



플러터 : 7월 3일

☀ 상태	시작 전
🎯 프로젝트	<u>플러터</u>
🏷 태그	

플러터 개발환경 설정 및 플러터 개념에 대해 학습하고 정리

Flutter 애플리케이션을 만들 때 필요한 Dart언어

Dart 코드를 일관되게 작성하는 모범 사례를 학습하여 잘 구조화된 Flutter 애플리케이션을 구축

Dart는 진정한 객체 지향 언어

Dart의 컴파일러

기본 플랫폼: 모바일 및 데스크톱 장치를 대상으로 하는 앱의 경우

Dart에는 JIT(Just-In-Time) 컴파일 기능이 있는 **Dart VM**과 기계 코드 생성을 위한 **AOT**(Ahead-of-Time) 컴파일러가 모두 포함

웹 플랫폼: 웹을 대상으로 하는 앱의 경우 Dart는 개발 또는 프로덕션 목적으로 컴파일할 수 있음

웹 컴파일러는 **Dart**를 **JavaScript**로 변환

sdk 설치

PowerShell 관리자 권한으로 실행하기

```
choco install dart-sdk
```

main.dart > main

```
Run | Debug
1 void main(){
2   print('hello world!');
3
4 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

[Running] dart "c:\Users\82102\Desktop\flutter\main.dart"
hello world!

[Done] exited with code=0 in 1.064 seconds

#1.0

```
void main() {
  var name = '안우섭';
  // 변수 타입 구체화할 필요 없음
  // 변수 업데이트 가능
  name = "□□L○";

  dynamic tmp;
  if (tmp is String) {
    //tmp는 string의 많은 옵션을 가짐
  }

  final something = 'asd';
  // final로 선언하면 바꿀 수 없음
```

```

late final String name2;
name2 = '안우섭';
// 변수를 먼저 만들고, 데이터를 나중에 받을 수 있음
// 실수를 막아줌 print(name2)

const name3 = '안우섭';
// compile-time constant
// const는 컴파일할 때 알고 있는 값을 사용해야 함
// 만약 어떤 값인지 모르고, 그 값이 API로부터 오거나 사용자가 화면에서
// var, final이어야 함

//const: 컴파일 시점에 바뀌지 않는 값 (상수)
//final: 컴파일 시점에 바뀌는 값 (API에서 받아온 값, 사용자 입력값)

String? name4 = 'asd';
name4 = null;
//null도 가능하게하고 싶으면 type뒤에 ? 붙여주면 됨
}

```

#2.0 Basic Data Types

기본 데이터 타입

아래 타입을 포함한 거의 대부분의 타입들이 객체로 이루어져 있다. (함수도 객체)

이것이 Dart가 진정한 객체 지향 언어로 불리는 이유

```

void main() {

String name = "tom";

bool isPlay = true;

int age = 10;

```

```
double money = 52.55;

num x = 12;

num y = 1.2;

// int와 double 은 num이라는 자료형을 상속받은 자료형
// 그래서 num은 그 값이 integer 일 수도 있고, double 일 수도 있음
```

list

```
List numbers = [1, 2, 3];
var number2 = [4, 5, 6];

//List는 collection if와 collection for를 지원한다.
//collection if는 List를 만들 때, if를 통해 존재할 수도 안 할 수도 있

var giveMeFive = true;
var item = [
  1,
  2,
  3,
  if (giveMeFive) 10, // giveMeFive가 true이면 10을 넣음
];
print(item);
}

list[1,2,3]을 list[1,2,3,]처럼 ', '를 넣어주면 자동 줄바꿈
```

string interpolation

```

void main() {
  bool isKorAge = false;
  List numberList = [
    1,
    if (isKorAge) 30 else 31,
  ];

  String myName = "Ahn";

  String greeting =
    "Hello, My name is $myName and I am ${numberList.last} years old.

  print(greeting);
}

Hello, My name is Ahn and I am 31 years old.

//리스트의 프로퍼티는 가져올때는 $와 함께 {}
//그냥은 $

// 작은따옴표로 감싸는데 문장 내부에 I'm 등 쓰게되면 I\'m 처리 해줘야함

```

collection for

```

// Dart는 조건문(if) 및 반복(for)을 사용하여 컬렉션을 구축하는 데
// 사용할 수 있는 컬렉션 if 및 컬렉션 for도 제공

void main() {
  var oldFriends = ["nico", "lynn"];
  var newFriends = [
    "tom",
    "jon",
    for (var friend in oldFriends) "❤️ $friend"
  ];
}

```

```
print(newFriends); // [tom, jon, ❤️ nico, ❤️ lynn]
}
```

// List comprehension과 비슷

Maps

일반적으로 맵은 key와 value를 연결하는 객체입니다.

키와 값 모두 모든 유형의 객체가 될 수 있습니다.

각 키는 한 번만 발생하지만 동일한 값을 여러 번 사용할 수 있습니다.

``

```
var gifts = {
// Key: Value
'first': 'partridge',
'second': 'turtledoves',
'fifth': 'golden rings'
};
```

```
void main() {
  var gifts2 = Map();
  gifts2['first'] = 'a';
  gifts2['second'] = 'b';
  gifts2['fifth'] = 'c';

  print(gifts2.containsKey('first')); // true
  print(gifts2['first']);
}
```

Sets

sets ⇒ items are always unique

lists ⇒ items are not always unique

