Introduction to ML

Tutorial 10

Outline

- Functional Programming
- Standard ML of New Jersey
- Getting Started with SML
- Types in ML
    - Tuple / List
- Function
    - Recursive Function
        - Pattern Matching
    - Higher Order Function
- Assignment 4 Q3

Functional Programming

- Functional programs are made up of **functions** applied to **data**. The building blocks are functions.
- ML is primarily a functional language.

Functional Programming

- ML has no operations that permanently change the value of some variable, like

  `a = b+c (side-effect)`

- ML evaluates an expression like `b+c`, the value of which is associated with the name `a`.

  `Expression rather than commands.`

Functional Programming Language

- Example:

```
- val x=3;
- fun addx(a)=a+x;
- val x=10;
- addx(2);
val it = 5 : int
```

Side-Effect Freedom

val-declaration always creates a new entry. The later declaration does not affect the previous declaration.

Standard ML of New Jersey

- SML/NJ is a full compiler, with associated libraries, tools, an interactive shell, and documentation.
- https://www.smlnj.org/
- Developed by various parties including Princeton University.

- Download and install SML/NJ.

Start Your Program with SML

- Interative Mode
  - To invoke SML, go to sparc machine, type
    - `$ sml`
  - To load the code in the file “myfile”
    - `- use "myfile";`
  - Remember end of statement (`;`)
  - `Ctrl-D` to exit

- Another way to direct execute the program
  - `$ sml < myfile`

Getting Started in ML

- ML as a programmable calculator

```
- 2*3;
val it = 6 : int
- length [1, 2, 3, 4, 5];
val it = 5 : int
- "house" ^ "cat";
val it = "housecat" : string
```

Types in ML

- Primitive Type
  - int : 3, ~4 (~ as minus/negative)
  - real : 3.5, ~9.4
  - string : “House"
  - char : #"c"
  - bool : true, false
- Composite Type
  - Tuple
  - List
  - Function
  - Union

Concatenate two strings:

```
- “House” ^ “cat”
val it = “Housecat” : string
```

Combining Logical Values:
orelse / andalso

```
- 1<2 orelse 3>4
val it = true : bool
```

short circuit evaluation

3>4 is not evaluated since 1<2 is true

Type Consistence

- ML is a strongly typed language in that it requires types of operators and operands to be consistent.

```
- 1 + 2.0
Error: operator and operand do not agree
```

- ML is also a statically typed language.
  - Type inference

```
- fun double(x) = 2*x;
val double = fn : int -> int;
```

ML can deduce the type of x as integer.

Then ML can deduce the argument type and return type of the function.

Type Inference

- Type inference for some overloaded operators.

```
- fun add(x,y) = x+y;
val add = fn: int * int -> int
add(1., 2.)
Error;
```

add sign is overloaded for integer type and real type. Thus, the compiler might not deduce them properly because there are no clues.

- We need to give some clues for ML to deduce the type properly.

```
- fun add(x:real,y) = x+y;
val add = fn: real * real -> real
```

Tuple

- Fixed number of components, possibly mixed typed
- Enclosed by parentheses

```
- val x = (true,3.5,"x")
val x = (true,3.5,"x") : bool * real * string
- #1 x;
val it = true: bool
- #3 x;
val it = “x”: string
```

It takes constant time to access the element of the tuple.

List

- Sequence of identically typed components of any length
- Enclosed by square brackets

```
["Andrew","Ben"]      : string list
[(2,3),(2,2),(9,1)] : (int*int) list
```

“T list” is the list type, whose elements are of the same type T.

List

- nil is the empty list
- a::b = head item a + tail list b

List in ML is a LinkedList.

It takes linear time to access the elements.

`nil`	`[]`
`1::nil`	`[1]`
`2::(1::nil)`	`[2,1]`
`3::2::1::nil`	`[3,2,1]`
`4::3::2::1`	Error

List - built-in functions

- `a@b` = concatenation of 2 lists `a`, `b`
- `hd(L)` = 1st element (head) of `L`
- `tl(L)` = List without head of `L`
- `null(L)` is true if `L` = `nil`
- `length(L)` = number of elements in `L`

Function

- Function Type
  - Parameter type -> Return type

```
- fun double(x:int):int = 2*x;
val double = fn : int -> int;
```

explicitly define parameter type and return type

  - Recall type Inference

```
- fun double(x) = 2*x;
val double = fn : int -> int;
```

- Declaring the parameter type and return type is always a good practice that lets other people easier to understand the code. Please specify the parameter type and return type in the assignment.

