다트 기초를 효과적으로 학습하기 위한 실습 프로젝트를 준비했습니다! 이론보다 실제 코드를 작성하면서 개념을 익히는 것이 훨씬 효과적이니 좋은 접근법이에요.

**다트 기초 실습 프로젝트 - "다트 마스터하기"**

이 프로젝트는 지금까지 배운 다트의 주요 개념들을 실습할 수 있는 단계별 과제로 구성되어 있습니다. 각 단계마다 여러 개념들을 종합적으로 활용해볼 수 있도록 설계했습니다.

**프로젝트 1: 다트 기본 개념 실습 - "다트 타입 마스터"**

**과제 1: 변수와 타입 다루기**

// 1. 다음과 같은 변수들을 선언하세요:

// - 정수형 변수 age (값: 25)

// - 문자열 변수 name (값: '홍길동')

// - 실수형 변수 height (값: 175.5)

// - nullable 문자열 변수 nickname (초기값: null)

// - dynamic 타입 변수 something (초기값: '안녕하세요')

// 2. 각 변수의 타입을 출력하는 코드를 작성하세요 (runtimeType 활용)

// 3. nullable 변수 nickname에 '길동이'를 할당하고 출력하세요

// 4. dynamic 변수 something에 숫자 42를 할당하고 출력하세요

// 5. 문자열 템플릿을 사용하여 "제 이름은 [name]이고, 나이는 [age]세, 키는 [height]cm입니다." 형식으로 출력하세요

**과제 2: 리스트, 셋, 맵 활용하기**

// 1. 다음 정보를 담는 리스트, 셋, 맵을 만드세요:

// - fruits: 사과, 바나나, 딸기, 포도, 사과 (리스트)

// - uniqueFruits: fruits에서 중복을 제거한 셋

// - fruitPrices: 과일 가격 정보를 담은 맵 (사과: 2000, 바나나: 3000, 딸기: 4500, 포도: 5000)

// 2. fruits 리스트에 '키위'와 '오렌지'를 추가하세요

// 3. uniqueFruits 셋에 '멜론'을 추가하세요

// 4. fruitPrices 맵에 '키위': 3500을 추가하세요

// 5. 다음 작업을 수행하세요:

// - fruits 리스트에서 forEach를 사용해 모든 과일 이름 출력

// - uniqueFruits 셋의 크기(길이) 출력

// - fruitPrices 맵에서 가격이 3000원 이상인 과일만 출력 (힌트: where 메서드 활용)

**프로젝트 2: 함수와 파라미터 - "함수 마스터"**

**과제 1: 다양한 파라미터 타입 활용**

// 1. 다음 조건에 맞는 함수를 작성하세요:

// - 이름: greeting

// - 필수 파라미터: String name

// - 옵셔널 파라미터: String greeting (기본값: '안녕하세요')

// - 반환: "[greeting], [name]님!" 형태의 문자열

// 2. 다음 조건에 맞는 함수를 작성하세요:

// - 이름: calculateTotal

// - 네임드 파라미터:

// \* required double price

// \* required int quantity

// \* double discountRate (기본값: 0.0)

// - 반환: 최종 가격(price \* quantity \* (1 - discountRate))

// 3. 두 함수를 다양한 인자로 호출하고 결과를 출력하세요

// (최소 3가지 다른 방식으로 호출)

**과제 2: 화살표 함수와 typedef**

// 1. 다음 typedef를 정의하세요:

// - MathOperation: 두 개의 정수를 받아 정수를 반환하는 함수 타입

// 2. 다음 화살표 함수를 MathOperation 타입으로 정의하세요:

// - add: 두 수를 더하는 함수

// - subtract: 두 수를 빼는 함수

// - multiply: 두 수를 곱하는 함수

// - divide: 두 수를 나누는 함수 (결과는 정수로 반환, 힌트: ~/ 연산자 사용)

// 3. 다음 함수를 작성하세요:

// - 이름: performOperation

// - 파라미터:

