Day 6 of Training at Ansh Info Tech

Topics Covered

- **List Comprehension**
- **Dictionary Comprehension**
- **Functions in Python**
- **Function Argument Types in Python**
- **User Defined Functions**
- **In Built Functions**
- **Lambda Functions, Recursive Functions**
- **20 Practice Questions on Python Functions**
- **20 Practice Questions on List and Dictionary Comprehension**
- **4 Real World Applications**
- **Text Adventure Game, Hangman Game, Contact Application, Expense Tracker**

Summary

List Comprehension

List comprehension offers a concise way to create lists. It consists of brackets containing an expression followed by a `for` clause and can include multiple `for` or `if` clauses.

Dictionary Comprehension

Dictionary comprehension is a method for transforming one dictionary into another. It works similarly to list comprehension but uses `{}` instead of `[]`.

Functions in Python

Functions are blocks of code that perform a specific task and can be reused. They are defined using the `def` keyword.

Function Argument Types in Python

Functions can have various types of arguments:

- **Default arguments**: Provide default values.
- **Keyword arguments**: Pass arguments by parameter name.
- **Arbitrary arguments**: Accept variable numbers of arguments using `*args` and `**kwargs`.

User Defined Functions

These are functions that are defined by the user to perform specific tasks, making code modular and reusable.

In Built Functions

Python provides several built-in functions like `print()`, `len()`, and `range()`, which are always available.

Lambda Functions, Recursive Functions

- **Lambda functions**: Small anonymous functions defined with the `lambda` keyword.
- **Recursive functions**: Functions that call themselves to solve a problem.

Practice Questions on Python Functions

1. Write a function to calculate the factorial of a number.
2. Create a lambda function to add two numbers.
3. Write a recursive function to compute the nth Fibonacci number.
4. **And many more...**

Practice Questions on List and Dictionary Comprehension

1. Use list comprehension to create a list of squares of the first 10 natural numbers.
2. Use dictionary comprehension to create a dictionary from two lists.
3. **And many more...**

Real World Applications

- **Text Adventure Game**: Create a simple text-based adventure game.
- **Hangman Game**: Develop a console-based hangman game.
- **Contact Application**: Build an application to manage contacts.
- **Expense Tracker**: Implement an expense tracker to manage daily expenses.

FUNCTIONS

In Python, a function is defined using the def keyword, followed by the function name, parentheses containing any parameters, and a colon. The function body is indented and contains the code to be executed. .

Types of Functions

1.Built-in Functions: These are pre-defined functions provided by a programming language or environment. Examples include print(), len() in Python, and printf() in C.

2.User-defined Functions: These are functions created by the programmer to perform specific tasks. They follow a defined structure with a name, parameters, and a body.

3.Anonymous Functions (Lambda Functions): These are functions without a name, often used for short, simple operations

4.Recursive Functions: These functions call themselves within their definition, useful for tasks that can be divided into similar subtasks.

Double-click (or enter) to edit

```
#define a function
#syntax
def function_name(paramenters):
  #statement
  pass
```

```
#Lambda function
add=lambda x,y:x+y
print(add(2,3))
```

```
#Recursive function
def factorial(n):
  if n ==1:
    return 1
  else:
    return n*factorial(n-1)
```

Arguments in Functions

Arguments are values passed to a function when it is called. They can be categorized as follows:

1.Positional Arguments: These are arguments that are passed to a function in a specific order.

2.Keyword Arguments: These are arguments passed to a function by explicitly naming each parameter and its corresponding value.

3.Default Arguments: These are arguments that assume a default value if no value is provided during the function call

4.Variable-length Arguments: *args (Non-keyword arguments): Allows a function to accept any number of positional arguments.

