

OPEN SOURCE SOFTWARE LAB (15B17CI575)
Lab Assignment 2 (Practice Lab)
ODD 2025
Topic Coverage: Python

Question 1: Create a Python program that initializes a list of five different fruits. Use indexing to print the first and last fruit from the list. Then, use slicing to print the middle three fruits.

- Example:
 - o First fruit: "Apple" (or whichever fruit is first in your list)
 - o Last fruit: "Mango" (or whichever fruit is last in your list)
 - o Middle fruits: ["Banana", "Orange", "Grapes"] (or whichever fruits are in the middle of your list)

CODE:

```
fruits = ["Apple", "Banana", "Orange", "Grapes", "Mango"]
```

```
print("First fruit:", fruits[0])
print("Last fruit:", fruits[-1])
print("Middle fruits:", fruits[1:4])
```

| | | | | | |
|--|--|--|--|-----|--|
| main.py | | | | Run | Output |
| 1 fruits = ["Apple", "Banana", "Orange", "Grapes", "Mango"] 2 3 print("First fruit:", fruits[0]) 4 print("Last fruit:", fruits[-1]) 5 print("Middle fruits:", fruits[1:4]) 6 | | | | | First fruit: Apple Last fruit: Mango Middle fruits: ['Banana', 'Orange', 'Grapes'] == Code Execution Successful == |

Question 2: Write a Python program that creates a dictionary with five key-value pairs, where the keys are the names of students and the values are their ages. Access and print the age of a specific student using their name as the key. Also, add a new student to the dictionary and print the updated dictionary.

- Example
 - o Age of the specific student: 20 (or the age of the student you accessed)
 - o Updated dictionary: {'Alice': 21, 'Bob': 22, 'Charlie': 20, 'David': 23, 'Eve': 19, 'Frank': 24} (or your updated dictionary with the new student added)

CODE:

```

students = {'Alice': 21, 'Bob': 22, 'Charlie': 20, 'David': 23, 'Eve': 19}

print("Age of the specific student:", students['Charlie'])

students['Frank'] = 24

print("Updated dictionary:", students)

```

```

main.py [Run] Share Output Clear
1 students = {'Alice': 21, 'Bob': 22, 'Charlie': 20, 'David': 23, 'Eve': 19}
2
3 print("Age of the specific student:", students['Charlie'])
4
5 students['Frank'] = 24
6
7 print("Updated dictionary:", students)
8

Age of the specific student: 20
Updated dictionary: {'Alice': 21, 'Bob': 22, 'Charlie': 20, 'David': 23, 'Eve': 19, 'Frank': 24}

== Code Execution Successful ==

```

Question 3: Write a function “duplicate” to find all duplicates in a list of 10 integers.

CODE:

```

def duplicate(numbers):
    duplicates = []
    for num in numbers:
        if numbers.count(num) > 1 and num not in duplicates:
            duplicates.append(num)
    return duplicates

```

Example usage

```

nums = [1, 2, 3, 4, 5, 2, 6, 7, 3, 8]
print("Duplicates:", duplicate(nums))

```

```

main.py [Run] Share Output Clear
1 def duplicate(numbers):
2     duplicates = []
3     for num in numbers:
4         if numbers.count(num) > 1 and num not in duplicates:
5             duplicates.append(num)
6     return duplicates
7
8 # Example usage
9 nums = [1, 2, 3, 4, 5, 2, 6, 7, 3, 8]
10 print("Duplicates:", duplicate(nums))

Duplicates: [2, 3]

== Code Execution Successful ==

```

Question 4: Write a function group (list, size) that takes a list and splits into smaller lists of given size.

CODE:

```
def group(lst, size):
    return [lst[i:i + size] for i in range(0, len(lst), size)]
```

Example usage

```
data = [1, 2, 3, 4, 5, 6, 7, 8]
print(group(data, 3))
```

```
main.py
```

[[1, 2, 3], [4, 5, 6], [7, 8]]
==== Code Execution Successful ===

```
1 def group(lst, size):
2     return [lst[i:i + size] for i in range(0, len(lst), size)]
3
4 # Example usage
5 data = [1, 2, 3, 4, 5, 6, 7, 8]
6 print(group(data, 3))
7
```

Question 5: Write a function “lensort” to sort a list of strings based on length. 4. Write a function extsort to sort a list of files based on extension.