.

Function

- Functions in ML takes only one argument.

- Single parameter function:

```
fun adda s = s^"a";
```

- Multiple parameter function:
  - Using tuple to include all parameters

```
- fun add(x : int,y) = x+y;
val add = fn : int * int -> int;
```

Recursive Function

- The recursive functions in ML substitute for most of the iterations such as while-loops or for loops.
  - ML

```
fun reverse L =
       if L = nil then nil
       else reverse(tl(L)) @ [hd(L)];
```

  - C++

```
int* reverse(int* L, int len){
       for (int i = 0; i < len/2; i ++)
          swap(L, i, len-i);
       return L;
}
```

Recursive Function

- Pattern Matching + Recursion

```
–  fun reverse nil = nil
=   |  reverse (h::t) = reverse(t) @ [h];
```

`h::t` pattern matches a list of at least one element.
h matches the head element, t matches the tail list.

The general form for a function defined by patterns involves the symbol |

```
fun <identifier> <pattern 1> = <expression 1>
|   <identifier> <pattern 2> = <expression 2>
|    ...
|   <identifier> <last pattern> = <last expression>
```

Do matching
from top to down

Recursive Function

- Pattern Matching + Recursion
  - Anonymous Variable “_” matches any value.

```
- fun comb(_, 0) = 1
= |   comb(n, m) =
=         if m=n then 1
=         else comb(n-1, m) + comb(n-1, m-1);
```

```
Our first attempt might be
fun comb(_, 0) = 1
|   comb(n, n) = 1
|   comb(n, m) = comb(n-1,m) +
comb(n-1, m-1);
```

Unfortunately, this code leads the error message

`Error: duplicate variable in pattern(s): n`

So we are forced to use a conditional expression

Higher Order Functions

Functions can take functions as arguments.

- C/C++
  - `int inc(int x) {return 1+x; }`
  - `int double(int x) { return 2*x; }`
  - `int square(int x) { return x*x; }`
  - `int inc2(int x) {return inc(inc(x)); }`
  - `int quad(int x) {return double(double(x));}`
  - `int fourth(int x) {return square(square(x));}`

- ML
  - `fun inc(x) = x + 1;`
  - `fun double(x) = x * 2;`
  - `fun square(x: int) = x * x;`
  - `fun applytwice f = fn x => f( f(x) );`
  - `val inc2 = applytwice inc;`
  - `val quad = applytwice double;`
  - `val fourth = applytwice square;`

Higher order function

Anonymous function `fn`

Let Expression and Nested Environment

- Let expressions are one way to introduce local environments.
- Given n, return the nth Fibonacci number

local declarations inside the let expression

```
fun fib n =
  let
     fun fibi (a,b,0) = a
     | fibi (a,b,n) = fibi (b,(a+b),(n-1))
  in
     fibi (1,1,n)
  end;

fibi(1,1,n);
Error: unbounded variable or constructor
```

Union Type

- Definition:

```
datatype money = cash of real | cheque of string * real;
```

- Usage:

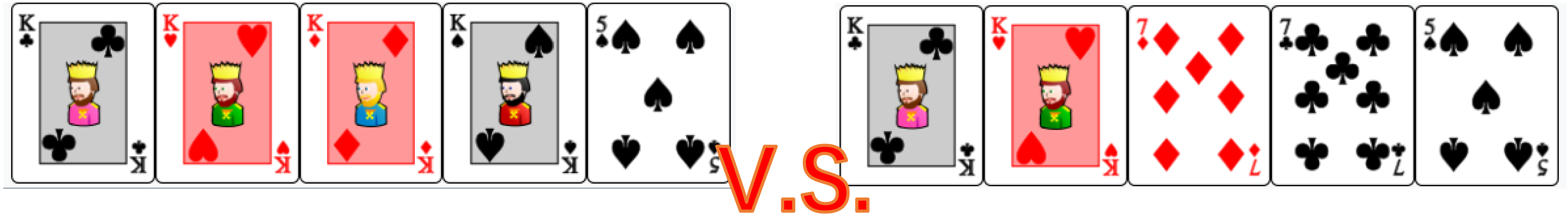
```
- val lunch = cash 45.;
- val car   = cheque(“HSBC",36000.0);
```