**kwargs (Keyword arguments): Allows a function to accept any number of keyword arguments

**bold text

```
#Arbitary keyword
def info(**kwargs):
  for key,value in kwargs:
    print("kwargs.iteam()")


#positinal argument
def subtract(a, b):
    return a - b

#result = add(2, 3)  # 2 and 3 are positional arguments


#keyword arguments
def greet(name, message):
    return f"{message}, {name}!"

#result = greet(name="Alice", message="Hello")  # name and message are keyword arguments


Greeting = print(greet(name = "Harsimar", message = "Hi"))
```

```
Hi, Harsimar!
```

```
Greeting1 = print(greet(message = "Hello", name = "Divyansh"))
```

```
Hello, Divyansh!
```

```
#default arguments
def greet(name, message="Hello"):
    return f"{message}, {name}!"

result1 = greet("Alice")          # Uses default message "Hello"
result2 = greet("Bob", "Hi")      # Uses provided message "Hi"


Greet = print(greet("Mehakpreet"))
```

```
Hello, Mehakpreet!
```

```
#variable length arguments(*args)
def sum_all(*a):
    return sum(a)

#result = sum_all(1, 2, 3, 4)  # Accepts multiple positional arguments
```

```
Summation = print(sum_all(67, 12, 178, 19, 90, 14, 25, 12, 45))
```

```
462
```

```
#**kwargs
def print_details(**g):
    for key, value in g.items():
        print(f"{key}: {value}")

print_details(name="Alice", age=30, city="New York")
```

```
name: Alice
age: 30
city: New York
```

```
print_details(Locality = "Model Town", Address = "SCF, BLOCK-3, Third floor", H_no= "3", St_
```

```
Locality: Model Town
Address: SCF, BLOCK-3, Third floor
H_no: 3
St_no: 4
```

PRACTICE QUESTIONS ON PYTHON FUNCTIONS:

1.Write a Python function that takes a string as input and returns the reverse of the string.

2.Create a list of numbers. Write a function that finds and returns the maximum value in the list without using the built-in max() function.

3.Define a function that accepts a list of integers and returns a new list containing only the even numbers from the original list.

4.Implement a Python function to check if a given word is a palindrome (reads the same backward as forward).

5.Create a dictionary with student names as keys and their corresponding ages as values. Write a function to find and print the names of students who are above a certain age.

6.Develop a Python function that calculates the sum of squares for a given range of numbers.

7.Write a recursive function that calculates the Fibonacci sequence for a given term.

8.Create a class called Rectangle with methods to calculate the area and perimeter. Initialize the class with the length and width as attributes.

9.Implement a function that takes a list of strings and returns a new list with the strings sorted by their lengths in ascending order.

10.Use a lambda function to filter a list of integers and return a new list containing only the numbers greater than 10.

11.Write a function that takes a list of numbers and returns a new list containing only the unique elements.

12.Create a Python function that accepts a string and counts the occurrences of each character. Return the result as a dictionary.

13.Implement a function to calculate the average of a list of numbers without using the built-in sum() and len() functions.

14.Write a function that checks if a given year is a leap year. A leap year is divisible by 4 but not divisible by 100 unless it is divisible by 400.

15.Design a function that takes a list of strings and returns a new list with only the strings that have more than 5 characters.

16.Create a function that accepts a sentence and counts the number of vowels (a, e, i, o, u) in it.

17.Write a Python function that takes a number and checks whether it is a prime number.

18.Implement a function that reverses the order of words in a given sentence.

19.Create a function to find and return the second-largest number in a list of integers.