// \* int a

// \* int b

// \* MathOperation operation

// - 동작: operation 함수를 사용하여 a와 b에 대한 연산 수행

// 4. 위에서 정의한 모든 함수를 호출하고 결과를 출력하세요

**프로젝트 3: 클래스와 객체지향 프로그래밍 - "클래스 마스터"**

**과제 1: 기본 클래스 구현**

// 1. Person 클래스를 만드세요:

// - 필드:

// \* final String name

// \* final int age

// \* String? address (nullable)

// - 생성자: 필수 파라미터로 name과 age를 받고, 선택적으로 address를 받음

// - 메서드:

// \* getInfo(): "[name], [age]세, 주소: [address]" 형태의 문자열 반환

// \* 주소가 null이면 '주소 정보 없음'으로 표시

// 2. Person 클래스를 상속받는 Student 클래스를 만드세요:

// - 추가 필드:

// \* final String schoolName

// \* final int grade

// - 생성자: Person의 필드와 Student의 추가 필드 초기화

// - 메서드:

// \* getInfo() 오버라이드: 기존 정보에 "학교: [schoolName], [grade]학년" 추가

// 3. Person과 Student 인스턴스를 각각 생성하고 정보를 출력하세요

**과제 2: 고급 클래스 기능**

// 1. 간단한 상품(Product) 클래스를 만드세요:

// - 필드:

// \* final String name

// \* final double price

// \* int stock

// - 생성자: 모든 필드 초기화

// - 메서드:

// \* sell(int quantity): 재고에서 quantity만큼 감소, 재고가 부족하면 예외 발생

// \* restock(int quantity): 재고에 quantity만큼 추가

// \* getInfo(): 상품 정보 문자열 반환

// 2. ShoppingCart 클래스를 만드세요:

// - 필드:

// \* Map<Product, int> items (상품과 수량을 매핑)

// - 생성자: 빈 맵으로 items 초기화

// - 메서드:

// \* addItem(Product product, int quantity): 장바구니에 상품 추가

// \* removeItem(Product product): 장바구니에서 상품 제거

// \* getTotalPrice(): 장바구니 총 가격 계산

// \* checkout(): 장바구니의 모든 상품 구매 처리(sell 메서드 호출)

// 3. 여러 상품을 생성하고, 장바구니에 추가/제거한 후, 결제 과정을 시뮬레이션하세요

**프로젝트 4: 함수형 프로그래밍 - "함수형 마스터"**

**과제 1: 컬렉션 처리**

// 다음과 같은 정수 리스트가 있습니다:

final numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

// 1. map을 사용하여 numbers의 각 요소를 제곱한 새 리스트를 만드세요

// 2. where를 사용하여 numbers에서 짝수만 필터링한 리스트를 만드세요

// 3. reduce를 사용하여 numbers의 모든 요소의 합을 계산하세요

// 4. fold를 사용하여 numbers의 모든 요소를 곱한 값을 계산하세요

// 5. 다음 조건을 충족하는 새 리스트를 만드세요:

// - numbers의 홀수 요소만 선택

// - 선택된 요소를 3배

// - 결과 값이 15보다 큰 요소만 필터링

// (힌트: where, map, where를 체이닝)

**과제 2: 고급 함수형 패턴**

// 다음과 같은 사용자 데이터 리스트가 있습니다:

final users = [

{'name': '김철수', 'age': 25, 'active': true},

{'name': '이영희', 'age': 31, 'active': false},

{'name': '박민수', 'age': 28, 'active': true},

{'name': '정지원', 'age': 22, 'active': true},

{'name': '홍길동', 'age': 35, 'active': false},

];

// 1. map을 사용하여 사용자의 이름만 추출한 리스트를 만드세요

// 2. where를 사용하여 active가 true인 사용자만 필터링하세요

// 3. 함수형 메서드를 사용하여 다음을 계산하세요:

// - 사용자들의 평균 나이

// - 가장 나이가 많은 사용자의 이름

// - 가장 나이가 적은 active 사용자의 이름

// 4. 사용자 데이터를 다음과 같이 변환하세요:

