Lesson-8: Modular Programming

Modules:

- Module
- Interface
- Client

Python Modules

Program: 1

```
import math

print( math.factorial(5) )
print( math.__doc__ )
print( )
print( math.factorial.__doc__ )
```

Output:

```
120
This module provides access to the mathematical functions
defined by the C standard.

Find x!.

Raise a ValueError if x is negative or non-integral.
```

This is my first module in python:

```
#save it as myfirstmodule.py

""" This is my first module in python """
print("I am your first python module")
def sample_module( ):
    """ This is my first module function in python """
    print("Hello Module")
```

Program: 2

```
import myfirstmodule

myfirstmodule.sample_module( )
print( myfirstmodule.__doc__ )
print( myfirstmodule.sample_module.__doc__ )
print( __name__ )
```

Output:

```
I am your first python module
Hello Module
This is my first module in
python
This is my first module function
in python
__main__
```

```
#myfirstmoduleone.py
def fun( ):
    print("Hello fun - 1”)

#myfirstmoduletwo.py
def fun( ):
    print("Hello fun - 2”)
```

Program: 3

Output:

```
import myfirstmoduleone, myfirstmoduletwo

fun( )
```

```
#moduleone.py
def fun( ):
    print("Hello fun - 1”)

#moduletwo.py
def fun( ):
    print("Hello fun - 2”)
```

Program: 4

Output:

```
import moduleone, moduletwo

moduleone.fun( )
moduletwo.fun( )
```

Guess The Output:

```
>>> factorial(5)
```

```
>>> math.factorial(5)
```

```
>>> import math
>>> factorial(5)
```

```
>>> math.factorial(5)
```

```
#mymodule.py
def fun_one( ):
    print("Hello fun - 1")
def fun_two( ):
    print("Hello fun - 2")
```

Program: 5

Output:

```
from mymodule import fun_one

fun_one( )
```

```
#mymodule.py
def fun_one( ):
    print("Hello fun - 1")
def fun_two( ):
    print("Hello fun - 2")
```

Program: 6

Output:

```
import mymodule
from mymodule import fun_one

fun_two( )
```

```
#mymodule.py
def fun_one( ):
    print("Hello fun - 1")
def fun_two( ):
    print("Hello fun - 2")
```

Program: 7

Output:

```
#check_module_program.py
import mymodule
from mymodule import fun_one

mymodule.fun_two()
```

```
#mymodule.py
def fun_one( ):
    print("Hello fun - 1")
def fun_two( ):
    print("Hello fun - 2")
```

Program: 8

Output:

```
import mymodule
from mymodule import fun_one

def fun_one( ):
    print("I am your function\n")

fun_one( )
mymodule.fun_two( )
```

```
#mymodule.py
def fun_one( ):
    print("Hello fun - 1")
def fun_two( ):
    print("Hello fun - 2")
```

Program: 9

Output:

```
import mymodule
from mymodule import fun_one

fun_one( )
mymodule.fun_two( )

def fun_one( ):
    print("I am your function\n")
```

```
#mymodule.py
def fun_one( ):
      print("Hello fun - 1")
def fun_two( ):
      print("Hello fun - 2")
```

Program: 10

Output:

```
import mymodule
from mymodule import fun_one

def fun_one( ):
      print("I am your function\n")

fun_one( )
mymodule.fun_two( )
```

Program: 11

Output:

```
from math import factorial as f
def factorial(n):
      print(n)
factorial(5)
print(f(5))
```

Program: 12

Output:

```
from math import factorial as f
def f(n):
      print(n)
f(5)
```

```
#mymodule.py
n = 20
def fun_one( ):
      print("Hello fun - 1")
def fun_two( ):
      print("Hello fun - 2")
```

Program:13

Output:

```
from mymodule import n

print(n)
```

```
#mymodule
__n__ = 20
def fun_one( ):
    print("Hello fun - 1")
def fun_two( ):
    print("Hello fun - 2")
```

Program: 14

```
from mymodule import *

print( __n__ )
```

Output:

IMPORTANT NOTE: When the **from module_name import *** form of import is used to import all the identifiers of a module's namespace, names beginning with double underscores are not imported. Thus, such entitites become inaccessible from within the importing module.

Guess The Output:

1.

```
def sum(n1, n2, n3):
    total = n1 + n2 + n3
    return total

res = sum ([1,2,3])
print(res)
```

2.

```
def sum(n1, n2, n3):
    total = n1 + n2 + n3
    return total

res = __builtins__.sum ([1,2,3])
print(res)
```

```
#grade_calc module
#grade_calc.py

def max(grades):
    largest = 0

    for k in grades:
        if k > 100:
            largest = 100
        elif k > largest:
            largest = k
    return largest

def gradesHighLow(grades):
    return (min(grades), max(grades))
```

Program: 15

#classgrades (main module)

```
from grade_calc import *

class_grades = [86, 72, 94, 102, 89, 76, 96]

low_grade, high_grade = gradesHighLow(class_grades)
print('Highest adjusted grade on the exam was', high_grade)
print('Lowest grade on the exam was', low_grade)

print('The highest grade on exam was', max(class_grades))
print('Actual highest grade on exam was', __builtins__.max(class_grades))
```

Program 16 (stack.py)

#stack module (LIFO)

```
def getStack():
    """ Creates and returns an empty stack. """

    return [ ]

def isEmpty(s):
    """ Returns True if stack empty, otherwise returns False."""

    if s == [ ]:
        return True
    else:
        return False

def top(s):
    """ Returns value of the top item of stack, if stack not empty. Otherwise, returns None."""

    if isEmpty(s):
        return None
    else:
        return s[len(s) - 1]

def push(s, item):
    """ Pushes item on the top of stack. """
    s.append(item)

def pop(s):
    """Returns top of stack if stack not empty. Otherwise, returns None."""
    if isEmpty(s):
```

```
                return None
        else:
                item = s[len(s) - 1]
                del s[len(s) - 1]
                return item
```

#main module - Using Stack - Client Program

import stack

```
mystack = stack.getStack()

for item in range(1,5):
        stack.push(mystack, item)
        print('Pushing', item, 'on stack')

while not stack.isEmpty(mystack):
        item = stack.pop(mystack)
        print('Popping', item, 'from stack')
```

<u>Program 17</u> (Palindrome or Not)

import stack

#welcome

```
print("This program can determine if a given string is a palindrome\n")
print("(Enter return to exit)")
```

#init

```
char_stack = stack.getStack()
empty_string = "
```

#get string from user

```
chars = input( "Enter string to check: ")

while chars != empty_string:
        if len(chars) == 1:
                print('A one letter word is by definition a palindrome\n')
        else:
                #init
                is_palindrome = True
```

```
                #to handle strings of odd length
                compare_length = len(chars) // 2

                #push second half of input string on stack
                for k in range(compare_length, len(chars)):
                        stack.push(char_stack, chars[k])

                #pop chars and compare to first half of string
```

```
            k = 0
            while k < compare_length and is_palindrome:
                  ch = stack.pop(char_stack)
                  if chars[k].lower() != ch.lower():
                        is_palindrome = False

                  k = k + 1

            #display results
            if is_palindrome:
                  print( chars, " is a palindrome\n" )
            else:
                  print( chars, " is not a palindrome" )

      #get next string from user
      chars = input("Enter string to check: " )
```

Output:

```
This program can determine if a given string is a palindrome

(Enter return to exit)
Enter string to check: bool
bool is not a palindrome

Enter string to check: madam
madam is a palindrome

Enter string to check:
>>>
```