CMP4266: Computer Programming UG1
Lab Session 4: Functions and Modules

Objectives

a. Experiment with different features of Python functions, including
    - Passing keyword arguments
    - Changing parameter values
    - Returning multiple values
    - Variable scopes

b. Experiment with different built-in and library functions.

1. Reminder about functions

1.1 Passing keyword arguments

Most programming languages (including Python) supports the passing of arguments by position to the corresponding parameter variables in the function. More specifically the first argument is passed to the first parameter variable, the second argument is passed to the second parameter variable and so on. Consider the following function definition and the call to the function

```
def print_name(first, last):
        print("First name %s " %first)
        print("Last name %s " %last)


fn = "Albert"
ln= "Einstein"
print_name(fn, ln)
```

Output

```
First name Albert
Last name Einstein
```

Here the value of *fn* is passed to the parameter *first* and the value of *ln* is passed to the parameter *last*. However, Python supports a special syntax, known as keyword argument that allows to specify which parameter variable the argument should be passed to. The general format for keyword argument passing is shown below:

```
Function_name(parameter_name=value, ....)
```

When the keyword argument passing is used the order of the argument does not need to match with the order of the parameter variables. Try to call the *print_name* function as shown below and you will get the same output as before.

```
print_name(last="Einstein", first="Albert")
```

Python allows to mix positional arguments with keyword arguments, but in that case positional arguments must appear first followed by keyword arguments. Try to call the *print_name* function as shown below and observe the output.

```
print_name(fn, last="Einstein")
print_name(first="Albert", ln)
```

1.2 Changing parameter values

Changing the value of parameter variables inside a function does not affect the value of the corresponding argument. Try to execute to the following code snippet and observe the output,

```
def change_val(num):
	num += 100
	print("Inside parameter value %d" %num)

val = 100
print("Argument value %d" %val)
change_val(val)
print("Argument value after function call %d" %val)
```

1.3 Returning multiple values

Python supports the return of multiple values from a function. The general format to return multiple values is shown below:

```
return expression1, expression2, ...
```

Consider the following code snippet. The *get_name()* function prompts the user to enter the first name and last name. Entered names are stored as values of the variables *first* and *last* respectively. Then the function returns both the variables.

```
def get_name():
	first = input("Enter first name : ")
	last = input("Enter last name : ")
	return first, last
```

```
f_name, l_name = get_name()

print(f_name)
print(l_name)
```

In the above code *get_name* function is called and two variables (i.e. *f_name* and *l_name*) are used to hold the returned values. The following is the output from executing the above code:

Output

```
Enter first name : Thomas
Enter last name : Hardy
Thomas
Hardy
```

1.4 Variable scopes

A variable created inside a function is local to that function, i.e. the variable can only be accessed from the statements inside the function that comes after the variable declaration. Execute the following code snippet and observe the result.

```
def add_numbers(n1, n2):
        summation = n1 + n2
        print("Here is sum %d" %summation)

add_numbers(10, 5)
```

Output

```
Here is sum 15
```

1.5 Built-In and Library Functions

In this section you will be experimenting with some python built-in library functions. The following table shows a list of executable (nested) function calls using built-in functions. Read the specifications for these built-in functions at the following link.

https://docs.python.org/3/library/functions.html

The first column of the table shows the function calls that should be executed in the Spyder IDE interpreter. First try to think about the possible output for each execution in your mind or on paper and write down the expected output in the middle column. Then execute the functions in the Spyder IDE and write down the observed output in the third column and compare it with your expected output.

Functions	Expected Output	Observed Output
abs(-12)		
abs(min(max(-12, 5), -15))		
min(sum(range(1, 10, 2)),sum(range(2, 10, 2)))		

max(round(25.64,0), round(25.64,2))		
len("This is a test")		

The following table shows a list of executable (nested) function calls using the functions from math module and built-in functions. Read the specifications for these functions at the following link,

https://docs.python.org/3/library/math.html

Then do the similar exercises as the above table. Note that you need to execute the *import math* statement (once) before executing the function calls in the following table.

