OCaml Tutorial

2019년 가을 프로그래밍 언어

고현수

hsgo@ropas.snu.ac.kr

서울대학교 프로그래밍 연구실

차례

- OCaml 시작하기
  - 어떤 언어인가
  - 설치와 실행, 컴파일
- 기초
  - 이름 붙이기, 값, 타입, 함수 등
- 더 나아가기
  - pair / tuple / list
  - currying
  - Inductive type / polymorphic type
  - match-with 구문 / try - with - raise 구문
  - 모듈 시스템, reference 문법 구조
- 예제코딩

출처 : 11~18년도 튜토리얼

OCaml 시작하기

OCaml 언어란?

- 여러가지 사용법을 가지는 상위 (high-level) 언어
  - 절차형(imperative), 객체지향(object oriented), 함수형(functional) 등의 프로그래밍을 모두 지원
- 편의성
  - 자동으로 메모리 관리 (garbage collector)
- 안정성(safety)
  - 타입 시스템

출처 : 11~18년도 튜토리얼

절차형 vs 값 중심형

- 기계에게 순서대로 명령을 전달하는 절차형 언어
- 기계에게 식의 계산을 시키는 값 중심의 언어 (혹은, 함수형 언어)
- 주로 값 중심의 프로그래밍을 경험하시게 됩니다.

```
a = 1;
b = 2;
c = a + b;
print(c);
```

```
let a = 1 in
let b = 2 in
let c = a + b in
print(c)
```

OCaml 설치하기

- **방법 1) martini 등의 컴퓨터공학부 실습 서버 사용**
  - Linux 환경, OCaml 4.02.3
  - 가장 권장하는 방법입니다
- **방법 2)** https://OCaml.org/releases/
  - 소스를 받아서 직접 컴파일
  - 혹은 운영체제에 맞는 binary distribution 받기
- **방법 3)** apt-get(Ubuntu), brew(Mac) 등으로 OPAM 설치 https://opam.OCaml.org
- **아래 두 개 방법으로 설치할 경우, 채점 환경과 동일한 4.02.3 버전을 권장**

출처 : 11~18년도 튜토리얼

OCaml 실행하기(Interpreter)

- .ml 파일을 작성하고 'ocaml' 명령의 인자로 주어 실행할 수 있음

(hello.ml)

```
let _ = print_endline "Hello world!"
```

(console)

```
jschoi@ropas:~$ ocaml hello.ml
Hello world!
```

출처 : 11~18년도 튜토리얼

OCaml 실행하기(compile)

- 작성한 .ml 파일을 ocamlc 로 컴파일 가능

(console)

```
jschoi@ropas:~$ ocamlc hello.ml -o hello
jschoi@ropas:~$ ./hello
Hello world!
```

출처 : 11~18년도 튜토리얼

과제 제출시

- **(중요) 반드시 ocaml 명령으로 실행되거나, ocamlc/make로 컴파일 되는 파일을 제출 바랍니다**
  - 컴파일 오류 발생시 0점 처리 되므로, 제출 전에 꼭 확인
  - 과제 제출과 채점, 클레임 등에 대해서는 게시판에 올라온 별도 공지 참조
  - 프로그램 외적인 문제 수정기회 잘 활용

출처 : 11~18년도 튜토리얼

OCaml의 기초

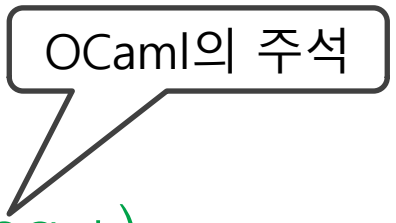
값에 이름 붙이기

- 정수 값
  - let i = 1
    - 앞으로 1이라는 값의 이름은 i 이다
    - 혹은, i 의 값은 1로 정의한다
- 문자열 값
  - let s = “hello world”
- Boolean 값
  - let b = true
- unit
  - let _ = print_endline “hello world”
- 값의 이름은 소문자로 시작해야 함