20.Write a function that takes a string and returns True if it is a palindrome (ignoring spaces and case), and False otherwise.

```python
# 1. Reverse a String
def reverse_string(s):
    return s[::-1]

# 2. Find Maximum Value in a List
def find_max(lst):
    max_value = lst[0]
    for num in lst:
        if num > max_value:
            max_value = num
    return max_value

# 3. Filter Even Numbers from a List
def filter_even_numbers(lst):
    return [num for num in lst if num % 2 == 0]

# 4. Check if a Word is a Palindrome
def is_palindrome(word):
    return word == word[::-1]

# 5. Find Students Above a Certain Age
def students_above_age(students, age_threshold):
    return [name for name, age in students.items() if age > age_threshold]

# 6. Sum of Squares for a Given Range
def sum_of_squares(start, end):
    return sum(i**2 for i in range(start, end + 1))

# 7. Fibonacci Sequence Using Recursion
def fibonacci(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

# 8. Rectangle Class
class Rectangle:
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

    def perimeter(self):
        return 2 * (self.length + self.width)

# 9. Sort Strings by Length
def sort_by_length(lst):
```

```
    return sorted(lst, key=len)

# 10. Filter Numbers Greater Than 10 Using Lambda
greater_than_10 = lambda lst: list(filter(lambda x: x > 10, lst))

# 11. Return Unique Elements from a List
def unique_elements(lst):
    unique_list = []
    for num in lst:
        if num not in unique_list:
            unique_list.append(num)
    return unique_list

# 12. Count Character Occurrences in a String
def count_characters(s):
    char_count = {}
    for char in s:
        if char in char_count:
            char_count[char] += 1
        else:
            char_count[char] = 1
    return char_count

# 13. Calculate Average Without Built-in sum() and len()
def calculate_average(lst):
    total = 0
    count = 0
    for num in lst:
        total += num
        count += 1
    return total / count if count != 0 else 0

# 14. Check if a Year is a Leap Year
def is_leap_year(year):
    return year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)

# 15. Filter Strings with More Than 5 Characters
def filter_long_strings(lst):
    return [s for s in lst if len(s) > 5]

# 16. Count Vowels in a Sentence
def count_vowels(sentence):
    vowels = "aeiouAEIOU"
    count = 0
    for char in sentence:
        if char in vowels:
            count += 1
    return count

# 17. Check if a Number is Prime
def is_prime(n):
```

```
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True

# 18. Reverse the Order of Words in a Sentence
def reverse_words(sentence):
    return ' '.join(sentence.split()[::-1])

# 19. Find the Second-Largest Number in a List
def second_largest(lst):
    first, second = float('-inf'), float('-inf')
    for num in lst:
        if num > first:
            first, second = num, first
        elif first > num > second:
            second = num
    return second

# 20. Check if a String is a Palindrome (Ignoring Spaces and Case)
def is_palindrome_ignore_spaces(s):
    cleaned = ''.join(char.lower() for char in s if char.isalnum())
    return cleaned == cleaned[::-1]

# Example usage of the functions
if __name__ == "__main__":
    print(reverse_string("hello"))  # Output: "olleh"
    print(find_max([1, 2, 3, 4, 5]))  # Output: 5
    print(filter_even_numbers([1, 2, 3, 4, 5]))  # Output: [2, 4]
    print(is_palindrome("racecar"))  # Output: True
    students = {"Alice": 20, "Bob": 22, "Charlie": 17}
    print(students_above_age(students, 18))  # Output: ['Alice', 'Bob']
    print(sum_of_squares(1, 5))  # Output: 55
    print(fibonacci(6))  # Output: 8
    rect = Rectangle(4, 5)
    print(rect.area())      # Output: 20
    print(rect.perimeter()) # Output: 18
    print(sort_by_length(["apple", "banana", "kiwi", "cherry"]))  # Output: ['kiwi', 'apple'
    print(greater_than_10([5, 11, 20, 3, 7]))  # Output: [11, 20]
    print(unique_elements([1, 2, 2, 3, 4, 4, 5]))  # Output: [1, 2, 3, 4, 5]
    print(count_characters("hello"))  # Output: {'h': 1, 'e': 1, 'l': 2, 'o': 1}
    print(calculate_average([1, 2, 3, 4, 5]))  # Output: 3.0
    print(is_leap_year(2020))  # Output: True
    print(filter_long_strings(["apple", "banana", "kiwi", "cherry"]))  # Output: ['banana',
    print(count_vowels("hello world"))  # Output: 3
    print(is_prime(11))  # Output: True
    print(reverse_words("hello world"))  # Output: "world hello"
    print(second_largest([1, 2, 3, 4, 5]))  # Output: 4
```

```
print(is_palindrome_ignore_spaces("A man a plan a canal Panama"))  # Output: True
```

```
olleh
5
[2, 4]
True
['Alice', 'Bob']
55
8
20
18
['kiwi', 'apple', 'banana', 'cherry']
[11, 20]
[1, 2, 3, 4, 5]
{'h': 1, 'e': 1, 'l': 2, 'o': 1}
3.0
True
['banana', 'cherry']
3
True
world hello
4
True
```

