Aim and Objectives

The aim of this lab is to provide students with a solid understanding of fundamental data structures in Python, such as **lists, tuples, sets,** and **dictionaries**, as well as the use of **docstrings** and **doctests**.

After successful completion of this lab, students should be able to:

- Demonstrate a clear understanding of **lists, tuples, sets,** and **dictionaries** in Python and their appropriate use cases.
- Implement Python programs that efficiently store, access, and manipulate data using these data structures.
- Apply best practices in Python programming, including the use of comments, **docstrings**, and structured code for clarity and maintainability, and employ **doctests** to validate functionality.
- Construct a menu-driven program that allows users to perform operations on different data structures interactively.

1 Data Structures

1.1 List

- A list holds an **ordered collection** of items, and items should be enclosed in **square brackets** [].
- After creating a list, one can add, remove, or search for items inside the list. Therefore, we call a list a **mutable** data type.

1. Since we already talked about Python lists in Lab 03, we are not going to cover the basic functionalities of Python lists.
2. Let's discuss some useful features in Python lists that will be used in calculations related to 2D data structures (**Matrices**)
3. In Python, we can create a Matrix using **Nested Lists**.

$$\begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 8 & 9 & 7 \end{bmatrix}$$

See the following program where we create a matrix as shown above and access elements inside the matrix.

```python
# Python Nested Lists
# CO1010 Lab 05

# Create the matrix using a nested list
matrix = [[1,2,3], [3,4,5], [8,9,7]]

# Checking the size of the matrix
rows = len(matrix)
columns = len(matrix[0])

print(f"The size of the matrix is {rows} x {columns}")

# Retrieve the element at the location 3,3 (Matrix(3,3))
element = matrix[2][2]

print(f"Element at Matrix(3,3) is {element}")

# Retrieve the second row of the matrix
second_row = matrix[1]
print(f"The second row of the matrix is {second_row}")
```

1.2 Tuple

- A tuple holds an **ordered collection** of items, and items should be enclosed in **round brackets ()**.
- After creating a tuple, one **cannot** add, remove, or change items inside the tuple (**read-only**). Therefore, a tuple is an **immutable** data type.

1. Let's write a simple Python script to observe the functionality of Python tuples.

```python
# Python Tuples
# CO1010 Lab 05

# Create a tuple
names_tuple = ("Thilina", "Isuru", "Shakthi", "Anjalee", "Isuri", "Akila")

# Get the first item
first_element = names_tuple[0]

print(f"The first element of the tuple is {first_element}.")

# Count the number of items inside the tuple

print(f"Number of elements: {len(names_tuple)}")

# Iterate through the tuple and print the elements
for name in names_tuple:
    print(name)
```

2. To see more functions related to Python tuple, use the inbuilt **help(tuple)**.

1.3 Set

- A set holds an **unordered collection** of **unique** items, and items should be enclosed in **curly brackets** {}.
- Since the items are unordered, there is **no index** to access items.
- Items **cannot be changed**, but existing items can be removed and new items can be added to the set.

1. Let's write a simple Python script to observe the functionality of Python sets.

```python
# Python Sets
# CO1010 Lab 05

# Create a set
prime_nums = {2,3,5,7}

# Print the set
print(f"The set of prime numbers: {prime_nums}.")

# Iterate through the set and print the elements
for item in prime_nums:
    print(item)

# Add an item to the set
prime_nums.add(11)

# Add multiple items to the set
prime_nums.update([13,17,19])

# Remove an item from the set
prime_nums.remove(3)

print(f"The updated set of prime numbers: {prime_nums}.")
```

2. To see more functions related to Python set, use **help(set)** method.

Pause and Reflect: What is the content and size of the **my_set** variable?

```
>>> my_set = {2,2,2,3,4,5,6,7,7,7}
```

1.4 Dictionary

- A dictionary is a collection of **key-value pairs** enclosed in **curly brackets** {}.
- Each entry contains a key and its corresponding value, separated by a colon.
- Unlike lists and tuples, dictionaries use keys instead of indices to access their elements.

1. Let's write a simple Python script to observe the functionality of Python dictionaries, i.e. **dict**.

```python
# Python Dictionary
# CO1010 Lab 05

# Create a dict
eng_students = {
    "E/22/000" : "Adithya",
    "E/22/470" : "Weerakoon",
    "E/22/500" : "Zabri"
    }

# Iterate through the dictionary and print all keys
for key in eng_students:
    print("Key in dict:" , key)

# Iterate through the dictionary and print all values
for value in eng_students.values():
    print("Value in dict:" , value)

# Get all keys as a list
key_ls = eng_students.keys()

print(f"All keys: {key_ls}")

# Print all items (key-value pairs)
for item in eng_students.items():
    print("Key-value in dict:", item)

# Add an item
eng_students["E/22/274"] = "Rajapaksha"

# Change an item
eng_students["E/22/000"] = "Abeykoon"

# Access an item
print(eng_students["E/22/274"])
student1 = eng_students.get("E/22/000")
print(student1)
```

2. To see more functions related to Python dict, use **help(dict)** method.

2 Documentation

1. Python **docstrings** are documentation strings that are used to document Python modules, functions, classes, or methods.
2. They provide a way to describe the purpose, usage, and behavior of the code to other developers or your future self.
3. Docstrings are enclosed in **triple quotes (either ''' or """)** and placed immediately after the definition of a module, function, class, or method.