출처 : 11~18년도 튜토리얼

타입

- 모든 값은 타입을 가짐
  - let i = 1 (* i : int *)
  - let s = “hello world” (* s : string *)
  - let b = true (* b : bool *)

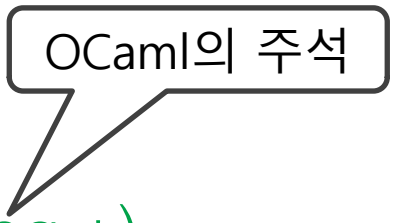


- 코드에 타입을 명시할 수도 있음
  - let i : int = 1
  - let s : string = “hello world”
  - let b : bool = true

출처 : 11~18년도 튜토리얼

타입

- 모든 값은 타입을 가짐
  - let i = 1 (* i : int *)
  - let s = “hello world” (* s : string *)
  - let b = true (* b : bool *)



- 코드에 타입을 명시할 수도 있음
  - let i : int = 1
  - let s : string = “hello world”
  - let b : bool = true

Q. 꼭 타입을 적어야 하나요? → 다음 장

출처 : 11~18년도 튜토리얼

타입 추론

- OCaml의 타입 추론 시스템
  - 자동으로 정확하게 타입을 추론해 주고
  - **프로그래밍에 타입 에러가 있다면 미리 잡아줌**
  - let x = 3 + “abc”

```
jschoi@ropas:~$ ocaml test.ml
File "test.ml", line 1, characters 12-17:
Error: This expression has type string but
an expression was expected of type
         int
```

- 따라서, 여러분이 코딩하실 때는 타입을 쓰지 않으셔도 됩니다
  - 모듈 타입, 레퍼런스 등 반드시 써야만 하는 경우도 **가끔** 있음
- 프로그램이 커지면 가독성을 위해 타입을 명시하는 것을 권장

출처 : 11~18년도 튜토리얼

함수 정의하기

- 함수 정의하고 사용하기
  - let incr x = x + 1
  - let y = incr 5
- 다른 스타일
  - let incr = fun x -> x + 1
  - let y = incr 5
- 이름 붙이지 않고 사용하는 것도 가능
  - let y = (fun x -> x + 1) 5
- 상황에 따라 편한 것을 선택

출처 : 11~18년도 튜토리얼

함수도 값

- 함수도 값으로 취급
  - let incr x = x + 1
  - let incr = fun x -> x + 1
  - “x를 받아서 x+1을 내놓는 함수”
  - 그 값(함수)에 incr 라는 이름을 붙이기
- 모든 값은 타입을 가지므로, 함수도 타입을 가짐
  - let incr : int -> int = fun x -> x + 1
  - let incr’ : float -> float = fun x -> x +. 1.0
  - 함수도 역시 OCaml이 자동으로 타입을 추론해 줍니다

출처 : 11~18년도 튜토리얼

let in

- 값을 정의하고 쓰기
  - let … = … in …
  - 들여쓰기(indent)도 신경 쓰면 가독성이 좋아집니다

```
let echo : unit -> unit = fun ( ) ->
  let i = read_int( ) in
  let str = string_of_int i in
  print_endline ("Your input : " ^ str)
  (* ^ : string concat operator *)
```

출처 : 11~18년도 튜토리얼

main 함수는?

- OCaml 코드는 main이 없음
- 일련의 정의(let)의 집합 (위에서부터 하나씩 실행)

(예) factorial 함수

```
let rec fact n = (* 'rec' for recursive *)
  if n <= 0 then 1 (* if ... then ... else ... *)
  else n * fact (n - 1)

let x = fact 10

let _ = print_endline (string_of_int x)
```

출처 : 11~18년도 튜토리얼

main 함수는?