Functions	**Expected Output**	**Observed Output**
math.sqrt(7)		
math.sqrt(-7)		
max(math.copysign(25, -2.0),sum([-5,-7,-15]))		
min(math.ceil(-4.5), math.floor(-5.2))		
max(math.trunc(-6.6), math.floor(-6.6))		

2. Getting started

2.1 Consider the following Python program. What output will this code produce?

```
def f(x):
   x=10
x=5
f(x)
print(x)
```

2.2 Consider the following Python program. What output will this code produce when run?

```
def mystery1(a, b):
 return a+b
def mystery2(a, b):
 if a > b:
    return a
 else:
    return b
def mystery3(a, b):
 if a < b:
    print(a)
    a = a + 2
    mystery3(a, b)
y = mystery1(10, 8)
print(y)
y = mystery2(10, 8)
print(y)
mystery3(10, 22)
```

3. Lab 4 Submission Exercises

3.1 **Create a calculator program**

Create a program that takes three parameters; two numbers and one string. The string is used to indicate the arithmetic operation to apply on the two numbers. The program output should look like the following:

Output

```
input number 1: 7

input number 2: 5

Select an operator * for multiplication, - for subtraction, + for addition, / for division: *

Results: multiplying 7 by 5 = 35
```

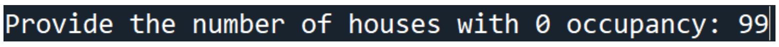
These are user inputs provided during the program runtime

4. The program must use an **if** statement that uses the sign operator string to decide on the arithmetic operation to be done on the two numbers (See above example).
5. The program should returns the answer plus an explanation. For example, if the user inputs 7, then 5 and then *, the program should return "**multiplying 7 by 5 = 35**"
6. The program should be able to call 4 functions to perform 4 arithmetic operations and these are **add**, **subtract**, **multiply** and **divide**.
7. Then create a function **calculator** to implement the actual calculator.
8. **For exercise 3.1 download the lab4_ex3.1.py from Moodle. Complete the prompted areas, and run it to make sure it works.**

Function	Parameters	What does it do?
`add`	`num1, num2` - int type	It returns the sum of num1 and num2.
`subtract`	`num1, num2` - int type	It returns the difference of num1 subtracts num2.
`multiply`	`num1, num2` - int type	It returns the product of num1 and num2.
`divide`	`num1, num2` - int type	It returns the result of num1 divided by num2.

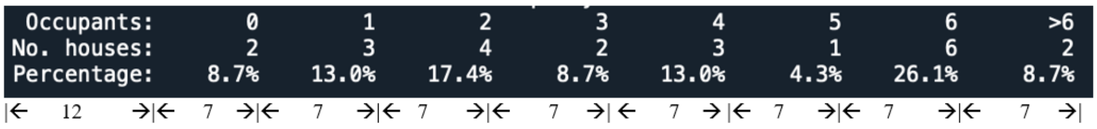
3.2 Designing a program which generates housing occupancy report

1. In this task, you will design a program which display a census report of the numbers and percentages of houses with particular numbers of occupants in a road.
2. When the program starts, it should ask the user 8 times about houses with occupancy of 0 to 6, plus 6+ to count the numbers of houses with particular numbers of occupants.
    - For each question, the user should enter a digit for the number of houses, e.g.:



```
Provide the number of houses with 0 occupancy: 99
```

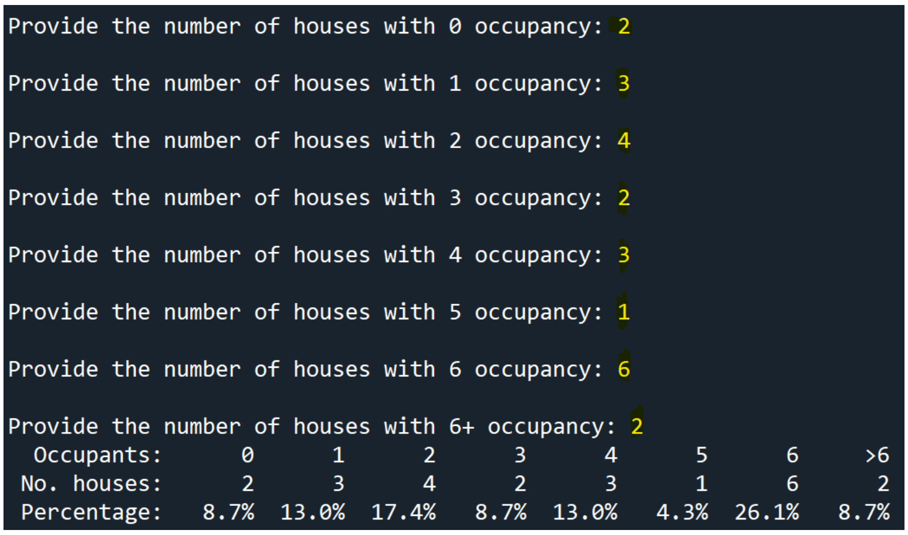
    - Please note there is a space after the colon mark.
    - This step should be implemented in a function named `get_data()`.
    - The function will return all the 8 integer values entered by the user.
3. After user entered the data, the program will calculate the percentage of each occupancy category.
    - For example, if the total house is 35 and the house with 0 occupant is 2, the percentage of houses with 0 occupancy is 5.7%. This is because (2/35) * 100 is 5.7%.
    - This step should be implemented in a function named `cal_percentage()`.
    - The function should take 8 parameters of the house number for each occupancy category. The order of the parameters must be the same as the user entered in step 2.
    - The function will return 8 float values for percentages of each occupancy category in the same order.
4. When the calculation finished, the program outputs the census result as below:



```
 Occupants:        0       1       2       3       4       5       6      >6
No. houses:        2       3       4       2       3       1       6       2
Percentage:     8.7%   13.0%   17.4%    8.7%   13.0%    4.3%   26.1%    8.7%
```

