PYTHON PROGRAMMING  
LAB MANUAL

Fundamentals of Python Data Types and Operations

Course: Python Programming  
Date: May 30, 2025

TABLE OF CONTENTS

1. Basic Data Types and Type Conversion...........3

2. String Operations..............................5

3. Lists Operations...............................7

4. Tuples Operations..............................9

5. Sets Operations................................11

6. Summary and Conclusion.........................13

LAB 1: BASIC DATA TYPES AND TYPE CONVERSION

Source File: start.ipynb

This lab introduces Python's fundamental data types including integers, floats, complex numbers,   
strings, booleans, None type, and lists. Students will learn to check variable types using the   
type() function and perform type conversions between different data types. The lab also covers   
basic user input handling and arithmetic operations.  
  
Learning Objectives:  
• Understand Python's basic data types  
• Learn type checking with type() function  
• Practice type conversion methods  
• Handle user input and perform basic operations

**Python Code:**

# Basic Data Types  
a = 5   
print(a)  
print(type(a))  
  
b = 3j  
print(b)  
print(type(b))  
  
c = 222.22  
print(c)  
print(type(c))  
  
d = "hello world"  
print(d)  
print(type(d))  
  
e = True  
print(e)  
print(type(e))  
  
f = False  
print(f)  
print(type(f))  
  
g = None  
print(g)  
print(type(g))  
  
h = [1, 2, 3, 4, 5]  
print(h)  
print(type(h))  
  
print('\n')  
  
# Type Conversion  
d = int(c)  
print(d)  
print(type(d))  
  
e = float(a)  
print(e)   
print(type(e))  
  
print('\n')  
  
cc = complex(a)  
print(cc)  
print(type(cc))  
  
# User Input and Basic Operations  
aa = input("Enter a number: ")  
ss = int(aa)  
print(ss+5)  
  
print('\n')  
  
bb = input("Enter a number: ")  
kk = int(bb)  
print(kk%5)  
  
print('\n')  
  
dd = input("Enter a number: ")  
mm = int(dd)  
print(mm/5)  
  
ff = input("Enter a number: ")  
nn = int(ff)  
print(nn\*5)

**Summary and Analysis:**

Key Takeaways:  
• Python supports multiple data types: int, float, complex, str, bool, NoneType, list  
• Use type() function to check variable types  
• Type conversion functions: int(), float(), complex(), str()  
• input() function returns string by default, requiring conversion for numerical operations  
• Python is dynamically typed - variables can change types during execution

LAB 2: STRING OPERATIONS

Source File: second.ipynb

This lab focuses on string manipulation in Python. Students will learn various string methods   
for case conversion, text cleaning, replacement, splitting, and counting occurrences. These   
operations are fundamental for text processing and data cleaning tasks.  
  
Learning Objectives:  
• Master string case conversion methods  
• Learn string manipulation techniques  
• Practice text processing operations  
• Understand string method chaining

**Python Code:**

# Getting User Input  
a = input("Enter a String: ")  
print(a)  
b = input("Enter a String: ")  
print(b)  
  
# String Case Conversion Methods  
print(a.upper())  
  
print(a.lower())  
  
print(a.capitalize())  
  
print(a.strip())  
  
# String Manipulation Methods  
print(a)  
  
print(a.replace("a", "b"))  
  
print(a.split("a"))  
  
print(a.split(" "))  
  
print(a.count("a"))

**Summary and Analysis:**

Key Takeaways:  
• String methods for case conversion: upper(), lower(), capitalize(), strip()  
• String manipulation methods: replace(), split(), count()  
• These methods return new strings (strings are immutable)  
• strip() removes whitespace from beginning and end  
• split() creates lists from strings based on delimiters

LAB 3: LISTS OPERATIONS

Source File: list.ipynb

This lab covers Python lists, which are ordered, mutable collections that can hold different   
data types. Students will learn list creation, indexing, slicing, concatenation, and various   
list methods essential for data manipulation.  
  
