CN Lab Manual 01

Introduction to Python Programming

Instructor: Muhammad Nouman Hanif

Department of Computer Science, FAST-NU, Lahore, Pakistan

National University of Computer and Emerging Sciences

Contents

1 Objectives

After completing this lab, you will be able to:

- Understand and use fundamental Python data types.
- Apply various Python operators including mathematical, comparison, and boolean operators.
- Implement conditional statements and loops for controlling program flow.
- Define and utilize functions to create modular and reusable code.

2 Task Distribution

The lab is designed to be completed within **170 minutes**, with the following time allocation for each section:

Topic	**Time Allotment**
Python Data Types	15 Minutes
Python Operators	10 Minutes
Python If-Else Conditions	10 Minutes
Python Loops	15 Minutes
Python Functions	20 Minutes
Exercise	90 Minutes
Online Submission	10 Minutes
Total Time	**170 Minutes**

3 Online Python Interpreter: Google Colab

In this lab, we will use **Google Colab** as our primary Python interpreter. Colab is a cloud-based service that allows you to write and execute Python code directly in your browser, with no setup required.

3.1 Setting up Google Colab

Follow these steps to create your first Colab notebook:

1. Go to **https://colab.research.google.com/**.
2. Sign in using your NU email address.
3. Once on the 'Welcome to Colaboratory' page, navigate to **File ¿ New Notebook**.
4. Rename the notebook to your roll number (e.g., '20L-1234.ipynb').
5. In the first cell, type the following code to print a "Hello World" message:

```
print('Hello World')

```

6. Execute the cell by clicking the play button or pressing **Ctrl+Enter**.

4 Python Fundamentals

4.1 Data Types

Python includes several built-in data types. A variable is created the moment you first assign a value to it, and Python automatically determines the type.

4.1.1 Built-in Types

The following table summarizes the standard data types built into Python.

Categories	Data Type	Examples
Numeric Types	int	-2, -1, 0, 1, int(20)
	float	-1.25, 0.0, 1.25, float(20.5)
	complex	1j, complex(1j)
Text Sequence Type	str	'a', 'Hello!', str("Hello World")
Boolean Type	bool	True, False, bool(5)
Sequence Types	list	["apple", "banana", "cherry"]
	tuple	("apple", "banana", "cherry")
	range	range(6)
Mapping Type	dict	{"name": "John", "age": 36}
Set Types	set	{"apple", "banana", "cherry"}
	frozenset	frozenset({"apple", "banana"})
Binary Types	bytes	b"Hello", bytes(5)
	bytearray	bytearray(5)
	memoryview	memoryview(bytes(5))

Built-in Types Example

```python
# Python automatically assigns the data type
my_integer = 101
my_float = 3.14
my_string = "Hello Python"
my_list = [1, "a", 3.0]
my_tuple = (1, "a", 3.0)
my_dict = {"name": "Alice", "age": 25}
my_set = {1, 2, 3, 4, 5} # Note: duplicates are automatically removed
my_bool = True

# We can check the type of any variable using the type() function
print(f"'{my_integer}' is of type {type(my_integer)}")
print(f"'{my_float}' is of type {type(my_float)}")
print(f"'{my_string}' is of type {type(my_string)}")
print(f"'{my_list}' is of type {type(my_list)}")
print(f"'{my_tuple}' is of type {type(my_tuple)}")
print(f"'{my_dict}' is of type {type(my_dict)}")
print(f"'{my_set}' is of type {type(my_set)}")
print(f"'{my_bool}' is of type {type(my_bool)}")
```

4.1.2 Concept: Mutability

In Python, data types can be either **mutable** or **immutable**.

- **Mutable** objects can be changed after they are created. **Lists**, **dictionaries**, and **sets** are mutable.
- **Immutable** objects cannot be changed after they are created. **Strings**, **integers**, and **tuples** are immutable.

4.2 Operators

4.2.1 Math Operators

These operators are used to perform mathematical calculations.

Math Operators Example

```
a = 10
b = 3

print(f"{a} ** {b} (Exponent) = {a ** b}")
print(f"{a} % {b} (Modulus) = {a % b}")
print(f"{a} // {b} (Integer Division) = {a // b}")
print(f"{a} / {b} (Division) = {a / b}")
print(f"{a} * {b} (Multiplication) = {a * b}")
print(f"{a} - {b} (Subtraction) = {a - b}")
print(f"{a} + {b} (Addition) = {a + b}")
```

4.2.2 Comparison Operators

These operators compare two values and return a boolean result (‘True‘ or ‘False‘).

Comparison Operators Example

```
x = 5
y = 10

print(f"{x} == {y} (Equal to) is {x == y}")
print(f"{x} != {y} (Not equal to) is {x != y}")
print(f"{x} < {y} (Less than) is {x < y}")
print(f"{x} > {y} (Greater than) is {x > y}")
print(f"{x} <= 5 (Less than or equal to) is {x <= 5}")
print(f"{y} >= 10 (Greater than or equal to) is {y >= 10}")
```

4.2.3 Boolean Operators

There are three Boolean operators: `and`, `or`, and `not`.