CODE:

```
def lensort(strings):
    return sorted(strings, key=len)
```

```
def extsort(files):
    return sorted(files, key=lambda x: x.split('.')[ -1] if '.' in x else '')
```

Example usage

```
words = ["apple", "kiwi", "banana", "fig"]
print(lensort(words))
```

```
files = ["test.py", "readme.txt", "script.js", "data.csv", "index.html"]
print(extsort(files))
```

Output

['fig', 'kiwi', 'apple', 'banana']
['data.csv', 'index.html', 'script.js', 'test.py', 'readme.txt']
==== Code Execution Successful ===

Question 6: Use Python Built-in Functions 'open', 'read', 'readline', 'write', 'writeline' to work with files.

```

Code
import io

file_sim = io.StringIO()

file_sim.write("Hello world\n")

file_sim.writelines(["Line 2\n", "Line 3\n"])

file_sim.seek(0)

print("Read all content:\n" + file_sim.read())

file_sim.seek(0)
print("\nRead one line using readline():")
print(file_sim.readline(), end="")
print(file_sim.readline(), end="")

file_sim.close()

```

The screenshot shows a Jupyter Notebook interface with two panes. The left pane contains the Python code from above. The right pane, titled 'Output', shows the results of the code execution. It displays the output of `file_sim.read()` followed by the outputs of two consecutive `readline()` calls. A message at the bottom indicates successful code execution.

```

main.py
2
3  file_sim = io.StringIO()
4
5  file_sim.write("Hello world\n")
6
7  file_sim.writelines(["Line 2\n", "Line 3\n"])
8
9  file_sim.seek(0)
10
11 print("Read all content:\n" + file_sim.read())
12
13 file_sim.seek(0)

Output
^ Read all content:
Hello world
Line 2
Line 3

Read one line using readline():
Hello world
Line 2

== Code Execution Successful ==

```

Question 7: Write a function to compute the number of characters, words and lines in a file.

```

def file_statistics():
    content = """This is line one.
This is line two.
Third line here."""
    num_chars = len(content)
    num_words = len(content.split())
    num_lines = content.count('\n') + 1
    return num_chars, num_words, num_lines

chars, words, lines = file_statistics()
print("Characters:", chars)
print("Words:", words)
print("Lines:", lines)

```

The screenshot shows a Python code editor with a dark theme. The code in `main.py` defines a function `file_statistics` that takes no arguments. It initializes a string `content` with three lines of text: "This is line one.", "Second line here.", and "Third line here.". It then calculates the number of characters, words, and lines in `content` using the `len` function and the `split` method. Finally, it returns a tuple containing the character count, word count, and line count. The output panel shows the results: Characters: 52, Words: 11, Lines: 3, followed by a success message: === Code Execution Successful ===.

```
1 def file_statistics():
2     content = """This is line one.
3 This is line two.
4 Third line here."""
5     num_chars = len(content)
6     num_words = len(content.split())
7     num_lines = content.count('\n') + 1
8     return num_chars, num_words, num_lines
9
10 chars, words, lines = file_statistics()
```

Question 8: Write a program named 'reverse.py' to print lines of a file in reverse order.

```
def reverse_lines():
    lines = [
        "First line\n",
        "Second line\n",
        "Third line\n"
    ]
    for line in reversed(lines):
        print(line.strip())

reverse_lines()
```

The screenshot shows a Python code editor with a dark theme. The code in `main.py` defines a function `reverse_lines`. Inside, it creates a list `lines` containing three lines of text: "First line", "Second line", and "Third line". It then iterates over the reversed version of `lines` using the `reversed` function, and prints each line after stripping whitespace. The output panel shows the printed lines: Third line, Second line, First line, followed by a success message: === Code Execution Successful ===.

```
1 def reverse_lines():
2     lines = [
3         "First line",
4         "Second line",
5         "Third line"
6     ]
7     for line in reversed(lines):
8         print(line.strip())
9
10 reverse_lines()
11
```

Question 9: Write a program to print each line of a file in reverse order.

```
def reverse_each_line():
    lines = [
        "Hello World",
        "Python is fun",
        "ChatGPT rocks"
    ]
    for line in lines:
        print(line[::-1])

reverse_each_line()
```

```

main.py
1 def reverse_each_line():
2     lines = [
3         "Hello World",
4         "Python is fun",
5         "ChatGPT rocks"
6     ]
7     for line in lines:
8         print(line[::-1])
9
10 reverse_each_line()
11

```

Output:

```

dlroW olleH
nuf si nohtyP
skcor TPGtahC
== Code Execution Successful ==

```

Question 10: Write a program ‘wrap.py’ that takes filename and width as arguments and wraps the lines longer than width.

```

def wrap_lines(width):
    line = "Python is an amazing programming language to learn"
    words = line.split()
    current = ""
    for word in words:
        if len(current) + len(word) + 1 > width:
            print(current.strip())
            current = word + " "
        else:
            current += word + " "
    if current:
        print(current.strip())

```

wrap_lines(20)

```

main.py
1 def wrap_lines(width):
2     line = "Python is an amazing programming language to learn"
3     words = line.split()
4     current = ""
5     for word in words:
6         if len(current) + len(word) + 1 > width:
7             print(current.strip())
8             current = word + " "
9         else:
10            current += word + " "
11     if current:
12         print(current.strip())
13
14 wrap_lines(20)
15

```

Output:

```

Python is an
amazing programming
language to learn
== Code Execution Successful ==

```

Question 11: Python provides a built-in function map that applies a function to each element of a list. Provide an implementation for map using list comprehensions.