- Pattern matching:

```
- fun worth(cash x) = x
=  |  worth(cheque(“HSBC”,x)) = 0.9*x
=  |  worth(cheque(_,_)) = 0.0;
```

Assignment 4 – Q3 (ML)

- This problem involves a card game invented just for this question. You will implement a program to based on several helper functions.
- Two players will get five cards respectively.
- Each card has two attributes, $i.e.$, suit and rank.
- The player who has the better hand will win the game.
- We refer the rule from https://www.pagat.com/poker/rules/ranking.html.



V.S.

Hand 1 Wins

Assignment 4 – Q3 (ML)

- Type Definition of Cards
  - For simplicity, we directly use **int** as card's **rank.** We convert all ranks to integers. i.e., A = 1, J = 11, Q = 12, and K = 13.

```
datatype suit = Clubs | Diamonds | Hearts | Spades;
```

  - We use a tuple to represent a card, like `(Clubs, 5), (Spades, 13)`

- Type Definition of Hands

```
datatype hand = Nothing | Pair | Two_Pairs | Three_Of_A_Kind |
Full_House | Four_Of_A_Kind | Flush | Straight;
```

Assignment 4 – Q3 (ML)

Hands	Description	Examples
Four of a Kind	Four cards of the same rank	♠3- ♥3- ♦3- ♦3- ♣A
Full House	Three cards of one rank and two cards of another rank	♦9- ♦9- ♠9- ♠4- ♣4
Flush	Five cards of the same suit	♠K- ♠J- ♠9- ♠3- ♠2
Straight	Five cards in sequence	♠Q- ♦J- ♥10- ♠9- ♣8
Three of a Kind	Three cards of the same rank plus two unequal cards	♠5- ♦5- ♥5- ♥3- ♣2
Two Pairs	Two pairs of cards, each pair are of same rank	♠Q- ♥Q- ♣5- ♠5- ♠4
Pair	Two cards of equal rank and three cards which are different from these and from one another	♥6- ♥6- ♣4- ♠3- ♦2
Nothing	Five cards which do not form any of the combinations listed above.	♣A- ♠J- ♣9- ♦5- ♠3

Card List Examples

- The input cards are already sorted in **descending** order according to the rank (with A always ordered last).
  - Two cards of same rank can be ordered arbitrarily.

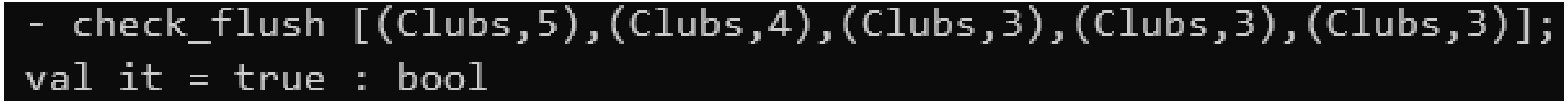
```
[(Clubs, 10), (Clubs, 9), (Hearts, 9), (Spades, 9), (Spades, 3)]
[(Diamonds, 11), (Spades, 11), (Clubs, 11), (Hearts, 11), (Hearts, 10)]
```

- The following card lists represent the same hand.

```
[(Clubs, 13), (Spades, 13), (Hearts, 6), (Spades, 1), (Diamonds, 1)]
[(Spades, 13), (Clubs, 13), (Hearts, 6), (Spades, 1), (Diamonds, 1)]
[(Clubs, 13), (Spades, 13), (Hearts, 6), (Diamonds, 1), (Spades, 1)]
[(Spades, 13), (Clubs, 13), (Hearts, 6), (Diamonds, 1), (Spades, 1)]
```

