Exam Preparation - COMP9021 25T3

1. Printing Patterns

1.1 Row-Based Printing - quiz 1; first few questions in your labs

Print data row by row, processing each line independently.

```python
# Example: Print a multiplication table
for i in range(1, 4):
    for j in range(1, 4):
        print(f"{i} × {j} = {i*j}")
    print()  # New line after each row
```

1.2 Column-Based Printing

Use `zip()` or `zip_longest()` to print data column by column.

```python
from itertools import zip_longest

# Example: Print names and scores side by side
names = ["Alice", "Bob", "Charlie"]
scores = [85, 92, 78]

for name, score in zip(names, scores):
    print(f"{name}: {score}")

# Output:
# Alice: 85
# Bob: 92
# Charlie: 78
```

1.3 Matrix Construction (2D Lists) - Practice 18

Create and manipulate nested lists to represent matrices.

```python
# Example: Create a 3x3 matrix filled with zeros
matrix = [[0 for _ in range(3)] for _ in range(3)]

# Set some values
matrix[0][0] = 1
matrix[1][1] = 2
matrix[2][2] = 3

# Print the matrix
for row in matrix:
    print(row)
```

```
# Output:
# [1, 0, 0]
# [0, 2, 0]
# [0, 0, 3]
```

1.4 Shape-Based Printing - Practice 6

Print triangles, diamonds, and other shapes using loops.

Python

```python
# Example: Print a right triangle
n = 5
for i in range(1, n + 1):
    print("*" * i)

# Output:
# *
# **
# ***
# ****
# *****

# Example: Print a pyramid
for i in range(1, n + 1):
    spaces = " " * (n - i)
    stars = "*" * (2 * i - 1)
    print(spaces + stars)

# Output:
#     *
#    ***
#   *****
#  *******
# *********
```

1.5 Useful Functions for Printing

Python

```python
from itertools import cycle, zip_longest
import string

# Character sets
print(string.ascii_lowercase)  # 'abcdefghijklmnopqrstuvwxyz'
print(string.ascii_uppercase)  # 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
print(string.digits)           # '0123456789'
```

---

2. String and Integer Operations

2.1 Integer Operations

Base Conversion - Lab 3

```python
# Example: Convert decimal to binary or 3/4/5/6...-base
num = 42

binary = bin(num)      # '0b101010'

# Convert back to decimal
decimal = int('101010', 2)  # 42
```

Using % and // (Modulo and Integer Division) - Lab 4_1

```python
# Example: Extract digits from a number
num = 456

# Get last digit
last_digit = num % 10       # 6

# Remove last digit
remaining = num // 10       # 45

# Using divmod() - returns quotient and remainder
quotient, remainder = divmod(456, 10)  # (45, 6)
```

Check if String Contains Only Digits

```python
text = "12345"
if text.isdigit():
    print("All digits!")
```

Prime Number Check - Lab 8 - Practice 25

```python
def is_prime(n):
    """Check if n is a prime number."""
    if n < 2:
        return False
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return False
    return True

print(is_prime(17))  # True
print(is_prime(20))  # False
```

Float Formatting - Lab 3

```python
# Example: Print float with specific decimal places
pi = 3.14159265359
```

```python
print(f"{pi:.2f}")   # 3.14
print(f"{pi:.4f}")   # 3.1416
print("{:.3f}".format(pi))  # 3.142
```

pascal triangle - Practice 41

```python
row = [1]
for _ in range(5):
    print(row)
    row = [1] + [row[i]+row[i+1] for i in range(len(row)-1)] + [1]
```

power - Practice 20 - Lab 2_6

prime decomposition - Practice 25

2.2 String Operations

Common String Methods - Lab 2_2

```python
text = "  Hello World  "

# Case conversion
print(text.lower())     # '  hello world  '
print(text.upper())     # '  HELLO WORLD  '
print(text.title())     # '  Hello World  '
print("hello world".capitalize()) # 'Hello world'

print (x)

# Whitespace removal
print(text.strip())     # 'Hello World'
print(text.lstrip())    # 'Hello World  '
print(text.rstrip())    # '  Hello World'

# Split and join
words = text.strip().split()  # ['Hello', 'World']
joined = "-".join(words)      # 'Hello-World'

# Alignment
print("Hello".ljust(10))   # 'Hello     '
print("Hello".rjust(10))   # '     Hello'
print("Hello".center(10))  # '  Hello   '
```

