Exercise 2: ML Warm-Up

# Exercise 2.1: Bindings

Python:

1. Language Design Time: Syntax and semantics are defined.
   1. Syntax and semantics are the root of how logic will be formed and followed in the language and so it is evaluated first.
2. Language Implementation Time: Built-in functions and standard library are established.
   1. These functions are implemented early so that when python is compiled into bytecode, all necessary outside functions are present to successfully run the program on other systems.
3. Compile Time: Compiled into bytecode.
   1. The interpreter of python compiles the code and translates it into the lower level code to be executed by python's virtual machine and any other machines.
4. Link Time: Modules are imported and linked.
   1. This is a dynamic step of linking imported modules and packages at run time. This is so that only necessary functions are loaded, reducing memory space needed.
5. Load Time: Loads bytecode and libraries into memory.
   1. This step makes the code ready to run, assigning physical space and making sure all data has its memory space as needed.
6. Run Time: Variables are bound to memory and code is executed.
   1. This is when a variable's type is determined and the code is put into hard memory and run.

# Exercise 2.2: ML

SOSML code:

fun isEven(x:int): bool = x mod 2 = 0;

> val **isEven = fn**: *int → bool*;

fun select(l:'a list, f:'a -> bool): 'a list = List.filter f l;

> val **select = fn**: *∀ 'a . 'a list \* ('a → bool) → 'a List.list*;

val numbers: int list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

> val **numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**: *int list*;

select(numbers, isEven);

> val **it = [2, 4, 6, 8, 10]**: *int List.list*;

val evenNumbers: int list = select(numbers, isEven);

> val **evenNumbers = [2, 4, 6, 8, 10]**: *int List.list*;

# Exercise 2.3: Types

The difference between type and datatype:

* Type: defines what sort of data belongs to a variable in a more general sense.
* Datatype: defines the exact format and size of a variable in memory.

The different kinds of ML type constructors:

* Tuple: uses \* as the constructor, it is parentheses sensitive.
* List: the type of the list defines the types of the contents in the list.
* Function: uses -> as the constructor, takes in a type and produces a type.

Algebraic or union types in ML and the of keyword:

* Algebraic data types (ADT): similar to C structs, storing multiple fields (point = (x,y)).
* Of keyword: defines the types of the variant of the ADT (point of int).

(\*categorizing archer competitors for a tournament at different ranges and shooting types\*)

(\*Enumerated Type\*)

datatype BowStyle = OlympicRecurve | Traditional | Compound;

datatype SkillLevel = Beginner | Intermediate | Expert;

(\*Algebraic Type\*)

datatype Distance =

TwentyMeters

| ThirtyMeters

| SixtyMeters;

(\*Record Type\*)

type Archer = {name: string, style: BowStyle, level: SkillLevel};

(\*Type Alias\*)

type Target = Distance;

(\*Function 1\*)

fun difficulty (distance: Distance) : int =

case distance of

TwentyMeters => 20

| ThirtyMeters => 30

| SixtyMeters => 60;

(\*Function 2\*)

fun categorizeArcher ({name, style, level}: Archer, distance: Distance) : string =

let val difficultyLevel = difficulty(distance)

in name ^ " shoots at a difficulty level of " ^ Int.toString(difficultyLevel) ^ "."

end;

(\*Values\*)

val archer1 = {name = "Alice", style = OlympicRecurve, level = Intermediate};

val archer2 = {name = "Bob", style = Traditional, level = Beginner};

val archer3 = {name = "Charlie", style = Compound, level = Expert};

val distance1 = TwentyMeters;

val distance2 = ThirtyMeters;

val distance3 = SixtyMeters;

val result1 = categorizeArcher(archer1, distance1);

val result2 = categorizeArcher(archer2, distance2);

val result3 = categorizeArcher(archer3, distance3);