다트 문법 정리

dartpad.dev/?로 실습가능

Hello World

```
void main() {
print("Hello World!);
}
```

변수 선언하기

```
void main() {
  var name = "다트";
  print(name);

  var name2 = "블랙핑크";
  print(name2);

  name = "플러터 프로그래밍"
  print(name);

  var name = "다트2" // 변수 같은 이름으로 재 선언 안됨
  print(name);
}
```

변수 타입

```
void main() {
  // 정수
  // integer
  int number1 = 10;
  print(number1);

  int number2 = 15;
  print(number2);

  int number3 = -20;
  print(number3);

  int number4 = 2;
  int number5 = 4;

  print(number4 + number5);
  print(number4 - number5);
```

```
print(number4 / number5);
print(number4 * number5);

// 실수
// double
double number6 = 2.5;
double number7 = 0.5;

print(number6 + number7);
print(number6 - number7);
print(number6 / number7);
print(number6 * number7);

//맞다 /틀리다
// Boolean
bool isTrue = true;
bool isFalse = false;

print(isTrue);
print(isFalse);

// 글자 타입
// String
String name = '블랙핑크';
String name2 = '다트';

print(name);
print(name2);

print(name + name2); // 문자열 붙이기 블랙핑크다트

print('${name} ${name2}'); // 블랙핑크 다트

print('$name.runtime $name2') //블랙핑크.runtimeType 타입

//함수가 들어가면 {}를 써야한다

// var String의 차이
var name3 = '블랙핑크'; // String 타입으로 유추됨
var number = 20;  // 정수 타입으로 유추됨

print(name3.runtimeType); // String 출력됨

dynamic name = '다트';
print(name);

dynamic number = 1;
print(number);

var name2 = '블랙핑크';
print(name2);

print(name.runtimeType);
print(name2.runtimeType);
```

```
  name = 2;
  name2 = 5; //오류

// var type은 한번 선언하면 선언할 때 그 타입으로 fix가 된다. 따라서 다른 type으로 변경이 불가능하다.
하지만 dynamic 타입은 아무거나 다 된다.
}
```

왜 var 타입을 안 쓰는가?

코드는 몇만, 몇 십만 줄 까지 가는데, 그런 코드 속에서 전부 var 타입으로 선언되있으면 보기가 힘들어진다. 어떤 타입인 줄 알 수가 있으면, 내 코드가 아니라 남의 코드일 경우에도 이해하기가 더 쉽다. 단 타입이 굉장히 복잡한 경우, 직접 명시하는 가치가 없을 경우 var을 쓰는 것이 적합하다.

Nullable vs Non-nullable

```
void main() {
  // nullable - null이 될 수 있다.
  // non-nullable - null이 될 수 업다.
  // null - 아무런 값도 있지 않다.
  String name = '다트'
  print(name);

  String? name2 = "블랙핑크';
  name2 = null;

  print(name2!);
  //물음표는 널이 들어갈 수 있다. 느낌표는 널이 절대 아니다.
}

## final vs const

'''
void main() {
  final String name = '다트';
  print(name);

  name = '블랙핑크' // 오류! final로 변수를 선언하면, 변수의 값을 선언한 뒤로 값을 변경할 수 없다.

  const String name2 = '블랙핑크';
  print(name2);

  name2 = '다트'; // 오류 final과 마찬가지로 const도 변수 값을 선언한 뒤로 변경할 수 없다.

  //  만일 var 타입일 경우 final과 const는 var 타입을 생략할 수 있다.
  final name = '다트';
  const name2 = '블랙핑크';
```

```
  DateTime now = DateTime.now();
  print(now); // DateTime.now() 코드가 실행될 때의 시간과 날짜를 출력한다.

  final DateTime now = DateTime.now(); // 실행됨 final은 빌드 타임에 값을 알 필요가
없다.
  const DateTime now = DateTime.now(); // 오류 const는 빌드 타임에 값을 알아야 한다.

}
```

빌드 타임이란?

