Modules in Python:

- We use modules to break down large programs into small manageable and organized files. Furthermore, modules provide reusability of code.
- Modules refer to a file containing Python statements and definitions.
- We can define our most used functions in a module and import it, instead of copying their definitions into different programs.

Module creation:

Type the following and save it as `calci.py`

```
def add(a, b):
   return a + b
def sub(a, b):
   return a - b
def mul(a, b):
   return a * b
def div(a, b):
   return a / b
```

Using created module:

```
import calci
a=20
b=10
c=calci.add(a,b)
print(c)
```

or

```
from calci import *
a=20
```

```
b=10
c=add(a,b)
print(c)
```

Python standard modules:

- We can import a standard module using the `import` statement and access the definitions inside it using the dot operator
- For example, there are many pre-defined functions in standard math module. We need to import math module to support them. Some of them are as follows:

  1. ceil() :- This function returns the **smallest integral value greater than the number**. If number is already integer, same number is returned.

  2. floor() :- This function returns the **greatest integral value smaller than the number**. If number is already integer, same number is returned.

  3. fabs() :- This function returns the **absolute value** of the number

  4. exp(a) :- This function returns the value of **e raised to the power a (e**a)** .

  5. log(a, b) :- This function returns the logarithmic **value of a with base b**. If base is not mentioned, the computed value is of natural log.

For Example:

```
import math

a = 2.3

# returning the ceil of 2.3
print ("The ceil of 2.3 is : ",
end="")
print (math.ceil(a))

# returning the floor of 2.3
print ("The floor of 2.3 is : ",
end="")
print (math.floor(a))
```

```
import math

# returning the exp of 4
print ("The e**4 value is : ",
end="")
print (math.exp(4))

# returning the log of 2,3
print ("The value of log 2 with
base 3 is : ", end="")
print (math.log(2,3))
```

Import with renaming:

```
# import module by renaming it

import math as m
print("The value of pi is", m.pi)
```

- We have renamed the `math` module as `m`. This can save us typing time in some cases.
- Note that the name `math` is not recognized in our scope. Hence, `math.pi` is invalid, and `m.pi` is the correct implementation.

Python from...import statement:

- We can import specific names from a module without importing the module as a whole. Here is an example.

```
# import only pi from math module

from math import pi
print("The value of pi is", pi)
```

- Here, we imported only the `pi` attribute from the `math` module.

Import all names:

- We can import all names(definitions) from a module using the following construct:

```
from math import *
```

```
print("The value of pi is", pi)
```

- Here, we have imported all the definitions from the math module.

Wikipedia:

We can now import Wikipedia in Python using Wikipedia module. Use the incessant flow of knowledge with Python for daily needs.
Install it as:

```
pip install wikipedia
```

And use it as:

```
import wikipedia
result = wikipedia.page("Bennett University")
print(result.summary)
```

If you wish to get a particular number of sentences from the summary, just pass that as an argument to the summary() function:

```
import wikipedia
print(wikipedia.summary("Debugging", sentences = 2))
```

Emoji:

Emojis have become a way to express and to enhance simple boring texts. For this, `emoji` module is needed to be installed.
In terminal. Use:

```
pip install emoji
```

To upgrade to the latest packages of emojis. Here's how it can be done:

```
pip install emoji -upgrade
```

```
from emoji import emojize
```

```
print(emojize(":thumbs_up:"))
```

File Handling Mode (Read, Write, Create, Append)

- Files are named locations on disk to store related information. They are used to permanently store data in a non-volatile memory (e.g. hard disk).
- Since Random Access Memory (RAM) is volatile (which loses its data when the computer is turned off), we use files for future use of the data by permanently storing them.
- When we want to read from or write to a file, we need to open it first. When we are done, it needs to be closed so that the resources that are tied with the file are freed.

Hence, in Python, a file operation takes place in the following order:

1. Open a file
2. Read or write (perform operation)
3. Close the file

Opening a file:

Python has a built-in `open()` function to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

```
f = open("test.txt")     # open file in current directory
f = open("C:/Python38/README.txt")  # specifying full path
```

We can specify the mode while opening a file. In mode, we specify whether we want to read `r`, write `w` or append `a` to the file. We can also specify if we want to open the file in text mode or binary mode. Different kinds of opening modes ae given below:

Modes	Uses	Definition
"r"	Read	Opens a file for reading, error if the file does not exist
"a"	Append	Opens a file for appending, creates the file if it does not exist
"w"	Write	Opens a file for writing, creates the file if it does not exist
"x"	Create	Creates the specified file, returns an error if the file exists

"t"	Open	Creates the specified file, returns an error if the file exists
"rb"	Read binary	Opens in text mode. (default)
"r+"	Reading and Writing	Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
"rb+"	Reading and Writing binary	Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
"wb"	Writing only in binary format	Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
"w+"	Both writing and reading	Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
"wb+"	Both writing and reading in binary	Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
"ab"	Appending in binary	Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
"a+"	Both appending and reading	Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
"ab+"	Both appending and reading in binary	Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the

		append mode. If the file does not exist, it creates a new file for reading and writing.

