National University of Sciences & Technology (NUST)
School of Natural Sciences (SNS)
Department of Mathematics

CS250: Data Structures and Algorithms

Class: BS Mathematics - 2021

Lab 01:

# Data Structures in Python
# Lists (Mutable Arrays), Tuple (Immutable Arrays), Sets and Dictionaries in Python

Date: 13 September, 2023

Time: 10:00am - 01:00pm

Instructor: Fauzia Ehsan

Lab Engineer: Mehwish Kiran

# Lab 01: Data Structures in Python

**Introduction**
The purpose of this lab is to practice built-in data structures and their corresponding methods in Python.

**Tools/Software Requirement**
Python 3/ VSCode / PyCharm IDE

## Built-in Data Structures:

As the name suggests, **data structures allow us to organize, store, and manage data for efficient access and modification. Data structures are the fundamental constructs around which you build your programs. Each data structure provides a particular way of organizing data so it can be accessed efficiently, depending on your use case. Python ships with an extensive set of data structures in its standard library.**

In this part, we are going to take a look at built-in data structures. There are four types of built-in data structures in Python: list, tuple, set, and dictionary.

### Key terms

- **Sequence**: a data structure in which data is stored and accessed in a specific order.
- **Index**: the location, represented by an integer, of an element in a sequence.
- **Iterable**: able to be broken down into smaller parts of equal size that can be processed in turn. You can loop through any iterable object e.g. strings, lists etc.
- **Slice**: a group of neighboring elements in a sequence.
- **Mutable**: an object that can be changed.

- **Immutable**: an object that cannot be changed. (*Many immutable objects appear mutable because programmers reuse their names for new objects*.) (See Figure 1 for more understanding)
- **List**: a mutable data type in Python that can store many types of data. The most common data structure in Python.
- **Tuple**: an immutable data type in Python that can store many types of data.
- **Range**: a data type in Python that stores integers in a fixed pattern.
- **String**: an immutable data type in Python that stores unicode characters in a fixed pattern. Iterable and indexed, just like other sequences.

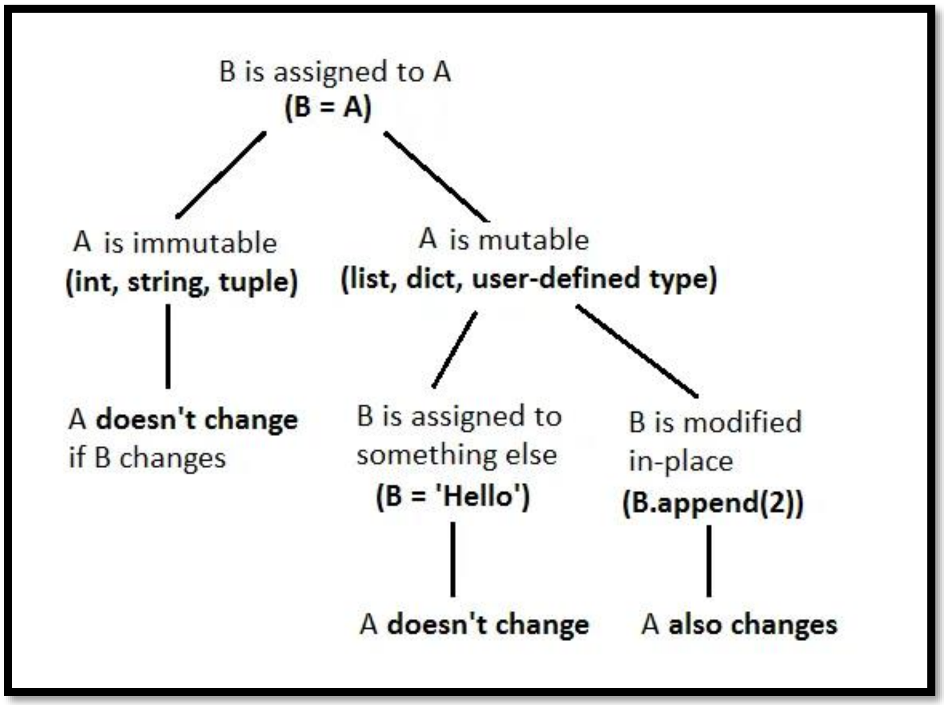


**Figure 1. Mutable vs Immutable**

## Contents

## 1. Python Lists:

Lists gave us an easy way to find sums of numbers, to store things, to sort things, and much more. Structures like this are commonly referred to as collections: collections collect data and encapsulate it. They give us useful methods to manipulate that data. A list in Python is an ordered container of any elements you want indexed by whole numbers starting at 0. Lists are mutable: this means you can add elements, change elements, and remove elements from a list. Meanwhile, tuples are immutable and cannot be changed once they are created.

