ITP20005
Good Programming / Racket Basics (1)

Lecture02
JC

Quiz1 answers from students

"I want to know how programming code changes into machine language"

⇒ This is what compilers do!!

"...want to learn a racket language…"

⇒ Racket is just a tool to study PL theories.

"Understanding the principles of PL is important to not only learn other languages but also understand how the program works…"

"...want to know how programming language is being made…"

Quiz1 answers from students

"...PL will not disappear unless the form or the medium might be changed..."
"...new PL can appear..."
"...PL will be same as our verbal language..."
"...in the future, computers can fully understand human languages...we don't need any PL in that day..."
"...programmers will disappear as AI can replace them. but PLs cannot be replaced..." ⇒ https://www.wired.co.uk/article/google-ai-language-create

"...I want to learn new language..."

⇒ We learn *language principles* which will *help you learn new language quickly*!

"...learn structures of many other languages..."

⇒ We learn a common notation (BNF) to understand language structures.

"...PL history"

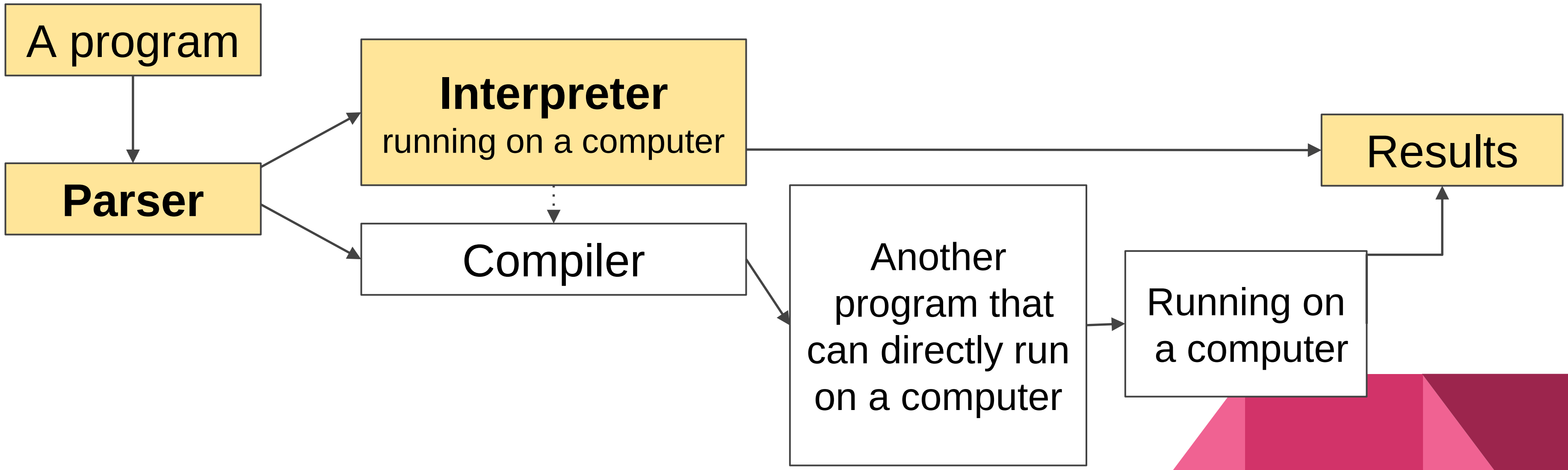
⇒ https://upload.wikimedia.org/wikipedia/commons/2/25/Genealogical_tree_of_pro ning_language

Teaching approach for this PL class

- By implementing interpreters
  - Write *programs* to learn *concepts*.
  - We are going to use Racket (formerly *Scheme*) and variants
  - Textbook (Free Ebook, Version 2007-04-26)
    - Programming Languages: Application and Interpretation
      - Shriram Krishnamurthi (Brown University)
      - http://www.plai.org First Edition
- Another approach we partially adopt
  - Survey approach
    - Good to learn the consequences of languages.
- Others
  - Definitional interpreter approach
  - Rigorous approach

Big Picture (modeling languages)

- Just write an interpreter to explain a language.
- By writing an interpreter, we can understand the language!
- Interpreter can be converted into a compiler!!!



