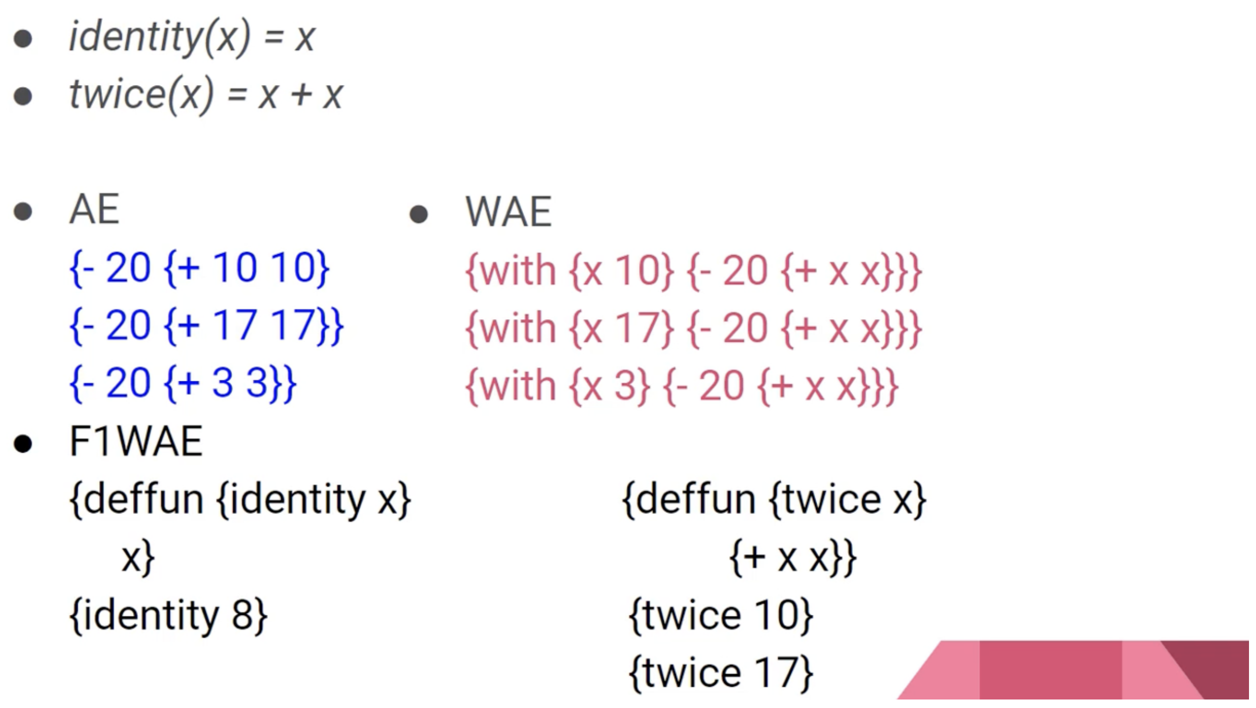
L8. Introduction to Function

유형	강의
복습 여부	✓
Status	Done

Think about simple functions



- *identity(x) = x*
- *twice(x) = x + x*

- AE
  ```
  {- 20 {+ 10 10}
  {- 20 {+ 17 17}}
  {- 20 {+ 3 3}}
  ```
- WAE
  ```
  {with {x 10} {- 20 {+ x x}}}
  {with {x 17} {- 20 {+ x x}}}
  {with {x 3} {- 20 {+ x x}}}
  ```
- F1WAE
  ```
  {deffun {identity x}
     x}
  {identity 8}
  ```
  ```
  {deffun {twice x}
            {+ x x}}
  {twice 10}
  {twice 17}
  ```

Improve ‘WAE’ to support functions → ‘F1WAE’

F1WAE: Concrete Syntax

```
<FunDef> ::= {deffun {<id> <id>} <F1WAE>}              ← for function definition
<F1WAE> ::= <num>
                   | {+ <F1WAE> <F1WAE>}
                   | {- <F1WAE> <F1WAE>}
                   | {with {<id> <F1WAE>} <F1WAE>}
                   | <id>
                   | {<id> <F1WAE>}                     ← for function call
```

