Dart 제어 흐름 기초

제어문과 예외처리

학습목표

- Dart의 기본적인 제어 흐름문 이해하기
- 조건문과 반복문의 구조와 활용법 습득하기
- 오류 처리를 위한 예외처리 구문 학습하기
- 열거형(Enum)의 사용법 이해하기
- Collection If의 활용법 이해하기

제어 흐름의 기본 개념

- 제어 흐름은 프로그램 실행 순서를 결정하는 핵심 요소
- 조건문과 반복문을 통해 프로그램의 논리적 흐름을 제어
- Python, JavaScript 등 다른 언어와 유사한 문법 구조 공유
- 효과적인 제어문 사용이 프로그래밍 실력 향상의 기초

조건문 활용하기

- if문을 사용하여 단순 조건 분기 처리 가능
- 논리 연산자 &&, ||를 활용한 복합 조건 설정
- collection if를 통한 컬렉션 내 조건부 요소 처리
- switch문으로 다중 조건 분기를 효율적으로 처리

if_condition.dart

```dart
void main() {
  bool isFootball = true; // 축구 경기 여부를 나타내는 변수

  if (isFootball) { // isFootball이 true일 경우
    print('Go Football!'); // "Go Football!" 출력
  } else { // isFootball이 false일 경우
    print('Go Sports!'); // "Go Sports!" 출력
  }
}
```

- `if` 문은 주어진 조건(`isFootball`)이 참인지 거짓인지에 따라 다른 코드 블록을 실행합니다.

switch_statement.dart

```dart
void main() {
  int myValue = 2; // 정수 값을 저장하는 변수

  switch (myValue) { // myValue 값에 따라 분기
    case 1: // myValue가 1일 경우
      print('Monday'); // "Monday" 출력
      break; // switch문 종료
    case 2: // myValue가 2일 경우
      print('Tuesday'); // "Tuesday" 출력
      break; // switch문 종료
    default: // myValue가 case에 해당하지 않을 경우
      print('Error: Value not defined?'); // "Error: Value not defined?" 출력
      break; // switch문 종료
  }
}
```

- `switch` 문은 변수(`myValue`)의 값에 따라 여러 `case` 중 하나를 선택하여 실행합니다. `default` 는 어떤 `case` 에도 해당하지 않을 때 실행됩니다.

Collection If

```dart
void main() {
  bool addMore = true;
  var list = [
    'Dart',
    'Flutter',
    if (addMore) 'More' // addMore가 true일 경우 'More' 추가
  ];
  print(list); // 결과: [Dart, Flutter, More]
}
```

- Collection If는 컬렉션(List, Set 등)을 생성할 때 조건을 만족하는 경우에만 요소를 추가할 수 있게 해 줍니다.

반복문 구현하기

- while문으로 조건 기반 반복 처리
- do-while문으로 최소 1회 이상 실행 보장
- for문을 통한 정해진 범위 반복
- forEach로 컬렉션 요소 순회 처리

while_loop.dart

```dart
void main() {
  bool isTrue = true; // 반복문 제어 변수

  while (isTrue) { // isTrue가 true인 동안 반복
    print ('Hello'); // "Hello" 출력
    isTrue = false; // isTrue를 false로 변경하여 반복문 종료
  }
}
```

- `while` 문은 주어진 조건(`isTrue`)이 참인 동안 코드 블록을 반복해서 실행합니다.

do_while_loop.dart

```dart
void main() {
  bool isTrue = false; // 반복문 제어 변수

  do {
    print('Hello'); // "Hello" 출력
  } while (isTrue); // isTrue가 true인 동안 반복 (최소 1번 실행)
}
```

- `do-while` 문은 코드 블록을 먼저 실행하고, 그 후에 조건을 검사합니다. 따라서 코드 블록은 최소한 한 번은 실행됩니다.

for_loop.dart

```dart
void main() {
  int maxIterations = 10; // 최대 반복 횟수
  for (var i = 0; i < maxIterations; i++) { // i가 0부터 maxIterations-1까지 반복
    print ('Iteration: $i'); // "Iteration: i" 출력
  }
}
```

- `for` 문은 초기화, 조건, 증감식을 사용하여 코드 블록을 정해진 횟수만큼 반복합니다.

forEach_example.dart

```dart
void main() {
  List daysOfWeek = ['Sunday', 'Monday', 'Tuesday']; // 요일 목록

  daysOfWeek.forEach((day) => print(day)); // 목록의 각 요소에 대해 print 함수 실행
}
```

- `forEach` 메서드는 리스트의 각 요소에 대해 주어진 함수를 실행합니다.

열거형 사용하기

```
enum Day { sun, mon, tues } // 요일을 나타내는 열거형

void main() {
  print(Day.values); // 열거형의 모든 값 출력
  print('${Day.values[0]}'); // 열거형의 첫 번째 값 출력
  print(Day.values.byName('mon')); // 이름으로 열거형 값 접근
}
```

- `enum`은 연관된 상수 값들을 그룹화하는 데 사용됩니다.

예외처리 구현하기

```
void main(){
  String name = "Dart"; // 문자열 변수

  try{ // 예외가 발생할 수 있는 코드 블록
    print ('Name: $name'); // "Name: Dart" 출력
    name.indexOf(name[0], name.length - (name.length+2)); // RangeError 발생
  } on RangeError catch (exception) { // RangeError 예외 처리
    print ('On Exception: $exception'); // 예외 메시지 출력
  }
  catch (exception) { // 그 외 예외 처리
    print ('Catch Exception: $exception'); // 예외 메시지 출력
  } finally { // 항상 실행되는 블록
    print ('Mission completed!'); // "Mission completed!" 출력
  }
}
```

- `try-catch` 블록은 예외가 발생할 수 있는 코드를 감싸고, 예외가 발생했을 때 적절한 처리를 할 수 있도록 합니다. `finally` 블록은 예외 발생 여부와 상관없이 항상 실행됩니다.

요약

Dart의 제어 흐름은 조건문, 반복문, 예외처리를 통해 프로그램의 실행 순서와 로직을 관리합니다. 기본적인 if 문과 switch문으로 조건 분기를 처리하고, while문과 for문으로 반복 작업을 수행하며, try-catch문으로 예외상황을 안전하게 처리할 수 있습니다. Collection If를 사용하여 조건부로 컬렉션에 요소를 추가할 수 있으며, Enum을 사용하여 관련된 상수 집합을 정의할 수 있습니다.

END