Dart Syntax

다트문법

다트패드

- 다트문법 연습 공식 제공 사이트
- 다트패드
- https://dartpad.dev/

기본문법 : 주석

주석

```
1     //주석
2
3      /**
4     * 주석
5      **/
6
7   /// 문서주석
```

문장

명령 단위. 세미콜론 ;으로 끝난다.

기본문법 : 변수

변수

data를 담는 상자. 종류는 type, 자료형이라고 함.dart는 int, double, String, bool을 기본 제공.사용자가 직접 타입을 정의할 수 있음.

```
1       String name;
2     name = "홍길동";
3     name = '홍길동';
4  //따옴표 종류 상관없이 가능
```

int, double은 num type의 하위 집합이라 int, double 대신 num으로 선언 가능.

num 타입에는 int, double 대입 가능.

```
1    int a = 1;
2 double b = 2.0;
3
4    num c = a;
5      c = b;
```

기본문법 : var, const

타입 추론

위와 같이 type을 직접 명시하지 않고, var로 대체할 수 있음. (JS같이)

일반적으로 많이 사용.

```
    var i = 10; //int
  var d = 10.0 //double
var s = "hello"; //String
 var b = true; //boolean
```

상수 final, const

값이 변하지 않는 경우는 상수 사용. 선언시 final을 제일 앞에 붙이면 값이 수정되지 않음. 타입 생략 가능.

```
final String name = "홍길동";
   final name = "홍길동";
```

기본문법 : 연산자

산술 연산자

+, -, *, /(나누기 - double), ~/(몫 - int), %(나머지 - int) 사용 가능.

+의 경우 string concat에서도 사용.

증감 연산자

1씩 증가(++) 또는 1씩 감소(--). 후위(식++)/전위(++식) 연산 모두 가능.

비교 연산자

==, !=, >, <, >=, <= 사용 가능.

논리연산자

boolean 타입으로 결과 반환.

&&, ||, ==, !, != 사용 가능.

기본문법 : 타입검사, 형변환

타입 검사

is를 사용.

[변수] is [type] => 같은 타입인지[변수] is! [type] => 다른 타입인지

```
    int a = 1;

 if (a is int){~}
if (a is! int){~}
```

형변환 (as)

type casting. as를 사용. 다른 타입끼리는 변환 불가. 상위 개념으로만 변환 가능.

특히, int, double은 num으로 묶여있지만 각자는 관계가 없어 형변화 불가.

```
        var c = 10.0
     num n = c as num;
num n = c //as num 생략 가능
 int d = c as int; //errpr
```

기본문법 : 함수

함수

function. 코드 묶음 단위.

함수 형태

입력 받는 문자 : 매개변수 parameter실제로 입력받는 값 : 인수 argument반환되는 값 : 반환값 Return

=> 일반적인 C문법과 동일하다.

변수 앞에 $를 붙여 문자열 내에 변수 삽입 가능. (파이썬에서 {}사용하는 것과 같다)

또한, ${~~}부분에 표현식 사용 가능.

```
1            String name = "이순신";
2                 int age = 20;
3
4                 void main(){
5              print('$name은 $age살 입니다.');
6       print("$name은 ${name.length}글자 입니다.");
7                         }
```

기본문법 : 익명함수, 람다식

함수와 메서드

최상위 함수 : class 밖에 작성하는 함수. 일반적으로 말하는 함수들. (main method처럼 가장 밖에 작성). 어디에서나 호출 가능.

method : class 내부에 작성하는 함수. 정의된 class에 관계된 기능을 수행. Static이 붙은 method는 정적 메서드가 되어 최상위 함수처럼 사용 가능.

익명 함수

anonymouse function.

([인수명]) { [동작 or Return 값] }

```
1            (number) {
2        return number % 2 == 0;
3              };
```

람다식

([인수명]) => [동작 or Return 값]

```
(number) => number%2 == 0;
```

기본문법 : 매개변수

선택 매개변수

정의에서 {}로 감싼 매개변수는 선택적으로 사용 가능. Named parameter.

```
void f(String name, {int age}) {}
```

{}로 감싸지 않은 매개변수를 필수적으로 넘겨주어야 하고, {}로 감싼 매개변수에 값을 주고 싶다면 반드시 매개변수의 이름을 붙여서 age : 5로 넘겨주어야 한다.