What do we learn?

Topics we cover and schedule (tentative)

- Racket tutorials (L2,3)
- Modeling languages (L4,5)
- Interpreting arithmetic (L5)
- Language principles
  - Substitution (L6)
  - Function (L7)
  - Deferring Substitution (L8, L9, HW)
  - First-class Functions (L10-L12)
  - Laziness (L13,14)
  - Recursion (L15,16)
  - Mutable data structures (L17,18)
  - Variables (L19,L20)
  - Continuations (L21,22,23)
  - Garbage collection (L24)
  - Semantics (L25)
  - Type (L25,26)
- Guest Video Lecture (L28)

No class: October 2 (Fri, Chuseok), October 9 (Fri, Hangul day)
Online only class can be provided.

What is a computer program?

What is good programming?

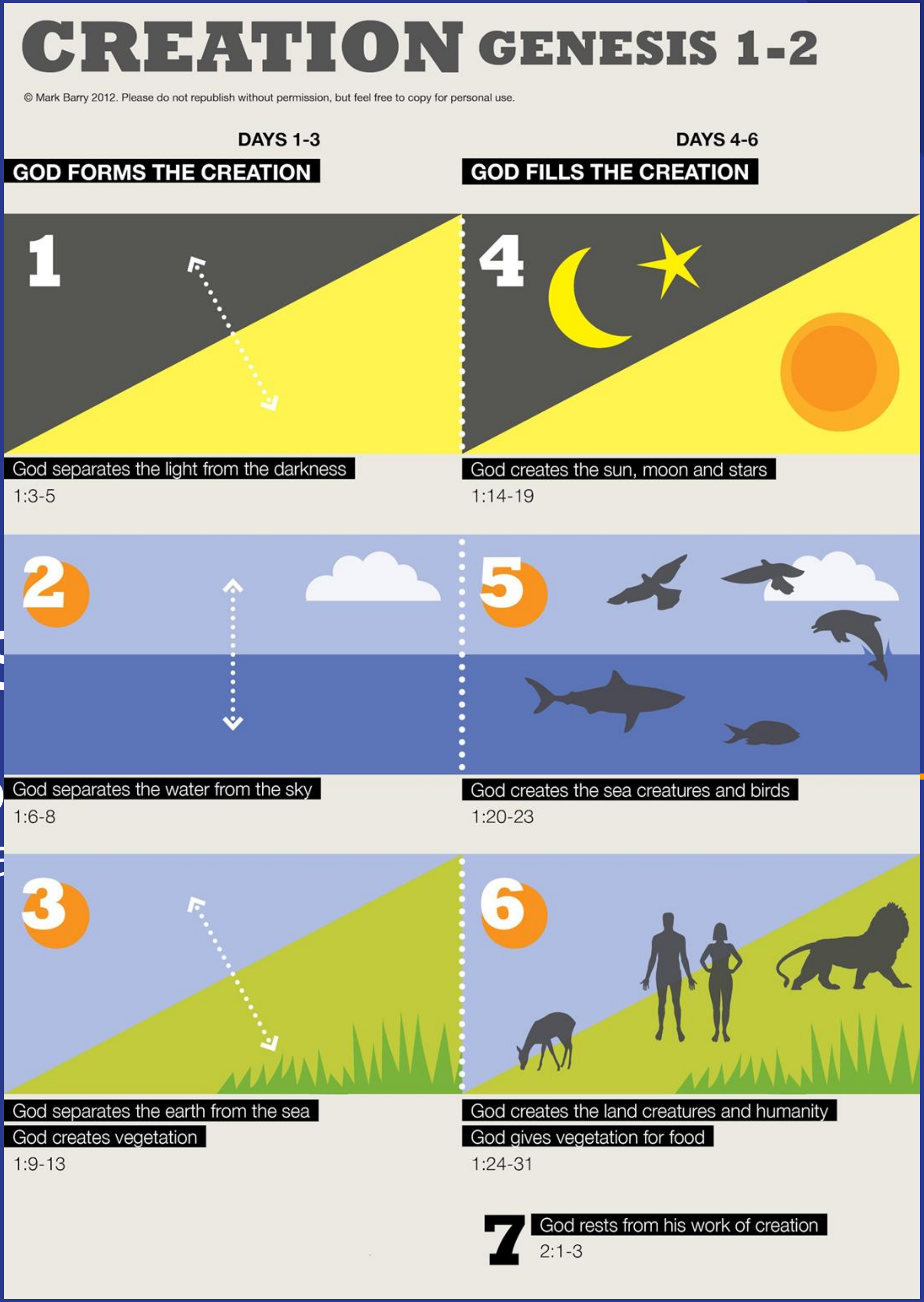
Tinker until it works????