Practice Questions on List Comprehension

1. You have a list of names. Use list comprehension to create a new list that contains only the names that start with the letter "A".
2. Given a list of temperatures in Fahrenheit, use list comprehension to convert each temperature to Celsius and create a new list with these values.
3. You are processing a list of numbers. Use list comprehension to create a new list that contains the squares of all the numbers in the original list.
4. You have a list of words. Use list comprehension to create a new list that contains only the words that have more than 5 characters.
5. Given a list of integers, use list comprehension to create a new list containing only the even numbers from the original list.
6. You have a list of prices. Use list comprehension to apply a 10% discount to each price and create a new list with the discounted prices.
7. You are working with a list of students' grades. Use list comprehension to create a new list that contains the grades that are greater than or equal to 60.

8. Given a list of strings, use list comprehension to create a new list where each string is reversed.
9. You have a list of tuples, each containing a product and its price. Use list comprehension to create a new list of products that cost more than $20.
10. You are given a list of mixed data types. Use list comprehension to create a new list that contains only the integer values.

Practice Questions on Dictionary Comprehension

1. You have a list of student names. Use dictionary comprehension to create a dictionary where the keys are the student names and the values are the lengths of those names.
2. Given a list of numbers, use dictionary comprehension to create a dictionary where the keys are the numbers and the values are the squares of those numbers.
3. You are given a list of fruits and their prices. Use dictionary comprehension to create a dictionary where the keys are the fruits and the values are the prices with a 5% tax added.
4. You have a list of words. Use dictionary comprehension to create a dictionary where the keys are the words and the values are the number of vowels in each word.
5. Given a list of employees and their salaries, use dictionary comprehension to create a dictionary where the keys are the employee names and the values are their salaries increased by 10%.
6. You are working with a list of cities and their populations. Use dictionary comprehension to create a dictionary where the keys are the cities and the values are their populations in millions.
7. Given a list of letters, use dictionary comprehension to create a dictionary where the keys are the letters and the values are their positions in the alphabet.
8. You have a list of product names and their corresponding stock quantities. Use dictionary comprehension to create a dictionary where the keys are the product names and the values are "In Stock" or "Out of Stock" based on the quantity (consider 0 as "Out of Stock").
9. Given a list of countries and their capitals, use dictionary comprehension to create a dictionary where the keys are the capitals and the values are the countries.
10. You are given a list of file names with their extensions. Use dictionary comprehension to create a dictionary where the keys are the file names (without extensions) and the values are the extensions.

```
# List Comprehension

# 1. Names starting with "A"
def names_starting_with_A(names):
    return [name for name in names if name.startswith('A')]

print(names_starting_with_A(["Alice", "Bob", "Amanda"]))  # Output: ["Alice", "Amanda"]

# 2. Convert Fahrenheit to Celsius
def fahrenheit_to_celsius(f_temps):
    return [(temp - 32) * 5/9 for temp in f_temps]

print(fahrenheit_to_celsius([32, 68, 100]))  # Output: [0.0, 20.0, 37.77777777777778]

# 3. Squares of numbers
def squares_of_numbers(numbers):
    return [num ** 2 for num in numbers]

print(squares_of_numbers([1, 2, 3, 4]))  # Output: [1, 4, 9, 16]

# 4. Words with more than 5 characters
def words_more_than_5_chars(words):