코드를 작성하면 이진수로 변환이 된다. 사람이 이진수로 코드를 짤 수는 없으니 사람이 이해할 수 있는 자연어로 프로그래밍 언어를 만든 것이다. 이 때 빌드가 프로그래밍 언어가 이진수로 변환하는 과정을 말한다.

그럼 빌드타임에 값을 알고 있어야 한다는 뜻은?

코드를 작성하는 순간에 코드의 값을 컴퓨터가 알고 있어야한다는 것이다. DateTime.now()는 실행이 되는 순간인 런타임에만 알 수 있다.

Operators

```
void main(){
  int number = 2;

print(number); // 2
print(number + 2);
print(number - 2);
print(number * 2);
print(number / 2);
print(number % 3); //2

print(number); // 2
number ++;
print(number); // 3
number--
print(number); //2

double number = 4.0;
print(number);

number += 1;
print(number);

number -= 1;
print(number);

number *= 2;
print(number);

number /= 2;
```

```
print(number);

double? number = 4.0;
print(number);

number = 2.0;
print(number);

number = null;

print(number);

numeber ?? = 3.0; // number가 만일 null이면 오른쪽 값으로 바꿔라
print(number);


int number1 = 1;
int number2 = 2;

print(number1 > number2);
print(number1 < number2);
print(number >= number2);
print(number1 <= number2);
print(number1 == number2);
print(number1 != number2);

int number = 1;

print(number1 is int);
print(number1 is String);
print(number1 is! int);
print(number1 is! String);

// && - and 조건
// || - or 조건
bool result = 12 > 10 && 1 > 0;
bool result2 12 > 10 && 0 > 1;
bool result3 12 > 10 || 1 > 0;
bool result4 12 < 10 || 0 > 1;
bool result5 12 < 10 || 0 > 1;
}
```

List

```
void main() {
  // List
  // 리스트
  List<String> blackPink = ['제니', '지수', '로제', '리사'];
  List<int> numbers = [1, 2, 3, 4, 5, 6];
  print(blackPink);
  print(numbers);
```

```
  // index
  // 순서
  // 0 부터 시작
  print(blackPink[0]);
  print(blackPink[1]);
  print(blackPink[2]);
  print(blackPink[3]);
  print(blackPink[4]); // 오류 Index out of range

  print(blackPink.length);

  blackPink.add('다트');

  print(blackPink);

  black.Pink.remove('다트');
  print(blackPink);

  print(blackPink.indexOf('로제'));
}
```

Map

```
void main(){
  // Map
  // Key / Value
Map<String, String> dictionary = {
  'Harry Potter' : '해리포터', 'Ronald Weasley : '론 위즐리', "Hermaione Granger" :
'헤르미온느 그레인저',
};

print(dictionary);

Map<String, bool> isHarryPotter = {
  'Harry Potter' : true, 'Ronald Weasley : true, 'Ironman' : false,
};

print(isHarryPotter);

isHarryPotter.addAll({
'Spiderman':false});

print(isHarryPotter);

print(isHarryPotter['Ironman']);

isHarryPotter['Hulk'] = false;
print(isHarryPotter);
isHarryPotter['Spiderman'] = true;
print(isHarryPotter);
```

```
isHarryPotter.remove('Harry Potter');

print(isHarryPotter);
print(isHarryPotter.keys); // 키값 모두 가져옴
print(isHarryPotter.values); // 밸류 값 모두 가져옴
}
```

Set

```
void main(){
  // Set
  final Set<String> names = {
  'Dart',
  'Flutter',
  'Black Pink',
  'Flutter',
  }; // 중복값 처리 해줌
  print(names);

  names.add('Jenny');

  print(names);

  names.remove('Jenny');

  print(names);

  print(names.contains('Flutter'));
}
```

If문

```
void main() {
  // if 문

  int number = 2;

  if(number % 3 == 0){
    print('나머지가 0입니다.');
  } else if(number % 3 == 1){
      print('나머지가 1입니다.');
  } else{
      print('나머지가 2입니다.');
  }

  // switch 문
  int number = 3;
```

```
  switch(number % 3){
    case 0:
      print('나머지가 0입니다');
      break;
    case 1:
      print('나머지가 1입니다');
    default:
      print('나머지가 2입니다');
      break;
  }
}
```

