Ocaml

KOSMOS

이거 좀 헷갈려요!

```
let foo : int -> int -> bool
= fun n t ->
  match n with
  ¦ t -> true
  ¦ _ -> false;;

foo 2 2;;
foo 3 4;;
```

```
Warning 11: this match case is unused.
val foo : int -> int -> bool = <fun>
- : bool = true
- : bool = true
```

Pattern matching

match with 구문은 이름 그대로 패턴을 찾는 구문이에요!

```
type foo =
  ¦ Foo of int
  ¦ FooFoo of foo;;

let rec func x =
  match x with
  ¦ Foo n ->  n
  ¦ FooFoo f -> func f;;
```

여기서 match 구문은 x의 값이 어떤 '형식'을 갖는지 물어보고 있는 거에요!

그래서 만약 형식이 맞는다면 새로 변수를 '선언'해주는 거에요!

만약에...

```
let x = Foo 1;;
func x;;
```

match 구문만 살펴본다면,

x 가 Foo 1 이므로 첫번째 패턴이겠죠?

그럼 Foo n = Foo 1이라고 저장하고,

즉, n에 1이 저장된 상태로 -> 뒤를 계산하는 거에요.

만약에...

```
let x = FooFoo (Foo 2);;
func x;;
```

여기서도 match 구문만 살펴 본다면

x가 FooFoo (Foo 2)이므로 두번째 패턴이겠죠?

그럼 FooFoo f = FooFoo (Foo 2)라고 저장하고,

즉, f에 Foo 2가 저장된 상태로 -> 뒤를 계산하는 거에요.

그럼 첫번째 예시에서

```
match n with
| t -> true
| _ -> false
```

여기서 n은 아래 식처럼 t라는 변수로 선언이 가능하죠?

```
let t = n;;
```

이런 경우는 형식이 일치한다고 생각하기 때문에

`| t -> true` 뒤의 구문은 실행이 안된다고 경고를 해주는 거에요!

어떻게 바꿀 수 있을까요?

```
let foo : int -> int -> bool
= fun n t ->
  match n = t with
  | true -> true
  | false -> false;;
```

```
let foo : int -> int -> bool
= fun n t -> if n = t then true else false;;
```

```
let foo : int -> int -> bool
= fun n t ->
  match n = t with
  | r -> r;;
```

```
let foo : int -> int -> bool
= fun n t -> n = t;;
```

이런 거 본 적 있죠?

```
type num =
  | Num of int
  | Add of num * num;;

let x = Num 7;;
let y = Add (Num 3, Num 5);;
```

```
type num = Num of int | Add of num * num
val x : num = Num 7
val y : num = Add (Num 3, Num 5)
```

사용자가 선언해주는 타입

type 키워드는 사용자가 직접 타입을 선언할 때 사용하는 키워드에요!

타입을 선언하는 형식은 아래처럼 하면 돼요.

```
type (타입 이름) =
  | (생성자) of (값의 타입)
  | …
```

앞선 예시에서 선언한 num 타입의 경우는

타입 이름은 num이고,

첫번째 생성자는 Num인데 int의 값을 갖고,

두번째 생성자는 Add인데 (num * num), 즉, num 2개로 되어있는 튜플을 값으로 갖는다고 해석할 수 있어요

사용자가 선언해주는 타입

이렇게 선언한 타입은 아래처럼 생성자를 이용해서 쓸 수 있어요!

(생성자)(값)

앞선 예시에서 x는 Num으로 생성했고, int 값인 7을 갖고 있고,
y는 Add로 생성했고 num * num 값인 (Num 3, Num 5)를 갖고 있어요.