```
//set도 두 가지 방법으로 정의할 수 있다.
```

```
//1. var을 사용
```

```
var numbers = {1, 2, 3};
```

```
//2. 자료형 명시
```

```
Set numbers = {1, 2, 3};
```

list는 대괄호를 쓰며 set은 중괄호를 쓴다는 것과 set의 요소들은 유니크하다
list는 같은 요소가 여러개 반복될 수 있지만, set은 중복이 허용되지 않는다

#3 FUNCTIONS

Dart는 진정한 객체 지향 언어이므로 **함수도 객체이며 타입이 Function**입니다.

이는 함수를 변수에 할당하거나 다른 함수에 인수로 전달할 수 있음을 의미합니다.

```
String sayHello(String name) {  
  return "Hello my name is $name nice to meet you";  
}
```

// 하나의 표현식만 포함하는 함수의 경우 아래와 같이 단축 구문을 사용할 수 있다.

```
String sayHello(String name) => "Hello ${name} nice to meet you";  
  
num plus(num a, num b) => a + b;  
  
void main() {  
  
  print(sayHello("sugar"));  
  
}
```

Named Parameters

Named parameters는 명시적으로 required로 표시되지 않는 한 선택 사항 기본값을 제공하지 않거나 Named parameters를 필수로 표시하지 않으면, 해당 유형은 기본값이 null이 되므로 null을 허용해야 함

```
String sayHello(  
  {required String name, required int age, required String coun  
  return "${name} / ${age} / ${country}";  
}  
  
void main() {  
  print(sayHello(name: "sugar", age: 10, country: "Korea"));  
}  
  
//default값을 줘도 됨  
  
String sayHello({  
  String name = 'anon',  
  int age = 99,  
  String country = 'wakanda',  
}) {  
  return "Hello $name, you are $age, and you come from $count  
}  
  
void main() {  
  print(  
    sayHello()); // 아무것도 전달하지 않아도 default value가 이미  
}
```

required을 사용하면 null safety를 적용할 수 있다.

(required를 쓰면 반드시 값이 있어야 한다)

Optional Positional Parameters

Dart에서 `[]` 은 optional, positional parameter를 명시할 때 사용된다.

name, age는 필수값이고 `[]` 를 통해 country를 optional값으로 지정해줄 수 있다.

```
String sayHello(String name, int age, [String? country = ""])
return 'Hello ${name}, You are ${age} from the ${country}';
}

void main() {
var result = sayHello("sugar", 10);
print(result);
}
```

`String? country='cuba']` → cuba

`[String country='cuba']` → cuba

`[String? country]` → null

`[String country]` → error (컴파일 안됨)

QQ Operator

?? 연산자를 이용하면 왼쪽 값이 null인지 체크해서

null이 아니면 왼쪽 값을 리턴하고

null이면 오른쪽 값을 리턴한다.

```
String capitalizeName(String? name) {
return name?.toUpperCase() ?? "";
}
// 이름을 대문자로 변환합니다. 만약 name이 null이면 빈 문자열("")을 반환합니다.
```

// ??= 연산자를 이용하면 변수 안에 값이 null일 때를 체크해서 값을 할당해줄 수 있다.

```
void main() {  
    String? name;  
    name ??= "sugar";  
  
    name = null;  
    name ??= "js";  
  
    print(name); // js  
}
```

즉, ??=는 해당 변수가 null이면, 오른쪽을 반환함

Typedef

자료형에 사용자가 원하는 **alias**

typedef ListOfInts = List

```
// 전  
List reverseListOfNumbers(List list) {  
    var reversed = list.reversed;  
    return reversed.toList();  
}  
  
// 후  
typedef ListOfInts = List; // List를 ListOfInts로 바꾼 것  
ListOfInts reverseListOfNumbers(ListOfInts list) {  
    var reversed = list.reversed;  
    return reversed.toList();  
}
```

4.0 Your First Dart Class

```
class Player {  
  
    final String name = 'jisoung';  
  
    final int age = 17;  
  
    void sayName(){  
  
        // class method안에서는 this를 쓰지 않는 것을 권장한다.  
        // 변수명이 겹치지 않는이상 this를 생략  
        print("Hi my name is $name")  
  
    }  
}  
  
void main(){  
    var player = Player();  
}
```

Constructors

```
class Player {  
    final String name;  
    int xp;  
  
    //생성자  
  
    Player(this.name, this.xp);  
  
    void sayHello() {  
        // class method안에서는 this를 쓰지 않는 것을 권장한다.  
        // 변수명이 겹치지 않는이상 this를 생략  
        print("Hi my name is $name");  
    }  
}
```

```

}

void main() {
    var player = Player("안우섭", 1500);
    player.sayHello();
    var player2 = Player("안우섭2", 1500);
    player2.sayHello();
}

```

Named Constructor Parameters

```

class Player {
    final String name;
    int age;
    String team;

    Player({
        required this.name,
        required this.age,
        required this.team,
    }); //{} 추가하기

    void sayHello() {
        print("Hi my name is $name");
    }
}

void main() {
    var player = Player(
        name: "nudge",
        age: 1,
        team: 'red',
    );

    player.sayHello();
}

```

Named Constructors

