

LU09.L02: Unittests auflisten

main.py

```
import json
import os
import subprocess
import sys

def main():
    """
    Main function
    :return: None
    """
    if len(sys.argv) < 2:
        print('Please provide the path to the project folder as an argument.')
        sys.exit(1)
    folder_path = sys.argv[1] # select_project()
    try:
        os.chdir(folder_path)
    except FileNotFoundError:
        print(f'The folder {folder_path} does not exist.')
        sys.exit(1)

    test_modules = find_test_modules(folder_path)

    test_functions = []
    for module in test_modules:
        new_functions = find_test_functions(file_name=module,
path=folder_path)
        test_functions.extend(new_functions)

    output_json(test_functions)

def select_project():
    """
    Asks the user to select the project folder
    :return: path to the project folder
    """
    try:
        # Run the zenity command to open a directory selection dialog
        result = subprocess.run(
            ["zenity", "--file-selection", "--directory", "--
title=Select a Directory"],
            text=True, # Capture output as string
```

```
        stdout=subprocess.PIPE, # Redirect standard output
        stderr=subprocess.PIPE # Redirect standard error
    )

    # Check if a directory was selected
    if result.returncode == 0: # Return code 0 means success
        return result[1].strip() # Return the selected directory
    else:
        print("No directory was selected.")
        return ""
except FileNotFoundError:
    print("Zenity is not installed. Please install it using 'sudo
apt install zenity'.")
    return ""
except Exception as e:
    print(f"An error occurred: {e}")
    return ""

def find_test_modules(path='.'):
    """
    Searches for files matching test_*.py or *_test.py in the specified
    folder using a Bash command.

    Parameters
    -----
    path : str
        The path to the folder to search in. Default is the current
    Returns
    -----
    List[str]
        A list of matching file paths.
    """

    # Define the Bash command to search for test files
    command = f'ls {path} | grep -E "test_.*\\.py|_test\\.py"'
    result = execute_bash(command)

    # Split the output into lines to get a list of file paths
    if result[1]:
        files = result[1].strip()
        file_list = files.split('\n')
        return file_list
    return []

def find_test_functions(file_name, path='.'):
    """
    Searches for test functions in the specified file using a Bash
    command.
```

```

Parameters
-----
path : str
    The path to the folder to search in. Default is the current
directory.
file_name : str
    The name of the file to search in.
Returns
-----
List[str]
    A list of test function names.
"""
# Define the Bash command to search for test functions in the file
command = f'grep -E "def test_*" "{path}/{file_name}"'
result = execute_bash(command)

# Split the output into lines to get a list of function names
if result[1]:
    functions = result[1].strip()
    function_list = functions.split('\n')
    for i in range(len(function_list)):
        function_list[i] = sanitize_function_name(function_list[i])
    return function_list

return []

def sanitize_function_name(function_name):
    """
    Sanitize the function name by removing the 'def ' prefix and any
    leading/trailing whitespace.
    Parameters
    -----
    function_name : str
        The function names to sanitize.
    Returns
    -----
    str
        The sanitized function names.
    """
    # remove the 'def ' prefix and any leading/trailing whitespace
    function_name = function_name.replace('def ', '').strip()
    # remove the arguments including the colon and brackets
    function_name = function_name.split('(')[0]
    return function_name

def output_json(test_functions):
    """
    Output the test functions as a JSON string.
    Parameters
    -----

```

```
test_functions : List[str]
    A list of test function names.
"""
json_str = json.dumps(test_functions, indent=4)
print(json_str)

def execute_bash(command):
    """
    Execute a Bash command and return the result.
    Parameters
    -----
    command : str
        The Bash command to execute.
    Returns
    -----
    subprocess.CompletedProcess
        The result of the command execution.
    """
    try:
        result = subprocess.run(
            command,
            shell=True, # Use a shell to interpret the command
            check=True, # Raise an error if the command fails
            text=True, # Capture output as a string
            stdout=subprocess.PIPE, # Redirect standard output
            stderr=subprocess.PIPE # Redirect standard error
        )
        return [result.returncode, result.stdout]
    except subprocess.CalledProcessError as e:
        print(f'Error executing the command: {e.cmd}')
        print(f'Error message: {e.stderr}')
        sys.exit(1)
    except Exception as ex:
        print(f'An unexpected error occurred: {ex}')
        sys.exit(1)

if __name__ == '__main__':
    main()
```

M122-LU09

 Marcel Suter

From:

<https://wiki.bzz.ch/> - **BZZ - Modulwiki**

Permanent link:

<https://wiki.bzz.ch/modul/m122/learningunits/lu09/loesungen/pytests>

Last update: **2024/12/10 07:07**