선택 매개변수는 기본값 지정이 가능하다.

```
void f(String name, {int age = 5}) {}
```

기본문법 : 분기문

분기와 반복

if else

```
        int a = 10;

    if (a % 2 == 0) {~}
else if (a % 3 == 0) {~}
        else {~}
```

삼항 연산

[조건] ? [참일 때] : [거짓일 때]

```
var number = a%2 ? a++ : a--
```

switch case

조건에 맞는 값이 여러 개일 때 유용. enum과 함께 사용할 땐 모든 케이스를 검사해야 하는 강제성이 생김.

Switch - case - break로 문법은 C와 동일

기본문법 : for문

for

요소를 반복.

```
for (var i = 0; i < 10 ; i++){
                print(i);
                }
```

기본문법 : 클래스

객체 지향 프로그래밍

dart는 객체지향 프로그래밍 언어!

클래스

객체 object : 저장 공간에 할당되어 값을 가지거나 식별자에 의해 참조되는 공간. (변수, 함수, 메서드)

인스턴스 instance : 객체를 메모리에 작성하는 것.

클래스 class : 인스턴스의 설계도.

속성 property : 클래스 안에 표현되는 속성.

클래스는 일종의 사용자 정의 타입!

```
1              class Person{
2                   String name;
3                   int age;
4                    }
5
6         var person = new Person();
7   var person2 = Person(); //new 생략 가능
```

기본문법 : 클래스

변수명 뒤에 . 연산자를 입력, 객체의 property에 접근 가능.

```
print(person.name);
print(person.age);
```

메서드 : 클래스 안에 작성하는 함수. 메서드는 class의 프로퍼티를 조작하는 등의 용도로 사용. 이 역시 .연산자를 붙여 접근 가능.

기본문법 : 접근 지정자

접근 지정자

java에서의 public, private와 같은 역할. 여기서는 _ 을 붙여 구분한다. (붙이면 private)

```
class Person{
    String name;
    int _age;

  void addOneYear(){
      _age++;
    }
  }

//=====다른 Dart file
import 'person.dart';

  void main(){
  var person = Person();
person._age = 10; //error!
    }
```

이때, private의 영역은 정의된 파일 내! 즉, _이 붙어 있어도 같은 파일 내라면 class의 밖에 있어도 자유롭게 접근이 가능하다.

기본문법 : 생성자

생성자

인스턴스화 하는 방법을 제공하는 일종의 메서드. 인스턴스가 생성될 때 가장 먼저 생성됨. 메서드 이름은 클래스명과 같다.

```
                    class Person{
                        String name;
                        int _age;

    Person({this.name, this._age}); //constructor
                         }

                var person = Person();
var person2 = Person(name : '홍길동', _age : 20);
```

기본문법 : Getter, Setter

getter, setter

_을 붙인 private 변수에 외부에서 접근하기 위해 만들어주는 통로같은 메서드.

```dart
//person.dart

class Person{
    String name;
  int _age = 10;

int get age => _age; //getter
set setAge(num value) => _age = value; //setter
}

//main.dart

void main(){
var person = Person();
print(person.age); //10
person.setAge = 5;
print(person.age); //5
}
```

기본문법 : 클래스 상속

상속

implement. 주는 쪽이 super class (부모), 받는 쪽이 sub class(자식)

```
                    //부모 class
                    class Hero{
                   String name = 'hero';

                     void run() {}
                         }

                //위를 상속받은 자식 클래스
            class SuperHero extend Hero{
                     @override //재정의
                      void run(){
                  super.run(); //부모의 run실행
        this.fly(); //추가로 자식이 따로 정의한 fly도 실행.
                           }

                     void fly() {}
                         }


                    void main(){
                   var hero = SuperHero();
                      hero.run();
                      hero.fly();
                print(hero.name); //hero 출력
                           }
```

기본문법 : 추상 클래스

추상 클래스

abstract class. 추상 메서드를 포함하는 클래스.

