Ocaml

KOSMOS

Ocaml은 이래요

- 모든 프로그램은 한 문장
- 모든 것은 타입을 갖는 객체
- 매우 엄격한 타입시스템

모든 프로그램은 한 문장(Expression)

이후 슬라이드는 expression의 종류에 대한 간단한 설명입니다.

이후 슬라이드에서 exp 자리에는 모든 expression이 들어갈 수 있어요!

Ocaml expression

- **true, false**
- exp = exp (boolean operator: =, !=(<>), >, <, <=, >=, &&, ||, not)

- integer (1, 2, …)
- exp + exp (arithmetic operator: +, -, *, /, mod)

- float (1.4, 3.21, …)
- exp +. exp (arithmetic operator: +., -., *., /.)

Ocaml expression

- `let var = exp1 in exp2`
  exp1을 계산한 값을 var에 저장한 후 exp2를 계산

- `fun var -> exp`
  var의 값을 활용해서 exp를 계산하는 함수 생성

- `if exp1 then exp2 else exp3`
  exp1의 계산 값이 true면 exp2 계산
  exp2의 계산 값이 false면 exp3 계산

Ocaml expression

```
- match exp with
  ¦ pat1 -> exp1
  ¦ pat2 -> exp2
  ¦ …
```

exp의 값이 pat1의 값의 형식에 일치하면 exp1을 계산
exp의 값이 pat2의 값의 형식에 일치하면 exp2를 계산

match … with

```
let x = 10 in
match x with
| 1 -> 2
| 2 -> 2
| 4 -> 9
| _ -> 0;;      ( ‘_’는 wildcard로 위의 모든 경우가 해당하지 않는 경우)


- : int = 0
```

이런 것도 가능해요!

```
let x =
  let pow n = n * n in
  let f a b = pow (a + b) in
  f 3 4
in x;;


- : int = 49
```

이런 것도 가능해요!

```
let x = 1 in
match let y = x + 1 in y + 1 with
| 1 -> 4
| 2 -> 5
| 3 -> 6
| _ -> 0;;


- : int = 6
```

모든 것은 타입을 갖는 객체

모든 것을 값으로 처리합니다.

함수도 값으로 처리돼요!

이후 슬라이드에서 type 자리에는 다른 type이 들어갈 수 있어요!

Ocaml type system

- int (정수)
- bool (true/false)
- float (실수)
- char (문자, ‘c’)
- string (문자열, “str”)
- type -> type (함수)
- type * type (튜플)
- type list (리스트)
- Unit (출력과 같이 반환 값이 없는 함수에서 사용돼요)

함수

type1 -> type2 의 형식으로 표시돼요.

type1의 값을 받아서 type2의 값을 출력하는 함수라고 생각하면 돼요.

예를 들어, 아래의 1번 형식의 경우는 int를 입력 받아서 float를 출력하는 함수를 나타내는 거예요.

```
1: int -> float
```

아래의 2번 형식의 경우는 int를 입력 받으면 int -> int 함수를
두개의 int를(int -> int) 입력 받으면 int를 출력하는 함수예요.

```
2: int -> int -> int
```

int -> int -> int

```
let f = fun a b -> a + b;;
f 1 2;;
f 1;; (* 이 친구는 int -> int 함수에요! *)

val f : int -> int -> int = <fun>
- : int = 3
- : int -> int = <fun>
```

int -> int -> int

```
let f = fun a b -> a + b;;
let g = f;; (* 이렇게 함수를 값처럼 사용할 수 있어요! *)
let h = g 1;;
h 3;;

val f : int -> int -> int = <fun>
val g : int -> int -> int = <fun>
val h : int -> int = <fun>
- : int = 4
```

함수도 값이에요

함수의 인자와 출력 결과로 함수가 올 수 있어요!

그런 함수를 하이어오더 함수(Higher-Order function)라고 해요.