// - 이름을 키로, 나이를 값으로 하는 Map<String, int> 생성

// (힌트: fold 또는 reduce 활용)

// 5. 캐스케이딩 연산자(.../cascade operator)를 사용하여

// 리스트에 여러 작업을 연속적으로 수행하는 예제를 작성하세요

**프로젝트 5: 비동기 프로그래밍 - "비동기 마스터"**

**과제 1: Future와 async/await**

// 1. 다음 함수를 구현하세요:

// - fetchUserData(): 2초 후에 사용자 데이터(Map)를 반환하는 Future

// - fetchUserPosts(): 1초 후에 사용자의 게시물 목록(List)을 반환하는 Future

// 2. 위 두 함수를 순차적으로 호출하고 결과를 출력하는 async 함수 getUserInfo()를 작성하세요

// 3. 위 두 함수를 병렬로 호출하고 모든 결과가 준비되면 출력하는 async 함수 getUserInfoParallel()을 작성하세요

// (힌트: Future.wait 활용)

// 4. try-catch를 사용하여 비동기 함수의 예외를 처리하는 예제를 작성하세요

// (힌트: 일부러 예외를 발생시키는 함수를 만들고 호출)

**프로젝트 5: 비동기 프로그래밍 - 기초**

**타스크 2: Stream 기초 (수정 버전)**

**2-1: 간단한 Stream 이해하기**

**dartimport 'dart:async';**

**// 1. 아래 함수를 완성하여 0부터 4까지의 숫자를 포함하는 Stream을 만드세요**

**Stream<int> createSimpleStream() async\* {**

**// 여기에 코드를 작성하세요 (힌트: for 루프와 yield 사용)**

**}**

**// 2. 위 Stream을 구독하고 각 숫자를 출력하세요**

**void testSimpleStream() async {**

**// 여기에 코드를 작성하세요 (힌트: await for 또는 listen 사용)**

**}**

**// main 함수 - 실행해보세요**

**void main() async {**

**await testSimpleStream();**

**}**

**2-2: 지연이 있는 Stream 만들기**

**dart// 1. 1초 간격으로 1부터 5까지 숫자를 생성하는 Stream을 만드세요**

**Stream<int> countStream(int maxCount) async\* {**

**// 여기에 코드를 작성하세요**

**// (힌트: for 루프, Future.delayed, await, yield 사용)**

**}**

**// 2. 위 Stream을 구독하고 "카운트: X"와 같은 형식으로 출력하세요**

**void printCountStream() async {**

**print("카운트 시작...");**

**// 여기에 코드를 작성하세요**

**print("카운트 완료!");**

**}**

**2-3: 기본 Stream 변환**

**dart// 1. countStream(10)에서 짝수만 필터링하는 코드를 작성하세요**

**void filterEvenNumbers() async {**

**final stream = countStream(10);**

**// 방법 1: await for 사용**

**// 여기에 코드를 작성하세요**

**// 또는**

**// 방법 2: where 메서드 사용**

**// 여기에 코드를 작성하세요**

**}**

**2-4: 간단한 StreamController 사용**

**dartimport 'dart:async';**

**// 1. StreamController를 생성하고 다음 단계를 수행하세요:**

**// - 컨트롤러 생성**

**// - 리스너 추가**

**// - 5개의 숫자 추가 (1~5)**

**// - 컨트롤러 닫기**

**void useStreamController() {**

**// 여기에 코드를 작성하세요**

**}**

**타스크 3: 비동기 프로그래밍 응용 (기초)**

**3-1: Future 체이닝**

**dart// 1. 다음 두 함수를 연결하여 사용하세요:**

**// - fetchUsername(): 1초 후 사용자 이름("홍길동")을 반환**

**// - fetchUserAge(String name): 1초 후 해당 이름의 나이를 반환 (예: "홍길동"->20)**

**// - 최종 출력: "홍길동님의 나이는 20세입니다."