What is good programming?

The creation of software that relies on **SYSTEMATIC** thought, planning, and understanding...

What is ... g?

The creatio ... TIC thought, pla ...


CREATION GENESIS 1-2

DAYS 1-3
GOD FORMS THE CREATION

DAYS 4-6
GOD FILLS THE CREATION

1 God separates the light from the darkness 1:3-5

4 God creates the sun, moon and stars 1:14-19

2 God separates the water from the sky 1:6-8

5 God creates the sea creatures and birds 1:20-23

3 God separates the earth from the sea
God creates vegetation
1:9-13

6 God creates the land creatures and humanity
God gives vegetation for food
1:24-31

7 God rests from his work of creation 2:1-3

Phases until a program runs

Until a program works...

[Design a program]

→ [Write the program by a **programming language**]

→ [Interpret or compile the program]

→ [Run it and see the results]

Elements for Systematic Program Design

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

Until a program works...

[Design a program]

→ ***[Write the program by a programming language]***

→ [Interpret or compile the program]

→ [Run it and see the results]

Programing Languages (by Krishnamurthi)

- Peculiar syntax (c.f. grammar = syntax + morphology)
- Some behaviors associated with each syntax
- Numerous useful libraries
- A collection of idioms that programmers of that language use

Grammar vs. Syntax in the view of natural language: https://orbi.kr/0003544888

Until a program works...

[Design a program]

→ [Write the program by a **programming language**]

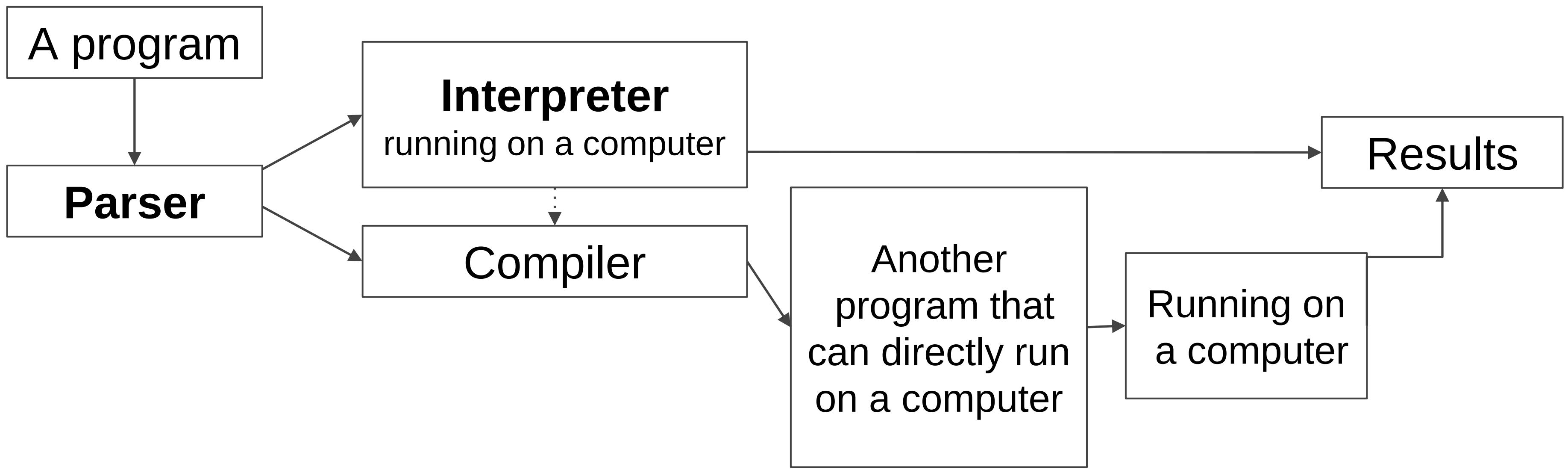
→ ***[Interpret or compile the program]***

