Advanced Python Exception Handling & String Formatting Tasks

```
# 1: Handling ZeroDivisionError
def safe_divide(a, b):
    try:
        return a / b
    except ZeroDivisionError:
        return "Error: Cannot divide by zero."

print(safe_divide(10, 2))
print(safe_divide(5, 0))

# 2: Handling Multiple Exceptions
try:
    num = int(input("Enter a number: "))
    str_input = input("Enter a string (will be converted to int): ")
    divisor = int(str_input)
    result = num / divisor
    print("Result:", result)
except ValueError:
    print("Error: Invalid input. String could not be converted to an integer.")
except ZeroDivisionError:
    print("Error: Division by zero.")

#  3: Using Try-Except-Finally
def read_file():
    try:
        file = open("data.txt", "r")
        print(file.read())
    except Exception as e:
        print("An error occurred:", e)
```

```
    finally:
        file.close()
        print("File closed.")

# Uncomment to test: read_file()

#  4: Custom Exception for Age Validation
class AgeError(Exception):
    pass

def check_age():
    age = int(input("Enter your age: "))
    if age < 18:
        raise AgeError("You must be at least 18 years old.")
    else:
        print("Access granted.")

try:
    check_age()
except AgeError as e:
    print(e)

#  5: Nested Try-Except Blocks
try:
    a = int(input("Enter first number: "))
    b = int(input("Enter second number: "))
    try:
        print("Result:", a / b)
    except ZeroDivisionError:
        print("Error: Cannot divide by zero.")
except ValueError:
```

```
    print("Error: Invalid input. Please enter integers.")
else:
    print("Operation successful.")

#  6: Raising Exceptions Manually
def validate_password(password):
    if len(password) < 8 or not any(char.isdigit() for char in password) or not any(char.isalpha() for char in password):
        raise ValueError("Password must be at least 8 characters and contain both letters and numbers.")
    else:
        print("Password is valid.")

try:
    validate_password("abc123")
except ValueError as e:
    print(e)

#  7: Exception Handling with Logging
import logging
logging.basicConfig(filename="error_log.txt", level=logging.ERROR)

def divide_with_logging(a, b):
    try:
        return a / b
    except Exception as e:
        logging.error("Error occurred: %s", e)

divide_with_logging(10, 0)
```

```
# 8 Exception Handling in Nested Functions
def inner_function():
    return 10 / 0

def outer_function():
    try:
        result = inner_function()
        print(result)
    except ZeroDivisionError:
        print("Handled ZeroDivisionError inside outer_function.")

outer_function()

#  9: Exception Handling for File Processing
def read_file(filename):
    try:
        with open(filename, 'r') as file:
            return file.read()
    except FileNotFoundError:
        return "Error: File not found."

print(read_file("sample.txt"))

#  10: Using Else with Try-Except
try:
    num = int(input("Enter a number: "))
except ValueError:
    print("Invalid input.")
else:
    print("Success! You entered:", num)
```

```
# 11: Format Strings with f-strings
name = input("Enter your name: ")
age = input("Enter your age: ")
print(f"Hello {name}, you are {age} years old!")

# 12: Formatting Decimal Places
def format_float(num):
    return "{:.2f}".format(num), f"{num:.2f}"

print(format_float(3.1415926))

# 13: Aligning Text in String Formatting
items = {"Apple": 40, "Banana": 10, "Mango": 25}
print("Item".ljust(10), "Price".rjust(5))
for item, price in items.items():
    print(item.ljust(10), str(price).rjust(5))

# 14: Dynamic String Formatting Using Dictionaries
person = {"name": "John", "age": 30, "city": "New York"}
print("{name} is {age} years old and lives in {city}.".format(**person))

#  15: Formatting Large Numbers
def format_large_number(num):
    return "{:,}".format(num)

print(format_large_number(1000000))
```

```
# 16: Combining String Formatting and Exception Handling
def validate_password(password):
    if len(password) < 8:
        raise ValueError(f"Error: The password '{password}' is too short. It must be at least 8 characters long.")
    elif not any(char.isdigit() for char in password) or not any(char.isalpha() for char in password):
        raise ValueError(f"Error: The password '{password}' must contain both letters and numbers.")
    else:
        print("Password is valid.")

try:
    validate_password("123")
except ValueError as e:
    print(e)

# 17: Formatting Date and Time
from datetime import datetime
now = datetime.now()
print("Today is", now.strftime("%B %d, %Y"), "and the time is", now.strftime("%I:%M %p"))

# 18: Formatting Multi-Line Strings
name = input("Enter your name: ")
age = input("Enter your age: ")
email = input("Enter your email: ")
print(f"""
User Details:
-------------
Name : {name}
Age  : {age}
Email: {email}
""")
```

```
# 19: Creating and Writing to a File
def create_file():
    with open("students.txt", "w") as f:
        f.write("Alice\nBob\nCharlie\n")

create_file()

# Task 20: Reading and Appending to a File
def read_and_append(filename, text):
    try:
        with open(filename, "r+") as f:
            content = f.read()
            f.write("\n" + text)
            f.seek(0)
            print(f.read())
    except FileNotFoundError:
        print("File not found.")

read_and_append("students.txt", "David")
```