|← 12 →|← 7 →|← 7 →|← 7 →|← 7 →|← 7 →|← 7 →|← 7 →|← 7 →|

    - The output must comply with the width specification given above.
    - Read **this article** to find out more about setting format.
    - This step should be implemented in a function named `display_result()`.
    - The function takes 16 parameters. The first 8 are for the house number of each occupancy category and second 8 are for the corresponding percentages.
    - The output of the function should follow the specification above.
5. To sum up, in the program, it should include definition of the three functions you have created and the necessary codes to run the program. See below for the result of the completed program (user inputs are highlighted in yellow):



```
Provide the number of houses with 0 occupancy: 2

Provide the number of houses with 1 occupancy: 3

Provide the number of houses with 2 occupancy: 4

Provide the number of houses with 3 occupancy: 2

Provide the number of houses with 4 occupancy: 3

Provide the number of houses with 5 occupancy: 1

Provide the number of houses with 6 occupancy: 6

Provide the number of houses with 6+ occupancy: 2
  Occupants:      0      1      2      3      4      5      6     >6
 No. houses:      2      3      4      2      3      1      6      2
 Percentage:   8.7%  13.0%  17.4%   8.7%  13.0%   4.3%  26.1%   8.7%
```

6. **For exercise 3.2 download the lab4_ex3.2.py from Moodle. Complete the prompted areas, and run it to make sure it works.**

4 Moodle submission

At the end of this lab session, zip the python files that you created in Sections 3 (3.1 and 3.2) and then upload the zip file in the Moodle. **Note:** It is important to complete the weekly labs in particular labs 1, 2, 3, 4, 5 and 6 because it contains questions that are part of the coursework. Failure to complete and submit the answers to these labs on the Moodle may mean giving you **zero** on for this assessment component.

The zip file should be named using the following format StudetName_studentID.zip For example: OgertaElezaj_123456789.zip

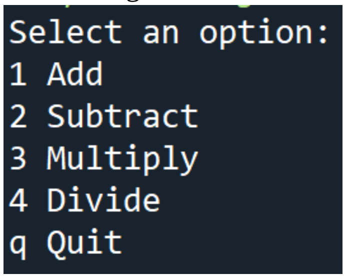
5 Additional exercises

5.2 Write a function named digits that takes one parameter that is a number (int) and returns a count of even digits, a count of odd digits and a count of zero digit. For example: digit (1234567890123) returns (5, 7, 1).

5.3 Write a Python function to check whether a number is perfect or not. (A perfect number is a positive integer that is equal to the sum of its proper positive divisors ex. 28 = 1 + 2 + 4 + 7 + 14)

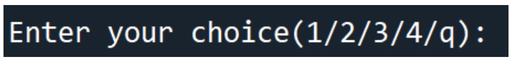
5.4 Modify exercise 3.1:

1. create a function `display_menu()` which displays the following menu:

- 

```
Select an option:
1 Add
2 Subtract
3 Multiply
4 Divide
q Quit
```

2. Then create a function `run_calculator()` to implement the actual calculator.
   - This function uses `display_menu()` to display the menu.
   - It asks for user input to choose an option and acts accordingly:

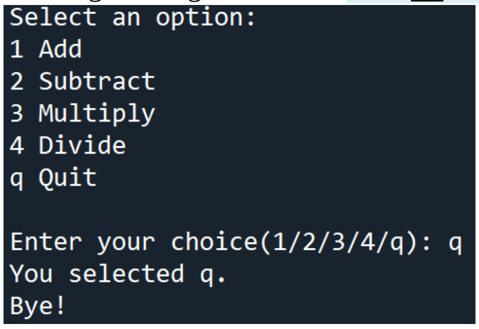


```
Enter your choice(1/2/3/4/q):
```

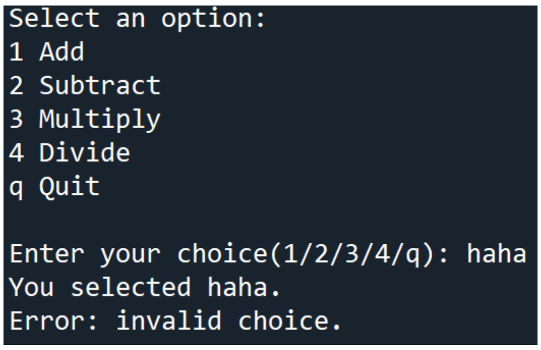
Choice	Action
1/2/3/4	Further asks user to **enter two integer** and prints out the results of the chosen operation.
q	Print outs "Bye!".
entered other choice	Print outs "Error: invalid choice.".

   - For division, it will display an error message when the divisor is 0.
   - For division, the result will be displayed with two decimal places.

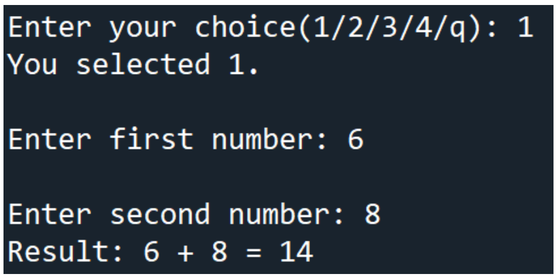
3. Write necessary codes to run the program. See the screenshots below for how the program runs:
   - **Note**. Put your testing/running code under `"if __name__ == "__main__":`
   - Selected "q":



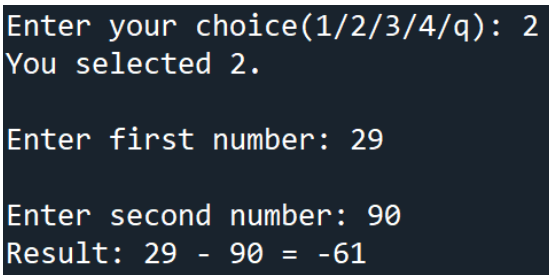
```
Select an option:
1 Add
2 Subtract
3 Multiply
4 Divide
q Quit

Enter your choice(1/2/3/4/q): q
You selected q.
Bye!
```

- Invalid choice: 

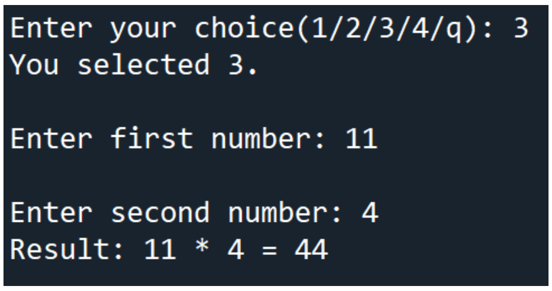
```
Select an option:
1 Add
2 Subtract
3 Multiply
4 Divide
q Quit

Enter your choice(1/2/3/4/q): haha
You selected haha.
Error: invalid choice.
```

- Selected "1": 

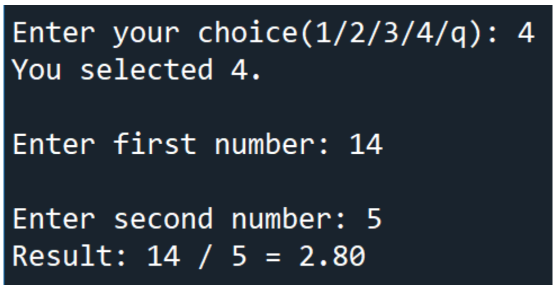
```
Enter your choice(1/2/3/4/q): 1
You selected 1.

Enter first number: 6

Enter second number: 8
Result: 6 + 8 = 14
```

- Selected "2": 

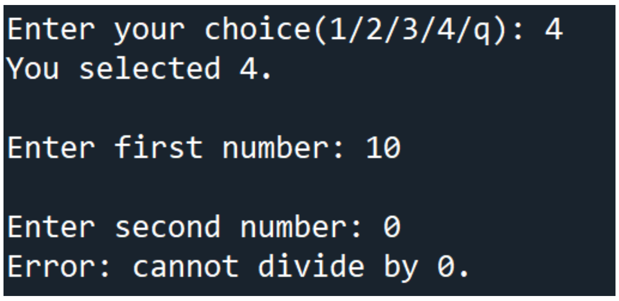
```
Enter your choice(1/2/3/4/q): 2
You selected 2.

Enter first number: 29

Enter second number: 90
Result: 29 - 90 = -61
```