A list is defined using square brackets and holds data that is separated by commas. The list is mutable and ordered. It can contain a mix of different data types.

See some useful functions for the list here.

## 2. Python Tuple

A **Tuple** is a collection of immutable **Python** objects separated by commas. Tuples are just like lists except it is immutable (basically a read-only list) i.e. we cannot change the elements of a tuple once it is assigned whereas in a list, elements can be changed. The main difference being that tuple manipulation are **faster** than list because tuples are immutable. Since they're immutable, they can be used as keys for a dictionary. Also, Tuples are used whenever you want to return **multiple results** from a function.

#### Creating Tuple with values

#### Example

```
a_tuple = ('East','West','North','South')
print(a_tuple)
```

## 3. Sets

Sets are a lot like lists, but their properties are a bit different. A set is just like a list, but ***no element can appear twice*** and ***it is unordered***. Additionally, they cannot contain mutable elements. Thus, a set cannot contain a list. Sets themselves are mutable.

A set is made using curly braces or from a list:

```
a = {5, 5, 4, 3, 2} # duplicates ignored
print(a) # output: {2, 3, 4, 5}
b = set([5, 5, 4, 3])
print(b) #output: {3, 4, 5}
```

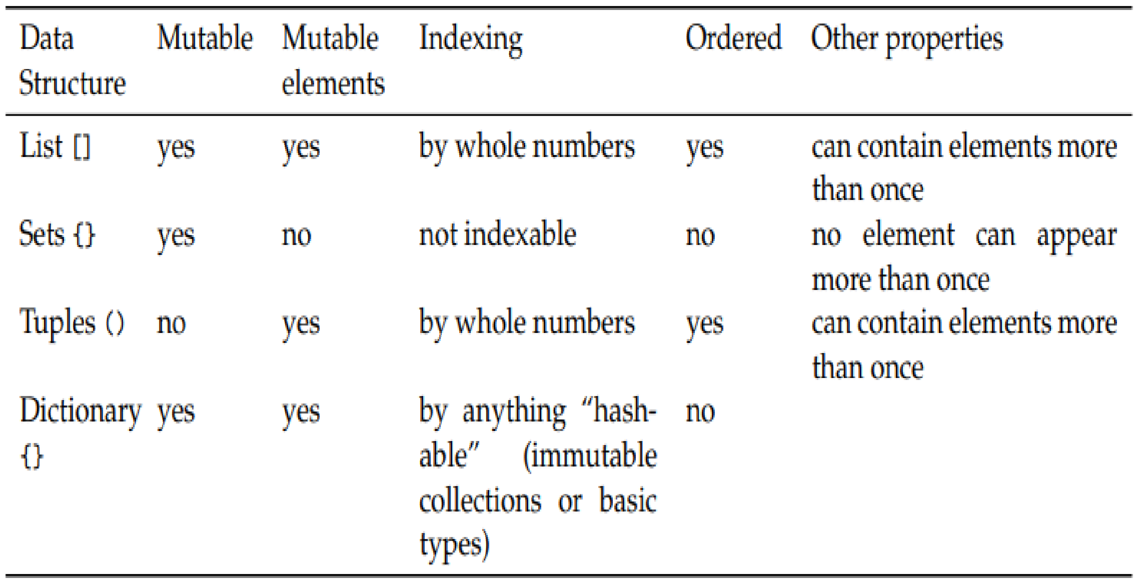
## 4. Dictionaries

Dictionaries are similar to lists, but with a different index system. The elements of a dictionary are stored using a key-value system. A key is used similarly to the indices of a list, but it must be of any **immutable type**. A value is simply the element associated with that key and can be anything. This means that dictionaries are useful for storing elements best identified by a description or name rather than a strict order.

