Programming Mental Health Data in Python

Course Content

- Python Basics
- Python Data Structures
- Looping and Function in Python
- Importing Datasets
- Data Wrangling
- Exploratory Data Analysis
- Data Visualization Using Python
- Univariate Analysis
- Model evaluation and Refinement

2.0 Python Data Structures

A data structure is a way of organizing and storing data in memory to allow for efficient access and manipulation

The following data structures are supported in python

- Lists
- Tuples
- Dictionaries
- Sets
- Strings

LISTS

Lists are ordered collections of elements, which can be of any data type.

Lists are mutable (elements can be modified after creation)

Lists are versatile and used for storing sequences of data

Creating a list

```python
# Creating a list
my_list = [1, 2, 3, 4, 5]
print(my_list)  # Output: [1, 2, 3, 4, 5]
```

List Operations

- Accessing elements in a list using indexing and slicing.
- Modifying elements using assignment, append(), extend(), insert(), remove(), pop(), and clear() methods.
- Combining lists using concatenation and list comprehensions.

```python
# Accessing elements
print(my_list[0])    # Output: 1
print(my_list[1:3])  # Output: [2, 3]

# Modifying elements
my_list[0] = 10      # Modify the first element
print(my_list)       # Output: [10, 2, 3, 4, 5]



# List methods

my_list.append(6)    # Append an element to the end
print(my_list)       # Output: [10, 2, 3, 4, 5, 6]
my_list.insert(1, 20)   # Insert 20 at index 1
print(my_list)          # Output: [10, 20, 2, 3, 4, 5, 6]

my_list.remove(3)       # Remove the first occurrence of 3
print(my_list)          # Output: [10, 20, 2, 4, 5, 6]

my_list.sort()          # Sort the list
print(my_list)          # Output: [2, 4, 5, 6, 10, 20]

# Copying lists
copy_of_list = my_list.copy()
print(copy_of_list)     # Output: [2, 4, 5, 6, 10, 20]

# Combining lists
another_list = [7, 8, 9]
combined_list = my_list + another_list
print(combined_list)  # Output: [10, 2, 3, 4, 5, 6, 7, 8, 9]
```

Iterating over Lists:

- Using for loops and list comprehensions to iterate over lists.

```
# Iterating over lists
my_list = [2,4,5,6,10,20]

for num in my_list:
    print(num)

# List comprehension
squared_nums = [num ** 2 for num in my_list]
print(squared_nums)      # Output: [4, 16, 25, 36, 100, 400]
```

List Manipulation Techniques:

- Filtering, mapping, and reducing lists using functional programming techniques.

```
my_list = [2, 4, 5, 6, 10, 20]
# Filtering lists
even_nums = [num for num in my_list if num % 2 == 0]
print(even_nums)        # Output: [2, 4, 6, 10, 20]

# Mapping lists
double_nums = [num * 2 for num in my_list]
print(double_nums)      # Output: [4, 8, 10, 12, 20, 40]

# Reducing lists
total_sum = sum(my_list)
print(total_sum)        # Output: 47
```

More Advanced Techniques using functools

```
# Nested lists
nested_list = [[1, 2], [3, 4], [5, 6]]
flattened_list = [num for sublist in nested_list for num in sublist]
print(flattened_list)   # Output: [1, 2, 3, 4, 5, 6]

# List comprehension with conditionals
# Functional programming with lists
my_list = [1,2,3,4,5,6,7,8,9,10]
filtered_nums = [num for num in my_list if num > 5]
print(filtered_nums)    # Output: [6, 7, 8, 9, 10]

# Functional programming with lists
my_list = [1,2,3,4,5]
from functools import reduce
product = reduce(lambda x, y: x * y, my_list)
print(product)    #Output : 120
```

TUPLES

- Tuples are ordered collections of elements, similar to lists, but they are immutable (cannot be changed or reassigned)
- Tuples have limited methods due to their immutability.

```
# Creating a tuple
my_tuple = (1, 2, 3, 4, 5)
print(my_tuple)  # Output: (1, 2, 3, 4, 5)


# Accessing elements
print(my_tuple[0])    # Output: 1
print(my_tuple[1:3])  # Output: (2, 3)


# Tuple methods
my_tuple = (1, 2, 3, 2, 4, 5, 2)
print(my_tuple.count(2))   # Output: 3 (count occurrences of 2)
print(my_tuple.index(4))   # Output: 4 (index of the first occurrence of
4)

# Packing and unpacking
coordinates = (10, 20, 30)
x, y, z = coordinates
```

```python
print(x, y, z)  # Output: 10 20 30

# Iterating over a tuple
for item in my_tuple:
    print(item)

# Attempting to modify a tuple (will raise an error)
my_tuple[0] = 100  # TypeError: 'tuple' object does not support item
assignment
```

DICTIONARIES

Dictionaries are unordered collections of key-value pairs in Python.

