UNIT IV: Files, Modules, and Packages

Subtopic 6: Illustrative Programs and Applications

Practical Applications of Files, Modules, and Packages

Programs

1. Word Count Program

A word count program reads the content of a file and counts the number of words.

Example: Word Count Program

Step 1: Create a text file (`textfile.txt`)

```
This is a sample text file.
It contains multiple lines of text.
Each line will be read and processed.
```

```
def word_count(filename):
    with open(filename, 'r') as file:
        content = file.read()
        words = content.split()
        return len(words)

filename = 'textfile.txt'
count = word_count(filename)
print(f"Number of words in {filename}: {count}")
# Output: Number of words in textfile.txt: 14
```

2. Copying Content from One File to Another

A program to copy content from one file to another.

Example: Copying File Content

Step 1: Create a source text file (`source.txt`)

```
This is the content of the source file.
It will be copied to the destination file.
```

```
def copy_content(source_file, destination_file):
    with open(source_file, 'r') as src:
        content = src.read()
    with open(destination_file, 'w') as dest:
        dest.write(content)

source_file = 'source.txt'
destination_file = 'destination.txt'
copy_content(source_file, destination_file)
```

```
# Verify the content of the destination file
with open(destination_file, 'r') as file:
    print(file.read())
# Output:
# This is the content of the source file.
# It will be copied to the destination file.
```

3. Voter's Age Validation

A program to validate a voter's age using custom exceptions.

Example: Voter's Age Validation

Step 1: Create a custom exception class

```
class AgeError(Exception):
    def __init__(self, age, message="Voter must be at least 18 years
old"):
        self.age = age
        self.message = message
        super().__init__(self.message)

    def __str__(self):
        return f'{self.age} -> {self.message}'
```

Step 2: Write the validation program

```
def validate_voter_age(age):
    if age < 18:
        raise AgeError(age)
    return "Voter is eligible"

try:
    age = 16  # Change this value for testing
    result = validate_voter_age(age)
    print(result)
except AgeError as e:
    print(e)
# Output: 16 -> Voter must be at least 18 years old
```

4. Marks Range Validation (0-100)

A program to validate that marks are within the range of 0-100 using custom exceptions.

Example: Marks Range Validation

Step 1: Create a custom exception class

```
class MarksError(Exception):
    def __init__(self, marks, message="Marks must be between 0 and
100"):
        self.marks = marks
        self.message = message
        super().__init__(self.message)

    def __str__(self):
        return f'{self.marks} -> {self.message}'
```

Step 2: Write the validation program

```
def validate_marks(marks):
    if not (0 <= marks <= 100):
        raise MarksError(marks)
    return "Marks are valid"

try:
    marks = 105  # Change this value for testing
    result = validate_marks(marks)
    print(result)
except MarksError as e:
    print(e)
# Output: 105 -> Marks must be between 0 and 100
```

Using Modules and Packages for Organization

Example: Organizing Programs into Modules

Step 1: Create a package structure

```
validation/
    __init__.py
    age_validation.py
    marks_validation.py
    file_operations.py
```

Step 2: Define the modules

`age_validation.py`

```
class AgeError(Exception):
    def __init__(self, age, message="Voter must be at least 18 years
old"):
        self.age = age
        self.message = message
        super().__init__(self.message)

    def __str__(self):
```

```
        return f'{self.age} -> {self.message}'

def validate_voter_age(age):
    if age < 18:
        raise AgeError(age)
    return "Voter is eligible"
```

marks_validation.py

```
class MarksError(Exception):
    def __init__(self, marks, message="Marks must be between 0 and
100"):
        self.marks = marks
        self.message = message
        super().__init__(self.message)

    def __str__(self):
        return f'{self.marks} -> {self.message}'

def validate_marks(marks):
    if not (0 <= marks <= 100):
        raise MarksError(marks)
    return "Marks are valid"
```

file_operations.py

```
def word_count(filename):
    with open(filename, 'r') as file:
        content = file.read()
        words = content.split()
        return len(words)

def copy_content(source_file, destination_file):
    with open(source_file, 'r') as src:
        content = src.read()
    with open(destination_file, 'w') as dest:
        dest.write(content)
```

Step 3: Use the package in the main program

main.py

```
from validation.age_validation import validate_voter_age, AgeError
from validation.marks_validation import validate_marks, MarksError
from validation.file_operations import word_count, copy_content

# Test Voter Age Validation
try:
    age = 16  # Change this value for testing
```

```
    result = validate_voter_age(age)
    print(result)
except AgeError as e:
    print(e)

# Test Marks Validation
try:
    marks = 105  # Change this value for testing
    result = validate_marks(marks)
    print(result)
except MarksError as e:
    print(e)

# Test Word Count
filename = 'textfile.txt'
count = word_count(filename)
print(f"Number of words in {filename}: {count}")

# Test Copy Content
source_file = 'source.txt'
destination_file = 'destination.txt'
copy_content(source_file, destination_file)

with open(destination_file, 'r') as file:
    print(file.read())
```

Summary

Practical applications of files, modules, and packages in Python involve tasks such as word counting, file copying, and validating voter age and marks ranges. These programs demonstrate the use of custom exceptions for robust error handling and the organization of code into modules and packages for better management and reusability. By mastering these techniques, you can create well-structured, maintainable, and efficient Python programs.