String Slicing and Reversing

```python
text = "Python"
```

```python
# Slicing
print(text[0:3])   # 'Pyt'
print(text[2:])    # 'thon'
print(text[:4])    # 'Pyth'

# Reversing
print(text[::-1])  # 'nohtyP'

# Every second character
print(text[::2])   # 'Pto'
```

Finding Substrings - Two Pointers - Practice 28

Python

```python
text = "Hello World"

# Find position
index = text.find("World")  # 6
index2 = text.find("Python") # -1 (not found)

# Check if substring exists
if "World" in text:
    print("Found!")
```

Character Codes

Python

```python
# ord() - get ASCII/Unicode value
print(ord('A'))   # 65
print(ord('a'))   # 97

# chr() - get character from code
print(chr(65))    # 'A'
print(chr(97))    # 'a'
```

Counting Characters

Python

```python
from collections import Counter

text = "abracadabra"
counter = Counter(text)

print(counter)  # Counter({'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1})
print(counter['a'])  # 5
print(counter.most_common(2))  # [('a', 5), ('b', 2)]
```

---

3. List Operations - lab 4 - lab 5 - Lab 6 - Practice 23

3.1 Basic List Operations - Lab 2

```python
# Create a list
numbers = [1, 2, 3, 4, 5]

# Append and extend
numbers.append(6)          # [1, 2, 3, 4, 5, 6]
numbers.extend([7, 8])     # [1, 2, 3, 4, 5, 6, 7, 8]

# Remove elements
numbers.remove(3)          # Removes first occurrence of 3
popped = numbers.pop()     # Removes and returns last element
```

3.2 List Comprehension - lab 5

```python
# Example: Create a list of squares
squares = [x**2 for x in range(1, 6)]
print(squares)  # [1, 4, 9, 16, 25]

# With condition
even_squares = [x**2 for x in range(1, 11) if x % 2 == 0]
print(even_squares)  # [4, 16, 36, 64, 100]

# Nested list comprehension - create a 3x3 matrix
matrix = [[i + j for j in range(3)] for i in range(3)]
print(matrix)  # [[0, 1, 2], [1, 2, 3], [2, 3, 4]]
```

3.3 Sorting Lists - Lab 3_6

```python
# Simple sorting
numbers = [3, 1, 4, 1, 5, 9, 2]
sorted_nums = sorted(numbers)  # [1, 1, 2, 3, 4, 5, 9]

# Sort in descending order
desc_nums = sorted(numbers, reverse=True)  # [9, 5, 4, 3, 2, 1, 1]

# Sort with custom key function
students = [("Alice", 85), ("Bob", 92), ("Charlie", 78)]
sorted_by_score = sorted(students, key=lambda x: x[1])
print(sorted_by_score)  # [('Charlie', 78), ('Alice', 85), ('Bob', 92)]
```

3.4 List Slicing - Lab 5_4 - Lab 6_1

```python
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

# Basic slicing
print(numbers[2:5])    # [2, 3, 4]
print(numbers[:4])     # [0, 1, 2, 3]
```

```python
print(numbers[6:])    # [6, 7, 8, 9]

# Reverse a list
print(numbers[::-1])  # [9, 8, 7, 6, 5, 4, 3, 2, 1, 0]

# Every second element
print(numbers[::2])   # [0, 2, 4, 6, 8]
```