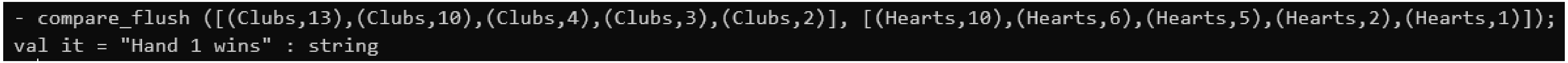
Assignment 4 – Q3 (ML)

- Write an ML function **check_flush**, which takes a list of five cards and returns if the hand is a flush.



```
- check_flush [(Clubs,5),(Clubs,4),(Clubs,3),(Clubs,3),(Clubs,3)];
val it = true : bool
```

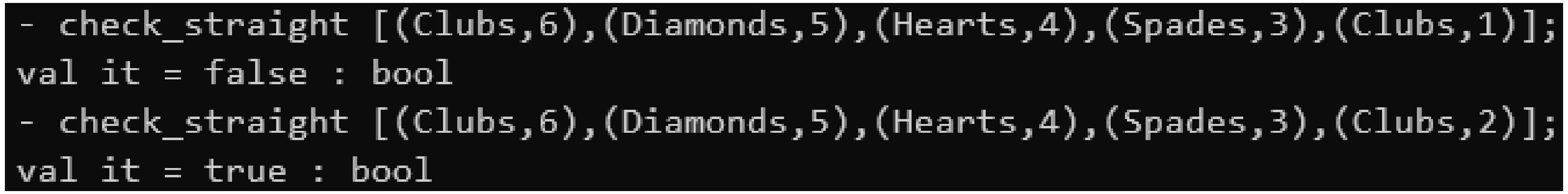
- Write an ML function **compare_flush**, which takes two flush card lists. The return value is a string selected from three candidates. i.e., "Hand 1 wins", "Hand 2 wins" and "This is a tie".



```
- compare_flush ([(Clubs,13),(Clubs,10),(Clubs,4),(Clubs,3),(Clubs,2)], [(Hearts,10),(Hearts,6),(Hearts,5),(Hearts,2),(Hearts,1)]);
val it = "Hand 1 wins" : string
```

Assignment 4 – Q3 (ML)

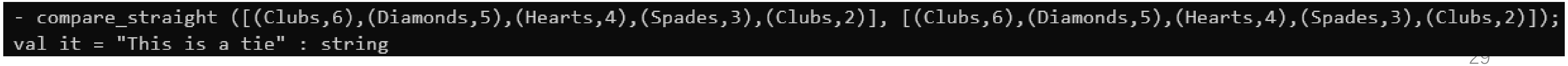
- Write an ML function **check_straight**, which takes a list of five cards and returns if the hand is a straight.



```
- check_straight [(Clubs,6),(Diamonds,5),(Hearts,4),(Spades,3),(Clubs,1)];
val it = false : bool
- check_straight [(Clubs,6),(Diamonds,5),(Hearts,4),(Spades,3),(Clubs,2)];
val it = true : bool
```

  - Note that Ace can count high or low in a straight although Ace is the smallest rank.
  - So, K-Q-J-10-A (largest straight) and 5-4-3-2-A (smallest straight) are valid straights, but K-Q-J-2-A is not.

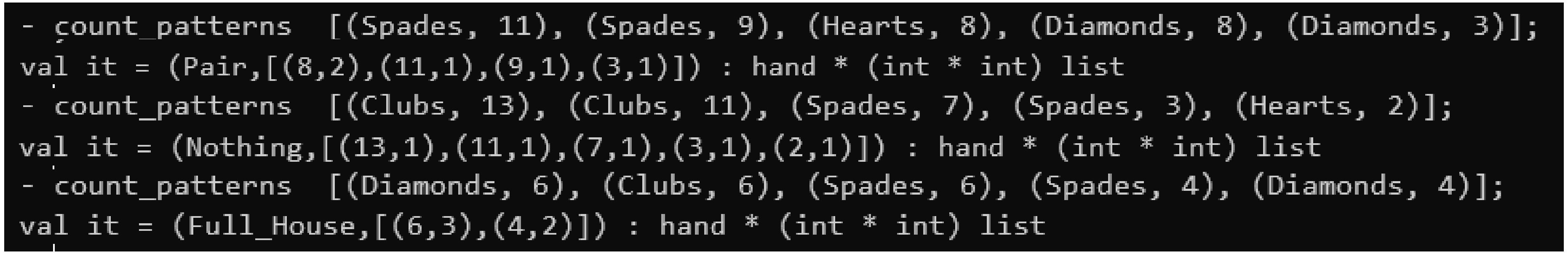
- Write an ML function **compare_straight**, which takes two straight card lists. The return value is a string selected from three candidates. i.e., “Hand 1 wins”, “Hand 2 wins” and “This is a tie”.



```
- compare_straight ([(Clubs,6),(Diamonds,5),(Hearts,4),(Spades,3),(Clubs,2)], [(Clubs,6),(Diamonds,5),(Hearts,4),(Spades,3),(Clubs,2)]);
val it = "This is a tie" : string
```

