1、
如果函数是这样的：
fun foo (alist, blist, intnum)
需要对 alist blist 做模式匹配的话，没必要先对 alist 做模式匹配，再对 blist 做模式匹配，可以一起模式匹配，比如(_,[],num)或者([],[],_)这样，可以对三个同时模式匹配都可以。
只关注我们关注的情况，其他就转到默认分支处理（抛出异常或者其它什么的）

fun foo (alist, intnum) =
    case alist of
x=>|y=>|x=>这种写法比
fun foo ([], intnum) =
| foo (alist, intnum)
好多了。。。下面那个，如果没有注释，你很难懂接受的参数是啥，很容易看的懵逼。

2、无用的局部变量绑定不是一个好的选择，对于只调用一次的函数，直接在调用处写就好了，没必要专门搞一个局部变量保存函数返回值。上面是建议的，下面是不建议的，因为这个函数只用了一次，那直接放到 case 里面就可以了，没必要新建一个局部变量。

```
fun get_substitutions1 (substitutions,str) =
    case substitutions of
        [] => []
      | x::xs => case all_except_option(str,x) of
                     NONE => get_substitutions1(xs,str)
                   | SOME y => y @ get_substitutions1(xs,str)

```

```
fun get_substitutions1 (substitutions,str) =
    case substitutions of
        [] => []
      | x::xs => let val foo = all_except_option(str,x)
                 in case foo of
                        NONE => get_substitutions1(xs,str)
                      | SOME y => y @ get_substitutions1(xs,str)
                 end
```

3、函数参数不需要显示声明类型， 系统会自动推导他的类型，我只要写一个单词表示这个变量的含义就好了，上面的代码是答案的，下面是我的。同样的，当函数只调用一次的时候，我就在调用处调用就好了，没必要新建一个局部变量去保存中间结果。而答案的参数是 name，非常直观，后面用一个变量绑定的时候就指出了这个 name 的类型了。

```
fun similar_names (substitutions,name) =
    let
        val {first=f, middle=m, last=l} = name
        fun make_names xs =
           case xs of
               [] => []
             | x::xs' => {first=x, middle=m, last=l}::(make_names(xs'))
    in
        name::make_names(get_substitutions2(substitutions,f))
    end
```

```
(* d  *)
fun similar_names (substitutions, {first=x, middle=y, last=z}) =
    let val namelist = get_substitutions2(substitutions, x)
        fun helper (namelist) =
            case namelist of
                [] => []
              | curname::namelist' => {first=curname, middle=y, last=z}::helper(namelist')
    in
        {first=x, middle=y, last=z}::helper(namelist)
    end
```

4、对于返回值是 bool 型的表达式，不需要通过 if then else 来判断，直接写下这个表达式就好了！！我的写法（上面的）就类似于 if true then true else false，这不多此一举吗？ 完全可以写成 e1 andalso e2 的形势。
还有就是一个简略的做法， 只列出特殊情况，其余的相同返回值的情况就用_，写代码的时候一定要清晰，有多少返回值不同的情况。！！

```
fun all_same_color cards =
    case cards of
        [] => true
      | fc::[] => true
      | fc::sc::cards' => if (card_color fc) = (card_color sc)
                          then all_same_color (sc::cards')
                          else false

(* e  *)
```

```
1  fun all_same_color cs =
2      case cs of
3          [] => true
4        | [_] => true
5        | head::neck::tail => card_color head = card_color neck
6            andalso all_same_color(neck::tail)
7
8  fun all_same_color cs =
9      case cs of
10         head::neck::tail => card_color head = card_color neck
11           andalso all_same_color(neck::tail)
12       | _ => true
13
```

5、只用一次的表达式也可以直接用括号扩住。。特别是只用一次的 if then else 语句，直接用括号扩住代码反而会清晰很多。

```
1  fun score (cs,goal) =
2      let
3          val sum = sum_cards cs
4      in
5          (if sum >= goal then 3 * (sum - goal) else goal - sum)
6          div (if all_same_color cs then 2 else 1)
7      end
```

```
fun score (held_cards, goal) =
    let val cardsum = sum_cards held_cards
        val presum = if cardsum > goal then 3 * (cardsum-goal) else goal - cardsum
    in
        if all_same_color held_cards
        then presum div 2
        else presum
    end
```