Learning Objectives:  
• Understand list creation and properties  
• Master list indexing and slicing  
• Learn list modification methods  
• Practice list concatenation and extension

**Python Code:**

# Creating Lists  
my\_list = ['orange', 'mango', 'banana', 'apple']  
print(my\_list)  
print(type(my\_list))  
  
# List Slicing  
print(my\_list[3:5])  
  
# Creating and Concatenating Lists  
my\_list2 = ['two','three']  
  
print(\*(my\_list + my\_list2), sep ='\n')  
  
# Adding Elements to Lists  
my\_list2.append('five')  
  
my\_list2.insert(0, 'one')  
my\_list2.insert(1, 'two')  
  
# List Methods - Count and Index  
print (my\_list2.count('five'))  
print (my\_list2.count('one'))  
print (my\_list2.count('two'))  
  
print (my\_list2.index('five'))  
  
# Extending Lists  
my\_list.extend(my\_list2)  
print(my\_list)

**Summary and Analysis:**

Key Takeaways:  
• Lists are ordered, mutable collections enclosed in square brackets  
• List slicing syntax: list[start:end] (end is exclusive)  
• Methods for adding elements: append(), insert(), extend()  
• Methods for finding elements: count(), index()  
• Lists can be concatenated with + operator  
• Use \* operator with print() for unpacking list elements

LAB 4: TUPLES OPERATIONS

Source File: tuples.ipynb

This lab covers Python tuples, which are ordered, immutable collections. Unlike lists, tuples   
cannot be modified after creation, making them useful for storing data that shouldn't change.   
Students will learn tuple operations and conversions.  
  
Learning Objectives:  
• Understand tuple immutability and use cases  
• Learn tuple creation and accessing methods  
• Practice tuple methods and operations  
• Master conversion between tuples and lists

**Python Code:**

# Tuple Basics  
my\_tuple = ('orange', 'mango', 'banana', 'apple')  
print("\n--- Tuple Basics ---")  
print(my\_tuple)  
print(type(my\_tuple))  
print(len(my\_tuple))  
  
# Accessing Tuple Elements  
print("\n--- Accessing Elements ---")  
print(my\_tuple[0])  
print(my\_tuple[-1])  
print(my\_tuple[1:3])  
  
# Modifying Tuples (Converting to List and Back)  
duplicat\_list = list(my\_tuple)  
  
duplicat\_list.append('kiwi')  
duplicat\_list.insert(0, 'xyz')  
  
updated\_tuple = tuple(duplicat\_list)  
print(updated\_tuple)  
  
# Tuple Methods  
print("\n--- Tuple Methods ---")  
repeated\_tuple = (1, 2, 3, 1, 2, 1)  
print(repeated\_tuple.count(1))  
print(repeated\_tuple.index(2))  
  
# Tuple Concatenation  
print("\n--- Tuple Concatenation ---")  
tuple1 = (1, 2, 3)  
tuple2 = ('a', 'b', 'c')  
combined = tuple1 + tuple2  
print(combined)  
  
# Nested Tuples  
print("\n--- Nested Tuples ---")  
nested = ((1, 2), ('a', 'b'), (True, False))  
print(nested)  
print(nested[1][0])  
  
# Type Conversions  
print("\n--- Conversions ---")  
list\_to\_convert = ['one', 'two', 'three']  
  
converted\_tuple = tuple(list\_to\_convert)  
print(f"List {list\_to\_convert} converted to tuple: {converted\_tuple}")  
back\_to\_list = list(converted\_tuple)  
print(f"Tuple converted back to list: {back\_to\_list}")  
  
# Printing Tuple Elements  
print("\n--- Printing Tuple Elements Line by Line ---")  
print(\*my\_tuple, sep='\n')

**Summary and Analysis:**

Key Takeaways:  
• Tuples are ordered, immutable collections enclosed in parentheses  
• Cannot modify tuples directly - convert to list for modifications  
• Limited methods: count() and index()  
• Support concatenation with + operator  
• Can contain nested structures  
• Useful for fixed data that shouldn't change  
• Convert between tuple() and list() functions

LAB 5: SETS OPERATIONS

Source File: set.ipynb

This lab covers Python sets, which are unordered collections of unique elements. Sets are   
useful for removing duplicates, membership testing, and mathematical set operations. Students   
will also learn about frozensets.  
  