- A dictionary is made using curly braces, followed by listing the key-value pairs that make up the dictionary. Keys and values are separated by colons, while key-value pairs are separated by commas.
- Values can be accessed, changed, or added by using their keys just as you would use the indices of a list.

```
food = {'breakfast': 'burrito','lunch': 'burger', 'dinner': 'tacos'}
print(food)
#{'dinner': 'tacos', 'lunch': 'burger', 'breakfast': 'burrito'}
food['breakfast'] = 'toast'
print(food)
#{'dinner': 'tacos', 'lunch': 'burger', 'breakfast': 'toast'}
food['brunch'] = 'biscuits'
print(food)
#{'dinner': 'tacos', 'brunch': 'biscuits', 'lunch': 'burger',
'breakfast': 'toast'}
```

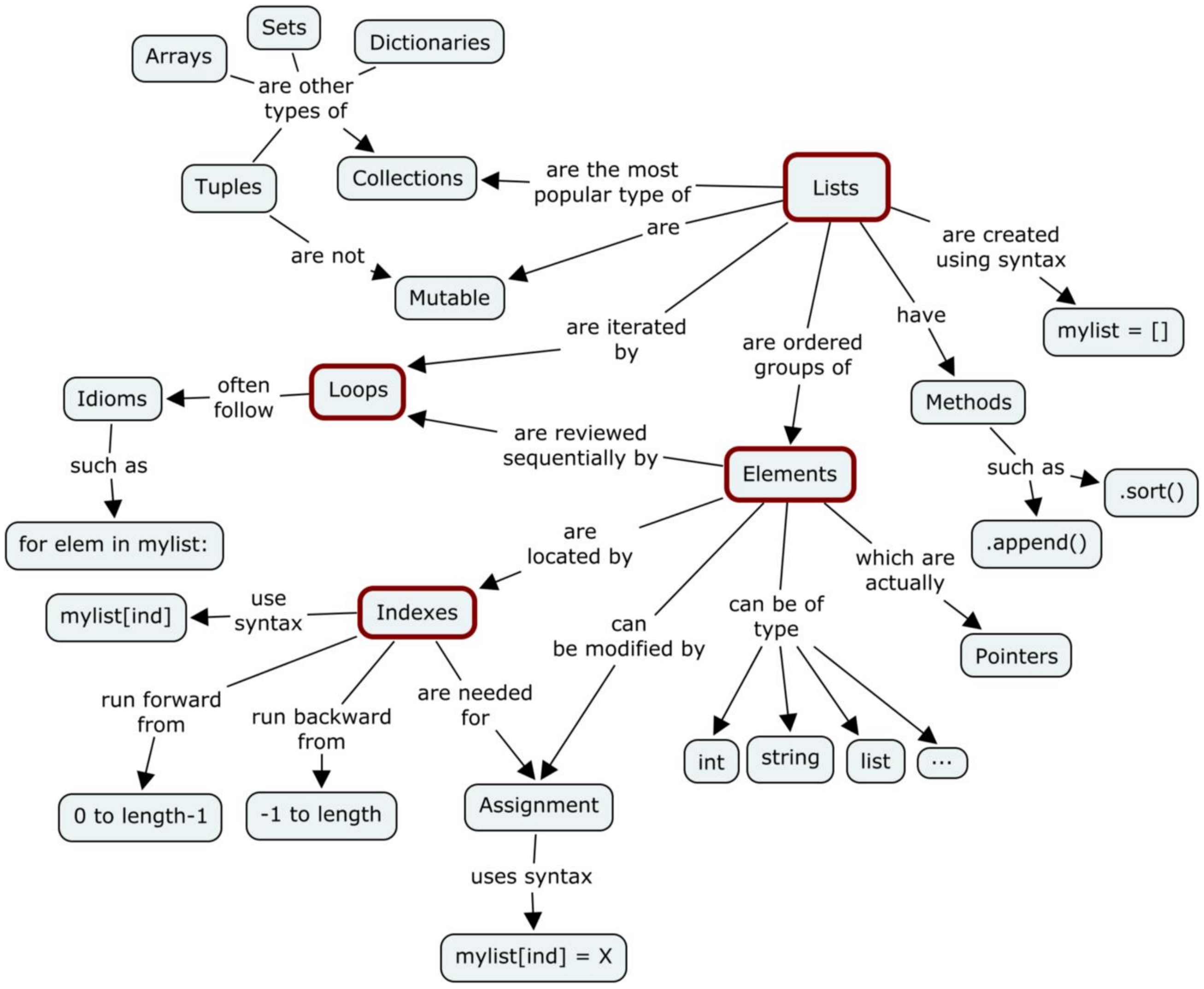
## 5. Collections Summary



| Data Structure | Mutable | Mutable elements | Indexing | Ordered | Other properties |
|---|---|---|---|---|---|
| List [] | yes | yes | by whole numbers | yes | can contain elements more than once |
| Sets {} | yes | no | not indexable | no | no element can appear more than once |
| Tuples () | no | yes | by whole numbers | yes | can contain elements more than once |
| Dictionary {} | yes | yes | by anything "hash-able" (immutable collections or basic types) | no | |

**Table 1 Summary of Data Structures in Python**

**Here is the mind map of all the data types and structures in Python:**



## Task 01 (To be submitted during lab timings):

1. **Create a new file called `long_rivers.py`**
2. **Copy and paste the given list of dicts to `long_rivers.py`**
3. **Solve the tasks seen below**
4. **Remember to take a screenshot/screenshots of the result!**
5. **If you can't fit all solutions on a single screenshot, you can zoom out in VS Code using CMD + - or CTRL + -**

```
rivers = [
 {"name": "Nile", "length": 4157},
 {"name": "Yangtze", "length": 3434},
 {"name": "Murray-Darling", "length": 2310},
 {"name": "Volga", "length": 2290},
 {"name": "Mississippi", "length": 2540},
 {"name": "Amazon", "length": 3915}
]
```

**Task:**

1. **In a for loop, print out each river's name!**
2. **In another for loop, add up and print out the total length of all the rivers!**
3. **Print out every river's name that begins with the letter M !**
