**Overview**

The age\_classifier.py script is a professional-grade Python application designed to classify a person’s age into one of several categories—infant, child, teenager, or adult. It demonstrates best practices in error handling, logging, command-line argument parsing, and modular design.

**Code Breakdown**

1. **Imports and Logging Configuration**

python

Copy

import logging

import argparse

* + **logging**: Used to record messages for debugging and operational purposes.
  + **argparse**: Allows the script to accept command-line arguments.

python

Copy

logging.basicConfig(

level=logging.DEBUG,

format='%(asctime)s - %(levelname)s - %(message)s',

handlers=[

logging.FileHandler("age\_classifier.log"),

logging.StreamHandler()

]

)

* + **Logging Setup**: Configured to output messages at the DEBUG level. Log messages include the timestamp, log level, and message.
  + **Handlers**: Logs are sent both to a file (age\_classifier.log) and the console.

1. **Function: classify\_age(age: int) -> str**

python

Copy

def classify\_age(age: int) -> str:

if age < 0:

raise ValueError("Age cannot be negative.")

if age <= 1:

return "infant"

elif 1 < age < 13:

return "child"

elif 13 <= age < 20:

return "teenager"

else:

return "adult"

* + **Purpose**: Determines the age group based on the input age.
  + **Error Handling**: Raises a ValueError if a negative age is provided.
  + **Classification Logic**:
    - Ages 0–1: Classified as "infant"
    - Ages 2–12: Classified as "child"
    - Ages 13–19: Classified as "teenager"
    - Ages 20 and above: Classified as "adult"

1. **Function: get\_user\_age() -> int**

python

Copy

def get\_user\_age() -> int:

while True:

try:

user\_input = input("Enter your age: ").strip()

age = int(user\_input)

if age < 0:

raise ValueError("Age must be non-negative.")

return age

except ValueError as e:

logging.error(f"Invalid input: {e}")

print("Invalid input. Please enter a valid non-negative integer for age.")

* + **Purpose**: Interactively prompts the user for their age.
  + **Input Validation**: Converts the input to an integer and ensures it’s non-negative.
  + **Error Handling**: If the input is invalid (non-numeric or negative), an error is logged and the user is prompted again.

1. **Function: parse\_arguments() -> argparse.Namespace**

python

Copy

def parse\_arguments() -> argparse.Namespace:

parser = argparse.ArgumentParser(description="Age Classification Utility")

parser.add\_argument("-a", "--age", type=int, help="Age to classify")

return parser.parse\_args()

* + **Purpose**: Enables the script to accept an optional age input from the command line.
  + **Usage**: If an age is provided as a command-line argument (e.g., python age\_classifier.py --age 25), it will be used instead of prompting the user.

1. **Main Function: main()**

python

Copy

def main():

args = parse\_arguments()

# Determine age: use command-line argument if provided; otherwise, prompt the user.

age: int = args.age if args.age is not None else get\_user\_age()

try:

category = classify\_age(age)

# Determine proper article ("an" for infant/adult, "a" for child/teenager)

article = "an" if category in ["infant", "adult"] else "a"

message = f"At age {age}, you are classified as {article} {category}."

print(message)

logging.info(f"Classified age {age} as {category}.")

except Exception as e:

logging.error(f"An error occurred: {e}")

print(f"An error occurred: {e}")

* + **Purpose**: Orchestrates the overall flow of the program.
  + **Age Input Determination**: Uses the command-line argument if available; otherwise, it calls get\_user\_age() to prompt the user.
  + **Classification**: Calls classify\_age() to determine the age category.
  + **Output**: Constructs a user-friendly message, prints it, and logs the classification.
  + **Error Handling**: Catches and logs any exceptions that might occur during execution.

1. **Program Execution**

python

Copy

if \_\_name\_\_ == "\_\_main\_\_":

main()

* + **Purpose**: Ensures that the main() function is executed only when the script is run directly, not when imported as a module.

**Summary**

* **Modular Design**: The code is organized into distinct functions for clarity and maintainability.
* **Error Handling & Logging**: Provides robust error management and detailed logging for troubleshooting.
* **User Interaction**: Supports both interactive and command-line inputs.
* **Type Safety**: Uses type hints to clarify function interfaces.