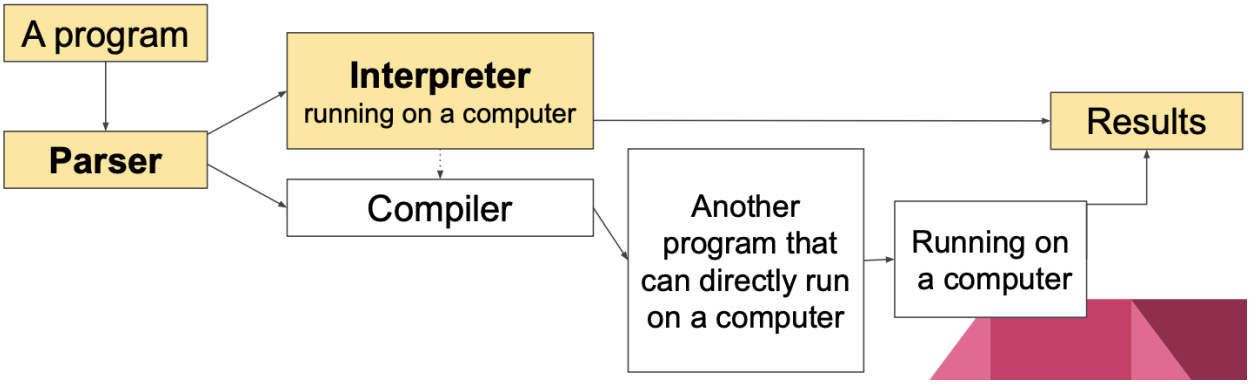
L2. Good Programming / Racket Basics

유형	강의
복습 여부	☑
Status	Done

Implement interpreters to learn programming concepts.

How can we make our programming language support the major elements ?



We are going to implement **parser** and **interpreters** for very simple language.

What do we learn?

- Racket tutorials (L2,3)
- Modeling languages (L4,5)
- Interpreting arithmetic (L5)
- Language principles
  - Substitution (L6-7)
  - Function (L8)
  - Deferring Substitution (L9)
  - First-class Functions (L10-L12)
  - Laziness (L13,14)
  - Recursion (L15,16)
  - Mutable data structures (L17,18,19,20)
  - Variables (L21,22)
  - Continuations (L23-26)
- Guest Video Lecture (L27)

What is a computer program?

a set of instructions

What is good programming?

the creation of software that relies on systematic thought, planning, and understanding

Phases until a program runs

1. Design a program
2. Write the program with a programming language
3. Interpret or compile the program
4. Run it and see the results

1. Design a program

- Problem Analysis and Data Definitions
- Contract (Signature), Purpose (Effect) statement, Header
- Functional Examples

- Function Definition
- Testing

2. Write the program with a programming language

- Peculiar syntax
- Some behaviors associated with each syntax = semantics
- Numerous useful libraries
- A collection of idioms that programmers of that language use

  idioms ≠ library

  idioms = programming patterns

3. Interpret or compile the program

- **Interpreter** : takes a program and produces a result
  - bash
  - Racket
  - Search engine
- **Compiler** : takes a program and produces a binary program
  - gcc
  - javac
  - Racket