Learning Objectives:  
• Understand set properties and unique element constraint  
• Learn set creation and modification methods  
• Master mathematical set operations  
• Practice with frozensets for immutable collections

**Python Code:**

# Creating Sets  
print("\n--- Creating Sets ---")  
set1 = {1, 2, 3, 4}  
set2 = set(["apple", "banana", "cherry"])  
empty\_set = set()  
print(f"Set1: {set1}")  
print(f"Set2: {set2}")  
print(f"Empty Set: {empty\_set}")  
  
# Adding Elements to Sets  
print("\n--- Adding Elements ---")  
set1.add(5)  
set1.update([6, 7, 8])  
print(f"Set1 after adding elements: {set1}")  
  
# Removing Elements from Sets  
print("\n--- Removing Elements ---")  
set1.discard(8)  
set1.remove(7)  
print(f"Set1 after removing elements: {set1}")  
  
# Accessing Elements in Sets  
print("\n--- Accessing Elements ---")  
for item in set2:  
 print(item)  
  
# Set Operations  
print("\n--- Set Operations ---")  
set\_a = {1, 2, 3, 4}  
set\_b = {3, 4, 5, 6}  
print(f"Union: {set\_a | set\_b}")  
print(f"Intersection: {set\_a & set\_b}")  
print(f"Difference (A - B): {set\_a - set\_b}")  
  
# Clearing Sets  
print("\n--- Clearing a Set ---")  
set1.clear()  
print(f"Set1 after clearing: {set1}")  
  
# Frozensets (Immutable Sets)  
print("\n--- Frozenset (Immutable Set) ---")  
frozen = frozenset([1, 2, 3])  
print(f"Frozenset: {frozen}")

**Summary and Analysis:**

Key Takeaways:  
• Sets store unique elements only (automatically remove duplicates)  
• Created with curly braces {} or set() function  
• Methods for adding: add(), update()  
• Methods for removing: discard(), remove(), clear()  
• Set operations: union (|), intersection (&), difference (-)  
• Sets are unordered - no indexing  
• Frozensets are immutable versions of sets  
• Useful for membership testing and mathematical operations

COURSE SUMMARY AND CONCLUSION

This comprehensive lab manual covered the fundamental data structures and operations in Python programming.   
Through hands-on exercises, students have gained practical experience with:  
  
CORE CONCEPTS MASTERED:  
• Data Types: Understanding integers, floats, complex numbers, strings, booleans, and None type  
• Type Conversion: Converting between different data types using built-in functions  
• String Operations: Text manipulation, case conversion, and string methods  
• Lists: Mutable sequences with indexing, slicing, and modification methods  
• Tuples: Immutable sequences for fixed data storage  
• Sets: Unique collections with mathematical operations  
  
PROGRAMMING SKILLS DEVELOPED:  
• Variable declaration and type checking  
• User input handling and validation  
• Data structure selection based on requirements  
• Method chaining and functional programming concepts  
• Problem-solving with appropriate data structures  
  
PRACTICAL APPLICATIONS:  
These fundamental concepts form the foundation for:  
• Data analysis and manipulation  
• Web development and API design  
• Scientific computing and research  
• Database operations and data modeling  
• Algorithm implementation and optimization  
  
NEXT STEPS:  
Students should continue practicing these concepts and explore:  
• Advanced data structures (dictionaries, nested structures)  
• File handling and data persistence  
• Object-oriented programming principles  
• Error handling and debugging techniques  
• Third-party libraries and frameworks  
  
This lab manual serves as a reference guide for future Python programming endeavors.