Assignment 4 – Q3 (ML)

- Write an ML function **count_patterns**, which takes a list of five cards and returns the **hand type** (Nothing, Pair, Two Pairs, Three of a Kind, Full House, Four of a Kind) except Straight and Flush and a list of **rank-quantity pairs**.



```
- count_patterns  [(Spades, 11), (Spades, 9), (Hearts, 8), (Diamonds, 8), (Diamonds, 3)];
val it = (Pair,[(8,2),(11,1),(9,1),(3,1)]) : hand * (int * int) list
- count_patterns  [(Clubs, 13), (Clubs, 11), (Spades, 7), (Spades, 3), (Hearts, 2)];
val it = (Nothing,[(13,1),(11,1),(7,1),(3,1),(2,1)]) : hand * (int * int) list
- count_patterns  [(Diamonds, 6), (Clubs, 6), (Spades, 6), (Spades, 4), (Diamonds, 4)];
val it = (Full_House,[(6,3),(4,2)]) : hand * (int * int) list
```

Note that they are sorted by the count, then by the rank.

Hints for `count_pattern`

- Define some helper functions.
  1. Get the list of unsorted rank-quantity pairs
     - `count(L: card list): (int*int) list`
  2. Get the hand type and sorted list of rank-quantity pairs.
     - Considering all possible hand types.
       - **e.g.**

         Four_Of_A_Kind : [(a, 4), (b, 1)] or [(a, 1), (b, 4)]

         Three_Of_A_Kind: [(a, 3), (b, 1), (c, 1)] or [(a, 1), (b, 3), (c, 1)] or [(a, 1), (b, 1), (c, 3)]

         Full_House: ?

         Two_Pairs: ?

         Pair: ?

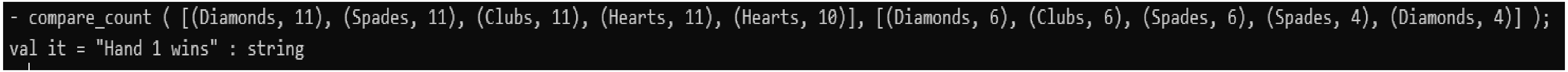
         Other hand types?

Pattern Matching can be helpful in this function.

Assignment 4 – Q3 (ML)

- Write an ML function **`compare_count`**, which takes two card lists and returns a string selected from three candidates. i.e., “Hand 1 wins”, “Hand 2 wins” and “This is a tie” except flush and straight.

Nothing < Pair < Two Pairs < Three of a Kind < Full House < Four of a Kind



```
- compare_count ( [(Diamonds, 11), (Spades, 11), (Clubs, 11), (Hearts, 11), (Hearts, 10)], [(Diamonds, 6), (Clubs, 6), (Spades, 6), (Spades, 4), (Diamonds, 4)] );
val it = "Hand 1 wins" : string
```

Four of a Kind

Full House

Hints for `compare_count`

- Define some helper functions
  - `compare_rank`
    - Compare two card lists according to the rank-quantity list.
  - `compare`
    - Considering all possible situations from `count_patterns`
    - **Please check the comparisons order according to the certain hand type**
      - **e.g.**
        1. Four_Of_A_Kind always wins hands of other types.
        2. Nothing always loses hands of other types.
        3. When two card lists are of same hand type, directly call `compare_rank`
        4. What if two card lists are of different types ?

How to do pattern matching here ?

Good luck for your final exam!