추상 메서드 : 선언만되고 정의가 없는 메서드.

추상 클래스는 다른 클래스에서 implement해서 완성하는 상속 재료로 사용된다. (틀같은 느낌? 템플릿?)

이때 대상 클래스에는 implements, 메서드에는 @override 키워드 사용(

여러 추상 클래스를 한번에 implement할 수도 있고, 추상 클래스를 구현할 때는 모든 추상 메서드를 재정의 해야 한다.

```
        abstract class Monster{
                void attack();
                  }

        abstract class Flyable{
                 void fly();
                  }

class Bat implements Monster, Flyable(
                  @override
               void attack(){
                print('할퀴기!');
                     }

                 @override
                void fly(){
               print('펄럭펄럭');
                     }
                  }
```

기본문법 : 믹스인

믹스인

mixin. with를 사용하면 상속하지 않고 다른 클래스 기능 overwritte가능.

```
                    class Goblin implements Monster{
                                    @override
                                void attack(){
                                print('고블린 공격');
                                    }
                                }

                class DarkGoblin extends Goblin with Hero{

} //다크 고블린은 고블린이기도 하며, 히어로이기도 하며, 몬스터이기도 함. => 다형성!
```

기본문법 : 열거형

열거형

enum type. 상수를 정의하는 특수한 형태의 클래스. 상수처럼 사용이 가능하다.

```
 enum Status { login, logout }

var authStatus = Status.logout;

      switch(authStatus){
          case Status.login:
           print('login');
               break;
        case Status.logout:
           print('logout');
               break;
             }
```

기본문법 : 컬렉션 List

컬렉션

list, map, set등의 collection을 기본 제공함.

List

순서가 있는 자료를 담음. Dart는 배열 (Array)를 별도로 제공하지 않는다.

```
List<String> items = ['a', 'b', 'c'];
    var items = ['a', 'b', 'c'];

items[0] = 'd'; //0부터 시작하는 index.

      print(items.length); //3
```

👉 ※ dynamic
모든 타입을 대변하는 특수 타입! 여러 타입을 한 리스트에 넣거나 일반 변수를 선언할 때도 사용 가능.

List<dynamic> list = [1, 2, 'a'];
var list = [1, 2, 'a'];

기본문법 : 컬렉션 Map

스프레드 연산자

spread. '...' 연산자. 컬렉션을 펼쳐준다. 다른 컬렉션 안에 컬렉션을 삽입할 때 사용.

```
            var items = ['a', 'b', 'c'];

var items2 = ['d', ...items, 'e']; //d, a, b, c, e
```

Map

순서 없음. 탐색 빠른 자료구조. key-value의 쌍.

```
//Map<String, String> cityMap = {...} 과 같음.
                var cityMap = {
                   'korea': 'busan',
                 'japan': 'tokyo',
                'china': 'Beijing'
                      };

          cityMap['korea'] = 'seoul';

          print(cityMap.length); //3

cityMap['America'] = 'Washington'; //새 값 추가
```

기본문법 : 컬렉션 Set

Set

집합 표현. => 중복 불허용

add(), remove()로 추가/삭제 가능.

contains() : 찾는 자료가 집합에 있는지 없는지 bool로 반환.

```
        // Set<String> citySet = {}과 같음
var citySet = {'서울', '부산', '광주', '대전', '울산'}

              citySet.add('대구');
            citySet.remove('서울');

      print(citySet.contatins('울산'); // true
```

비어있는 Set이나 map을 작성할 때는 주의! 그냥 {}만 쓰면 Map으로 인식해버림.

```
      var mySet = <String>{}; //set으로 인식
var mySet2 = {}; //dynamic, dynamic인 map으로 인식
```

기본문법 : 함수형 프로그래밍

함수형 프로그래밍

Dart는 객체지향 프로그래밍과 함수형 프로그래밍의 특징을 모두 제공

함수형 프로그래밍 : 자료 처리를 수학적 하수의 계산으로 취급하는 패러다임. (상태와 가변 데이터 X)

일급 객체

함수를 값으로 취급. 즉, 다른 변수에 함수를 대입할 수 있음.

```
1          void greeting(String text){
2                      print(text);
3                        }
4
5  void main(){ // 함수를 다른 변수에 대입할 수 있음
6                     var f = greeting;
7                      f('hello');
8                        }
```

다른 함수의 인수로 함수 자체를 전달하거나 함수를 반환받을 수도 있음.

