FUNCTION

- arrow function

```
void sayHello(String name){
	print("Hello $name nice to meet you");
}

String sayHello(STring name) => "Hello $name"

num plus(num a, num b) => a + b;

void main() {

}
```

fat arrow function 은 return 을 한다는 의미

- named parameters

```
String sayHello(String name, int age, String country){
	return "Hello $name, you are $age, and you come from $country";
}

void main(){
	print(sayHello('nice', 19, 'cuba');
}
```

위는 직관적이지 않다. 게다가 사용자는 반드시 순서를 지켜야 하는 불편함이 있다.

따라서 코드를 아래 와 같이 변경하여 직관적으로 만들어 줄 수 있다.

순서와 상관없이 parameter 를 이름으로 입력할 수 있다.

```
String sayHello({
	String name = 'anon', // null safefy 에 걸리지 않기 위해
	int age = 99,
	String country = 'wakanda'
}){
```

```
	return "Hello $name, you are $age, and you come from $country";
}

void main(){
	print(sayHello(
		age : 12,
		country: 'cuba',
		name: 'nico',
		);
}
```

parameter 에 default value 를 넣어주는 이유는 사용자가 parameter 를 모두 입력한다는 보장이 없기 때문에 dart 의 null safety 에 걸려 에러가 발생하는 것을 방지하기 위해서이다.

또 다른 방법으로 null 을 방지하기 위해 required 를 사용하여 사용자에게 필수 값으로 지정해줄 수 있다.(추천)

```
String sayHello({
	required String name, // null safefy 에 걸리지 않기 위해
	required int age,
	required String country,
}){
	return "Hello $name, you are $age, and you come from $country";
}

void main(){
	print(sayHello(
		age : 12,
		country: 'cuba',
		name: 'nico',
		);
}
```

- optional positional parameters

```
String sayHello(String name, int age, [String? country = 'cuba']) => 'Hello $name, you are $age years old from $country';

void main(){
	sayHello('nico', 12);
}
```

위와 같이 작성하면 country 를 입력하지 않아도 함수를 호출 할 수 있다.

(잘 사용 안 함. named 가 훨씬 유용)

- QQ operator

```
String capitalizeName(String? name){
        if(name != null){
                return name.toUpperCase();
        }
        return 'ANON';
}

String capitalizeName(String? name) => name != null ? name.toUpperCase() :
'ANON';

String capitalizeName(String? name) => name?.toUpperCase() ?? 'ANON';

void main(){
        capitalizeName('nico');
        capitalizeName(null);
}
```

left ?? right 에서 left == null 이면 right 를 리턴하고, left ≠ null 이면 left 를 리턴하는 연산자이다.

```
void main(){
        String? name;
        name ??= 'nico'; // 만약 name 이 null 이면 nico 를 넣는다.
        name ??= 'another'; // 만약 named null 이면 another 를 넣는다.

}
```

따라서 위 코드에서는 name ??= ‘another’ 이라는 코드는 실행될 일이 없어 에러가 발생한다.

```
void main(){
        String? name;
        name ??= 'nico'; // 만약 name 이 null 이면 nico 를 넣는다.
        name = null;
        name ??= 'another'; // 만약 named null 이면 another 를 넣는다.
```

}

하지만 둘 사이에 name = null 을 넣어준다면 모든 코드가 실행돼

name 은 null → nico → null → another 이라는 값을 가지게 된다.

- typedef : 자료형이 헷갈릴 때 도움이 되는 alias 를 만드는 방법

```
typedef ListOfInts = List<int>;

ListOfInts reverseListOfNumbers(ListOfInts list){
        var reversed = list.reversed;
        return reversed.toList();
}
```

typedef 를 통해 원하는 자료형에 이름을 붙여 사용할 수 있다.