- OCaml 코드는 main이 없음
- 일련의 정의(let)의 집합 (위에서부터 하나씩 실행)

(예) factorial 함수

```
let rec fact n = (* 'rec' for recursive *)
  if n <= 0 then 1 (* if ... then ... else ... *)
  else n * fact (n - 1)

let x = fact 10

let _ = print_endline (string_of_int x)
```

(중요) 과제 제출시에는, 테스트 케이스나 디버그용 출력 등을 지우고 제출해 주세요

출처 : 11~18년도 튜토리얼

더 나아가기

중요한 개념들

- Pair / tuple / list
- Currying
- Inductive type 과 match-with 구문
- Polymorphic type
- try - with - raise 구문
- 모듈 시스템
- Reference

출처 : 11~18년도 튜토리얼

Pair

- 두 개의 값을 한번에 묶기
- 직관적이고 편리함
- 정의는 (x, y) 형태, 타입은 a * b 형태로 표기
- 풀 때는 fst 와 snd 함수 사용
  - let p : (int * string) = (1, “a”)
  - let i = fst p (* int, 1 *)
  - let s = snd p (* string, “a” *)
  - let (i, s) = p 로도 같은 효과

출처 : 11~18년도 튜토리얼

Tuple

- Pair의 연장선 - 여러 값을 한번에 묶기
- Pair의 fst, snd 와 같은 함수는 제공되지 않음

- let t : (int * string * float) = (1, “a”, 1.5)
- let (i, s, f) = t
- let (_ , _ , f) = t

출처 : 11~18년도 튜토리얼

List (1/2)

- 정말 많이 사용하는 자료구조
- 리스트는 …
  - 빈 리스트 [ ] 이거나
  - 원소와 리스트의 결합 x :: y
    - x가 int 타입이면, y는 int list 타입
- 두 가지 방법으로 리스트 정의하기 (같은 의미)
  - let x : int list = 1 :: 2 :: 3 :: [ ]
  - let x : int list = [1; 2; 3]

출처 : 11~18년도 튜토리얼

List (2/2)

- List 다루기 기초
  - let head_elem : int = List.hd [1 ; 2 ; 3] (* 1 *)
  - let tail_list : int list = List.tl [1 ; 2 ; 3] (* [2 ; 3] *)
  - let elem : int = List.nth [1 ; 2 ; 3] 1 (* 2 *)
  - if (List.mem 1 [1 ; 2 ; 3]) then … else …
- List 의 원소는 모두 같은 타입이어야 함
  - cf. tuple은 다른 타입의 여러 값들도 묶을 수 있음

출처 : 11~18년도 튜토리얼

Currying (1/2)

- 함수가 여러 개의 인자를 받아야 할 때
  - Pair로 묶어서 한번에 받기
  - let sum (x, y) = x + y (* (int * int) -> int *)
  - 두 인자를 차례대로 받기 (**Currying**)
  - let sum x y = x + y (* int -> int -> int *)
- 두 함수의 타입은 다릅니다
- **(중요) 반드시 과제에서 요구하는 타입에 맞춰서 함수를 작성해 주세요**
  - **타입이 다른 함수를 작성하여 컴파일 오류 발생시 0점입니다**

출처 : 11~18년도 튜토리얼

Currying (2/2)

- Currying 타입에 대한 이해 (왜 pair인자와 다른가?)
  - let sum x y = x + y (* sum : int -> int -> int *)
  - 쉽게 생각하면 : int를 두 번 받아서 int를 내 놓는 함수
  - 정확한 실체 : int를 받아서, “int -> int 타입 함수” 를 내 놓는 함수
    - 다음과 같은 것도 가능
    - let incr = sum 1 (* incr : int -> int *)

출처 : 11~18년도 튜토리얼

Inductive Type

- 사용자가 직접 타입을 정의할 수 있다
- 이 타입 정의는 inductive한 형태도 가능
- 예) 정수 이진 트리 타입

```
type tree = Leaf of int
          | Node of int * tree * tree
let t1 : tree = Leaf 1
let t2 : tree = Node (1, Leaf 2, Leaf 3)
```

