'r')`.
- To access the contents, chain the `read()` method: `data = open('data.txt', 'r').read()` 【4:6†transcript】.

Downloading Files using `gdown`

- The `gdown` command is used to download files from Google Drive in environments like Google Colab. It requires the file ID from Google Drive 【4:0†transcript】.

Regular Expressions in Python

Introduction to Regular Expressions

- Regular Expressions (Regex) are sequences of characters that form search patterns, often used for string-searching algorithms.
- Python supports regex through the `re` module: `import re`.

Basic Regex Operations

```
match = re.search(pattern, text)
```

- Returns the first occurrence of the match【4:14†transcript】.

2. **Match**: Checks for a match only at the beginning.

```
match = re.match(pattern, text)
```

- Not often used as it checks from the start【4:14†transcript】.

3. **Findall**: Returns a list of all matches.

```
matches = re.findall(pattern, text)
```

- Useful for extracting all instances of a pattern【4:15†transcript】.

Metacharacters in Regular Expressions

- Metacharacters have special functions in regular expressions:
  - `.` : Matches any single character except newline【4:3†transcript】.
  - `\d` : Matches any digit; `\D` matches non-digits【4:4†transcript】.
  - `\w` : Matches word characters (alphanumeric & underscore); `\W` matches non-word characters【4:12†transcript】.
  - `\s` : Matches whitespace; `\S` matches non-whitespace【4:12†transcript】.
  - `^` : Asserts position at the start of a string.
  - `$` : Asserts position at the end of a string【4:4†transcript】.

Character Sets and Ranges

- **Character Set**: `[...]` matches any one of the characters inside the brackets.
  - Example: `[abc]` matches any of 'a', 'b', or 'c'.
  - Can define ranges: `[a-z]` matches any lowercase letter【4:5†transcript】.

Quantifiers

- `*` : Matches 0 or more occurrences.
- `+` : Matches 1 or more occurrences.
- `?` : Matches 0 or 1 occurrence.

- `{n,m}` : Matches between n and m occurrences【4:4†transcript】.

Groups and Capturing

- Parentheses `()` are used to capture groups.
- Groups allow sections of a pattern to be extracted or applied upon【4:7†transcript】.

Practical Applications: Matching Patterns

Email Validation

- Email pattern:

```
pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
```

- This pattern verifies the standard structure of an email【4:17†transcript】.

Phone Number Extraction

- Example Pattern: `\d{3}[-.\s]?\d{3}[-.\s]?\d{4}` to extract US format numbers【4:8†transcript】.

Practical Exercises and Considerations

- Always test patterns with edge cases to ensure they work correctly.
- Explore using `re.IGNORECASE` if case insensitivity is desired.
- Test regex with functions and understand their output to grasp practical usage【4:18†transcript】.

By understanding these principles and examples, learners can effectively utilize Python's capabilities for text processing and automation tasks that require pattern matching.