4. **The length of the rivers is in miles. Print out every river's length in kilometres! (1 mile is roughly 1.6 km)**

## Task 02 (To be submitted during lab).

Write the following functions:

`overlap()` : **Given two lists, find a list of the elements common to both lists and return it.**

`join()` : **Given two lists, join them together to be one list without duplicate elements and return that list.**

```
E.g.
List 1 is [1.0, 2.0, 4.5]
List 2 is [2.0, 4.5, 5.0]
Overlap is [2.0, 4.5]
Join is [1.0, 2.0, 4.5, 5.0]
```

## Task 03.

**Download the data_structures_food.py file from LMS and run it in VS code. Your goal is to practice manipulating sequences with the Python tools.**

**In data_structures_food.py, there is a <u>list of dictionaries</u> representing different spicy foods.**

```
spicy_foods = [
    {
        "name": "Green Curry",
        "cuisine": "Thai",
        "heat_level": 9,
    },
    {
        "name": "Buffalo Wings",
        "cuisine": "American",
        "heat_level": 3,
    },
    {
        "name": "Mapo Tofu",
        "cuisine": "Sichuan",
        "heat_level": 6,
    },
]
```

**Practice using loops and Python list comprehensions alongside list and dict methods to solve these deliverables.**

**(Note: if you don't know list comprehensions just go with loops)**

### 1. get_names()

Define a function **get_names()** that takes a list of spicy_foods and returns a list of strings with the names of each spicy food.

```
get_names(spicy_foods)
# => ["Green Curry", "Buffalo Wings", "Mapo Tofu"]
```

## 2. get_spiciest_foods()

Define a function **get_spiciest_foods()** that takes a list of spicy_foods and returns a **list of dictionaries** where the heat level of the food is greater than 5.

## 3. get_spiciest_foods(spicy_foods)

```
# => [{"name": "Green Curry", "cuisine": "Thai", "heat_level": 9}, {"name": "Mapo Tofu", "cuisine": "Sichuan", "heat_level": 6}]
```

## 4. print_spicy_foods()

Define a function **print_spicy_foods()** that takes a list of spicy_foods and output to the terminal each spicy food in the following format using print():

Buffalo Wings (American) | Heat Level: 🌶🌶🌶.

HINT: you can use times (*) with a string to produce the correct number of "🌶" emojis.