- 사용자 정의타입의 constructor (Leaf, Node 등) 는 대문자로 시작해야 함

출처 : 11~18년도 튜토리얼

match-with 구문

- 패턴 매칭 : 케이스를 나누어 계산하는 편리한 문법 요소
- 예) 앞에서 정의한 정수 이진 트리의 원소 합을 구하기
  - 타입 눈여겨보기

```
type tree = Leaf of int
          | Node of int * tree * tree

let rec sum_of_tree : tree -> int = fun tree ->
  match tree with
   |Leaf i -> i
   |Node (i, ltree, rtree) ->
     i + (sum_of_tree ltree) + (sum_of_tree rtree)
```

출처 : 11~18년도 튜토리얼

match-with 주의사항

- 중첩시켜 사용할 경우 괄호를 잘 써줘야 함

```
let foo x =
  match x with
   |A -> ...
   |B -> ...
   |C y ->
     (match y with
      |K -> ...
      |L -> ... )
   |D -> ...
```

- Match-with뿐만 아니라 대부분의 syntax 에러는 괄호를 제대로 써주지 않아서 발생

출처 : 11~18년도 튜토리얼

Polymorphic Type (1/2)

- 문제 : 정수 리스트의 길이를 구하는 함수와, 문자열 리스트 길이를 구하는 함수를 따로 짜야 할까?
  - let len = List.length [1 ; 2 ; 3] (* 3 *)
  - let len = List.length ["a"; "b" ; "c" ; "d"] (* 4 *)
- List 가 다형(polymorphic) 타입이라서 가능
  - type 'a list = [ ] | 'a :: 'a list

    (* Pseudocode, does not compile *)
  - 'a 가 int면 정수 리스트, 'a가 string면 문자열 리스트 …

출처 : 11~18년도 튜토리얼

Polymorphic Type (2/2)

- 다음과 같은 함수를 한 번만 정의하고 쓰면 됨
- 비슷한 일을 한꺼번에 해 주므로 효율적

```
let rec list_len : 'a list -> int = fun l ->
  match l with
   |head :: tail -> 1 + list_len tail
   |[] -> 0
```

- 실제 OCaml 의 List 라이브러리도 비슷하게 (= 다형 타입 덕분에 효율적으로) 구현되어 있습니다
  - List.mem, List.nth 등

출처 : 11~18년도 튜토리얼

try-with(-raise) 구문

- 예외 처리
  - Java / C++ 등의 예외 처리와 비슷
  - try 로 예외가 생길만한 부분을 묶고, with로 예외를 처리하며, raise는 예외를 발생시킴

```
let do_div : int -> int -> unit = fun x y ->
  try print_endline (string_of_int (x/y)) with
  Division_by_zero -> print_endline "Div by 0"

let f x =
  if x < 0 then
    raise (Failure "invalid input")
  else ...
```

출처 : 11~18년도 튜토리얼

모듈

- 관련 있는 코드들의 묶음
  - 예) 리스트를 다루는 함수들을 List 모듈에 모으기
  - List.hd, List.length 등으로 사용 가능
- Signature (모듈에서 드러내고 싶은 것만 드러내기)
- Functor (모듈을 받아서 모듈을 내놓는 것)
- 대부분의 경우, 조교팀에서 모듈을 설계해서 뼈대 코드를 제공합니다
  - 여러분은 모듈에 정의된 함수 내용만 채워넣으면 되도록

출처 : 11~18년도 튜토리얼

Reference

- OCaml은 값 중심 언어이지만, 명령형(imperative) 스타일로 쓰는 요소도 지원
- 변수를 선언하고 값을 저장하는 스타일
  - 0을 가리키는 참조자(reference) 정의
  - let int_ref = ref 0
  - 값을 가져오려면 ! 연산자를 사용
  - let value_of_ref : int = !int_ref
  - 새 값을 저장할 때는 := 연산자를 이용한다
  - let _ = int_ref := !int_ref + 1 (* cf. "i++" in C *)