DrRacket

our programming language to study PL

Racket

https://racket-lang.org/

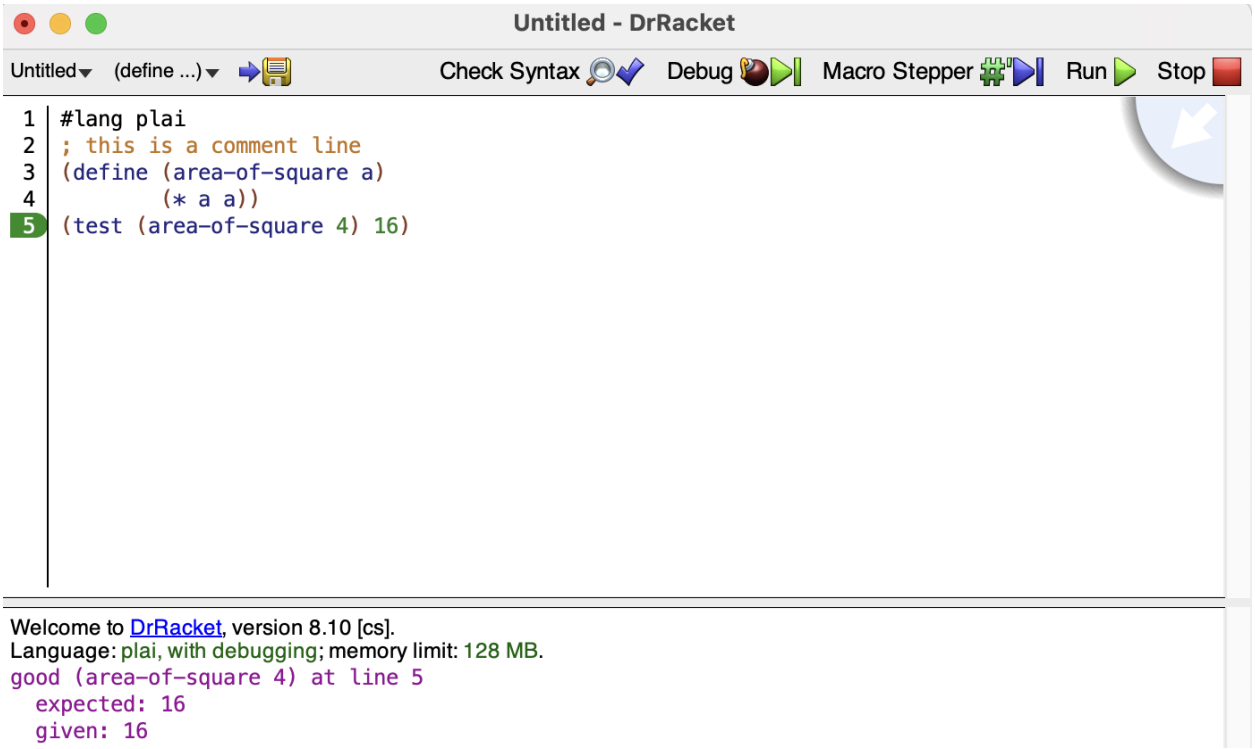
- Language level: plai
  - Each file should be prefixed with: #lang plai

> Programming Languages: Application and Interpretation
>
> This is the documentation for the software accompanying the textbook Programming Languages: Application and
> Interpretation (PLAI). The full book can be found on the Web at:
>
> https://docs.racket-lang.org/plai/index.html

After downloading DrRacket,

type the following in the Definitions window and Run

```
#lang plai
; this is a comment line
(define (area-of-square a)
  (* a a))
(test (area-of-square 4) 16)
```



Racket basic syntax

```
;expression/block/statement
(operator operand1 operand2 ...)

; Summate 1 and 2
(+ 1 2)

;; Define a function, 'jc' with a parameter 'b' and its body summates 1 and 'b'
(define (jc b) (+ 1 b))

;; jc 3!
(jc 3)
```

What kinds of PL elements exist for Computers?

- Numbers and Arithmetic
- Variables and Functions
- Conditional Expressions
- Conditional Functions
- Symbols
- Type Definitions
- Type Deconstruction
- Lists

Racket 기준으로 위 elements 설명 예정!

Numbers and Arithmetic

(operator operand$_1$ operand$_2$ ...) ; expression/block/statement

- numbers can be integers, rationals, reals, or complex

Ex. 42, 22/7, 3.141, 2+3i

- Arithmetic operations with operators and various arguments

```
(+ 1 2 3)
(/ 22 7)
(modulo 23 3)
(max 1 4 3 5 6)
(min 2 5 3 4 5)
(abs -6)
(sqrt 4)
```

6

$3\frac{1}{7}$

2

6

2

6

2

Variables and Functions

```
(define (function-name param1 ...)  ; define a function
        body)
(function-name args1 ... )          ; call the function
```

Ex1. A square of side-length a has the area a^2

```
(define (area-of-square a)
  (* a a))

(area-of-square 5)
(area-of-square 3)
```

```
25
9
```

Ex2. A disk of radius r has the approximate area 3.14 * r^2

```
(define (area-of-disk r)
  (* 3.14 (* r r)))

(area-of-disk 5)
(area-of-disk 3)
```

```
78.5
28.26
```