함수를 매개변수로 전달, 수정, 변수에 대입하기가 가능한 객체를 '일급 객체', first-class object라고 함.

기본문법 : forEach(), where

for문과 forEach()함수

for : 외부 반복

forEach() : 내부반복 =>(E element) {} 형태의 함수를 인수로 받음.

```
items.forEach(print); // 1, 2, 3, 4, 5

          items.forEach((e) {
                    print(e);
                  });

    itmes.forEach((e) => print(e));

         items.forEach(print);
```

where

조건을 필터링 할 때 사용. 함수형 프로그래밍을 지원하는 함수들은 결과를 반복 가능한 타입으로 반환하여 메서드 체인으로 연결해서 사용 가능.

```
items.where((e) => e % 2 == 0).forEach(print); //2, 4
```

기본문법 : map, toList, toSet

map

반복되는 값을 다른 형태로 변환하는 방법을 제공.

```
items.where((e) => e % 2 == 0).map((e) => '숫자 $e').forEach(print);
```

toList

함수형 프로그래밍을 지원하는 함수 대부분은 Iterable<T> 인터페이스 타입 인스턴스를 반환.

하지만 실제 사용할 때는 대부분 리스트 형태로 변환해야 하는 경우가 많음. => 결과를 리스트로 바꿔야함.

```
final result = itmes.where((e) => e % 2 == 0).toList();
```

toSet

리스트에 중복된 데이터가 있을 경우, 중복을 제거한 리스트를 얻고 싶을 때 집합인 set을 사용.

```
final result = itmes.where((e) => e % 2 == 0).toSet().toList();
```

기본문법 : any, reduce

any

리스트에 특정 조건을 충족하는 요소가 있는지 없는지 검사할 때 사용하는 함수.

```
print(items.any((e) => e % 2 == 0));
```

reduce

반복 요소를 줄여가면서 결과를 만들 때 사용하는 함수.

```
1  // 최댓값을 구할 때, 순차적으로 비교하는 로직
2
3   final result = items.reduce(max); //5
```

기본문법 : ..연산자, 컬렉션if

기타

계단식 표기법 .. 연산자

cascade notation .. 연산자. 동일 객체에서 일련의 작업을 수행 가능.

.. 연산자를 사용하면 메서드를 수행한 객체의 참조를 반환.

```
1            print(items
2                  ..add(6)
3      ..remove(2)); //1, 3, 4, 5, 6
```

컬렉션 if

조건에 의해 컬렉션의 값을 조정하거나 다르게 사용하고 싶을 때 사용.

```
1                    bool promoActive = true;
2
3  print([1, 2, 3, 4, 5, if (promoActive) 6]); // true일 때만 6이 추가됨
```

기본문법 : 컬렉션for,null처리

컬렉션 for

컬렉션 문법 안에서 for 문을 사용 가능.

```
1       var listOfInts = [1, 2, 3];
2          var listOfStrings = [
3                      '#0',
4       for (var i in listOfInts) '#$i'
5                      ];
6             //#0, #1, #2, #3
```

null 처리

다트는 null을 처리할 수 있는 여러가지 방법을 제공. (Dart에서는 모든 것이 객체. int, double, bool 같은 타입들도 모두 객체라 null값을 가질 수도 있음.)

?. 연산자를 사용하면 null 여부 파악 가능.

```
print(name?.lentgh); //null출력
```

??연산자는 객체가 null일 때 작동을 간단히 구현하는데 사용.

```
print(name?.length ?? 0); //name이 null이면 0을 출력
```