    return [word for word in words if len(word) > 5]

print(words_more_than_5_chars(["apple", "banana", "kiwi", "cherry"]))  # Output: ['banana',

# 5. Even numbers
def even_numbers(numbers):
    return [num for num in numbers if num % 2 == 0]

print(even_numbers([1, 2, 3, 4, 5, 6]))  # Output: [2, 4, 6]

# 6. Prices with 10% discount
def discounted_prices(prices):
    return [price * 0.9 for price in prices]

print(discounted_prices([100, 200, 300]))  # Output: [90.0, 180.0, 270.0]

# 7. Grades greater than or equal to 60
def passing_grades(grades):
    return [grade for grade in grades if grade >= 60]

print(passing_grades([50, 65, 70, 45, 90]))  # Output: [65, 70, 90]

# 8. Reversed strings
def reversed_strings(strings):
    return [s[::-1] for s in strings]

print(reversed_strings(["hello", "world"]))  # Output: ['olleh', 'dlrow']

# 9. Products costing more than $20
```

```
def expensive_products(products):
    return [product for product, price in products if price > 20]

print(expensive_products([("apple", 10), ("banana", 25), ("kiwi", 30)]))  # Output: ['banana

# 10. Only integer values
def only_integers(items):
    return [item for item in items if isinstance(item, int)]

print(only_integers([1, "two", 3.0, 4, "five"]))  # Output: [1, 4]

# Dictionary Comprehension

# 1. Student names and their lengths
def student_name_lengths(names):
    return {name: len(name) for name in names}

print(student_name_lengths(["Alice", "Bob", "Amanda"]))  # Output: {'Alice': 5, 'Bob': 3, 'A

# 2. Numbers and their squares
def number_squares(numbers):
    return {num: num ** 2 for num in numbers}

print(number_squares([1, 2, 3, 4]))  # Output: {1: 1, 2: 4, 3: 9, 4: 16}

# 3. Fruits and their prices with tax
def prices_with_tax(fruits_prices):
    return {fruit: price * 1.05 for fruit, price in fruits_prices.items()}

print(prices_with_tax({"apple": 1.0, "banana": 0.5}))  # Output: {'apple': 1.05, 'banana': 0

# 4. Words and the number of vowels
def count_vowels(words):
    vowels = "aeiouAEIOU"
    return {word: sum(1 for char in word if char in vowels) for word in words}

print(count_vowels(["apple", "banana", "kiwi", "cherry"]))  # Output: {'apple': 2, 'banana':

# 5. Employee salaries with 10% increase
def increased_salaries(employees_salaries):
    return {employee: salary * 1.10 for employee, salary in employees_salaries.items()}

print(increased_salaries({"Alice": 50000, "Bob": 60000}))  # Output: {'Alice': 55000.0, 'Bob

# 6. Cities and populations in millions
def populations_in_millions(cities_populations):
    return {city: population / 1_000_000 for city, population in cities_populations.items()}

print(populations_in_millions({"CityA": 1000000, "CityB": 2500000}))  # Output: {'CityA': 1.

# 7. Letters and their positions in the alphabet
```

```
def letter_positions(letters):
    return {letter: ord(letter.lower()) - 96 for letter in letters}

print(letter_positions(["a", "b", "c", "z"]))  # Output: {'a': 1, 'b': 2, 'c': 3, 'z': 26}

# 8. Products and their stock status
def stock_status(products_quantities):
    return {product: ("In Stock" if quantity > 0 else "Out of Stock") for product, quantity

print(stock_status({"apple": 10, "banana": 0}))  # Output: {'apple': 'In Stock', 'banana': '

# 9. Capitals and their countries
def capitals_to_countries(countries_capitals):
    return {capital: country for country, capital in countries_capitals.items()}

print(capitals_to_countries({"USA": "Washington", "France": "Paris"}))  # Output: {'Washingt

# 10. File names and their extensions
def file_extensions(files):
    return {file.split('.')[0]: file.split('.')[1] for file in files}

print(file_extensions(["file1.txt", "file2.doc", "image.png"]))  # Output: {'file1': 'txt',
```

1. Text-Based Adventure Game Assignment Description: Develop a text-based adventure game where the player can navigate through different rooms, pick up items, and interact with objects and characters. Implement at least three different rooms and a simple inventory system.