**

**Future<String> fetchUsername() async {**

**// 여기에 코드를 작성하세요**

**}**

**Future<int> fetchUserAge(String name) async {**

**// 여기에 코드를 작성하세요**

**}**

**// 두 가지 방법으로 구현해보세요:**

**// 방법 1: then() 사용**

**void chainWithThen() {**

**// 여기에 코드를 작성하세요**

**}**

**// 방법 2: async/await 사용**

**void chainWithAsync() async {**

**// 여기에 코드를 작성하세요**

**}**

**3-2: 간단한 타이머 기반 이벤트**

**dartimport 'dart:async';**

**// 1. 타이머를 사용하여 5초 동안 1초마다 메시지를 출력하는 함수를 작성하세요**

**// (힌트: Timer.periodic 사용)**

**void simpleTimerEvents() {**

**int count = 0;**

**// 여기에 코드를 작성하세요**

**}**

**3-3: 비동기 작업 오류 처리**

**dart// 1. 랜덤하게 성공 또는 실패하는 Future 함수를 만들고 오류를 처리하세요**

**Future<String> fetchDataWithPossibleError() async {**

**// 50% 확률로 성공 또는 실패하는 코드**

**// 여기에 코드를 작성하세요**

**}**

**// 오류 처리 예제**

**void handleAsyncError() async {**

**try {**

**// 여기에 코드를 작성하세요**

**} catch (e) {**

**// 여기에 코드를 작성하세요**

**} finally {**

**print("작업이 완료되었습니다.");**

**}**

**}**

**타스크 4: 비동기 프로그래밍 미니 프로젝트**

**4-1: 간단한 다운로드 진행 상황 시뮬레이션**

**dart// 1. 파일 다운로드 진행 상황을 시뮬레이션하는 코드를 작성하세요:**

**// - downloadFile(String fileName) 함수 구현**

**// - 0%부터 100%까지 20% 단위로 진행 상황 보고**

**// - 각 단계는 0.5초 소요**

**// - 완료 시 "다운로드 완료" 메시지 출력**

**Future<void> downloadFile(String fileName) async {**

**// 여기에 코드를 작성하세요**

**}**

**// 다운로드 실행 및 진행 상황 출력**

**void simulateFileDownload() async {**

**print("파일 다운로드를 시작합니다...");**

**// 여기에 코드를 작성하세요**

**}**

**4-2: 간단한 비동기 퀴즈 앱**

**dart// 1. 간단한 퀴즈 앱을 구현하세요:**

**// - getQuestion(): 1초 후 퀴즈 질문을 반환하는 함수**

**// - getAnswer(String question): 1초 후 해당 질문의 답을 반환하는 함수**

**// - 3개의 질문을 순차적으로 표시하고 답변 확인**

**Future<String> getQuestion(int questionNumber) async {**

**// 여기에 코드를 작성하세요 (미리 정의된 질문 반환)**

**}**

**Future<String> getAnswer(String question) async {**

**// 여기에 코드를 작성하세요 (질문에 맞는 답변 반환)**

**}**

**// 퀴즈 실행**

**void runQuizApp() async {**

**print("퀴즈를 시작합니다!");**

**// 여기에 코드를 작성하세요**

**print("퀴즈가 종료되었습니다!");**

**}**

**학습 가이드**

**단계적 접근: 가장 간단한 과제부터 시작하여 점진적으로 진행하세요.**

**개념 이해 중심: 각 단계에서 코드가 작동하는 방식을 이해하는 데 집중하세요.**

**실험 장려: 코드를 변경하고 다양한 시나리오를 테스트해보세요.**

**시각화: 비동기 작업의 흐름을 주석이나 print 문으로 추적해보세요.**

**개념 정리: 각 과제 후에는 배운 내용을 자신의 언어로 요약하세요.**

**프로젝트 6: Dart 3.0 신기능 - "최신 다트 마스터"**

**과제 1: Records와 패턴 매칭**

// 1. 다음 정보를 담는 Record 타입을 정의하고 값을 할당하세요:

// - 영화 정보: (제목, 감독, 개봉년도, 평점)

// 2. 위에서 만든 Record에서 destructuring을 사용하여 각 필드를 변수로 추출하세요

// 3. 다음과 같은 함수를 작성하세요:

// - getMovieInfo(): 영화 제목을 받아 해당 영화의 정보를 Record로 반환

// (최소 3개 이상의 영화 정보를 포함)

// 4. switch 표현식과 패턴 매칭을 사용하여 영화 평점에 따라 다른 메시지를 출력하는 함수를 작성하세요

// - 4.5 이상: "최고의 영화!"

