Files

```python
# "r", reading
with open("sample.txt", "r") as file:
    lines = file.read() # Read the entire file
    lines = file.read(100)  # Read the first 100 characters

    lines = file.readlines()  # Read all lines and store them in a list
    for line in file:
        print(line.strip(), end="") # Print each line after removing extra whitespace

# "w", writing
with open("sample.txt", "w") as file:
    file.write("Hello, world!")  # Write to the file, creating it if it doesn't exist,  overwriting it if it does

# "a", appending
with open("sample.txt", "a") as file:
    file.write("Hello, world!")  # Append to the file, creating it if it doesn't exist, keeping the existing content
```

File methods and modes

- `close()` - Closes the file and frees up system resources.
- `seek()` - Moves the file cursor to a specific position.
- `tell()` - Returns the current position of the file cursor.
- `truncate()` - Truncates the file to a specified size (in bytes).
- `flush()` - Forces the buffer to write its content to the file immediately.
- `with` **statement** - Ensures proper opening and closing of files automatically.

Summary Table

Mode	Description	File Must Exist	Overwrites	Creates New File
`'r'`	Read-only	Yes	No	No
`'w'`	Write-only	No	Yes	Yes
`'a'`	Append-only	No	No	Yes
`'x'`	Create-only	No	No	Yes
`'r+'`	Read and Write	Yes	No	No
`'w+'`	Write and Read	No	Yes	Yes
`'a+'`	Append and Read	No	No	Yes
`'x+'`	Exclusive Create for Read and Write	No	No	Yes

```python
def load_result(): # returns a list
    students = []
    # implement the load result logic here
    try:
        with open("results.txt", "r") as file1:
            for lines in file1:
                data = lines.strip().split(",") # returns a list of split up strings
                name = data[0]
                student = Student(name)
                student.math, student.chinese, student.english, student.science = list(map(int, data[1:]))
                students.append(student)
        return students
```

concept: put objects into list/dictionary

Read line by line

map(__,__)

```python
import random
import string

def generate_random_string():
    # Define the character sets
    special_chars = "!@#$%^&*"
    numbers = string.digits
    lowercase_letters = string.ascii_lowercase

    # Randomly select one character from each required set
    special_char = random.choice(special_chars)
    number = random.choice(numbers)
    lower_letters = ''.join(random.choices(lowercase_letters, k=4))

    # Combine all characters and shuffle them
    all_chars = special_char + number + lower_letters
    random_string = ''.join(random.sample(all_chars, len(all_chars)))

    return random_string

# Generate and print the random string
print(generate_random_string())
```

Import random
randint()

```python
class Counter:
    count1 = 0 # Class attribute shared by all objects
    def __init__(self):
        self.count2 = 0
    def increase_count2(self):
        self.count2 += 1 # Access object attribute using self
    def increase_count1(self):
        self.__class__.count1 += 1 # Access class attribute using __class__
```

Class and object attributes

String methods

1. **Case Conversion**:
   - `upper()`: Convert to uppercase.
   - `lower()`: Convert to lowercase.
   - `capitalize()`: Capitalize the first letter.
   - `title()`: Capitalize the first letter of each word.
   - `swapcase()`: Swap case of all characters.
2. **Trimming & Padding**:
   - `strip()`: Remove leading/trailing whitespace or characters.
3. **Search & Replace**:
   - `find(sub)`: Find index of first occurrence of `sub` (-1 if not found).
   - `index(sub)`: Find index of first occurrence (raises error if not found).
   - `replace(old, new)`: Replace occurrences of `old` with `new`.
   - `count(sub)`: Count occurrences of `sub`.
- `isalpha()`: Check if all characters are alphabetic.
- `isdigit()`: Check if all characters are digits.
- `isalnum()`: Check if all characters are alphanumeric.
- `isspace()`: Check if all characters are whitespace.

5. **Splitting & Joining**:
   - `split(separator)`: Split string into a list.

List Methods

- `join(iterable)`: Join elements of an iterable with the string as separator.

6. **Built-in Functions**:
   - `len(list)`: Return the number of items in the list.
   - `max(list)`: Return the largest item.
   - `min(list)`: Return the smallest item.
   - `sum(list)`: Return the sum of all numeric items.

3. **Finding Items**:
   - `index(item)`: Return the index of the first occurrence of an item.
   - `count(item)`: Count occurrences of an item in the list.

