## Answer Key

---

## Section A: Multiple Choice Questions

1. **b) 2**
   Explanation: The `~/` operator performs integer division, so `5 ~/ 3` results in `2`.
2. **b) Start the variable name with _.**
   Explanation: Variables or methods prefixed with _ are private in Dart.
3. **c) Dictionary**
   Explanation: Dart does not have a `Dictionary` type; it uses `Map` instead.
4. **a)** `void myFunc(Function func)`
   Explanation: This is the correct syntax to pass a function as a parameter.
5. **a) 10**
   Explanation: The `x++` post-increment operator returns the current value of `x` before incrementing.
6. **b) To delay variable initialization until it is first used.**
   Explanation: The `late` keyword is used to declare variables that will be initialized later.
7. **c)** `List myList = {1, 2, 3};`
   Explanation: `{}` is used for `Set` or `Map`, not `List`.
8. **a) Declares a synchronous function that returns an iterable.**
   Explanation: The `sync*` keyword is used for generator functions that yield iterable sequences.
9. **c) It allows any type of value to be assigned.**
   Explanation: Variables of `dynamic` type can hold values of any type.
10. **c) Both a and b**
    Explanation: Both methods are valid for creating a `Map` in Dart.

---

## Section B: True/False Questions

1. **True**
   Explanation: Dart supports both positional and named parameters.
2. **True**
   Explanation: The `dart:io` library is used for file and I/O operations.
3. **False**
   Explanation: A `Set` in Dart does not allow duplicate values.
4. **False**
   Explanation: Functions in Dart cannot return multiple values directly but can return a collection like a `List` or `Map`.
5. **True**
   Explanation: The `is` keyword checks the runtime type of a variable.
6. **False**
   Explanation: The `Stream` class is used to handle asynchronous sequences of data.
7. **True**
   Explanation: Dart is a strongly-typed language with a type-checking system.

8. **False**
   Explanation: A `Future` in Dart is used for handling asynchronous operations, not synchronous ones.
9. **True**
   Explanation: The `null` keyword represents the absence of a value.
10. **True**
    Explanation: The `List` class in Dart supports both fixed-length and growable lists.

---

## Section C: Short Answer Questions

1. **Difference Between List and Set**:
   - `List`: Allows duplicate values and maintains order.
     Example: `List<int> myList = [1, 2, 2, 3];`
   - `Set`: Does not allow duplicate values and does not guarantee order.
     Example: `Set<int> mySet = {1, 2, 3};`
2. **Exception Handling**:

```
try {
  int result = 10 ~/ 0; // Division by zero
} catch (e) {
  print('Error: $e');
}
```

8. **Null Safety**:
   - Null safety ensures variables cannot contain null unless explicitly allowed with `?`.
     Example:
   - `int? nullableVar = null; // Allowed`
   - `int nonNullableVar = 10; // Cannot be null`
9. **Extension Methods**:

```
extension StringExtension on String {
  String reverse() => split('').reversed.join('');
}
void main() {
  print('hello'.reverse()); // Output: olleh
}
```

16. **Asynchronous Function**:

```
Future<void> fetchData() async {
  await Future.delayed(Duration(seconds: 2));
  print('Data fetched');
}
```

21. **Typedef Usage**:

```
typedef MathOperation = int Function(int a, int b);
int add(int a, int b) => a + b;
void main() {
  MathOperation operation = add;
  print(operation(5, 3)); // Output: 8
}
```

28. **Inheritance Example**:

```
class Animal {
  void sound() => print('Animal sound');
}
class Dog extends Animal {
  @override
  void sound() => print('Bark');
```

```
}
```

36. **Generics Example**:

```
class Box<T> {
  T content;
  Box(this.content);
}
void main() {
  Box<int> intBox = Box(5);
  print(intBox.content); // Output: 5
}
```

45. **Stream Class**:
    - The `Stream` class is used for handling asynchronous data.
46. **Using Mixins**:

```
mixin Flyable {
  void fly() => print('Flying');
}
class Bird with Flyable {}
```

---

## Section D: Programming Questions

1. **Custom Linked List**:

```
class Node {
  int value;
  Node? next;
  Node(this.value);
}

class LinkedList {
  Node? head;

  void add(int value) {
    if (head == null) {
      head = Node(value);
    } else {
      Node current = head!;
      while (current.next != null) {
        current = current.next!;
      }
      current.next = Node(value);
    }
  }

  void remove(int value) {
    if (head == null) return;
    if (head!.value == value) {
      head = head!.next;
      return;
    }
    Node current = head!;
    while (current.next != null && current.next!.value != value) {
      current = current.next!;
    }
    if (current.next != null) {
      current.next = current.next!.next;
    }
  }

```

```
38.    void display() {
39.      Node? current = head;
40.      while (current != null) {
41.        print(current.value);
42.        current = current.next;
43.      }
44.    }
45.  }
```

46. **Library Management System**:

```
47.  class Book {
48.    String title, author;
49.    bool isAvailable;
50.    Book(this.title, this.author, this.isAvailable);
51.  }
52.
53.  class Library {
54.    List<Book> books = [];
55.
56.    void addBook(Book book) => books.add(book);
57.
58.    void borrowBook(String title) {
59.      for (var book in books) {
60.        if (book.title == title && book.isAvailable) {
61.          book.isAvailable = false;
62.          print('You borrowed "$title".');
63.          return;
64.        }
65.      }
66.      print('Book not available.');
67.    }
68.
69.    void returnBook(String title) {
70.      for (var book in books) {
71.        if (book.title == title && !book.isAvailable) {
72.          book.isAvailable = true;
73.          print('You returned "$title".');
74.          return;
75.        }
76.      }
77.      print('Invalid return.');
78.    }
79.
80.    void displayBooks() {
81.      for (var book in books) {
82.        print('${book.title} by ${book.author} - ${book.isAvailable ?
   'Available' : 'Borrowed'}');
83.      }
84.    }
85.  }
```