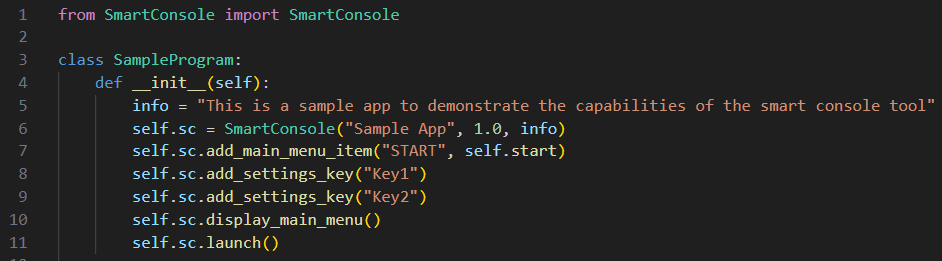
1. **Overview**
   1. The Smart-Console is a Python 3 library designed to make console applications for various purposes.
   2. Smart-Console offers a variety of functions that can save the developer a lot of code writing.
2. **How to Use**
   1. Download Smart-Console.py from: <https://github.com/VladFeldfix/Smart-Console>
   2. Place the file in your Python folder.  
      (typically: C:\Python\Python313\Lib\SmartConsole.py)
   3. Once the SmartConsole.py is among your other libraries, you can import it.
   4. Below is a template for a program using the smart console.

 Illustration 1 – implementing smart console in your code

* 1. Once Smart-Console is implemented as an object as presented in illustration 1, the developer can use the smart console functions as described in section 3

1. **Commands list**
   1. **SETUP – the setup functions must be placed in the code in the sequence they are presented in this document.**
      1. **\_\_init\_\_ (name, version, info)** – this is the constructor function and it requires the following arguments:
         1. **Name (str)** – the name of your application
         2. **Version (any)** – the version of your application
         3. **Info (str)** – short description of the application
      2. **add\_main\_menu\_item (function name, function)** – the main menu of the application must be set before displayed, and it should contain at least one item. there are several functions that will automatically be added to the end of the list: SETTINGS (if relevant), HELP, VERSION HISTORY, EXIT.
         1. **Function name (str)** – the name of the function as displayed in the main menu (for example: "START")
         2. **Function (method)** – the function called when user selects this menu item. (for example: self.start)
      3. **add\_settings\_key (key)** – optional function. Adds a key to settings.
         1. **Key (str)** – the name of the key. (for example: My files location)
      4. **display\_main\_menu ()** – displays the main menu
      5. **launch ()** – starts tkinter's main loop. Must be placed after the setup.
      6. **restart ()** – this function will be placed at the end of each main menu function to say "Done!" and display the main menu again.
   2. **MESSAGES**
      1. **print (text)** – this function displays a messages
         1. **Text (str) –** the text of the message to display
      2. **input (text)** – this function displays a message and activates the user's entry box allowing the user to type and send a response
         1. **Text (str) –** the text of the message to display
         2. **RETURN VALUE (str)** – the text of the user's response
      3. **question (text)** – this function displays a message with a yes or no question and activates the user's entry box allowing the user to type and send a response
         1. **Text (str) –** the text of the message to display
         2. **RETURN VALUE (bool)** – True or False based on the user's answer
      4. **choose (text, options)** – this function displays a message with a multiple-choice question and activates the user's entry box allowing the user to type and send a response
         1. **Text (str) –** the text of the message to display
         2. **Options (tuple) –** a list of choices to select
         3. **RETURN VALUE (str)** – the option the user selected
   3. **SETTINGS**
      1. **get\_settings\_value (key)** – this function returns the current value of the requested key from settings.
         1. **Key (str) –** the requested key
   4. **SCRIPT**
      1. **run\_script (path, functions)** – runs a script with the given functions. See Illustration 2 implementation example.
         1. **Path (str) –** the location of the script file (will display a message saying the file is not found if that's the case)
         2. **Functions (dict) –** a dictionary of functions and expected arguments. The structure of the dict must be as following:  
            key: FUNCTION NAME (str)  
            value: ( FUNCTION TO CALL (method), LIST OF ARGUMENTS (tuple))  
            For example:   
            { "SAY HELLO": (self.say\_hello, ("argument1", " argument2")) }
         3. **RETURN VALUE (bool) –** True or False whether the code went successfully without internal errors

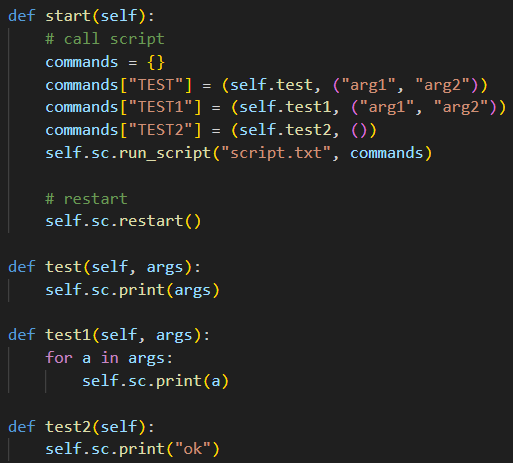
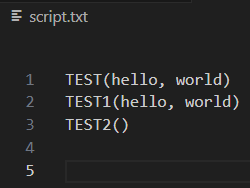
 

Illustration 2 – run script implementation example

* 1. **DATABASE**
  2. **DATE**

1. **Document version history**
   1. v1.0 created on 2025-03-17
   2. v2.0 created on 2025-10-20