출처 : 11~18년도 튜토리얼

간단한 예제

sum.ml

```
let rec sum_of_list (l : int list) : int =
  match l with
  |[]->0
  |hd::tl-> hd + sum_of_list tl


(* test 1 : interpreter *)


(* test 2 : let _ = *)
let _ = (print_int (sum_of_list [1;2;3;4;5]));print_newline ()


(* test 3 : test function *)
let test (f : 'a -> 'b) (input : 'a) (output : 'b) : unit =
  if ((f input) = output)
    then ((print_string ("correct answer")); (print_newline ()))
    else ((print_string ("wrong answer")); (print_newline ()))


let _ =
  let test_sum = test sum_of_list in
  (test_sum [1;2;3;4;5] 15);
  (test_sum [1;2;3;4;5;6;7;8;9;10] 55)
```

출처 : 17년도 튜토리얼

sum.ml 실행결과

- sum.ml 작성
- Interpreter로 실행
- 함수 타입확인
- 컴파일, 실행

```
donk0501@martini:~$ ls
sum.ml  useless
donk0501@martini:~$ ocaml -init sum.ml
        OCaml version 4.02.3
15
correct answer
correct answer
# test;;
- : ('a -> 'b) -> 'a -> 'b -> unit = <fun>
# sum_of_list;;
- : int list -> int = <fun>
# test sum_of_list;;
- : int list -> int -> unit = <fun>
# exit 1;;
donk0501@martini:~$ ocamlc sum.ml -o sum
donk0501@martini:~$ ./sum
15
correct answer
correct answer
donk0501@martini:~$
```

마지막으로…

참고자료 (1/2)

- 모듈 관련 참고자료
  - http://caml.inria.fr/pub/docs/manual-ocaml/moduleexamples.html
- OCaml 라이브러리
  - 앞에서 언급한 list 등 유용한 자료구조와 함수들 제공
  - http://caml.inria.fr/pub/docs/manual-ocaml/stdlib.html
  - http://caml.inria.fr/pub/docs/manual-ocaml/libref/List.html
  - google “OCaml list”, “OCaml set”, etc.
  - 리스트 관련해서 필요한 것이 생기면 여기서 먼저 검색

출처 : 11~18년도 튜토리얼

참고자료 (2/2)

- Introduction to OCaml
  - http://caml.inria.fr/pub/docs/manual-ocaml-4.02/
- 그 외 조교팀 홈페이지에 올라와있는 여러 자료들
  - http://ropas.snu.ac.kr/~ta/4190.310/19
  - **본 튜토리얼 자료도 여기 업로드됩니다**
  - 몇몇 예제 코드들
  - 이전 OCaml 튜토리얼
- Google, stackoverflow, 작년도 게시판

출처 : 11~18년도 튜토리얼

주의사항 정리

- OCaml을 직접 설치할 경우, 채점 환경과 동일한 4.02.3버전을 권장
- 반드시 ocaml 명령으로 실행되거나, ocamlc/make로 컴파일 되는 파일을 제출 바랍니다
- 과제 제출시에는, 테스트 케이스나 디버그용 출력 등을 지우고 제출해 주세요
- 과제에서 요구하는 타입에 맞춰서 함수를 작성해 주세요
  - let sum x y = ... vs let sum (x, y) = ...

출처 : 11~18년도 튜토리얼

출처

- 본 자료는 11년도 OCaml 튜토리얼 자료를 바탕으로 역대 TA들이 조금씩 살을 붙여나가며 완성한 자료입니다.(http://ropas.snu.ac.kr/~ta/4190.310/19)
  - 11년도 : 이원찬, 윤용호, 김진영
  - 13년도 : 최준원, 강동옥
  - 15년도 : 최재승
  - 17년도 : 이동권
  - 18년도 : 이동권, 배요한
  - 19년도 : 고현수

한 학기 동안 유익한 강의가 되기 바랍니다

감사합니다