Kim Sokhom (Group 1)

Answer

1. Type Inference

EXPLAIN: Explain how Dart infers the type of a variable.

> Type inference is a feature of Dart which allows the compiler to automatically ddetermine the type of a variable based on its first assign value.

Code

```
var x = 'Cat';  //Dart infers this as String

or

const y = 18;   //Dart infers this as integer

or

var name = ['Mia', 'Joe', 'Bob'];   //Dart infers this as List<String>

or

var z;  //Dart infers this as dynamic
```

Note: You can't assign a value with different type from the initial value.

Example

```
var x = 'Cat';
x = 123;    //This would throw an error
```

2. Nullable and Non-Nullable Variables

EXPLAIN: Explain nullable variables.

> Nullable variables are variables that can hold either normal value or `null` value. We can declare a variable as nullable by adding a "?" after the data type.
> In the other case, Non-Nullable variables are variables that cannot hold a `null` value.

EXPLAIN: When is it useful to have nullable variables?

> Nullable variables are useful in situations as below:
>
> - When dealing with optional fields in forms or waiting to receive data.

> - When working with external data sources like web APIs, nullable variables allow us to handle missing data.
> - Functions often have optional parameters, so nullable variables can represent these optional values.
> - When working with database, some data might be optional.
> - When we need to declare a variable and only initialize it after some condition is met.
> - User Input.

Code

```
void main() {
  // Declare a nullable integer variable and assign it a null value
  int? nullInteger = null;
  print(nullInteger);

  // Declare a non-nullable integer variable and assign it a value
  int age = 30; //cannot be null if we try to assgin 'null' to this variable, it
will throw an error.
  print(age);

  // Assign a new value to the nullable variable
  nullInteger = 2;
  print(nullInteger);
}
```

3. Final and const

EXPLAIN: Describe the difference between final and const.

> final:
>
> - The value of final variable is set at runtime.
> - we use final variable when we don't know the value ahead of time, it means that we can assign the value to it later during the execution of the program.

> const:
>
> - Its value is set at compile time.
> - we use it when we know the value ahead of time.

Code

```
void main() {
  // Declare a final variable and assign it the current date and time
  final currentDate = DateTime.now();
  print(currentDate);

  // Can you declare this variable as const? Why?
  // => No because const requires value at compile time and DateTime.now() is
```

```
known at runtime.

  // Declare a const variable with a integer value
  const integer = 10;
  print(integer);

  // Can you reassign the value of this final variable? Why?
  // => No because both final and const can only be assign once, after that their
value will be fixed and cannot change.
}
```

4. String, List and Maps

Strings:

```
void main() {
  // Declare two strings: firstName and lastName and an integer:age
  String firstName = 'Kim';
  String lastName = 'Sokhom';
  int age = 19;

// Concatenate the 2 strings and the age
  String result = 'I am $firstName' + ' ' + '$lastName' + ' ' + 'and I am $age
year olds.';

// Print result
  print(result);
}
```

Lists:

```
void main() {
  // Create a list of integers
  List<int> integer = [1, 2, 3];

  // Add a number to the list
  integer.add(4);


  // Remove a number from the list
  integer.remove(3);

  // Insert a number at a specific index in the list
  integer.insert(1, 18);

  // Iterate over the list and print each number
  integer.forEach((intItem) {
    print(intItem);
  });
```

```
}
```

Maps:

```
void main() {
  // Create a map with String keys and integer values
  Map<String, int> stuAge = {
    'Kosal': 90,
    'Bob': 40,
    'Somnang': 75,
    'Jummy': 95,
  };

  // Add a new key-value pair to the map
  stuAge['Yashi'] = 100;

  // Remove a key-value pair from the map
  stuAge.remove('Bob');

  // Iterate over the map and print each key-value pair
  stuAge.forEach((key, value) {
    print('$key: $value');
  });

}
```

5. Loops and Conditions

```
void main() {
  // Use a for-loop to print numbers from 1 to 5
  for(int i = 1; i <= 5; i++){
    print(i);
  }

  // Use a while-loop to print numbers while a condition is true
  int num = 5;
  while(num > 0){
    print(num);
    num--;
  }
  // Use an if-else statement to check if a number is even or odd
  int Number = 12;
  if(Number % 2 == 0){
    print('$Number is Even Number!');
  }
  else { print('$Number is Odd Number!'); }
}
```

6. Functions

EXPLAIN: Compare positional and named function arguments.

> `Positional function arguments` is strictly care about the order of the arguments passed to the function, it means that we must provide the arguments in the correct order to the function and all arguments are required.
> `Named function arguments` isn't care about the order of the arguments, it means that we can pass the arguments in any order and we can make some arguments optional if needed.

EXPLAIN: When and how to use arrow syntax for functions.

> We use arrow syntax for functions when the function contains a single expression.
> To use arrow syntax for functions we replace the body of the function with `=>`.

Note: The arrow syntax will automatically implies a `return` statement for the expression.

Code

- Defining and Invoking a Function:

```dart
void main() {
  // Define a function that takes two integers and returns their sum
  int sum(int a, int b){
    return a + b;
  }

// Call the function and print the result
  int result = sum(2,3);
  print(result);
}
```

- Positional vs Named Arguments:

```dart
void main() {
  // Define a function that uses positional arguments
  String getName (String firstName, String lastName){
    return ('$firstName $lastName');
  }

  // Define another function that uses named arguments with the required keyword
(ex: getArea with rectangle arguments)
  double getArea ({required double width, required double height}){
    return width * height;
  }

  // Call both functions with appropriate arguments
  print(getName('Kim', 'Sokhom'));
  print(getArea(width: 10, height: 5.22));
}
```

EXPLAIN: Can positional arguement be ommited? Show an example

> Positional arguments cannot be ommited if they are required but if they are not required we can also make them optional by wrapping them with square brackets `[]`.

Example

```dart
void main() {
    //if they are required
    void hello(String arg1, String arg2){
        print('Hello $arg1 $arg2');
    }
    //call the function
    hello('Kim', 'Sokhom');

    //if they are not required
    void sayHi (String arg1, [String arg2 = '']) {
        print('Hi $arg1 $arg2');
    }
    //call the function
    sayHi('Kim');
}
```

EXPLAIN: Can named arguement be omitted? Show an example.

> Named arguments cannot be omitted if they are marked with required keyword and they can be omitted if they are not maked as required.

Example

```dart
void main() {
    //if they are marked as required
    void hello({required String arg1, required String arg2}){
        print('Hello $arg1 $arg2');
    }
    //call the function
    hello(arg1: 'Kim', arg2: 'Sokhom');

    //if they are not maked as required
    void sayHi ({String arg1 = 'Yas', String arg2 = 'Yashi'}) {
        print('Hi $arg1 $arg2');
    }
    //call the function
    sayHi(arg1: 'Yoo'); //arg2 will take the default value which in 'Yashi'
}
```

- Arrow Syntax:

```dart
void main() {
   // Define a function using arrow syntax that squares a number
  double square(double a) => a*a;

// Call the arrow function and print the result
  double result = square(10);
  print(result);
}
```