1. Write a Python program that reads all the Python source code files from a given Python package and reports the Lines Of Code (LOC) as per the following requirements. LOC excludes the lines that are only comments. • Total code size (Total LOC of the package) • Number of modules and total size of each module (sub-directory) • Number of sub-modules under each module (Each file under the sub-directory) and total size of each sub-module • Number of independent functions (defined outside a class) in each sub-module and total size of each function • Number of classes in each sub-module and total size of each class • Number of methods (functions defined within a class) in each class and total size of each method • Total size of other lines of code in each sub-module

Code:

```
"""
    This module provides the functionality for counting the
    number of lines of code of each class, each method and
    each function and other lines of code for each python source code.

    Original Author: Pranesh Kumar

    Created On: 25 Apr 2023
"""

# importing the os module to find out the python files
import os


def read_python_file(absolute_path):
    """
    Opens the file passed to the function in read mode and returns the
    data present in the file as a string

    Args:
        absolute_path (str): absolute path of the python program

    Returns:
        str: source code present in the python file
    """
    with open(absolute_path, "r") as filehandle:
        data = filehandle.read()
    filehandle.close()

    return data


def breakdown_contents(filecontents, filehandle=None):
    """
    Breaks down the source code and finds out the number
    of blank lines, single line comments, multi line comments,
    classes, methods and functions.

    Args:
        filecontents (str): source code of the python program
        filehandle (TextIOWrapper): file handle of the file, to where the
output is written.
                                    Defaults to None.
```

```
    Returns:
        dict: dictionary containing the number
        of blank lines, single line comments, multi line comments,
        classes, methods and functions.
    """
    counter = {"LinesFiltered": 0,
               "Multiline": 0,
               "Comments": 0,
               "BlankLines": 0,
               "Functions": 0,
               "Classes": 0,
               "Methods": 0}
    isdocstring = False
    lines = filecontents.split("\n")
    print("Total LOC:", len(lines), file=filehandle)
    for line in lines:
        if isdocstring:
            if line.strip().endswith('"""') or
line.strip().endswith("'''"):
                isdocstring = False
                counter["Multiline"] += 1
                continue
            counter["Multiline"] += 1
            continue
        if line.strip().startswith("#"):
            counter["Comments"] += 1
            continue
        elif line.strip() == "":
            counter["BlankLines"] += 1
            continue
        if line.strip().startswith("'''") or
line.strip().startswith('"""'):
            counter["Multiline"] += 1
            isdocstring = True
            continue
        else:
            counter["LinesFiltered"] += 1

        if line.startswith("def"):
            counter["Functions"] += 1
        elif line.startswith("class"):
            counter["Classes"] += 1
        elif line.startswith("    def"):
            counter["Methods"] += 1

    return counter


def class_code_counter(src):
    """
    Counts the number of lines in each class and returns the name of each
class along with its count.

    Args:
        src (str): source code of the python program

    Returns:
        dict: Dictionary containing the name of the class as key and number
of lines in that respective class as value
    """
```

```
    isdocstring = False
    classcodecounter = 0
    lines = src.split("\n")
    classcodedict = {}
    classname = None
    for line in lines:
        if line.strip().endswith('"""') or line.strip().endswith("'''"):
            isdocstring = False
        if isdocstring:
            if line.strip().endswith('"""') or
line.strip().endswith("'''"):
                isdocstring = False
                continue
            continue
        if line.strip().startswith("#"):
            continue
        elif line.strip() == "":
            continue
        if line.strip().startswith("'''") or
line.strip().startswith('"""'):
            isdocstring = True
            continue
        if line.strip().startswith("class"):
            classname = line.strip().split(" ")[1].split("(")[0]
            if isdocstring:
                if line.strip().endswith('"""') or
line.strip().endswith("'''"):
                    isdocstring = False
                    continue
                continue
            if line.strip().startswith("#"):
                continue
            elif line.strip() == "":
                continue
            if line.strip().startswith("'''") or
line.strip().startswith('"""'):
                isdocstring = True
                continue
            if line.startswith("    "):
                classcodecounter += 1
        else:
            if line.startswith("    "):
                classcodecounter += 1
            if line.split(" ")[0].isalpha():
                classcodedict[classname] = classcodecounter
                classname = None
                continue
    try:
        del classcodedict[None]
    except KeyError:
        pass
    return classcodedict


def method_code_counter(src):
    """
    Counts the number of lines in each method and returns the name of each
method along with its count.

    Args:
        src (str): source code of the python program
```

```
    Returns:
        dict: Dictionary containing the name of the method as key and
number of lines in that respective method as value
    """
    isdocstring = False
    methodcodecounter = 0
    lines = src.split("\n")
    methodcodedict = {}
    methodname = None
    for line in lines:
        if isdocstring:
            if line.strip().endswith('"""') or
line.strip().endswith("'''"):
                isdocstring = False
                continue
            continue
        if line.strip().startswith("#"):
            continue
        elif line.strip() == "":
            continue
        if line.strip().startswith("'''") or
line.strip().startswith('"""'):
            isdocstring = True
            continue
        if not line.startswith("    def"):
            if line.startswith("        "):
                methodcodecounter += 1
            if line.strip().split(" ")[0].isalpha() and line.split(" ")[0]
!= "class" and not line.startswith(
                    "        "):  # here is the change required
                methodcodedict[methodname] = methodcodecounter
                methodname = None
                methodcodecounter = 0
                continue
        elif line.strip().split(" ")[0].isalpha() and line.split(" ")[0] !=
"class" and not line.startswith(
                "        "):  # here is the change required
            methodcodedict[methodname] = methodcodecounter
            methodcodecounter = 0
            methodname = line.lstrip().split(" ")[1].split("(")[0]
            if isdocstring:
                if line.strip().endswith('"""') or
line.strip().endswith("'''"):
                    isdocstring = False
                    continue
                continue
            if line.strip().startswith("#"):
                continue
            elif line.strip() == "":
                continue
            if line.strip().startswith("'''") or
line.strip().startswith('"""'):
                isdocstring = True
                continue
            if line.startswith("        "):
                methodcodecounter += 1
    try:
        del methodcodedict[None]
    except KeyError:
        pass
```

```
    return methodcodedict


def function_code_counter(src):
    """
    Counts the number of lines in each function and returns the name of
each function along with its count.