Loops

```
void main() {
// for loop
  for(int i = 0; i < 10; i++){
    print(i);
  }

  int total = 0;

  List<int> numbers = [1, 2, 3, 4, 5, 6];
  for(int i = 0; i < numbers.length; i++){
    total += numbers[i];
  }
  print(total);

  total = 0;

  for(int number in numbers){
    total += number;
  }

  //while loop

  int total = 0;

  while(total < 10){
    total += 1;

    if(total == 5){
    break;
    }
  }

  print(total);

  do {
  total += 1;
```

```
  } while(total < 10);

  print(total);

  for(int i = 0; i< 10; i++){
    total += 1;
    if(total == 5){
    break;
    }
  }

  for(int i = 0; i < 10; i++){
    if(i == 5){
      continue; // 현재 loop만 스킵함
    }
  }
}
```

enum

```
enum Status{
  approved, // 승인
  pending, // 대기
  rejected, // 거절
}

void main() {
  Status status = Status.pending;

  if(status == Status.approved){
    print('승인입니다');
  }else if(status == Status.pending){
    print('대기입니다.');
  }else{
    print('거절입니다.');
  }
}
```

enum을 쓰는 이유

```
void main() {
  String status = 'pending';

  if(status == approved){
    print('승인입니다');
  }else if(status == pending){
    print('대기입니다.');
  }else{
    print('거절입니다.');
```

```
  }
}
```

이런 식으로 그냥 String 타입 써서 문자열로 처리할 수 있지 않는가? 왜 굳이 enum을 써야하는가?

왜 enum을 쓰냐면은

1. 정확히 approved, pending, rejected 3개만 존재한다는 것을 다른 개발자 또는 미래의 나에게 알려주기 위해서
2. 오타가 하나라도 나면 오류가 나기 때문에 enum을 쓰면 자동완성을 사용하여 오류를 방지할 수 있다.

함수

함수는 반복되는 로직(코드)을 한 번만 작성하고서 재활용 할 수 있게 하는 것이다.

```
void main(){
  addNumbers(10, 20, 30);
  addNumbers(x: 10, y : 20, z : 30);
  addNumbers(20, 30, 40);
  addNumbers(y : 30, x : 10, z : 40);
}

// 세개의 숫자 (x, y, z)를 더하고 짝수인지 홀수인지 알려주는 함수
// parameter / argument - 매개변수, 인수
// positional parameter - 순서가 중요한 파라미터
// optioanl parameter - 있어도 되고 없어도 되는 파라미터를 대괄호를 쌓으면 된다. 대신
null 변수로 바꾸거나 기본값을 넣어주어야한다.
// named parameter - 이름이 있는 파라미터 (순서가 중요하지 않다.)
// arrow function - 화살표 함수 화살표 다음에 오는 값이 리턴 값이다.

addNumbers(int x, {required int y, int z = 30,}){
// named parameter
}
addNumbers(int x, {required int y, int z = 30,}) = > x + y + z

addNumbers(int x, [int y = 20, int z = 30]){
int sum  = x + y + z;

print('x : $x);
print('y : $y');
print('z : $z');

if(sum %2 == 0){
  print('짝수입니다.');
} else{
  print('홀수입니다.');
}

}
```

type def

함수를 편리하게 사용할 수 있는 기능 함수와 비슷하지만, 함수의 바디가 없다.

```
void main(){
Operation operation = add;

int result operation(10,20,30);

print(result);

operation = subtract;

int result2 = operation(10,20, 30);
print(result2);

int result3 = operation(10, 20, 30, add);

print(result3);

int result4 = operation(10, 20, 30, sub);

print(result4);

}


// 시그니처
type def Operation = int Function(int x, int y, int z);

// 더하기
int add(int x, int y, int z) => x + y +z;

// 빼기
int subtract(int x, int y, int z) => x - y - z;

int calculate(int x, int y, int z, Operation operation){
  return operation(x, y, z);
}
```