# Introduction

PLC: Chapter 1



# Basic Terminology

## Meta and Object Language

💡 **Definition of Meta and Object Language**
An object language is a language you study (e.g., C#, C) using a meta language.

A meta language is the language in which we conduct our discussions, in this course F#

## Interpreter

💡 **Interpreter**
An interpreter for a program written in some language is one that evaluates the abstract syntax.

## Environment

💡 **Environment**
When we work with variables we need an environment which maps variable names to values. Typically done with a tuple list:

```
let env = [("a", 3); ("c", 78); ("baf", 666); ("b", 111)];;
```

## Syntax and Semantics

💡 **Syntax**
Syntax deals with form: is this program text well-formed?

We distinguish between two kinds of syntax:

- Concrete Syntax
  - The representation of a program as a text, with whitespace, curly braces etc.
- Abstract Syntax
  - The representation of a program as a tree, e.g., F# datatype `CstI` etc.

💡 **Semantics**
Semantics deals with meaning: What does this well-formed program mean?

We distinguish between two kinds of semantics:

- Dynamic Semantics
  - The meaning or effect of a program at run-time; what happens when it is executed. May be expressed by `eval` functions.

For the rest of the course we take the following approach:

- **Abstract Syntax:** F# datatypes

- **Concrete Syntax:** Lexer and Parser specifications
- **Semantics:** F# functions, both static semantics (checks) and dynamic semantics (execution). Can be described by direct interpretation using functions, or by compilation to another language.

# Simple Expr Language

Found in Chapter 1: `Inro2.fs`

A simple expression language with support for:

- Integer constants
- Variables
- Primitive operators +, -, *

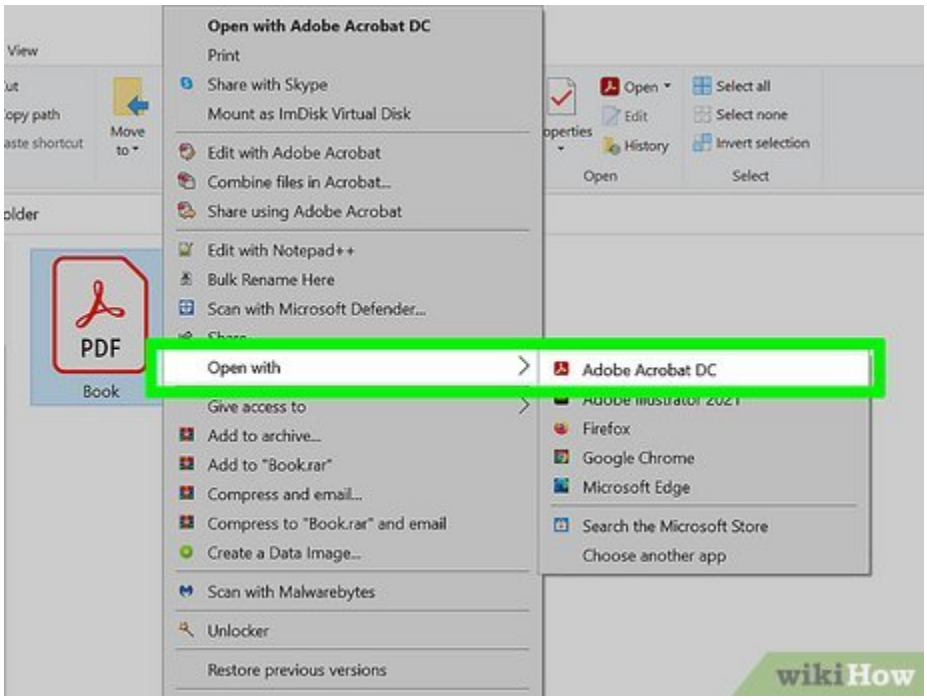
Variable mappings are stored in an environment list.

```
module Intro2

type expr =
  | CstI of int
  | Var of string
  | Prim of string * expr * expr;;

let rec lookup env x =
    match env with
    | []        -> failwith (x + " not found")
    | (y, v)::r -> if x=y then v else lookup r x

let rec eval e (env : (string * int) list) : int =
    match e with
    | CstI i            -> i
    | Var x             -> lookup env x
    | Prim("+", e1, e2) -> eval e1 env + eval e2 env
    | Prim("*", e1, e2) -> eval e1 env * eval e2 env
    | Prim("-", e1, e2) -> eval e1 env - eval e2 env
    | Prim _            -> failwith "unknown primitive"
```


Open with Adobe Acrobat DC
Print
Share with Skype
Mount as ImDisk Virtual Disk
Edit with Adobe Acrobat
Combine files in Acrobat...
Share using Adobe Acrobat
Edit with Notepad++
Bulk Rename Here
Scan with Microsoft Defender...
Open with
Give access to
Add to archive...
Add to "Book.rar"
Compress and email...
Compress to "Book.rar" and email
Create a Data Image...
Scan with Malwarebytes
Unlocker
Restore previous versions
Adobe Acrobat DC
Firefox
Google Chrome
Microsoft Edge
Search the Microsoft Store
Choose another app
PDF
Book
Open
Edit
History
Select all
Select none
Invert selection
Select
wikiHow