→ [Run it and see the results]

Interpreters and Compilers

- An interpreter takes a program and produces a result
  - bash
  - Racket
  - Search engine (Google, Naver,...)???
- A compiler takes a program and produces a (binary) program
  - gcc
  - javac
  - Racket

Big Picture


A program
Parser
Interpreter
running on a computer
Compiler
Another program that can directly run on a computer
Running on a computer
Results

Understand preliminary language elements for computers
(in Racket)

What kinds of PL elements exist for Computers?
(e.g., variable,...)

Basic programming language elements

(used in this class and Racket)

- Numbers and Arithmetic
- Variables and Functions
- Conditional Expressions
- Conditional Functions
- Symbols

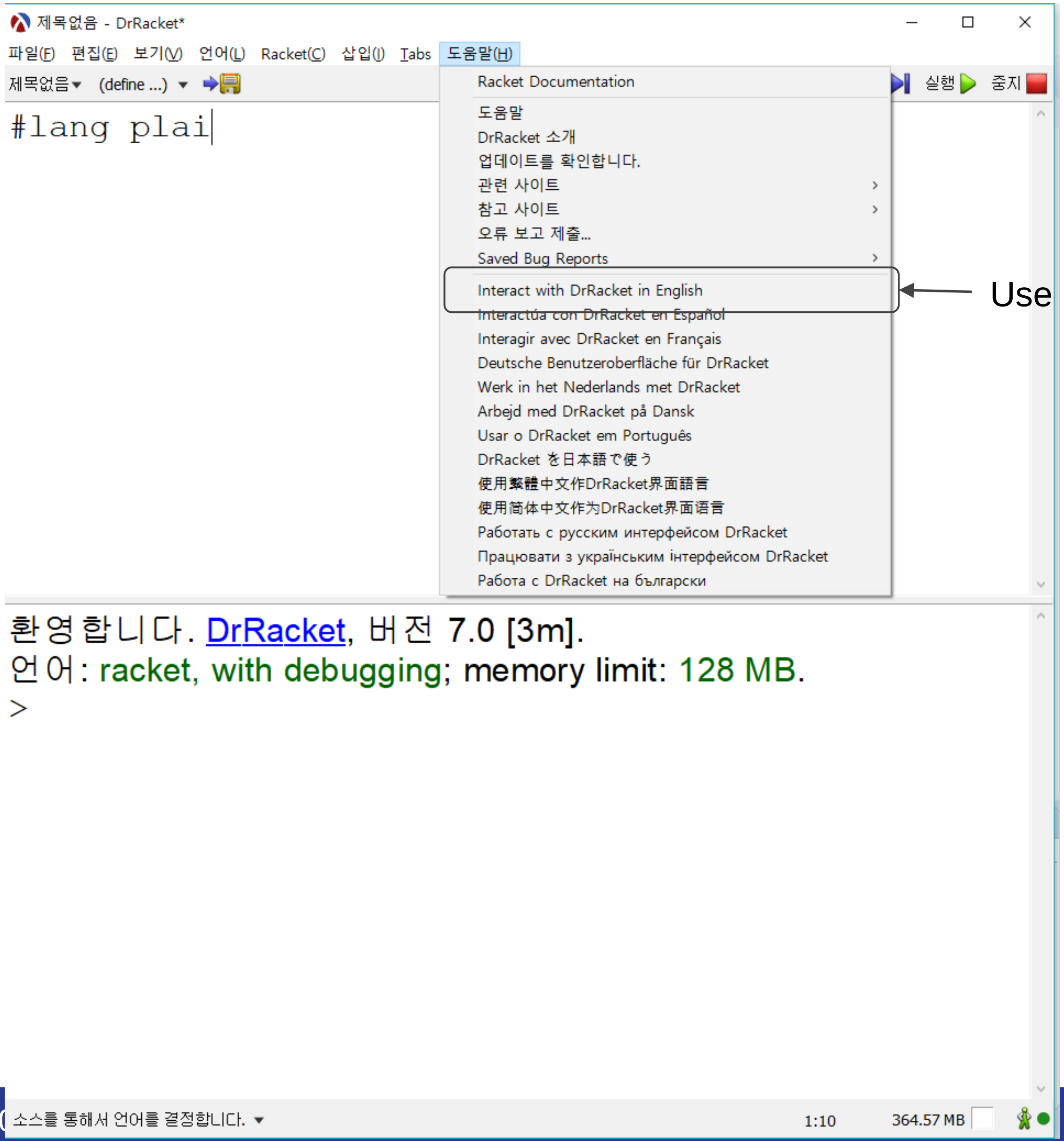
DrRacket

- Our programming language to study PL
- Install Dr. Racket
  - http://racket-lang.org/
- Language level: plai (https://docs.racket-lang.org/plai/index.html)
- Each file should be prefixed with: #lang plai
- Example
  - Type the following in the Definitions window:

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

  - Click Run

DrRacket (2)


제목없음 - DrRacket*
파일(F) 편집(E) 보기(V) 언어(L) Racket(C) 삽입(I) Tabs 도움말(H)
제목없음 (define ...)
실행 중지
Racket Documentation
도움말
DrRacket 소개
업데이트를 확인합니다.
관련 사이트
참고 사이트
오류 보고 제출...
Saved Bug Reports
Interact with DrRacket in English
Interactúa con DrRacket en Español
Interagir avec DrRacket en Français
Deutsche Benutzeroberfläche für DrRacket
Werk in het Nederlands met DrRacket
Arbejd med DrRacket på Dansk
Usar o DrRacket em Português
DrRacket を日本語で使う
使用繁體中文作DrRacket界面語言
使用简体中文作为DrRacket界面语言