Dictionary Operations

The table below shows the commonly used dictionary operations.

len(dictionary)	Returns the number of elements in the dictionary.
list(dictionary.keys())	Return a list of the keys in the dictionary.
list(dictionary.values())	Return a list of the values in the dictionary.
list(dictionary.items())	Return a list of tuples containing the keys and values for each entry in the dictionary.
dictionary.clear()	Removes all the keys.

Use `dict.values()` for values only.

Use `dict.items()` for both keys and values.

Use `for key in dict` with `dict[key]` to access values by keys.

```python
try:
    file = open('file.txt', 'r')
except IOError:
    print("File error")

else:
    print("No exceptions")

finally:
    file.close()
```

If there are not exceptions

this will always be executed

SyntaxError - Error in Python syntax.
TypeError - Operation applied to an inappropriate type.
ValueError - Invalid value for a function's argument.
IndexError - Accessing an invalid index in a list or tuple.
KeyError - Accessing a non-existent dictionary key.
AttributeError - Failing attribute reference or assignment.
ZeroDivisionError - Division by zero.
FileNotFoundError - File or directory does not exist.
IOError - General input/output operation failure.
ImportError - Import statement failure.

Custom Error

```python
3 usages
class CustomError(Exception):
    def __init__(self, message: str):
        super().__init__(message)
        self.__message = message

    def __str__(self):
        return f"{self.__message}"

try:
    num = int(input("Enter a number"))
    if num < 0:
        raise CustomError("Number Less than 0")
    elif num > 9:
        raise CustomError("Number is more than 9")
except CustomError as e:
    print(e)
```

Shelve

1. Store and Retrieve Simple Key-Value Pairs

Practice basic usage of the `shelve` module to store and retrieve simple data.

```python
import shelve

def store_data():
    with shelve.open("example_shelve") as db:
        db["name"] = "Alice"
        db["age"] = 25
        db["city"] = "New York"
        print("Data stored successfully!")

def retrieve_data():
    with shelve.open("example_shelve") as db:
        print("Name:", db.get("name"))
        print("Age:", db.get("age"))
        print("City:", db.get("city"))

store_data()
retrieve_data()
```

2. Add, Update, and Delete Data

Practice adding, updating, and deleting key-value pairs in a shelve.

Keys in shelve must be strings

```python
import shelve

def manage_data():
    with shelve.open("example_shelve") as db:
        # Add new data
        db["language"] = "Python"
        print("Added:", db["language"])

        # Update data
        db["language"] = "JavaScript"
        print("Updated:", db["language"])

        # Delete data
        del db["language"]
        print("Deleted language key")

manage_data()
```

3. Store and Retrieve Custom Objects

Practice storing and retrieving custom objects like classes in a shelve.

```python
import shelve

class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"{self.name}, {self.age} years old"

def store_person():
    with shelve.open("example_shelve") as db:
        person = Person("Alice", 25)
        db["person"] = person
        print("Person stored successfully!")

def retrieve_person():
    with shelve.open("example_shelve") as db:
        person = db.get("person")
        if person:
            print("Retrieved:", person)

store_person()
retrieve_person()
```

Updating shelve values

```python
if id in db:
    # Retrieve the record
    record = db[id]  # This is a mutable object, e.g., a list or dictionary

    # Update the grade
    grade = int(input("Enter new grades: "))
    record[1] = grade
```

```python
def add():
    title = input("Enter title of the book: ").capitalize()
    author = input("Enter author of the book: ").capitalize()
    year = int(input("Enter year of publication: "))
    book1 = Book(title, author, year)
    with shelve.open("book") as db:
        db[title] = book1
        print("Book added successfully")

def display():
    try:
        title = input("Enter title to display it's details: ").capitalize()
        with shelve.open("book") as db:
            if title in db:
                book = db.get(title)
                print(f"Title: {book.get_title()}, Author: {book.get_author()}, Year: {book.get_year()}")
            else:
                raise KeyError
    except KeyError:
        print("Title not found")

1 usage
def delete():
    try:
        title = input("Enter title to delete: ").capitalize()
        with shelve.open("book") as db:
            if title in db:
                del db[title]
                print("Deleted successfully")
            else:
                raise KeyError
    except KeyError:
        print("Title not found")
```