The ‘and‘ Operator’s Truth Table

Expression	Evaluates to
True and True	True
True and False	False
False and True	False
False and False	False

The ‘or‘ Operator’s Truth Table

Expression	Evaluates to
True or True	True
True or False	True
False or True	True
False or False	False

The ‘not‘ Operator’s Truth Table

Expression	Evaluates to
not True	False
not False	True

Boolean Operators Example

```
age = 25
is_student = True

# 'and' example: checks if both conditions are true
if age > 18 and age < 65:
    print("You are an adult of working age.")

# 'or' example: checks if at least one condition is true
if age < 18 or is_student:
    print("You qualify for a discount.")

# 'not' example: inverts the boolean value
if not is_student:
    print("You are not currently a student.")
```

4.3 Control Flow

4.3.1 Conditional Statements

Python uses 'if', 'elif', and 'else' to execute code based on certain conditions.

Conditional Statements Example

```
age = 18

if age < 13:
    print("You are a child.")
elif age < 18:
    print("You are a teenager.")
else:
    print("You are an adult.")
```

4.3.2 Loops

Python provides 'for' and 'while' loops to repeat a block of code.

Loops Example

```
# A 'while' loop runs as long as a condition is true
count = 0
print("While loop:")
while count < 5:
    print(f"  Count is {count}")
    count += 1 # Increment count

# A 'for' loop iterates over a sequence (like a list or a range)
print("\nFor loop with range:")
for i in range(5):
    print(f"  Number {i}")

print("\nFor loop with a list:")
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(f"  I like {fruit}s")
```

4.4 Functions

Functions are reusable blocks of code that perform a specific action. They are defined using the 'def' keyword.

Functions Example

```python
# A simple function that takes a name and prints a greeting
def greet(name):
    print(f"Hello, {name}! Welcome.")

# A function that takes two numbers, adds them, and returns the result
def add_numbers(a, b):
    return a + b

# Calling the functions
greet("Nouman")
sum_result = add_numbers(5, 7)
print(f"The sum of 5 and 7 is {sum_result}.")
```

5 Exercise (25 Marks)

5.1 Task 1: Move Zeroes (5 Marks)

Given a list of integers ‘nums‘, write a function to move all ‘0‘’s to the end of it while maintaining the relative order of the non-zero elements. You must do this **in-place** without making a copy of the list.

Example

Input: `[0, 1, 0, 3, 12]`
Output: `[1, 3, 12, 0, 0]`

Conceptual Hint

Use a two-pointer approach. One pointer (the ”write” pointer) keeps track of the position where the next non-zero element should be placed. The second pointer (the ”read” pointer) iterates through the entire list. When the read pointer finds a non-zero element, you move it to the write pointer’s position and advance the write pointer. After the loop, fill the rest of the list with zeroes.

5.2 Task 2: Two Sum (10 Marks)

Given a list of integers ‘nums‘ and an integer ‘target‘, return the **indices** of the two numbers such that they add up to ‘target‘. You may assume that each input has *exactly one solution*, and you may not use the same element twice.

Example

Input: `nums = [2, 7, 11, 15], target = 9`
Output: `[0, 1]`
Explanation: Because nums[0] + nums[1] == 9, we return [0, 1].

Conceptual Hint

A dictionary (or hash map) is very effective here. Iterate through the list. For each number, calculate its ”complement” (i.e., `target - current_number`). Check if this complement already exists in your dictionary. If it does, you’ve found a pair. If not, add the current number and its index to the dictionary.

5.3 Task 3: FizzBuzz (10 Marks)

Write a function that returns a list of strings representing the numbers from 1 to ‘n‘. For multiples of three, use the string ‘”Fizz”‘ instead of the number. For multiples of five, use ‘”Buzz”‘. For numbers that are multiples of both three and five, use ‘”FizzBuzz”‘.

Example

Input: `n = 15`
Output: `["1", "2", "Fizz", "4", "Buzz", "Fizz", "7", "8", "Fizz", "Buzz", "11", "Fizz", "13", "14", "FizzBuzz"]`

Conceptual Hint

Use a ‘for‘ loop to iterate from 1 to ‘n‘. Inside the loop, use the modulus operator (‘%‘) to check for divisibility. The order of your ‘if/elif/else‘ conditions is important. Check for the most specific condition first (divisible by both 3 and 5) before checking for divisibility by 3 or 5 alone.

6 Submission Instructions

1. Rename your Jupyter notebook to your roll number (e.g., '20L-1234.ipynb').
2. Download the notebook by navigating to **File ¿ Download .ipynb**.
3. Submit the `.ipynb` file on Google Classroom under the designated assignment.
4. **Do not** compress the file (e.g., in a ZIP or RAR archive).
5. Late submissions will not be accepted.