Работать с русским интерфейсом DrRacket
Працювати з українським інтерфейсом DrRacket
Работа с DrRacket на български
소스를 통해서 언어를 결정합니다.
1:10
364.57 MB

```
#lang plai
```

```
환영합니다. DrRacket, 버전 7.0 [3m].
언어: racket, with debugging; memory limit: 128 MB.
>
```

Use DrRacket in English

Racket basic syntax

(operator operand_1 operand_2 ...) ; expression/block/statement

```
(+ 1 2) ; Summate 1 and 2

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

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

Basic programming language elements

- Numbers and Arithmetic
  - In Racket, numbers can be integers, rationals, reals, or complex.

    42 22/7 3.141 2+3i
  - Arithmetic operations with operators and various arguments

    (opr args1 …)

    (ftn args1 …)

Basic programming language elements

- Numbers and Arithmetic
    - In Racket, numbers can be integers, rationals, reals, or complex.

      ```
      42    22/7    3.141    2+3i
      ```

    - Arithmetic operations with operators and various arguments:

      ```
      (+ 1 2 3 )
      (- 5.1 4)
      (/ 22 7)
      (modulo 23 3)
      (max 1 4 3 5 6)
      (min 2 5 3 4 5)
      (abs -6)
      (sqrt 4)
      ...
      ```

      https://docs.racket-lang.org/reference/generic-numbers.html#%28part._.Arithr

Basic programming language elements

- Variables and Functions

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

Basic programming language elements

- Variables and Functions
  - 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)
```

Basic programming language elements

- Variables and Functions
  - A disk of radius r has the approximate area $3.14 \cdot r^2$