Code:

```

def my_map(func, lst):
    return [func(x) for x in lst]

```

```
# Example
result = my_map(lambda x: x.upper(), ["hello", "world"])
print(result)
```

```
main.py [ ] Share Run Output
1 def my_map(func, lst):
2     return [func(x) for x in lst]
3
4 # Example
5 result = my_map(lambda x: x.upper(), ["hello", "world"])
6 print(result)
7
```

['HELLO', 'WORLD']
== Code Execution Successful ==

Question 12: Python provides a built-in function filter (f, a) that returns items of the list a for which

```
def my_filter(func, lst):
    return [x for x in lst if func(x)]
```

Example

```
result = my_filter(lambda x: x > 10, [4, 11, 8, 20, 3])
print(result)
```

```
main.py [ ] Share Run Output
1 def my_filter(func, lst):
2     return [x for x in lst if func(x)]
3
4 # Example
5 result = my_filter(lambda x: x > 10, [4, 11, 8, 20, 3])
6 print(result)
7
```

[11, 20]
== Code Execution Successful ==

Question 13: Write a function triplet that takes a number n as an argument and returns a list of triplets such that the sum of first two elements of the triplet equals the third element using numbers below n.

Please note that (a, b, c) and (b, a, c) represent the same triplet.

Code:

```
def triplet(n):
    result = []
    for a in range(n):
        for b in range(a, n):
            c = a + b
            if c < n:
                result.append((a, b, c))
    return result

print(triplet(10))
```

```

main.py [ ] Share Run Output Clear
1 def triplet(n):
2     result = []
3     for a in range(n):
4         for b in range(a, n):
5             c = a + b
6             if c < n:
7                 result.append((a, b, c))
8     return result
9
10 print(triplet(10))
11

```

[(0, 0, 0), (0, 1, 1), (0, 2, 2), (0, 3, 3), (0, 4, 4), (0, 5, 5), (0, 6, 6), (0, 7, 7), (0, 8, 8), (0, 9, 9), (1, 1, 2), (1, 2, 3), (1, 3, 4), (1, 4, 5), (1, 5, 6), (1, 6, 7), (1, 7, 8), (1, 8, 9), (2, 2, 4), (2, 3, 5), (2, 4, 6), (2, 5, 7), (2, 6, 8), (2, 7, 9), (3, 3, 6), (3, 4, 7), (3, 5, 8), (3, 6, 9), (4, 4, 8), (4, 5, 9)]
== Code Execution Successful ==

Question 14: Write a python function parse_csv to parse csv (comma separated values) files.

14. Write a function mutate to compute all words generated by a single mutation on a given word. A mutation is defined as inserting a character, deleting a character, replacing a character, or swapping 2 consecutive characters in a string. For simplicity consider only letters from a to z.

Code:

```
from io import StringIO
```

```
def parse_csv_from_string(data):
    result = []
    file = StringIO(data)
    for line in file:
        line = line.strip()
        values = line.split(',')
        result.append(values)
    return result
```

```
csv_data = """name,age,city
Alice,21,Delhi
Bob,22,Mumbai"""

print(parse_csv_from_string(csv_data))
```

```

main.py [ ] Share Run Output Clear
1 from io import StringIO
2
3 def parse_csv_from_string(data):
4     result = []
5     file = StringIO(data)
6     for line in file:
7         line = line.strip()
8         values = line.split(',')
9         result.append(values)
10    return result
11

```

[['name', 'age', 'city'], ['Alice', '21', 'Delhi'], ['Bob', '22', 'Mumbai']]
== Code Execution Successful ==

