

---

# Dart 기초 문법

---

# 목차

---

1. 자료형 - 타입 추론
2. 연산자
3. 함수
4. **typedef**
5. 문자열 변수 사용
6. 상수
7. **enum**
8. **null**

```

1 ▼ void main() {
2     //변수 type
3     String name = "Bob";
4     var friend = "Tom"; //type 추론
5     dynamic bestFriend = "Kan"; //type 변경
6
7
8
9     //List
10    List<int> Nums = [1,2,3];
11    Nums[0] = 9; //변경
12
13    //Map key & value pair
14 ▼ Map<String, int> NameAge = {
15        "qwer" : 20,
16        "asdf" : 30
17    };
18    NameAge["qwer"] = 40; //key 변경
19 }

```

▶ Run

## 자료형(type)

- int: 정수형
- double: 실수형
- num: int, double을 포함하는 타입
- bool: true, false
- String: 문자열

## - 집합 자료형

- List: 중복을 허용하며 순서가 있는 집합
- Set: 중복을 허용하지 않고 순서가 없는 집합
- Map: key-value 쌍으로 구성된 집합

**dynamic** - 모든 타입을 대변하는 특수 타입!

여러 타입을 한 리스트에 넣거나 일반 변수를 선언할 때도 사용 가능.

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

# num

---

```
void main() {  
    //자료형  
    int age = 25; //정수  
    double weight = 65.4; //실수  
  
    num age1 = age; //num타입은 int, double형 모두 대입가능  
    num weight1 = weight;  
    print(age1);  
    print(weight1);  
}
```

▶ Run

Console

25

65.4

num은 int, double 모두 받을 수 있지만,  
반대는 불가능하다

# var

---

```
void main() {  
  //자료형 - 타입 추론  
  var age = 23;  
  var weight = 78.4;  
  var word = "dart";  
  var t = true;  
  var x = word.isEmpty;  
  print("${age.runtimeType}, ${x.runtimeType}");  
}
```

▶ Run

Console

int, bool

var 를 사용하게 되면,  
선언 후 타입이 추론되어 결정된다  
cf.runtimeType - 타입 반환 함수

```
//var, dynamic 타입
void main() {

//var로 선언하여 문자열 할당
  var word = '랜덤값';
  print(word.runtimeType);

//이번에는 var로 선언하여 숫자값을 할당
  var number = 1;
  print(number.runtimeType);

//var형은 타입이 정해진 이후, 형을 변환할 수 없다
  //int word = 2; => 컴파일 에러

//위에 var로 선언 후, 다시 문자열을 재할당 해보자.
  word = '랜덤값 변경';

//문제없이 변경된 값으로 출력된다.
  print(word);
}
```

▶ Run

Console

String  
int  
랜덤값 변경

Documentation

var타입의 특이점-[DartPad](#)

var선언자는 내가 할당한 값에 의해 타입이 정해지고, 그 이후에는 타입을 바꿀 수 없다.

# dynamic

---

```
▼ void main() {  
    //이번에는 dynamic이라는 선언자를 살펴보자.  
    dynamic Name = '다이나믹값';  
  
    //var 선언자를 사용했을때 처럼 문제없이 출력된다.  
    print(Name);  
  
    //var 선언자와 달리  
    //dynamic 선언자를 사용하면 어려없이 값의 종류를 바꿀수 있다|  
    //dynamic은 할당된 변수값의 종류에 영향없이 타입이 변경되는 선언자이다  
    Name = 1;  
    print(Name);  
}
```

▶ Run

Console

다이나믹값  
1

## 연산자

---

논리 연산자

&&: 그리고

||: 또는

증감 연산자

전위 연산: ++ [식] --[식]

후위 연산: [식] ++, [식]--

증감연산자퀴즈-[DartPad](#)

```
void main() {  
    //산술 연산자(+, -, *, /, ~/ , %)  
    var a = 5;  
    var b = 2;  
    print(a ~/ b); // 몫(int)  
    //증감 연산자  
    print(a += 4);  
    //비교, 논리 연산자  
    if (a >= 8 && b == 2) {  
        print("true");  
    }  
}
```



## 삼항 연산자

```
void main() {  
    int a = 3;  
    int b = 4;  
    int bigger;  
  
    if (a > b) {  
        bigger = a;  
    } else {  
        bigger = b;  
    }  
}
```

if-else 사용



```
void main() {  
    int a = 3;  
    int b = 4;  
    int bigger;  
  
    bigger = (a > b) ? a : b;  
}
```

삼항 연산자 사용

Condition ? A : B

Condition이 참이면 A 실행,  
거짓이면 B 실행

if-else 가 있는데 삼항 연산자를 쓰는  
이유는 코드의 길이를 줄일 수 있는  
경우도 있지만,

if/else - 문(statement) 으로서 if  
자체로는 아무런 값을 만들어내지  
않는다.