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

(area-of-disk 5)

(area-of-disk 3)
```

Basic programming language elements

- Variables and Functions
    - Design the function for the area of a ring

Basic programming language elements

- Variables and Functions
  - Design the function for the area of a ring

```
; area-of-ring: number number -> number          [Contract]

; to compute the area of a ring whose radius     [Purpose]
; outer and whose hole has a radius of inner

; (area-of-ring 5 3 ) should produce 50.24       [Example]

(define (area-of-ring outer inner)                [Header]
        (- (area-of-disk outer)                   [Body]
           (area-of-disk inner)))
```

Basic programming language elements

- Variables and Functions
  - Design the function for the area of a ring

```
; area-of-ring: number number -> number        [Contract]

; to compute the area of a ring whose radius   [Purpose]
; outer and whose hole has a radius of inner

(define (area-of-ring outer inner)             [Header]
        (- (area-of-disk outer)                [Body]
           (area-of-disk inner)))

(test (area-of-ring 5 3) 50.24)                [Tests]
```

Basic programming language elements

- Conditional Expressions
  - Booleans and relations

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

Basic programming language elements

- Conditional Expressions
    - Booleans and relations
    - 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)
(test (is-5? 8) false)
```

Basic programming language elements

- Conditional Expressions
  - Booleans and relations
  - Functions that test conditions

```
;; is-between-5-6-or-over-10?: number -> boolean
;; 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)))
```

Basic programming language elements

- Conditional Expressions
  - Booleans and relations
  - Functions that test conditions

```
;; is-between-5-6-or-over-10?: number -> boolean
;; 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 (is-between-5-6? n) (>= n 10)))

;; is-between-5-6-or-over-10?: number -> boolean
;; to determine whether n is between 5 and 6 (exclusive)
(define (is-between-5-6? n)
          (and (< 5 n) (< n 6)))
```

Basic programming language elements

- Conditional Functions

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

Basic programming language elements

- Conditional Functions

  Suppose the bank pays 4% for deposits of up to $1,000 (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.

Basic programming language elements

- Conditional Functions

  Suppose the bank pays 4% for deposits of up to $1,000 (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.

```
;; interest-rate : number -> number
;; to determine the interest rate for the given amount
(define (interest-rate amount) …)
…
(test (interest-rate 1000) 0.040)
(test (interest-rate 2000) 0.045)
```

Basic programming language elements

- Conditional Functions

  Suppose the bank pays 4% for deposits of up to $1,000 (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.

```
;; interest-rate : number -> number
;; to determine the interest rate for the given amount
(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)
```

Basic programming language elements

- Conditional Functions

  Suppose the bank pays 4% for deposits of up to $1,000 (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.

```
;; interest-rate : number -> number
;; to determine the interest rate for the given amount
(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)
```

Basic programming language elements

- Symbols
  - A symbol is an identifier preceded by a single forward quotation mark:
    'the 'dog 'cat 'two^3 'and%so%on?
- Only one basic operation on symbols: symbol=?

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

Then, how we can design a program?

Steps to Good Programming

The Design Recipe for functions

- 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)))
```

Test-Driven Development

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

Test-Driven Development (2)

- Code and Tests are two wheels in a bike.
  - Tests are a great source of documents.
    - Precise
    - Executable and always in sync with code
  - Help us keep the design simple.
  - Help us make incremental progress.
  - Protect our code!!!

Test-Driven Development (3)

- 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?)
  - (test/regexp result_expr error_message_regexp)

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

More resources

1. http://plrg.kaist.ac.kr/doku.php?id=home:lectures:cs320_2018_2
2. http://www.cs.brown.edu/courses/cs173/2008/
3. http://pl.barzilay.org/
4. http://www.eng.utah.edu/~cs5510/f09/
5. http://users.eecs.northwestern.edu/~robby/courses/395-2010-winter/

JC

jcnam@handong.edu

https://lifove.github.io

* Slides are from Prof. Sukyoung Ryu's PL class in 2018 Spring or created by JC based on the main text book.