Before starting, what is this course about?

This course is about learning programming language concepts, data and procedures representations and interpreters. It is about mastering on programming. You cannot be a master before how your program is compiled/interpreted in your compiler/interpreter.

**Question 1:** What is a **compiler**, what is an **interpreter**?

This thing scan the entire program, generates an intermediate object to link program to memory and translates it to machine code.

So, this is a thing before creating a programming language. Are we going to produce a compiled or interpreted language? In this course we use Scheme, a dialect of LISP, as it offers a minimalist design, allows creating easy abstraction and functional and object-oriented programs in a dynamic environment.

Scheme is our minimalist tool to create other languages. Actually, as we see in the lab, we are using `#lang eopl` which is based on Scheme.

This tool includes 227 functions.

Example: you used the following function in Lab 4.
define-datatype is a function from eope

AS THE NAME OFFERS -> IT DEFINES A DATA TYPE

(define-datatype id predicate-id
  (variant-id (field-id predicate-expr) ... ...))

id → is your datatype
predicate-id → is your function that returns # t for instances of this datatype.
variant-id → is a constructor function that creates an instance of the datatype.
field-id → takes a) any argument that constructor takes.
predicate-expr → each argument is checked by the function.

What are the datatypes in LET lang?

Program and Expression.

a-program (const-exp)
is a variant of Program.
are variants of Expression.
check, if the datatype is the kind of instance
extracts the corresponding portion

define-datatype program program?
(a-program
 (expr expression!)))

WHAT ARE THESE?
LET is our very simple language. We are designing this language do not confuse LET with scheme-based let.

This is our language.

This function creates a local binding.

To define LET:

We need to specify syntax (it is how you structure your language)

  specify values (we have integer and decimal values)

  define an environment (this will affect how you make local binding)

  specify the behavior (implementation)

This part is where you use value of.

In our implementation we are defining two behavior functions for two types: value-of-program and value-of.

This note is a summary of the points about your questions and confusions. I did not answer all questions because they were so trivial or because you clearly you did not open any page of your textbook.

Your textbook is GREAT! Please read it carefully.

Best,
Alpay Sebnecoglu