Requirements:

- Use functions to handle different rooms and actions.
- Store items in a dictionary.
- Include user input for navigation and actions.
- Ensure the game loops until the player decides to quit.

```
def create_rooms():
    rooms = {'Hall': {
            'description':'You are in the Hall. There is a door to the east',
            'east':'Kitchen',
            'items': ['key']
        },
             'Kitchen': {
                 'description': 'You are in the Kitchen. There is a door to the west and sou
                 'west': 'Hall',
                 'south': 'Garden',
                 'items': ['apple', 'banana']
        },
             'Garden': {
                 'description': 'You are in the Garden. There is a door to the north',
                 'north': 'Kitchen',
                 'items': ['flower']
        }
    }
    return rooms

def process_command(rooms, current_room, inventory, command):
    command = command.lower().split()
    if not command:
        print("unknown command")
        return current_room
    action = command[0]
    if action == 'go':
        if(len(command)>1):
            direction = command[1]
            if(rooms[current_room][direction]):
                current_room = rooms[current_room][direction]
                return current_room
            else:
                print("There is no way")
                return current_room
        else:
            print("where do you want to go?")
            return current_room
    elif action == 'look':
        print(rooms[current_room]['description'])
        print(f"You see the Following Items: {', '.join(rooms[current_room]['items'])}" )
    elif action == 'pick':
        if(len(command)>1):
            item = command[1]
            if item in rooms[current_room]['items']:
                inventory.append(item)
            else:
                print("Unknown Item")
        else:
            print("Pick up What?")
    elif action == 'inventory':
```

```
        if (not inventory):
            print("Your Inventory is empty")
        else:
            print(f"Your Inventory: {', '.join(inventory)}")
    elif action == 'quit':
        print("Thanks For Playing")
    else:
        print("unknown command")
    return current_room


def main():
    rooms = create_rooms()
    inventory = []
    current_room = "Hall"
    print("Welcome to the text based Adventure Game")
    print("Type 'quit' to Exit the Game \n")
    while True:
        print(f"You are in the {current_room}")
        command = input("Enter a Command:")
        current_room = process_command(rooms, current_room, inventory, command)

if __name__ == "__main__":
    main()
```

2. Contact Book Application Assignment Description: Create a contact book application that allows users to add, remove, search, and list contacts. Ensure that contacts persist by saving them to a file.

Requirements:

- Implement functions for adding, removing, searching, and listing contacts.
- Use file I/O to save and load contacts.
- Handle exceptions for file operations and invalid input.

```
import json

CONTACT_FILE = "contacts.json"

def add_contact(contacts):
    name = input("Enter contact name: ").strip()
    phone = input("Enter contact phone number: ").strip()
    email = input("Enter contact email: ").strip()
    contact = {"name": name, "phone": phone, "email": email}
    contacts.append(contact)
    save_contacts(contacts)
    print(f"Contact '{name}' added successfully.")

def remove_contact(contacts):
    name = input("Enter the name of the contact to remove: ").strip()
    for contact in contacts:
        if contact['name'].lower() == name.lower():
            contacts.remove(contact)
            save_contacts(contacts)
            print(f"Contact '{name}' removed successfully.")
            return
    print(f"Contact '{name}' not found.")

def search_contact(contacts):
    name = input("Enter the name of the contact to search for: ").strip()
    found_contacts = [contact for contact in contacts if name.lower() in contact['name'].l
    if found_contacts:
        for contact in found_contacts:
            print(f"Name: {contact['name']}, Phone: {contact['phone']}, Email: {contact['e
    else:
        print(f"No contacts found matching '{name}'.")

def list_contacts(contacts):
    if not contacts:
        print("No contacts found.")
        return
    for contact in contacts:
        print(f"Name: {contact['name']}, Phone: {contact['phone']}, Email: {contact['email

def save_contacts(contacts):
    try:
        with open(CONTACT_FILE, 'w') as file:
            json.dump(contacts, file)
    except IOError as e:
        print(f"An error occurred while saving contacts: {e}")

def load_contacts():
    try:
        with open(CONTACT_FILE, 'r') as file:
            return json.load(file)
    except FileNotFoundError:
```

```
        return []
    except json.JSONDecodeError:
        return []
    except IOError as e:
        print(f"An error occurred while loading contacts: {e}")
        return []

def main():
    contacts = load_contacts()
    while True:
        print("\nContact Book")
```