{deffun {identity x} x}

{identity 8}

{deffun {twice x} {+ x x}}

{- 20 {twice 10}}

F1WAE: Abstract Syntax

```
(define-type FunDef
        [fundef   (fun-name symbol?)
                         (arg-name symbol?)
                         (body F1WAE?)])

(define-type F1WAE
        [num      (n number?)]
        [add      (lhs F1WAE?) (rhs F1WAE?)]
        [sub      (lhs F1WAE?) (rhs F1WAE?)]
        [with     (name symbol?) (named-expr F1WAE?) (body F1WAE?)]
        [id                      (name symbol?)]
        [app      (ftn symbol?)       (arg F1WAE?)])
```

```
; F1WAE
(define-type FunDef
  [fundef (fun-name symbol?) (arg-name symbol?) (body F1WAE?)])

(define-type F1WAE
  [num (n number?)]
  [add (lhs F1WAE?) (rhs F1WAE?)]
  [sub (lhs F1WAE?) (rhs F1WAE?)]
  [with (name symbol?) (named-expr F1WAE?) (body F1WAE?)]
  [id (name symbol?)]
  [app (ftn symbol?) (arg F1WAE?)])
```

```
(fundef 'identity 'x (id 'x))
(test (app 'identity (num 8)) (app 'identity (num 8)))

(fundef 'twice 'x (add (id 'x) (id 'x)))
(test (app 'twice (num 10)) (app 'twice (num 10)))
(test (app 'twice (num 17)) (app 'twice (num 17)))
```

```
(fundef 'identity 'x (id 'x))
good (app 'identity (num 8)) at line 16
  expected: (app 'identity (num 8))
  given: (app 'identity (num 8))

(fundef 'twice 'x (add (id 'x) (id 'x)))
good (app 'twice (num 10)) at line 19
  expected: (app 'twice (num 10))
  given: (app 'twice (num 10))

good (app 'twice (num 17)) at line 20
  expected: (app 'twice (num 17))
  given: (app 'twice (num 17))
```

F1WAE: Parser

```
; parse-fd : sexp -> FunDef
(define (parse-fd sexp)
  (match sexp
    [(list 'deffun (list f x) b) (fundef f x (parse b))]))

; parse : sexp -> F1WAE
(define (parse sexp)
  (match sexp
    [(? number?) (num sexp)]
    [(list '+ l r) (add (parse l) (parse r))]
    [(list '- l r) (sub (parse l) (parse r))]
    [(list 'with (list i v) e) (with i (parse v) (parse e))]
    [(? symbol?) (id sexp)]
    [(list f a) (app f (parse a))]
    [else (error 'parse "bad syntax: ~ a" sexp)]))
```

```
(test (parse-fd '{deffun {twice x} {+ x x}}) (fundef 'twice 'x (add (id 'x) (id 'x))))
(test (parse '{- 20 {twice 10}}) (sub (num 20) (app 'twice (num 10))))
```

```
good (parse-fd '(deffun (twice x) (+ x x))) at line 39
  expected: (fundef 'twice 'x (add (id 'x) (id 'x)))
  given: (fundef 'twice 'x (add (id 'x) (id 'x)))

good (parse '(- 20 (twice 10))) at line 40
  expected: (sub (num 20) (app 'twice (num 10)))
  given: (sub (num 20) (app 'twice (num 10)))
```

F1WAE: Interpreter

```
; interp: F1WAE list-of-FuncDef -> number
(define (interp f1wae fundefs)
  (type-case F1WAE f1wae
    [num (n) n]
    [add (l r) (+ (interp l fundefs) (interp r fundefs))]
    [sub (l r) (- (interp l fundefs) (interp r fundefs))]
    [with (x i b) (interp (subst b x (interp i fundefs)) fundefs)]
    [id (s) (error 'interp "free identifier")]
    [app (f a) (local
                 [(define a_fundef (lookup-fundef f fundefs))]
                 (interp (subst (fundef-body a_fundef)
                                (fundef-arg-name a_fundef)
                                (interp a fundefs))
                         fundefs
                         )
                 )
         ]
    )
  )

; lookup-fundef : symbol list-of-FunDef -> FunDef
(define (lookup-fundef name fundefs)
  (cond
    [(empty? fundefs) (error 'lookup-fundef "unknown function")]
    [else (if (symbol=? name (fundef-fun-name (first fundefs)))
              (first fundefs)
              (lookup-fundef name (rest fundefs)))]))

; subst
(define (subst f1wae idtf val)
  (type-case F1WAE f1wae
    [num (n) f1wae]
    [add (l r) (add (subst l idtf val) (subst r idtf val))]
    [sub (l r) (sub (subst l idtf val) (subst r idtf val))]
    [with (i v e) (with i (subst v idtf val) (if (symbol=? i idtf) e (subst e idtf val)))]
    [id (s) (if (symbol=? s idtf) (num val) f1wae)]
    [app (f a) (app f (subst a idtf val))]))
```

```
(test (interp (add (num 1) (num 1)) (list (fundef 'f 'x (add (id 'x) (num 3))))) 2)
(test (interp (app 'f (num 1)) (list (fundef 'f 'x (add (id 'x) (num 3))))) 4)
(test (interp (app 'f (num 10)) (list (fundef 'f 'x (sub (num 20) (app 'twice (id 'x)))) (fundef 'twice 'y (add (id 'y) (id 'y))))) 0)
```

```
good (interp (add (num 1) (num 1)) (list (fundef 'f 'x (add (id 'x) (num 3))))) at line 81
  expected: 2
  given: 2

good (interp (app 'f (num 1)) (list (fundef 'f 'x (add (id 'x) (num 3))))) at line 82
  expected: 4
  given: 4

good (interp (app 'f (num 10)) (list (fundef 'f 'x (sub (num 20) (app 'twice (id 'x)))) (fundef 'twice 'y (add (id 'y) (id 'y))))) at line 83
  expected: 0
  given: 0
```