For example:

```
"hello" * 3 == "hellohellohello"
# => True
```

## 5. print_spicy_foods(spicy_foods)

```
# => Green Curry (Thai) | Heat Level: 🌶🌶🌶🌶🌶🌶🌶🌶🌶
# => Buffalo Wings (American) | Heat Level: 🌶🌶🌶
# => Mapo Tofu (Sichuan) | Heat Level: 🌶🌶🌶🌶🌶🌶
```

## 6. get_spicy_food_by_cuisine()

Define a function **get_spicy_food_by_cuisine()** that takes a list of spicy_foods and a string representing a cuisine, and returns a single dictionary for the spicy food whose cuisine matches the cuisine being passed to the method.

```
get_spicy_food_by_cuisine(spicy_foods, "American")
# => {"name": "Buffalo Wings", "cuisine": "American",
"heat_level": 3}
```

7. **get_spicy_food_by_cuisine(spicy_foods, "Thai")**

```
# => {"name": "Green Curry", "cuisine":
"Thai", "heat_level": 9}
```

## 8. print_spiciest_foods()

Define a function **print_spiciest_foods()** that takes a list of spicy_foods and outputs to the terminal ONLY the spicy foods that have a heat level greater than 5, in the following format:

```
Buffalo Wings (American) | Heat Level: 🌶️🌶️🌶️.
```

Try to use functions you've already written to solve this!

## 9. print_spiciest_foods(spicy_foods)

```
# => Green Curry (Thai) | Heat Level:
🌶️🌶️🌶️🌶️🌶️🌶️🌶️🌶️🌶️

# => Mapo Tofu (Sichuan) | Heat Level:
🌶️🌶️🌶️🌶️🌶️🌶️
```

## 10. get_average_heat_level()

Define a function average_heat_level() that takes a list of spicy_foods and returns an integer representing the average heat level of all the spicy foods in the array. Recall that to derive the average of a collection, you need to calculate the total and divide number of elements in the collection.

```
average_heat_level(spicy_foods)

# => 6
```

## 11. create_spicy_food()

Define a function **create_spicy_food()** that takes a list of spicy_foods and a new spicy_food and returns the original list with the new spicy_food added.

**Example:**

```
create_spicy_food(
    spicy_foods,
    {
        'name': 'Griot',
        'cuisine': 'Haitian',
        'heat_level': 10,
    }
)

# => [
# =>     {
# =>         "name": "Green Curry",
# =>         "cuisine": "Thai",
# =>         "heat_level": 9,
# =>     },
# =>     {
# =>         "name": "Buffalo Wings",
# =>         "cuisine": "American",
# =>         "heat_level": 3,
# =>     },
# =>     {
# =>         "name": "Mapo Tofu",
# =>         "cuisine": "Sichuan",
# =>         "heat_level": 6,
# =>     },
# =>     {
# =>         'name': 'Griot',
# =>         'cuisine': 'Haitian',
# =>         'heat_level': 10,
# =>     },
# => ]
```

**Check all the functions by calling in main function, submit your work as first_name_filenameTask03.py.**

**Add screenshots where necessary.**

## Grade Criteria

This lab is graded. Min marks: 0. Max marks: 30

| Tasks Shown during Lab (At least 2 Tasks must be shown to LE during lab working) 10 marks | Modern Tool Usage 5 marks | Lab Ethics 5 Marks | Lab Report and Tasks Final Submission 5 marks | Lab Viva/Quiz 5 Marks |
|---|---|---|---|---|
| | | | | |
| Total Marks: 30 | Obtained Marks: _____ | | | |

## Happy Coding!

## Deliverables

**Comment** your program heavily. Intelligent comments and a clean, readable formatting of your code account for 20% of your grade.

Copy paste your code and screen shot of console window as a solution of each task Name your submission file as given below and submit this file in **PDF format** on LMS before the deadline. At the end of each lab or in the next lab, there will be a viva related to the tasks. The viva has a weightage of 50% of total lab marks. You must show the implementation of the tasks in the designing tool, along with your complete document to get your work graded.

**Submit all your codes and report in one zip file as `firstname_secondname_lab01.zip`**

**Name – Registration No. – Section**

**Note:** Students are required to upload the lab on LMS before deadline.

Use proper indentation and comments. Lack of comments and indentation will result in deduction of marks.

Useful link:

- Python List/Array Methods - W3Schools
- Python Dictionary Methods - W3Schools
- When to Use a List Comprehension in Python - Real Python
- https://docs.python.org/3.4/library/stdtypes.html#tuple