

Theory Exercise

Ans - 1:

1. **int** : A whole number without decimals (like 1, 42, or -7). Use it when you need to count or index things.
2. **double** : A number with decimals (e.g., 3.14, -0.5). Use it for measurements, prices, or anything that needs fractions.
3. **num** : A number that can be either int or double. Use it if you don't care whether it's whole or has decimals—but it's better to pick int or double when possible for safety.
4. **String** : A sequence of letters, words, or characters like "Hello" or 'Dart'. Used for text. You can use single or double quotes.
5. **bool**: A true-or-false value (true or false). Commonly used for yes/no, on/off, or checking conditions.
6. **List**: An ordered group of values, like a numbered bucket.
Example: [1, 2, 3], ['apple', 'banana'].
You can change its items and access elements by position.
7. **Map** : A collection of key–value pairs. Keys must be unique, but values can repeat. Like a dictionary: {'name': 'Alice', 'age': 25 }.
8. **Set**: An unordered collection of **unique** items. Example: { 'red', 'green', 'blue' }.
Great for storing things once without caring about order.

Ans -2:

1. if / else : Checks a condition and runs code based on whether it's true or false.

- Use if when you need to do something only when a condition is true.
- Add else for what happens when it's false.
- You can also add multiple checks using else if Example:

```
if (age >= 18) {  
  print('Adult'); } else {  
  print('Minor'); }
```

for loop : Repeats a block of code a set number of times. Example:

```
for (int i = 0; i < 5; i++) { print(i); // prints  
0,1,2,3,4 }
```

You can also loop over items in a list:

Example:

```
for (String fruit in ['apple', 'banana']) { print(fruit); }
```

2. while loop : Keeps running as long as a condition stays true. Checks the condition at the start. If false, it won't run even once

Example:

```
int count = 0; while (count <  
5) { print(count);  
count++; }
```

3. do-while loop : Runs the block first, then checks the condition. So it runs at least once. Even if the condition is false at first, it runs once .

Example: `int num = 0; do
{ print(num); num++; }
while (num < 5);`

4. switch statement : Chooses among many options based on one value. **case** checks each value. **break** stops after a match. **default** runs if none match. Example:

```
String fruit = 'apple'; switch (fruit)
{ case 'banana':
    print('Banana!'); break;
  case 'apple':  print('Apple!');
  break; default:
    print('Unknown fruit'); }
```

Ans - 3:

1. Class : A blueprint for creating objects (things) with properties and actions. Example:

```
class Animal { String name;
  Animal(this.name);
  void speak() {
    print('$name makes a sound');
  } }
```

2. Inheritance : One class (child) can use code from another class (parent). Example:

```
class Dog extends Animal {
  Dog(String name) : super(name);
```

```
void speak() { print('$name barks');
}
}
```

3. Polymorphism: A single call can do different things based on the object. Example:

```
Animal a = Dog('Rex');
a.speak(); // prints: "Rex barks"
```

4. Interfaces : Dart uses classes as interfaces: any class can promise to do something. Example:

```
class Flyer { void fly() {
print('Can fly');
}
}

class Bird implements Flyer {
void fly() { print('Bird flies');
}}
```

Ans - 4:

1. Future : A Future is like a promise that a value will be available later—maybe after a network request, file read, or timer. It represents a single value you'll get sometime in the future.

Example without await:

```
Future<String> fetchData() =>

Future.delayed(Duration(seconds: 2), () => 'Hello');

void main() async { print('Start');
```

```
    fetchData(); // returns a Future immediately    print('End'); // runs  
before 'Hello' is ready  
}
```

O/P:

Start

End

2. async & await : Use async on a function to mark it as asynchronous (returns a Future). Inside it, use await to pause the code until the Future finishes, making async code feel like regular code.

Example:

```
Future<String> fetchData() async {  
    await Future.delayed(Duration(seconds: 2));    return 'Hello';  
}  
  
void main() async {    print('Start');  
  
    String message = await fetchData();    print(message);  
  
    print('End');  
}
```

O/P:

Start

Hello

End

Here, main() waits for fetchData() before moving on

You can also use try-catch around await to handle errors

3. Stream: A Stream gives you *multiple* values over time, like a series of events (e.g., user input, timer ticks). It's like an async list of values.

Example with await for:

```
Stream<int> count(int to) async* { for (int i = 1; i <= to; i++) { await
Future.delayed(Duration(seconds: 1)); yield i;
}
}

void main() async { await for (int i in count(3)) {
print(i);
}
}
```

O/P:

This prints numbers 1, 2, 3, each after a 1-second delay