3.5 Composing Lists with zip() - Lab 4

Python

```python
# Example: Combine multiple lists
names = ["Alice", "Bob", "Charlie"]
ages = [25, 30, 35]
cities = ["New York", "London", "Tokyo"]

combined = list(zip(names, ages, cities))
print(combined)
# [('Alice', 25, 'New York'), ('Bob', 30, 'London'), ('Charlie', 35, 'Tokyo')]

# Unzip lists
unzipped_names, unzipped_ages, unzipped_cities = zip(*combined)
```

3.6 Flattening a List

Python

```python
# Example: Flatten a nested list
nested = [[1, 2, 3], [4, 5], [6, 7, 8, 9]]

# Method 1: List comprehension
flat = [item for sublist in nested for item in sublist]
print(flat)  # [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Method 2: Using sum() with an empty list
flat2 = sum(nested, [])
print(flat2)  # [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

3.8 Subsequence - Practice 28 - Practice 34

3.9 Sets - Lab 4_2

Python

```python
# Remove duplicates
numbers = [1, 2, 2, 3, 4, 4, 5]
unique = list(set(numbers))
```

3.10 Tuple

---

4. Dictionary Operations

4.1 Basic Dictionary Operations

```python
# Create a dictionary
student = {
    "name": "Alice",
    "age": 20,
    "major": "Computer Science"
}

# Access values
print(student["name"])        # 'Alice'
print(student.get("age"))     # 20
print(student.get("grade", "N/A"))  # 'N/A' (default if key not found)
```

4.2 Iterating Through Dictionaries

```python
scores = {"Alice": 85, "Bob": 92, "Charlie": 78}

# Iterate through keys
for name in scores.keys():
    print(name)

# Iterate through values
for score in scores.values():
    print(score)

# Iterate through key-value pairs
for name, score in scores.items():
    print(f"{name}: {score}")
```

4.3 Sorting Dictionaries

```python
scores = {"Alice": 85, "Bob": 92, "Charlie": 78}

# Sort by key (alphabetically)
sorted_by_name = dict(sorted(scores.items()))
print(sorted_by_name)  # {'Alice': 85, 'Bob': 92, 'Charlie': 78}

# Sort by value (score)
sorted_by_score = dict(sorted(scores.items(), key=lambda x: x[1]))
print(sorted_by_score)  # {'Charlie': 78, 'Alice': 85, 'Bob': 92}

# Sort in descending order
sorted_desc = dict(sorted(scores.items(), key=lambda x: x[1], reverse=True))
print(sorted_desc)  # {'Bob': 92, 'Alice': 85, 'Charlie': 78}
```

4.4 Counting with Dictionaries

```python
# Example: Count word frequency
text = "apple banana apple cherry banana apple"
words = text.split()

# Manual counting
word_count = {}
for word in words:
    word_count[word] = word_count.get(word, 0) + 1

print(word_count)  # {'apple': 3, 'banana': 2, 'cherry': 1}
```

4.5 Default Dictionary

```python
from collections import defaultdict

# Example: Group students by grade
students = [("Alice", "A"), ("Bob", "B"), ("Charlie", "A"), ("David", "B")]

# Using defaultdict
grade_groups = defaultdict(list)
for name, grade in students:
    grade_groups[grade].append(name)

print(dict(grade_groups))  # {'A': ['Alice', 'Charlie'], 'B': ['Bob', 'David']}
```

---

5. Recursion - Lab 7; Quiz 7

5.1 Basic Recursion Concepts

Recursion is when a function calls itself. Every recursive function needs:

1. **Base case**: The condition that stops the recursion
2. **Recursive case**: The function calling itself with a modified input

5.2 Island Problem (Grid Path Finding)

A classic problem using Depth-First Search (DFS) to find connected regions in a grid.