Ex3. Design the function for the area of a ring

```
; [contract] area-of-ring: number number -> number
; [purpose] to compute the area of a ring whose radius
; outer and whose hole has a radius of inner
; [tests] (area-of-ring 5 3) should produce 50.24

(define (area-of-ring outer inner)
  (- (area-of-disk outer)
     (area-of-disk inner)))

(test (area-of-ring 5 3) 50.24)
```

Conditional Expressions

- Booleans and relations

```
(and (> 4 3) (<= 10 100))
(or (> 4 3) (= 10 100))
(not (= 2 3 ))
```

- Functions that test conditions

```
;; [contract] is-5?: number -> boolean
;; [purpose] to determine whether n is equal to 5
(define (is-5? n)
    (= n 5))

(test (is-5? 5) true)
(test (is-5? 7) false)
```

Ex. Check whether a given number is between 5 and 6, or over 10

```
; [contract] is-between-5-6-or-over-10?: number -> boolean
; [purpose] to determine whether n is between 5 and 6
; (exclusive) or larger than or equal to 10

(define (is-between-5-6-or-over-10? n)
  (or (and (< 5 n) (< n 6))
      (>= n 10)))

(test (is-between-5-6-or-over-10? 8) false)
(test (is-between-5-6-or-over-10? 12) true)
```

```
good (is-between-5-6-or-over-10? 8) at line 54
  expected: #f
  given: #f

good (is-between-5-6-or-over-10? 12) at line 55
  expected: #t
  given: #t
```

Conditional Functions

```
(define (function-name param1 ...)
    (cond
            [ce1 body1]
            ...
            [else body]))
```

Ex. Suppose the bank pays 4% for deposits of up to $1000 (inclusive), 4.5% for deposits of up to $5,000 (inclusive), and 5% for deposits of more than $5,000. Write the function interest-rate which calculates the interest rate for a given amount.

```
; [contract] interest-rate: number -> number
; [purpose] to calculate the interest rate for a given amount
; [tests] (interest-rate 1000) should produce 0.040

; solution 1
(define (interest-rate amount)
  (cond
    [(<= amount 1000) 0.040]
    [(<= amount 5000) 0.045]
    [(> amount 5000) 0.050]))

(test (interest-rate 1000) 0.040)
(test (interest-rate 2000) 0.045)
```

```
good (interest-rate 1000) at line 69
  expected: 0.04
  given: 0.04

good (interest-rate 2000) at line 70
  expected: 0.045
  given: 0.045
```

```
; solution 2
(define (interest-rate amount)
  (cond
    [(<= amount 1000) 0.040]
    [(<= amount 5000) 0.045]
    [else 0.050]))

(test (interest-rate 1000) 0.040)
(test (interest-rate 2000) 0.045)
```

```
good (interest-rate 1000) at line 81
  expected: 0.04
  given: 0.04

good (interest-rate 2000) at line 82
  expected: 0.045
  given: 0.045
```

Symbols

A symbol is an identifier preceded by a single forward quotation mark:

Ex. 'the, 'dog, 'two^3, 'and%so%on?

```
(define (reply s)
  (cond
    [(symbol=? s 'GoodMorning) 'Hi]
    [(symbol=? s 'HowAreYou?) 'Fine]
    [(symbol=? s 'GoodAfternoon) 'INeedANap]
    [(symbol=? s 'GoodEvening) 'BoyAmITired]))

(test (reply 'GoodMorning) 'Hi)
```

The Design Recipe for function

- Contract (Signature)
  ; area-of-ring: number number -> number
- Purpose
  ; to compute the area of a ring whose radius is
  ; outer and whose hole has a radius of inner
- Tests
  (test (area-of-ring 5 3) 50.24)
- Header
  (define (area-of-ring outer inner)
- Body
  (- (area-of-disk outer)
    (area-of-disk inner)))

- Write test cases before writing programs
- If your code doesn't do everything you want it to, write more tests and repeat

Testing in Racket with the PLAI setting

(test result_expression expected_expression)
(test (area-of-square 4) 16)

produces
good (area-of-square 4) "at line 3"
    expected: 16
    given: 16

```
(test/exn result_expr error_message)
(test/exn (error "/: division by zero") "by zero")
(test/pred result_expr pred?)
```

1 PLAI Scheme

https://docs.racket-lang.org/plai/plai-scheme.html