4. To understand Python **docstring**, let's create a **module** named **mathfuns** to implement simple mathematical functions. First, create a file named ***mathfuns.py***. Then, include some mathematical functions with documentation inside the file as shown below:

```python
# mathfuns.py module with docstrings
# CO1010 Lab 05

'''
Simple Mathematical Function Module
Example for CO1010: Programming for Engineers I (2024)
'''

def add_nums(n1, n2):
    '''
    A function to add two numbers
    '''
    return n1 + n2

def abs_nums(n):
    '''
    A function to find the absolute value of a number
    '''
    if n < 0:
        return (-1)*n
    else:
        return n
```

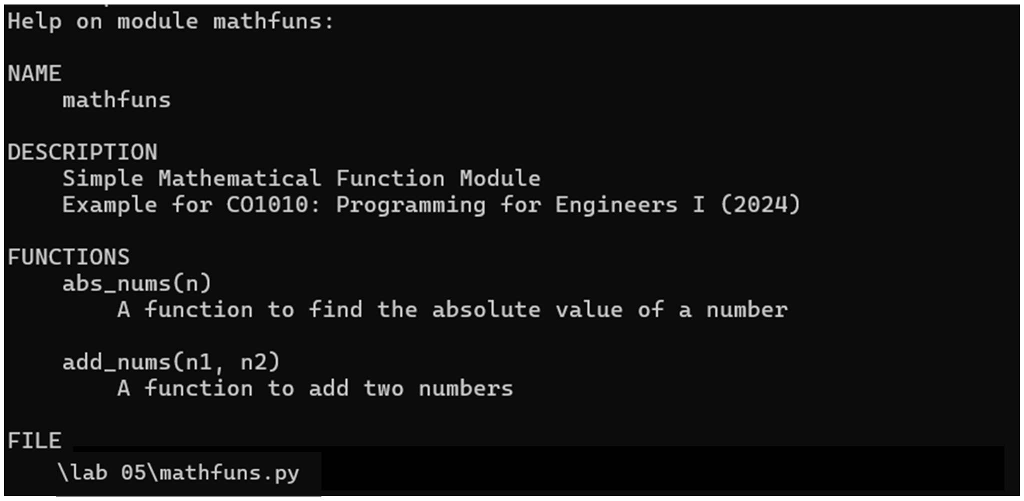
5. Let's view the documentation using the **help()** module.

```python
# Import the module
import mathfuns

# Call the help function
help(mathfuns)
```

Note: If you are using the Python terminal, remember to open the terminal from the same location where you created your mathfuns.py module.

The output from the help function should be similar to below:



```
Help on module mathfuns:

NAME
    mathfuns

DESCRIPTION
    Simple Mathematical Function Module
    Example for CO1010: Programming for Engineers I (2024)

FUNCTIONS
    abs_nums(n)
        A function to find the absolute value of a number

    add_nums(n1, n2)
        A function to add two numbers

FILE
    \lab 05\mathfuns.py
```

Figure 1: Output from help(mathfuns) module

3 Testing

1. It is a good programming practice to test our implemented functions before using them. For this, Python provides a special module called **doctest**.
2. Just like **docstring** inline documentation, we can add inline test cases to the module and test them using the **doctest** module.
3. When writing testcases, we have to think carefully about the expected functionality and the types of arguments that the user will use while calling the functions. Let's add some testcases with example inputs and expected outputs for the two functions in the ***mathfuns.py*** module.

```python
# mathfuns.py module with docstrings and doctests
# CO1010 Lab 05

'''
Simple Mathematical Function Module
Example for CO1010: Programming for Engineers I (2024)
'''

def add_nums(n1, n2):
    '''
    A function to add two numbers
    Tests:
    >>> add_nums(1,2)
    3
    >>> add_nums(-1,-2)
    -3
    '''
    return n1 + n2

def abs_nums(n):
    '''
    A function to find the absolute value of a number
    Tests:
    >>> abs_nums(-1)
    1
    >>> abs_nums(0)
    0
    >>> abs_nums(1)
    1
    '''
    if n < 0:
        return (-1)*n
    else:
        return n
```

4. To run the tests, we have to import and call the **doctest module** in the script containing the modules you are testing. Add the below code snippet to the mathfuns.py script and run the script.

```python
if __name__ == "__main__":
    import doctest
    doctest.testmod(verbose = True)
```

Lab Tasks

Complete the exercises below, show them to an instructor, and get them marked within the lab hours. Make sure to include meaningful comments in your Python scripts.