Writing to file

- In order to write into a file in Python, we need to open it in write `w`, append `a` or exclusive creation `x` mode.
- We need to be careful with the `w` mode, as it will overwrite into the file if it already exists. Due to this, all the previous data are erased.
- Writing a string or sequence of bytes (for binary files) is done using the `write()` method. This method returns the number of characters written to the file.

```
with open("test.txt",'w',encoding = 'utf-8') as f:
   f.write("my first file\n")
   f.write("This file\n\n")
   f.write("contains three lines\n")
```

This program will create a new file named `test.txt` in the current directory if it does not exist. If it does exist, it is overwritten. Following syntax has been given for different file operations.

File Handling Functions (Open, Read, Write, Remove)

Functions	Syntax	Definition
open()	f = open("demofile.txt")	The open() function takes two parameters; *filename*, and *mode*.
read()	f = open("demofile.txt", "r") print(f.read())	The open() function returns a file object, which has a read()method for reading the content of the file
write()	f = open("demofile2.txt", "a") f.write("Now the file has more content!")	To write to an existing file, you must add a parameter to the open() function

`remove()`	`import os` `os.remove("demofile.txt")`	To delete a file, you must import the OS module, and run its `os.remove()` function

Closing the file:

- After performing all the actions file must be closed.
- It will free up the resources that were tied with the file. It is done using the `close()` method available in Python.
- Python has a garbage collector to clean up unreferenced objects but we must not rely on it to close the file.

```
f = open("test.txt")
# perform file operations
f.close()
```

- This method is not entirely safe. If an exception occurs when we are performing some operation with the file, the code exits without closing the file, hence we can use:

```
try:
    f = open("test.txt")
    # perform file operations
finally:
    f.close()
```

This way, we are guaranteeing that the file is properly closed even if an exception is raised that causes program flow to stop.

Problems:

1. How is the `from` statement related to the `import` statement?

   Ans: The from statement imports an entire module, like the import statement, but as an extra step it also copies one or more variables from the imported module into the scope where the from appears. This enables you to use the imported names directly (name) instead of having to go through the module (module.name).

2. What will be the output of following code:

test.txt Content:

```
aaa
bbb
ccc
ddd
eee
fff
ggg
```

Code:

```
f = open("test.txt", "r")
print(f.readline(3))
```

Ans: aaa

3. The contents of file names.txt are as follows:

```
Moana
Cinderella
Tiana
```

Write the lines of code which will print all the names in names.txt?

Ans:

```
names = open("names.txt", "r")
for line in names:
    print(line)
```

4. Using the following directory structure, write the relative path to the *feline* folder. Assume that the current working directory is in the root folder where the *animals* directory resides.

```
animals/
|
├── feline/
|   ├── lions.gif
|   └── tigers.gif
|
├── ursine/
|   └── bears.gif
|
└── animals.csv
```

Ans: animals/feline

5. What error is returned by the following statement if the file does not exist?

```
f=open("A.txt")
```

Ans: FileNotFoundError

6. What will be the output of the following code snippet:

```
f=open("test.txt","w+")
f.write("FileHandling")
f.seek(5)
a=f.read(5)
print(a)
```

Ans: andli

7. Write a module which should contain all the functions necessary for creating a simple calculator. You need to provide functions for the addition, subtraction, multiplication and division. Addition and multiplication functions should be able to be applied on more than numbers. Use these functions in a separate file which will implement the functionality of calculator.

```
#======== module.py=============
def addNumbers(lst):
    sum=0
    for i in lst:
        sum=sum+int(i)
    return sum

def subtractNumbers(num1,num2):
    return num1-num2

def multiplyNumbers(lst):
    result=1
    for i in lst:
        result=result*int(i)
    return result

def divideNumbers(num1,num2):
    if num2==0:
        return "Divisor can't be zero"
    return num1/num2
```

```
#============= user_file.py ==============
import random
import module
```

```
randomList=random.sample(range(10,30),5)
print(randomList)
print(module.addNumbers(randomList))
print(module.subtractNumbers(randomList[0],randomList[1]))
print(module.multiplyNumbers(randomList))
print(module.divideNumbers(randomList[0],randomList[1]))
```

