Here's a simple Python program for managing tasks:

# Task Manager Program

# Initialize an empty list to store tasks

tasks = []

def display\_tasks():

print("Tasks:")

for i, task in enumerate(tasks, start=1):

status = "Completed" if task["completed"] else "Pending"

print(f"{i}. {task['description']} - {status}")

def add\_task():

description = input("Enter task description: ")

tasks.append({"description": description, "completed": False})

print("Task added!")

def complete\_task():

task\_number = int(input("Enter the task number to complete: ")) - 1

if task\_number < len(tasks):

tasks[task\_number]["completed"] = True

print("Task completed!")

else:

print("Invalid task number!")

def remove\_task():

task\_number = int(input("Enter the task number to remove: ")) - 1

if task\_number < len(tasks):

del tasks[task\_number]

print("Task removed!")

else:

print("Invalid task number!")

while True:

print("\nTask Manager")

print("1. Display tasks")

print("2. Add task")

print("3. Complete task")

print("4. Remove task")

print("5. Quit")

choice = input("Choose an option: ")

if choice == "1":

display\_tasks()

elif choice == "2":

add\_task()

elif choice == "3":

complete\_task()

elif choice == "4":

remove\_task()

elif choice == "5":

break

else:

print("Invalid option!")

Here's how the program works:

1. The program initializes an empty list **tasks** to store task dictionaries, each with a **description** and a **completed** status.
2. The **display\_tasks** function prints out the list of tasks, showing their descriptions and completion status.
3. The **add\_task** function prompts the user to enter a task description, adds it to the list, and sets its completion status to **False**.
4. The **complete\_task** function prompts the user to enter a task number, marks the corresponding task as completed, and updates its status.
5. The **remove\_task** function prompts the user to enter a task number, removes the corresponding task from the list, and updates the task numbers.
6. The program uses a **while** loop to repeatedly display a menu and prompt the user to choose an option:
   * Display tasks (1)
   * Add task (2)
   * Complete task (3)
   * Remove task (4)
   * Quit (5)
7. Based on the user's choice, the program calls the corresponding function and updates the task list accordingly.

To use this program, simply run it and follow the menu options to add, complete, and remove tasks!

Note: This is a simple program to introduce basic Python concepts, such as lists, dictionaries, and functions. In a real-world application, you would want to add more features, error handling, and data persistence.

Let's play a fun guessing game!

I'm thinking of a number between 1 and 100. You can try to guess the number, and I'll give you hints to help you out.

To play, simply type a number between 1 and 100, and I'll respond with one of the following hints:

* "Too high!" if your guess is above the number I'm thinking of
* "Too low!" if your guess is below the number I'm thinking of
* "You got it!" if you guess the number correctly

Ready to start? Go ahead and guess a number!

(Note: You can ask for a hint at any time if you're stuck)

Here's a Python script to generate strong and secure passwords:

import string

import secrets

def generate\_password(length):

characters = string.ascii\_letters + string.digits + string.punctuation

password = ''.join(secrets.choice(characters) for \_ in range(length))

return password

length = int(input("Enter the desired password length (at least 12 characters recommended): "))

if length < 12:

print("Warning: Password length is less than 12 characters. This may not be secure.")

password = generate\_password(length)

print("Generated password: ", password)

Here's how the script works:

1. The **string** module provides a set of string constants, including **ascii\_letters**, **digits**, and **punctuation**, which we use to create a pool of characters for the password.
2. The **secrets** module is used to generate cryptographically strong random numbers, which we use to select characters from the pool.
3. The **generate\_password** function takes a **length** parameter, which specifies the desired length of the password.
4. The function uses a list comprehension to generate a password by selecting **length** number of characters from the pool using **secrets.choice**.
5. The generated password is returned as a string.
6. The script prompts the user to enter the desired password length, and generates a password of that length.
7. If the user enters a length less than 12, the script warns them that the password may not be secure.

To use this script, simply run it and enter the desired password length when prompted. The script will generate a strong and secure password for you!

Note: It's a good idea to use a password manager to store and generate unique, complex passwords for each of your accounts. This script is for educational purposes only