Detailed Solutions: Python Basics II

Code Crafters

1. Functions

1.1. Task 1: Square Calculator

Solution:

```python
def square(n):
    """
    Accepts an integer n and returns its square.
    """
    return n * n

# Example usage
result = square(7)
print(f"The square of 7 is {result}.")
```

Explanation:

- The function `square(n)` takes an integer as input.
- It computes the square using `n * n`.
- The result is returned, and the usage example demonstrates how to call the function.

1.2. Task 2: Check Even or Odd

Solution:

```python
def is_even(n):
    """
    Checks if a number is even.
    Returns True if even, otherwise False.
    """
    return n % 2 == 0

# Example usage
print(is_even(4))  # Output: True
print(is_even(5))  # Output: False
```

Explanation:

- The modulo operator `%` checks divisibility by 2.
- If the result is 0, the number is even.
- The function outputs `True` for even numbers and `False` for odd numbers.

1.3. Task 3: String Repeater

Solution:

```python
def repeat_string(s, times):
    """
    Repeats a given string a specified number of times.
    """
    return s * times

# Example usage
result = repeat_string("Hello", 3)
print(result)  # Output: HelloHelloHello
```

Explanation:

- The string multiplication operator `*` repeats the string.
- `s * times` concatenates `s` with itself `times` times.

2. Indexing and String Manipulation

2.1. Task 4: Extract Year from Date

Solution:

```python
date = "2024-12-08"

# Extract year, month, and day
year = date[:4]
month = date[5:7]
day = date[8:]

print(f"Year: {year}, Month: {month}, Day: {day}")
```

Explanation:

- String slicing extracts parts of the string using `start:end` indices.
- `date[:4]` gets the first 4 characters for the year.
- Similarly, `date[5:7]` and `date[8:]` extract the month and day.

2.2. Task 5: String Replacement

Solution:

```python
sentence = "I love Python programming."
modified_sentence = sentence.replace("Python", "JavaScript")
print(modified_sentence)  # Output: I love JavaScript programming.
```

Explanation:

- The `replace()` method replaces all occurrences of ”Python” with ”JavaScript”.
- The result is stored in `modified_sentence`.

3. Collections

3.1. Task 6: Calculate Average Marks

Solution:

```python
marks = [80, 90, 85, 95, 88]
average = sum(marks) / len(marks)
print(f"The average marks are {average:.2f}.")
```

Explanation:

- `sum()` calculates the total of all marks.
- `len()` gets the number of subjects.
- The average is computed as `sum(marks) / len(marks)`.

3.2. Task 7: Manage To-Do List

Solution:

```python
to_do_list = []
while True:
    task = input("Enter a task (type ’done’ to finish): ")
    if task.lower() == ’done’:
        break
    to_do_list.append(task)

print("Your tasks:")
for task in to_do_list:
    print(f"- {task}")
```

Explanation:

- The program uses a loop to accept tasks.
- It exits when the user types ”done”.
- Each task is appended to the list using `append()`.

4. Dictionaries

4.1. Task 8: Phone Book

Solution:

```python
phone_book = {"Alice": "12345", "Bob": "67890"}

name = input("Enter a name: ")
if name in phone_book:
    print(f"{name}’s phone number is {phone_book[name]}.")
else:
    print("Name not found in the phone book.")
```

Explanation:

- A dictionary stores names as keys and phone numbers as values.
- The program checks if the name exists using `in`.

4.2. Task 9: Student Grades

Solution:

```
grades = {"Alice": "A", "Bob": "B", "Charlie": "C"}

name = input("Enter a student’s name: ")
if name in grades:
    print(f"{name}’s grade is {grades[name]}.")
else:
    print("Student not found.")
```

Explanation:

- Grades are stored in a dictionary.
- Input is matched against dictionary keys.

5. 2D Lists

5.1. Task 10: Diagonal Sum

Solution:

```
matrix = [
    [2, 4, 6],
    [1, 3, 5],
    [7, 9, 11]
]

# Calculate the diagonal sum
diagonal_sum = 0
for i in range(len(matrix)):
    diagonal_sum += matrix[i][i]

print(f"The sum of the diagonal elements is {diagonal_sum}.")
```

Explanation:

- The loop iterates over the indices of the matrix.
- The diagonal element is accessed using `matrix[i][i]`.
- The sum of these elements is accumulated in `diagonal_sum`.

5.2. Task 11: Transpose of a Matrix

Solution:

```python
matrix = [
    [2, 4, 6],
    [1, 3, 5],
    [7, 9, 11]
]

# Transpose the matrix
transpose = [[matrix[j][i] for j in range(len(matrix))] for i in range(len(matrix[0]))]

print("The transpose of the matrix is:")
for row in transpose:
    print(row)
```

Explanation:

- The transpose is calculated by swapping rows and columns.
- A nested list comprehension is used to build the transposed matrix.

6. Object-Oriented Programming

6.1. Task 12: Define a Bank Account

Solution:

```python
class BankAccount:
    def __init__(self, account_number, balance=0):
        """
        Initialize account number and balance.
        """
        self.account_number = account_number
        self.balance = balance

    def deposit(self, amount):
        """
        Add the specified amount to the balance.
        """
        self.balance += amount
        print(f"Deposited ${amount}. New balance is ${self.balance}.")

    def withdraw(self, amount):
        """
        Subtract the specified amount if funds are sufficient.
        """
        if amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew ${amount}. Remaining balance is ${self.balance}.")
        else:
            print("Insufficient funds.")

    def display(self):
        """
        Print the account details.
        """
        print(f"Account Number: {self.account_number}")
        print(f"Balance: ${self.balance}")

# Example usage
account = BankAccount("123456789", 100)
account.deposit(50)
account.withdraw(30)
account.display()
```

Explanation:

- The class includes methods for depositing, withdrawing, and displaying account details.
- Initial balance defaults to 0 if not provided.

6.2. Task 13: Define a Library System

Solution:

```python
class Library:
    def __init__(self):
        """
        Initialize an empty list of books.
        """
        self.books = []

    def add_book(self, book):
        """
        Add a book to the library collection.
        """
        self.books.append(book)
        print(f"Added book: {book}")

    def borrow_book(self, book):
        """
        Remove a book from the library if available.
        """
        if book in self.books:
            self.books.remove(book)
            print(f"You have borrowed: {book}")
        else:
            print(f"Sorry, {book} is not available.")

    def return_book(self, book):
        """
        Return a book to the library.
        """
        self.books.append(book)
        print(f"Returned book: {book}")

# Example usage
library = Library()
library.add_book("Python Programming")
library.add_book("Data Structures")
library.borrow_book("Python Programming")
library.return_book("Python Programming")
```

Explanation:

- Books are managed using a list.
- Methods handle adding, borrowing, and returning books.
- Availability is checked before borrowing.