// - 4.0 이상 4.5 미만: "매우 좋은 영화!"

// - 3.5 이상 4.0 미만: "괜찮은 영화"

// - 3.5 미만: "평범한 영화"

// 5. if-case 패턴 매칭을 사용하여 특정 감독의 영화만 출력하는 코드를 작성하세요

**과제 2: 다양한 클래스 유형과 Mixin**

// 1. base class Vehicle을 작성하세요:

// - 필드:

// \* String model

// \* int year

// - 메서드:

// \* String getInfo() (차량 정보 반환)

// 2. Vehicle을 상속받는 final class Car를 작성하세요:

// - 추가 필드:

// \* int doors

// \* String engineType

// - getInfo() 오버라이드

// 3. 다음 mixin을 정의하세요:

// - ElectricPowered: 전기차 관련 기능

// \* double batteryCapacity

// \* double getBatteryStatus()

// \* void charge()

// 4. sealed class Shape을 만들고, 이를 상속받는 Circle, Rectangle, Triangle 클래스를 작성하세요

// - 각 클래스는 면적을 계산하는 double getArea() 메서드를 구현

// 5. 패턴 매칭을 사용하여 주어진 Shape 객체의 타입에 따라 다른 메시지를 출력하는 함수를 작성하세요

**종합 프로젝트: "다트 미니 앱"**

이제 배운 모든 개념을 종합하여 간단한 콘솔 애플리케이션을 만들어 보세요. 이 프로젝트는 도서 관리 시스템을 구현합니다.

// 도서 관리 시스템

//

// 요구사항:

// 1. Book 클래스 구현

// - 필드: id, title, author, publishYear, isAvailable

// - 메서드: getInfo(), borrow(), return()

//

// 2. Library 클래스 구현

// - 필드: name, books (Book 리스트)

// - 메서드:

// \* addBook(): 도서 추가

// \* removeBook(): 도서 제거

// \* findBooksByAuthor(): 저자로 도서 검색

// \* findBooksByTitle(): 제목으로 도서 검색

// \* borrowBook(): 도서 대출 처리

// \* returnBook(): 도서 반납 처리

//

// 3. User 클래스 구현

// - 필드: id, name, borrowedBooks (Book 리스트)

// - 메서드:

// \* borrowBook(): 도서 대출

// \* returnBook(): 도서 반납

// \* getBorrowedBooks(): 대출 중인 도서 목록 반환

//

// 4. 함수형 프로그래밍 기능 활용

// - 도서 필터링 (출판년도, 저자 등)

// - 대출 가능한 도서만 표시

// - 특정 조건에 맞는 도서 카운트

//

// 5. 비동기 기능 구현

// - 도서 정보 로딩 시뮬레이션 (Future 활용)

// - 대출/반납 처리 시 지연 시뮬레이션

//

// 6. Dart 3.0 신기능 활용

// - Record를 사용한 도서 정보 반환

// - 패턴 매칭을 활용한 도서 상태 확인

// - sealed class를 활용한 대출/반납 결과 처리

//

// 7. 메인 함수에서 시스템 테스트

// - 도서 추가, 검색, 대출, 반납 기능 테스트

// - 다양한 시나리오 실행

여기까지가 다트 기초 개념을 실습할 수 있는 프로젝트입니다. 각 단계별로 진행하면서 개념을 정확히 이해하고 적용할 수 있을 것입니다. 정답은 직접 작성해보고 나중에 확인하는 방식으로 진행하면 학습 효과가 더 클 것입니다.

필요하시면 제가 특정 부분에 대한 힌트나 추가 설명을 제공해드릴 수 있습니다. 어떤 부분부터 시작해보실 계획인가요?