삼항 연산자 - 식(expression)  
으로서 값을 만들어낸다.

## null check 연산자

```
void main(){
```

```
String a;
```

```
String b = 'Hi';
```

```
String c = 'Good';
```

```
a = b ?? c;
```

```
//a = b ?? c;
```

```
//b가 null이 아니면 a에 배정한다.
```

```
print('a = a');
```

```
}
```

Run

Console

a = Hi

```
void main() {
```

```
int? a; //null
```

```
a ??= 2; //2 할당
```

```
print(a); //2 출력
```

```
//a ??= b;
```

```
//a가 null이면 b를 넣는다.
```

```
int? b = 1; //null 아님
```

```
b ??= 2; //2 할당
```

```
print(b); //에러는 안나지만 1 출력
```

```
}
```

Run

Console

2

1

Documen

# 함수

```
1 ▼ void main() {  
2   // 함수 - 리턴값 타입 함수명 (매개변수 타입 매개변수){}  
3 ▼ void introduce (String name, [String food = 'chocolate']) {  
4   //[파라미터] - optional parameter  
5   //[파라미터 = default] - 기본값 설정  
6   print('I am $name, I like $food!');  
7 }  
8 introduce ('Tom');  
9 introduce('Tom', 'chicken');  
10  
11 //named parameter - 순서 상관x  
12 ▼ add({  
13   required int x,  
14   required int y,  
15   required int z,  
16 ▼ }) {  
17   int sum = x+y+z;  
18   print (sum);  
19 }  
20 add(x: 10, y: 20, z: 30);  
21 add(y: 20, x: 10, z: 30);  
22 }
```

▶ Run

Console

```
I am Tom, I like chocolate!  
I am Tom, I like chicken!  
60  
60
```

Documentation

[named parameter, optional parameter 추가설명과 코드 보기](#)

```

void main() {
    Operation operation = add;

    int result = operation(10, 20, 30);

    print(result);

    operation = subtract;

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

    print(result2);
}

// signature
typedef 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;

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

```

## typedef

- 함수를 변수처럼 사용하기 위함
- 같은 타입의 파라미터, 리턴 값을 가진 함수는 모두 사용 가능함


마지막라인은 calculate 함수에 파라미터 x, y, z 와 operation 을 넣으면 리턴 값으로 operation 에 파라미터가 들어가서 operation의 함수 실행

[코드 실행 해보기](#)

## 문자열 변수 사용

---

```
void main() {  
    void printName(String name) {  
        print("I'm $name.");  
    } // $변수 - 문자열 내에서 어떤 변수의 값을 그대로 사용  
    printName("Bob");  
  
    void printAge(int age) {  
        print("I'm an ${age > 18 ? 'adult' : 'adolescence'}");  
        // 표현식을 써야 할 때는 반드시 ${ }  
        printAge(7);  
    }  
}
```



## 상수(final vs const)

---

```
void main() {  
    const DateTime rightNow = DateTime.now();  
    final DateTime rightNow = DateTime.now(); |  
}
```

현재 시간을 가져오는  
DateTime.now(); 는 런타임  
시점에서 결정되기 때문에  
const에서 에러가 난다

const는 컴퓨터 언어로 번역되는 컴파일 과정에서 상수가  
된다.

>> 런타임 시점에 결정되는 값은 const에 담을 수 없다.

final은 번역 후 실제로 프로그램이 실제 실행되는 과정에서  
상수가 된다.

```

▼ enum Status {
    approved,
    pending,
    rejected,
}
//enum {상수 값}

▼ void main() {

    Status status = Status.approved;

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

    print(Status.values); //상수 전체 값
}

```

## enum

---

한정된 상수 값 집합을 나타내기 위함

- 정확히 이 값만 존재
- 몇가지 타입만  
사용하도록 강제가능
- 오타 방지
- 따라서 직관적이고 에러없는 코드  
작성에 용이함

## nullable / non-nullable

---

```
▼ void main() {  
  
    String food = "chicken";  
    food = null;  
  
    //nullable  
    String? food2 = "hamberger";  
    food2 = null;  
    print(food2);  
  
    //non-nullable  
    String! food3 = "potato";  
    food3 = null;  
    print(food3);  
  
}
```

에러 이유 : 모든 자료형은 기본적으로  
non-nullable 이기 때문이다.

=> 물음표(?) 를 붙여주면 - nullable

=> ! - non-nullable