8. Write a module for implementing matrix transpose and multiplication operations. Use this module in a separate file to use both the functions.

```
#======== module.py=============
def multiplyMatrix(X,Y):
    result = [[0,0,0,0],
              [0,0,0,0],
              [0,0,0,0]]
    for i in range(len(X)):
        for j in range(len(Y[0])):
            for k in range(len(Y)):
                result[i][j] += X[i][k] * Y[k][j]
    print(result)
    return result

def getTranspose(X):
    result = [[0,0,0],
              [0,0,0]]
    for i in range(len(X)):
        for j in range(len(X[0])):
            result[j][i] = X[i][j]

    print(result)
    return result
```

```
#============= user_file.py ==============
# import random
import module


X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]
```

```
Y = [[5,8,1,2],
    [6,7,3,0],
    [4,5,9,1]]

result=module.multiplyMatrix(X,Y)
for r in result:
    print(r)

X1 = [[12,7],
    [4 ,5],
    [3 ,8]]
result1=module.getTranspose(X1)
for r in result1:
    print(r)
```

9. Write a program that takes in a filename, then takes in a series of lines of input until a blank line is entered, writing each line to the file with the given name. After the blank line is entered, properly close the file before ending the program.

```
f=open("write_user_inputs.txt",'w','utf-8')

while True:
    line=input()
    if line=="":
        break
    else:
        f.write(line)
        f.write("\n")
f.close()
```

10. Write a program that takes in a string as input. Then print how many times that string appears inside the chosen file.

```
searchStr=input("Enter search string: ")
f=open("D:\python_programs\python_course_bennett\\tutorial_preparation\s
ample.txt",'r')
count=0
for index, line in enumerate(f):
    count=count+line.strip().count(searchStr)
print(count)
```

11. Write a program that reads a list of temperatures in celsius from a file, converts those temperatures to Fahrenheit, and writes the results to a file.

```
file1 =
open("D:\python_programs\python_course_bennett\\tutorial_preparation\\ft
emps1.txt", 'a')
temperatures = [line.strip() for line in
open("D:\python_programs\python_course_bennett\\tutorial_preparation\\ft
emps.txt")]
for t in temperatures:
    temp=int(t)*9/5+32
    file1.write(str(temp))
    file1.write("\n")
file1.close()
```

12. You are given a file called `class_scores.txt`, where each line of the file contains a one-word username and a test score separated by spaces, like below:.

```
GWashington 83
JAdams 86
```

Write code that scans through the file, adds 5 points to each test score, and outputs the usernames and new test scores to a new file, scores2.txt.

```
file1 =
open("D:\python_programs\python_course_bennett\\tutorial_preparation\\cl
ass_scores_1.txt", 'a')
lines = [line.strip() for line in
open("D:\python_programs\python_course_bennett\\tutorial_preparation\\cl
ass_scores.txt")]
for line in lines:
    temp=line.split()
    file1.write(temp[0]+" "+str(int(temp[1])+5))
    file1.write("\n")
file1.close()
```

13. You are given a file called grades.txt, where each line of the file contains a one-word student username and three test scores separated by spaces, like below:.

```
GWashington 83 77 54
JAdams 86 69 90
```

Write code that scans through the file and determines how many students passed all three tests. A student will pass the test if his marks are greater than 70.

```
lines = [line.strip() for line in
open("D:\python_programs\python_course_bennett\\tutorial_preparation\\gr
ades.txt")]
count=0
for line in lines:
    temp=line.strip().split()
    print(temp)
    if int(temp[1].strip())>70 and int(temp[2].strip())>70 and
int(temp[3].strip())>70:
        count+=1
print("Count: ",count)
```

14. You are given a file called logfile.txt that lists log-on and log-off times for users of a system. A typical line of the file looks like this:

```
Van Rossum, 14:22, 14:37
```

Each line has three entries separated by commas: a username, a log-on time, and a log-off time. Times are given in 24-hour format. You may assume that all log-ons and log-offs occur within a single workday.

Write a program that scans through the file and prints out all users who were online for at least an hour.

```
from datetime import datetime
from datetime import timedelta
lines = [line.strip() for line in
open("D:\python_programs\python_course_bennett\\tutorial_preparation\\lo
g_file.txt")]
for line in lines:
    temp=line.strip().split(",")
    # print(temp)
    FMT = '%H:%M'
    tdelta = datetime.strptime(temp[2].strip(), FMT) -
datetime.strptime(temp[1].strip(), FMT)
    if tdelta.total_seconds()>=3600:
        print(temp[0].strip())
```