Dictionaries are created using curly braces {}, with key-value pairs separated by colons:

```python
# Creating a dictionary
my_dict = {'name': 'Alice', 'age': 30, 'city': 'New York'}
print(my_dict)  # Output: {'name': 'Alice', 'age': 30, 'city': 'New York'}


# Accessing elements
print(my_dict['name'])  # Output: Alice
print(my_dict['age'])   # Output: 30


# Modifying elements
my_dict['age'] = 35
print(my_dict)  # Output: {'name': 'Alice', 'age': 35, 'city': 'New York'}

# Adding new elements
my_dict['job'] = 'Engineer'
print(my_dict)  # Output: {'name': 'Alice', 'age': 35, 'city': 'New York',
'job': 'Engineer'}
```

Dictionary Methods

Dictionaries have various built-in methods for common operations, such as keys (), values (), items (), get (), pop (), and update ().

```
# Dictionary methods
print(my_dict.keys())    # Output: dict_keys(['name', 'age', 'city',
'job'])
print(my_dict.values())  # Output: dict_values(['Alice', 35, 'New York',
'Engineer'])
print(my_dict.items())   # Output: dict_items([('name', 'Alice'), ('age',
35), ('city', 'New York'), ('job', 'Engineer')])

print(my_dict.get('age'))  # Output: 35
my_dict.pop('job')         # Remove 'job' key-value pair
print(my_dict)             # Output: {'name': 'Alice', 'age': 35, 'city':
'New York'}

my_dict.update({'gender': 'Female', 'age': 30})  # Update multiple key-
value pairs
print(my_dict)                                  # Output: {'name':
'Alice', 'age': 30, 'city': 'New York', 'gender': 'Female'}
```

Iterating Over a Dictionary

```
for key in my_dict:
    print(key, ':', my_dict[key])

# Using items() method for key-value pairs
for key, value in my_dict.items():
    print(key, ':', value)
```

Dictionary Comprehensions:

Like lists, dictionaries support comprehensions for concise creation and transformation

```
# Dictionary comprehension
squared_values = {num: num**2 for num in range(1, 6)}
print(squared_values)  # Output: {1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
```

Nesting Dictionaries

```

# Nesting dictionaries
student1 = {'name': 'Alice', 'age': 20}
student2 = {'name': 'Bob', 'age': 22}
students = {'student1': student1, 'student2': student2}
```

```
print(students)  # Output: {'student1': {'name': 'Alice', 'age': 20},
'student2': {'name': 'Bob', 'age': 22}}
```

SETS

Sets are unordered collections of unique elements in Python.

They are similar to mathematical sets and support various set operations like union, intersection, and difference.

Sets are created using curly braces {} or the set() constructor.

```
# Creating a set
my_set = {1, 2, 3, 4, 5}
print(my_set)  # Output: {1, 2, 3, 4, 5}


# Set operations
my_set.add(6)        # Add an element to the set
print(my_set)        # Output: {1, 2, 3, 4, 5, 6}

my_set.remove(3)     # Remove an element from the set
print(my_set)        # Output: {1, 2, 4, 5, 6}

print(2 in my_set)   # Output: True (Check membership)

other_set = {4, 5, 6, 7, 8}
union_set = my_set.union(other_set)
print(union_set)     # Output: {1, 2, 4, 5, 6, 7, 8}

intersection_set = my_set.intersection(other_set)
print(intersection_set)  # Output: {4, 5, 6}
```

Set Methods

```
# Set methods
my_set.update({7, 8, 9})  # Update the set with another set
print(my_set)             # Output: {1, 2, 4, 5, 6, 7, 8, 9}

my_set.discard(5)          # Discard an element from the set (if present)
print(my_set)              # Output: {1, 2, 4, 6, 7, 8, 9}

my_set.clear()             # Clear all elements from the set
print(my_set)          # Output: set()
```

Iterating over Sets and Set comprehension

```
# Iterating over a set
for item in my_set:
    print(item)
# Set comprehension
squared_set = {num**2 for num in my_set}
print(squared_set)  # Output: {1, 4, 9, 16, 25}
```

STRINGS

Strings are sequences of characters enclosed within single quotes (') or double quotes (").

They are immutable, meaning they cannot be modified after creation.

```
# Creating strings
string1 = 'Hello, world!'
string2 = "Python programming"
print(string1)  # Output: Hello, world!
print(string2)  # Output: Python programming


# String operations
concatenated_string = string1 + ' ' + string2
print(concatenated_string)  # Output: Hello, world! Python programming

repeated_string = string1 * 3
print(repeated_string)      # Output: Hello, world!Hello, world!Hello,
world!

print(string1[0])           # Output: H (accessing individual characters)
print(string2[7:18])        # Output: programming (slicing)
```

String Methods

```
# String methods
print(string1.upper())          # Output: HELLO, WORLD!
print(string2.lower())          # Output: python programming
```

```python
trimmed_string = string1.strip()    # Remove leading and trailing
whitespace
print(trimmed_string)           # Output: Hello, world!

words = string2.split()            # Split string into a list of words
print(words)                     # Output: ['Python', 'programming']

joined_string = '-'.join(words)    # Join list of words into a single
string
print(joined_string)             # Output: Python-programming

replaced_string = string1.replace('Hello', 'Hi')  # Replace substring
print(replaced_string)           # Output: Hi, world!
```

String Formating

```python
# String formatting
name = 'Alice'
age = 30
formatted_string = "My name is {} and I am {} years old.".format(name, age)
print(formatted_string)         # Output: My name is Alice and I am 30 years old.

f_string = f"My name is {name} and I am {age} years old."  # f-string (Python 3.6+)
print(f_string)
```

String Escaping

```python
# String escaping
escaped_string = 'This is a single quote: \' and this is a backslash: \\'
print(escaped_string) # Output: This is a single quote: ' and this is a backslash: \
```

String Concatenation

```python
# String concatenation vs. joining
strings = ['Python', 'is', 'awesome']
concatenated_string = ' '.join(strings) # Efficient way to join strings
print(concatenated_string)              # Output: Python is awesome
```

String indexing and slicing

```python
# String indexing and slicing
string = 'Hello, world!'
print(string[0])         # Output: H
print(string[-1])        # Output: !
print(string[7:12])      # Output: world
```