    Args:
        src (str): source code of the python program

    Returns: dict: Dictionary containing the name of the function as key
and number of lines in that respective
    function as value
    """
    isdocstring = False
    functioncodecounter = 0
    lines = src.split("\n")
    functioncodedict = {}
    functionname = None
    for line in lines:
        if isdocstring:
            if line.strip().endswith('"""') or
line.strip().endswith("'''"):
                isdocstring = False
                continue
            continue
        if line.strip().startswith("#"):
            continue
        elif line.strip() == "":
            continue
        if line.strip().startswith("'''") or
line.strip().startswith('"""'):
            isdocstring = True
            continue
        if not line.startswith("def"):
            if line.startswith("    "):
                functioncodecounter += 1
            if line.strip().split(" ")[0].isalpha() and line.split(" ")[0]
!= "class" and not line.startswith(
                    "    "):  # here is the change required
                functioncodedict[functionname] = functioncodecounter
                functionname = None
                functioncodecounter = 0
                continue
        elif line.strip().split(" ")[0].isalpha() and line.split(" ")[0] !=
"class" and not line.startswith(
                "    "):  # here is the change required
            functioncodedict[functionname] = functioncodecounter
            functioncodecounter = 0
            functionname = line.lstrip().split(" ")[1].split("(")[0]
            if isdocstring:
                if line.strip().endswith('"""') or
line.strip().endswith("'''"):
                    isdocstring = False
                    continue
                continue
            if line.strip().startswith("#"):
                continue
            elif line.strip() == "":
                continue
```

```
            if line.strip().startswith("'''") or
line.strip().startswith('"""'):
                isdocstring = True
                continue
            if line.startswith("    "):
                functioncodecounter += 1
    try:
        del functioncodedict[None]
    except KeyError:
        pass
    return functioncodedict


def calcualte_other_lines(counterdict, classdict, functiondict):
    """Calculates other lines of code by removing the above calculated LOC
from total LOC

    Args: counterdict (dict): dictionary containing the number of blank
lines, single line comments, multi line
    comments, classes, methods and functions. classdict (dict): Dictionary
containing the name of the class as key
    and number of lines in that respective class as value functiondict
(dict): Dictionary containing the name of the
    function as key and number of lines in that respective function as
value

    Returns:
        int: other lines of code
    """
    totalfilteredlines = counterdict["LinesFiltered"]
    classlines = sum(classdict.values()) + len(classdict)
    functionlines = sum(functiondict.values()) + len(functiondict)
    otherlines = totalfilteredlines - (classlines + functionlines)
    return otherlines


def traversefolder(homepath, filehandle=None):
    """Traverses each folder and sub-folders recursively and finds out the
    python files, and does the above operations and writes the data
    to a file.

    Args: homepath (str): home path where the python files are to be
searched recursively. filehandle (TextIOWrapper,
    optional): File handle of the file where the output will be written.
Defaults to None.
    """
    pythonfiles = []
    for root, folders, files in os.walk(homepath):
        for file in files:
            if file.endswith(".py"):
                pythonfiles.append(os.path.join(root, file))

    for file in pythonfiles:
        print("=" * 26, file=filehandle)

        print("File:", file, file=filehandle)

        filecontents = read_python_file(file)
        counterdict = breakdown_contents(filecontents, filehandle)
        classdict = class_code_counter(filecontents)
        methoddict = method_code_counter(filecontents)
```

```
        functiondict = function_code_counter(filecontents)
        otherlines = calcualte_other_lines(counterdict, classdict,
functiondict)

        print(f"Filtered LOC: {counterdict['LinesFiltered']}",
file=filehandle)
        print(f"Single Line Comments: {counterdict['Comments']}",
file=filehandle)
        print(f"Multi Line Comments: {counterdict['Multiline']}",
file=filehandle)
        print(f"Number of Functions: {counterdict['Functions']}",
file=filehandle)
        print(f"Number of Classes: {counterdict['Classes']}",
file=filehandle)
        print(f"Number of Methods: {counterdict['Methods']}",
file=filehandle)

        print("*" * 26, file=filehandle)
        print("Classes:", file=filehandle)

        for classname, count in classdict.items():
            print(f"{classname} - {count}", file=filehandle)

        print("*" * 26, file=filehandle)
        print("Methods:", file=filehandle)

        for methodname, count in methoddict.items():
            print(f"{methodname} - {count}", file=filehandle)

        print("*" * 26, file=filehandle)
        print("Functions:", file=filehandle)

        for functionname, count in functiondict.items():
            print(f"{functionname} - {count}", file=filehandle)

        print("*" * 26, file=filehandle)
        print(f"Other Lines: {otherlines}", file=filehandle)

        print("=" * 26, file=filehandle)


# driver code
if __name__ == "__main__":
    homedir = input("Enter the home path to search for python files: ")
    with open(os.path.join(homedir, "loc.txt"), "w") as handle:
        traversefolder(homedir, handle)
    handle.close()
```