Obsidian for Researchers

Ji, Yong-Hyeon

24. 09. 10 (Thu)

Coding & Optimization Together (CO2)
Crypto & Security Engineering Lab (CSE)
Department of Information Security, Cryptology, and Mathematics

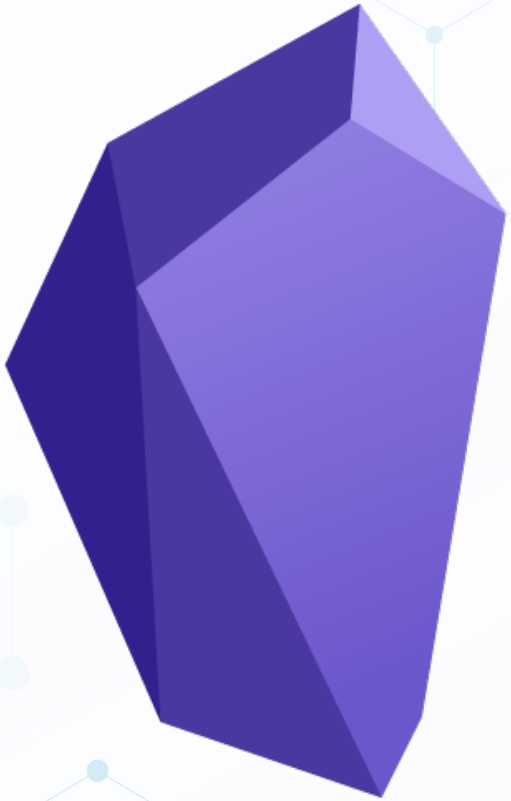

Table of Contents

section1

2.1 OCaml 기본 구성

▷ Function Expression (함수식)

$$\texttt{fun}\ x\ \texttt{->}\ e$$

- 함수의 예:
  * $\texttt{fun}\ x\ \texttt{->}\ x + 1$
  * $\texttt{fun}\ y\ \texttt{->}\ y * y$
  * $\texttt{fun}\ x\ \texttt{->}\ \texttt{if}\ x > 0\ \texttt{then}\ x + 1\ \texttt{else}\ x * x$
  * $\texttt{fun}\ x\ \texttt{->}\ \texttt{fun}\ y\ \texttt{->}\ x + y$
  * $\texttt{fun}\ x\ \texttt{->}\ \texttt{fun}\ y\ \texttt{->}\ \texttt{fun}\ z\ \texttt{->}\ x + y + z$

- Syntactic Sugar

$$\texttt{fun}\ x_1\ \ldots\ x_n\ \texttt{->}\ e$$

  * $\texttt{fun}\ x\ y\ \texttt{->}\ x + y$
  * $\texttt{fun}\ x\ y\ z\ \texttt{->}\ x + y + z$

▷ Function Call Expression (함수 호출식)

e_1 e_2

```
# (fun x -> x * x) 3;;
- : int = 9
# (fun x -> if x > 0 then x + 1 else x * x) 1;;
- : int = 2
# (fun x -> fun y -> fun z -> x + y + z) 1 2 3;;
- : int = 6
```

```
# (fun f -> f * 1) (fun x -> x * x);;
- : int = 1
# (fun x -> x * x) ((fun x -> if x > 0 then 1 else
2) 3);;
- : int = 2
```

▷ Let Expressions

값에 이름 붙이기!

$$\texttt{let } x = e_1 \texttt{ in } e_2$$

- e_1의 값을 x라고 하고 e_2를 계산
  - x: variable (변수, 값의 이름)
  - e_1: binding expression (정의식)
  - e_2: body expression (몸통식)
- e_2: scope of x (유효범위)

```
# let x = 1 in x + x;;
- : int = 2
# (let x = 1 in x) + x;;
Error: Unbound value x
# (let x = 1 in x) + (let x = 2 in x);;
- : int = 3
```

```
# let x = (let y = 1 in y + 1) in x + 1;;
- : int = 3
# let x = 1 in
        let y = 2 in
                x + y;;
- : int = 3
```

```
# let square = fun x -> x * x in square 2;;
- : int = 4
# let add x y = x + y in add 1 2;;
- : int = 3
```

```
# let rec factorial n =
        if n = 0 then 1
        else n * factorial (n - 1);;
val factorial : int -> int = <fun>
# factorial 5;;
- : int = 120
```

2.1 OCaml 기본

▷ Pattern Matching (패턴 매칭)

- 패턴 매칭을 이용한 값의 구조 분석

```
# let rec factorial n =
if n = 0 then 1 else n * factorial (n - 1);;
val factorial : int -> int = <fun>
```

```
# let factorial a =
match a with
0 -> 1
|_ -> a * factorial (a-1);;
val factorial : int -> int = <fun>
```

2.1 OCaml 기본

▷ Polymorphic Type (다형 타입)

```
# let id x = x;;
val id : 'a -> 'a = <fun>
# id 1;;
- : int = 1
# id "abc";;
- : string = "abc"
# id true;;
- : bool = true
```

To be continue ...