가장 많이 사용하는 하이어오더 함수 중에 대표적인 map을 가지고 설명할게요.

let map = …

map 함수는 이렇게 정의되어 있어요.
함수와 리스트를 하나 받아서 다른 리스트를 만들어 주는 함수에요.

```
let map : (‘a -> ‘b) -> ‘a list -> ‘b list
= fun f lst ->
  match lst with
  ¦ [] -> []
  ¦ hd::tl -> (f hd)::(map f tl)
```

map은 이렇게 사용해요

```
let orig = [1; 2; 3; 4; 5];;
map (fun x -> 2 * x) orig;;

val orig : int list = [1; 2; 3; 4; 5]
- : int list = [2; 4; 6; 8; 10]
```

이 코드는 리스트내에 모든 값을 2배 해주는 거예요.
값을 두배 시켜주는 함수를 인자로 넘겨주어서 처리했어요.

Ocaml이 타입을 판단해줘요

코드를 작성하면 알아서 해당 코드의 타입을 판단해줘요.
코드의 타입을 판단하는 방법은 나중에 프언 수업시간에 배울 거예요...
예를 들어, 아래의 코드에서 x는 float라는 걸 자동으로 판단해줘요.

```
let x =
  let foo = true in
  if foo then 3.14 else 2.71;;

val x : float = 3.14
```

저는 직접 지정하는게 편해요

명시적으로 타입을 지정해 줄 수 있어요!

콜론(:)을 쓰고 뒤에 타입을 정해주면 돼요.

예를 들어, 아래의 코드에서 x는 float라고 명시적으로 지정해준 거예요.

```
let x : float =
  let foo = 4 in
  if foo mod 2 then 3.14 else 2.71;;

val x : float = 3.14
```

함수도 타입을 지정해 줄 수 있어요

```
let foo : (int-> bool) -> int -> string
= fun f n ->
  match f n with
  ¦ true -> “given int satisfy condition”
  ¦ false -> “given int dissatisfy condition”

val foo : (int -> bool) -> int -> string = <fun>
```

매우 엄격한 타입 시스템

Ocaml은 매우 엄격한 타입 시스템을 가지고 있어요.
하나의 문장(Expression)은 무조건 하나의 값을 가져야 돼요.
자동 형 변환(Auto type casting)도 지원하지 않아요.

하나의 문장(if exp then exp else exp)

then과 else의 형식이 항상 같아야 해요.

아래의 코드를 실행하면 int자리에 float가 왔다고 오류라고 알려줘요.

```
if true then 3 else 3.1;;

Error: This expression has type float but an expression was
                 expected of type
         int
```

하나의 문장(match exp with …)

with 이후에 반환하는 값(-> 뒤에 exp를 계산해서 나오는 값)의 형식이 항상 같아야 돼요.

아래의 코드를 실행하면 bool자리에 int가 왔다고 오류라고 알려줘요.

```
match 1 with
| 0 -> true
| _ -> 1;;

Error: This expression has type int but an expression was expected
                 of type
         bool
```

자동 형 변환이 안돼요

C, 자바와는 다르게 자동으로 형 변환이 안돼요.

그래서 형을 변환하고 싶으면 명시적으로 지정해야 돼요.

심지어 자료형 별로 연산자가 따로 있어요!

명시적인 형 변환

OcamI에서 형을 바꾸고 싶으면 형 변환을 명시적으로 알려줘야 돼요.

예를 들어, int를 float로 바꾸고 싶거나, char를 int로 바꾸고 싶다면 아래와 같이 써야 돼요.

```
float_of_int 7;;
int_of_char ‘c’;;
```

```
- : float = 7.
- : int = 99
```

정수형 연산자

아래 5개가 정수형 기본 연산자예요.

- +
- -
- *
- /
- mod

정수형 연산자

이 정수형 연산자들은 양쪽에 모두 정수(int)여야 계산 할 수 있어요

예를 들여 아래에 int와 float의 덧셈은 불가능해요.

```
7 + 7.;;

Error: This expression has type float but an expression was
                 expected of type
         int
```

실수형 연산자

실수형 연산자는 정수형 연산자 뒤에 .을 붙이면 돼요.

아래 4개가 실수형 기본 연산자예요

- +.
- -.
- *.
- /.
- (* mod.는 없어요! *)