- Selected "3": 

```
Enter your choice(1/2/3/4/q): 3
You selected 3.

Enter first number: 11

Enter second number: 4
Result: 11 * 4 = 44
```

- Selected "4": 

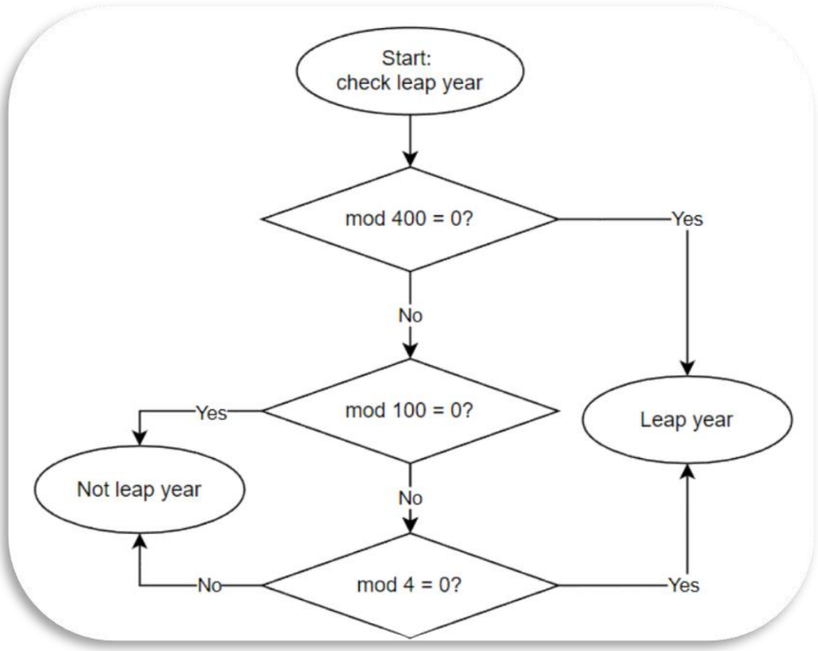
```
Enter your choice(1/2/3/4/q): 4
You selected 4.

Enter first number: 14

Enter second number: 5
Result: 14 / 5 = 2.80
```

- Selected "4" but divide by 0: 

```
Enter your choice(1/2/3/4/q): 4
You selected 4.

Enter first number: 10

Enter second number: 0
Error: cannot divide by 0.
```

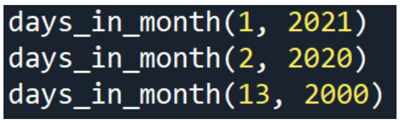
5.5 Create a script named `lab4_library.py`.

1. In **`lab4_library.py`**, create a function named **`is_leap(year)`**.
   - It takes a parameter **`year`** and the expected data type is int.
   - The function will return **`True`** or **`False`**, depends on whether the year is a leap year.
   - The algorithm to check leap year is showed as below:

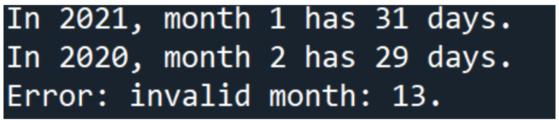


- "mod" is a math operation and the corresponding operator in Python is "**%**". For example, to calculate year mod 400, the Python code is: `year % 400`.

2. In `lab4_library.py`, create another function named `get_days_of_month(month, year)`.
   - The two parameters are both of data type int.
   - Similar to task 6 from Lab 2, this function returns the number of days for the given month.
   - The only difference is that, for February, it will return 29 for a leap year and 28 for a normal year.
3. Create a script named `lab4_days_in_month.py`.
4. In `lab4_days_in_month`, import `lab4_library`.
5. In `lab4_days_in_month`, create a function named `days_in_month(month, year)`.
   - It prints out a message of the days in month.
   - It prints out an error message when month is invalid.
   - See below for the format of the message:



```
days_in_month(1, 2021)
days_in_month(2, 2020)
days_in_month(13, 2000)
```



```
In 2021, month 1 has 31 days.
In 2020, month 2 has 29 days.
Error: invalid month: 13.
```