```python
def count_islands(grid):
    """
    Count the number of islands in a grid.
    '1' represents land, '0' represents water.
    """
    if not grid:
        return 0
```

```python
    # 1. Define directions (up, down, left, right)
    directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]

    # 2. Get grid boundaries
    height = len(grid)
    width = len(grid[0])

    # 3. Create visited grid
    visited = [[False for _ in range(width)] for _ in range(height)]

    # 4. DFS function
    def dfs(x, y):
        # (1) Check bounds
        if x < 0 or y < 0 or x >= height or y >= width:
            return

        # (2) Check conditions (is land and not visited)
        if grid[x][y] == "0" or visited[x][y]:
            return

        # (3) Mark current cell as visited
        visited[x][y] = True

        # (4) Explore all directions
        for dx, dy in directions:
            dfs(x + dx, y + dy)

    # Count islands
    island_count = 0
    for i in range(height):
        for j in range(width):
            if grid[i][j] == "1" and not visited[i][j]:
                dfs(i, j)
                island_count += 1

    return island_count

# Example usage
grid = [
    ["1", "1", "0", "0", "0"],
    ["1", "1", "0", "0", "0"],
    ["0", "0", "1", "0", "0"],
    ["0", "0", "0", "1", "1"]
]
```

```
print(count_islands(grid))  # 3
```

5.3 Path Finding in Nested Structures (n-ary tree) - Lab 5_2

5.4 Leetcode 79 word search / 212 word search II -check google drive

5.5 Generating Subsets - Practice 47

Python

```
from itertools import combinations

# Example: Generate all subsets of a list
def all_subsets(lst):
    """Generate all possible subsets."""
    result = []
    for i in range(len(lst) + 1):
        for combo in combinations(lst, i):
            result.append(list(combo))
    return result

numbers = [1, 2, 3]
print(all_subsets(numbers))
# [[], [1], [2], [3], [1, 2], [1, 3], [2, 3], [1, 2, 3]]
```

Python

```
def dfs(start_index, curr_sum, path):
    # Base case: We found a valid combination that sums to desired_sum
    if curr_sum == desired_sum:
        solutions.append(path[:])  # Save a copy of the current path
        return
    if curr_sum > desired_sum or start_index == n:
        return

    # Try selecting each digit from start_index to the end
    for i in range(start_index, n):
        # Make a choice: select the digit at position i
        path.append(digits[i])
        # Explore: recursively search from the next position with updated sum
        dfs(i + 1, curr_sum + int(digits[i]), path)
        # Undo: remove the digit we just added (backtrack)
        # This allows us to try other possibilities
        path.pop()
```

---

6. File Operations

6.1 Reading Files

```python
# Example: Read a file line by line
    """Read and process a text file."""
with open(file_name) as file:
    for line in file:
        # Skip empty lines
        if not line.isspace():
            # Remove leading/trailing whitespace
            line = line.strip()
            print(line)
```

---

7. Class - Lab 8; quiz 8

Something else

enumerate() to get index

```python
items = ['a', 'b', 'c']
for index, item in enumerate(items):
    print(f"{index}: {item}")
```

reversed() - Lab 4

```python
items = [1, 2, 3, 4, 5]
for item in reversed(items):
    print(item)
```

zip() - Lab 4

```python
names = ['Alice', 'Bob']
scores = [85, 92]
for name, score in zip(names, scores):
    print(f"{name}: {score}")
```

how to filter and transform list

```python
numbers = [1, 2, 3, 4, 5, 6]
even_squares = [x**2 for x in numbers if x % 2 == 0]
# [4, 16, 36]
```

abs()

permutations() and combinations

```python
from itertools import permutations, combinations

permutations('ABCD', 2)
-> AB AC AD BA BC BD CA CB CD DA DB DC

combinations('ABCD', 2)
-> AB AC AD BC BD CD
```

Thanks Everyone!

Thanks for all your participation and effort this term. Hope these notes make your revision a bit easier.

Good luck with the final! ;)

from you tutor,

Menghan Zhao (Silvana)