Question 15: Write a function nearly_equal to test whether two strings are nearly equal. Two strings a and b are nearly equal when a can be generated by a single mutation on b.

```
def nearly_equal(a, b):
```

```
    if a == b:
```

```
        return False
```

```

len_a = len(a)
len_b = len(b)
if len_a == len_b + 1:
    for i in range(len_a):
        if a[:i] + a[i+1:] == b:
            return True
if len_a + 1 == len_b:
    for i in range(len_b):
        if b[:i] + b[i+1:] == a:
            return True

if len_a == len_b:
    mismatch = 0
    for ch1, ch2 in zip(a, b):
        if ch1 != ch2:
            mismatch += 1
        if mismatch > 1:
            break
    if mismatch == 1:
        return True

if len_a == len_b:
    for i in range(len_a - 1):
        if (a[i] != b[i] or a[i+1] != b[i+1]) and \
            a[i] == b[i+1] and a[i+1] == b[i]:
            if a[:i] == b[:i] and a[i+2:] == b[i+2:]:
                return True

return False
print(nearly_equal("cat", "bat"))
print(nearly_equal("cat", "cta"))
print(nearly_equal("cat", "cat"))

```

The screenshot shows a Jupyter Notebook interface with a code cell containing Python code for a function named `nearly_equal`. The code is identical to the one above, but it includes a check for equality at index 0. The output cell shows the results of three print statements:

```

main.py      Run   Output  Clear
1 - def nearly_equal(a, b):
2 -     if a == b:
3 -         return False
4 -
5     len_a = len(a)
6     len_b = len(b)
7     if len_a == len_b + 1:
8         for i in range(len_a):
9             if a[:i] + a[i+1:] == b:
10                return True
11 if len_a + 1 == len_b:

```

```

True
True
False

```

Below the output, a message indicates successful code execution.

Question 16: Write a program to count the frequency of characters in a given file. Can you use character frequency to tell whether the given file is a Python program file, C program file or a text file?

Code:

```
def count_char_frequency_from_string(content):
    freq = {}
    for ch in content:
        freq[ch] = freq.get(ch, 0) + 1
    return freq

def detect_file_type(freq_dict):
    if not freq_dict:
        return "Unknown"

    combined = ".join(freq_dict.keys())"

    if 'def' in combined or ':' in freq_dict and '#' in freq_dict:
        return "Python Program File"
    elif '#include' in combined or ';' in freq_dict and '{' in freq_dict:
        return "C Program File"
    else:
        return "Text File"

content = """#include <stdio.h>
int main() {
    printf("Hello, World!");
    return 0;
}
"""

freq = count_char_frequency_from_string(content)
print("Character Frequencies:\n", freq)
print("\nFile Type Detected:", detect_file_type(freq))
```

```

main.py [ ] Share Run Output Clear
1 def count_char_frequency_from_string(content):
2     freq = {}
3     for ch in content:
4         freq[ch] = freq.get(ch, 0) + 1
5     return freq
6
7 def detect_file_type(freq_dict):
8     if not freq_dict:
9         return "Unknown"
10
11 combined = ''.join(freq_dict.keys())
12
13 if 'def' in combined or ':' in freq_dict and '#' in freq_dict:

```

Character Frequencies:

```

{'#': 1, 'i': 5, 'n': 5, 'c': 1, 'l': 4, 'u': 2, 'd': 3, 'e': 3, ' ': 13, '<': 1, 's': 1, 't': 4, 'o': 3, '.': 1, 'h': 1, '>': 1, '\n': 5, 'm': 1, 'a': 1, '(': 2, ')': 2, '{': 1, 'p': 1, 'r': 4, 'f': 1, '')': 2, 'H': 1, ',': 1, 'W': 1, 'I': 1, ';': 2, 'O': 1, 'T': 1}

```

File Type Detected: C Program File

== Code Execution Successful ==

Question 17: Write a program to find anagrams in a given list of words. Two words are called anagrams if one word can be formed by rearranging letters of another. For example, 'eat', 'ate' and 'tea' are anagrams.

Code:

```
from collections import defaultdict
```

```
def find_anagrams(word_list):
    anagram_groups = defaultdict(list)
```

```
for word in word_list:
    key = ''.join(sorted(word))
    anagram_groups[key].append(word)
```

```
return [group for group in anagram_groups.values() if len(group) > 1]
```

```
words = ['eat', 'tea', 'tan', 'ate', 'nat', 'bat', 'tab', 'hello', 'silent', 'listen']
anagram_sets = find_anagrams(words)
```

```
for group in anagram_sets:
    print(group)
```

```

main.py [ ] Share Run Output Clear
1 from collections import defaultdict
2
3 def find_anagrams(word_list):
4     anagram_groups = defaultdict(list)
5
6     for word in word_list:
7         key = ''.join(sorted(word))
8         anagram_groups[key].append(word)
9
10    return [group for group in anagram_groups.values() if len
11           (group) > 1]
12
12 words = ['eat', 'tea', 'tan', 'ate', 'nat', 'bat', 'tab',
13           'hello', 'silent', 'listen']

```

['eat', 'tea', 'ate']
['tan', 'nat']
['bat', 'tab']
['silent', 'listen']

== Code Execution Successful ==