While peer-learning is highly encouraged in our labs, copying someone else's codes will earn you zero marks for all lab exercises.

1. Create a Python module named ***stringfun.py*** that provides two functions for basic string manipulation ***without using built-in functions***:

   - **count_char(string, char)**:
     - Count the number of occurrences of **char** in **string**.
     - Doctests: Include cases with multiple occurrences, no occurrences, empty strings, etc.
   - **remove(string, index)**:
     - Remove the character at the specified **index** in the **string**.
     - Doctests: Include cases for removing from the start, middle, end, out-of-range indices, and empty strings.

   Include proper **docstrings** to describe each function and use **doctests** to verify their correctness, including corner cases.

2. Create a Python shopping cart program named shopping_cart.py to manage a shopping cart, allowing users to add, remove, and view items, as well as handle product pricing and see unique items in the cart.

   Use the following data structures in your program:

   - Use a **dictionary** to store product names and their prices.
   - Use a **list** to represent the shopping cart where items are added and removed.

   There are 7 menu items to implement in the shopping cart program. You will implement them in parts as described below.

   (a) **Begin with the Basic Functionalities:**
   Implement the following menu options first:
   - **1. Add Item to Cart:** Prompt the user to enter a product name to add to the shopping cart, ensuring the item exists in the product list.
   - **2. Remove Item from Cart:** Prompt the user to enter a product name to remove from the shopping cart, provided the item is already in the cart.
   - **3. View Cart:** Display all items currently in the shopping cart. If the cart is empty, show the message "Your shopping cart is empty."
   - **7. Exit:** Terminate the program with an appropriate exit message.

   Once completed, **demonstrate your code to the instructor** and get these functionalities marked before moving to the next stage.

   (b) **Implement the Checkout Option:** Calculate and display the total price of all items in the shopping cart, and then terminate the program.
   After completing this step, **show your updated program to the instructor** and get it marked.

   (c) **Implement the Additional Functionalities:**
   - **5. Add New Product:** Allow the user to add a new product and its price to the product list.
   - **6. Get Unique Items:** Display unique items in the shopping cart. *(Hint: Use a Python data structure that removes duplicates automatically.)*

   When finished, **present the final version to the instructor** for marking.

The program menu and list of available products should be displayed to the user **until the user terminates or checks out** as shown below [Hint: Use **while True**]:

```
Available Products:
        apple   $50
        banana  $30
        orange  $40
        grapes  $90
        mango   $70

What would you like to do?
        1. Add item to cart
        2. Remove item from cart
        3. View cart
        4. Checkout
        5. Add a new product to the price list
        6. Get unique items in cart
        7. Exit

Enter your choice (1-7): 1
Enter the item you want to add: Apple
Apple added to your cart.

Available Products:
        apple   $50 ...
```

You may use the skeleton code below as a guide:

```python
# Initialize empty shopping cart as a list
shopping_cart = []

# Initialize product prices as a dictionary
product_prices = {
    "apple": 50,
    "banana": 30,
    "orange": 40,
    "grapes": 90,
    "mango": 70
}
def add_item_to_cart(product):
    """
    Add a new item cart if the item exists in the product_prices list
    """
def remove_item(product):
    """
    Add item from cart if the item exists in the cart
    """
def view_cart():
    """
    Display all items currently in the shopping cart
    """
def checkout():
    """
    Calculate and display the total price of all items in the shopping cart
    """
def add_product(product, price):
    """
    Add a new product to the product_price list.
    """

def get_unique_items(cart):
    """
    Get unique items in the shopping cart.
    """

# TODO: Implement the main program using the above functions
```

Submission

Please follow these steps for submission:

1. Complete the tasks and save all your Python scripts.
2. Compress all scripts into a single ZIP folder.
3. Rename the ZIP folder to `E22xxx.ZIP`, where `xxx` represents the last three digits of your student registration number.
4. Submit the ZIP folder to the link provided on the FEeLS course page.

Steps for Creating a ZIP File

If you are unfamiliar with how to create a ZIP file, please follow the steps below:

1. Select the all Python scripts.
2. Right-click on one of the selected files.
3. Choose **Send to → Compressed (zipped) folder**.
4. A new ZIP folder will appear. Rename this folder to `E22xxx.ZIP`, replacing `xxx` with the last three digits of your registration number.