



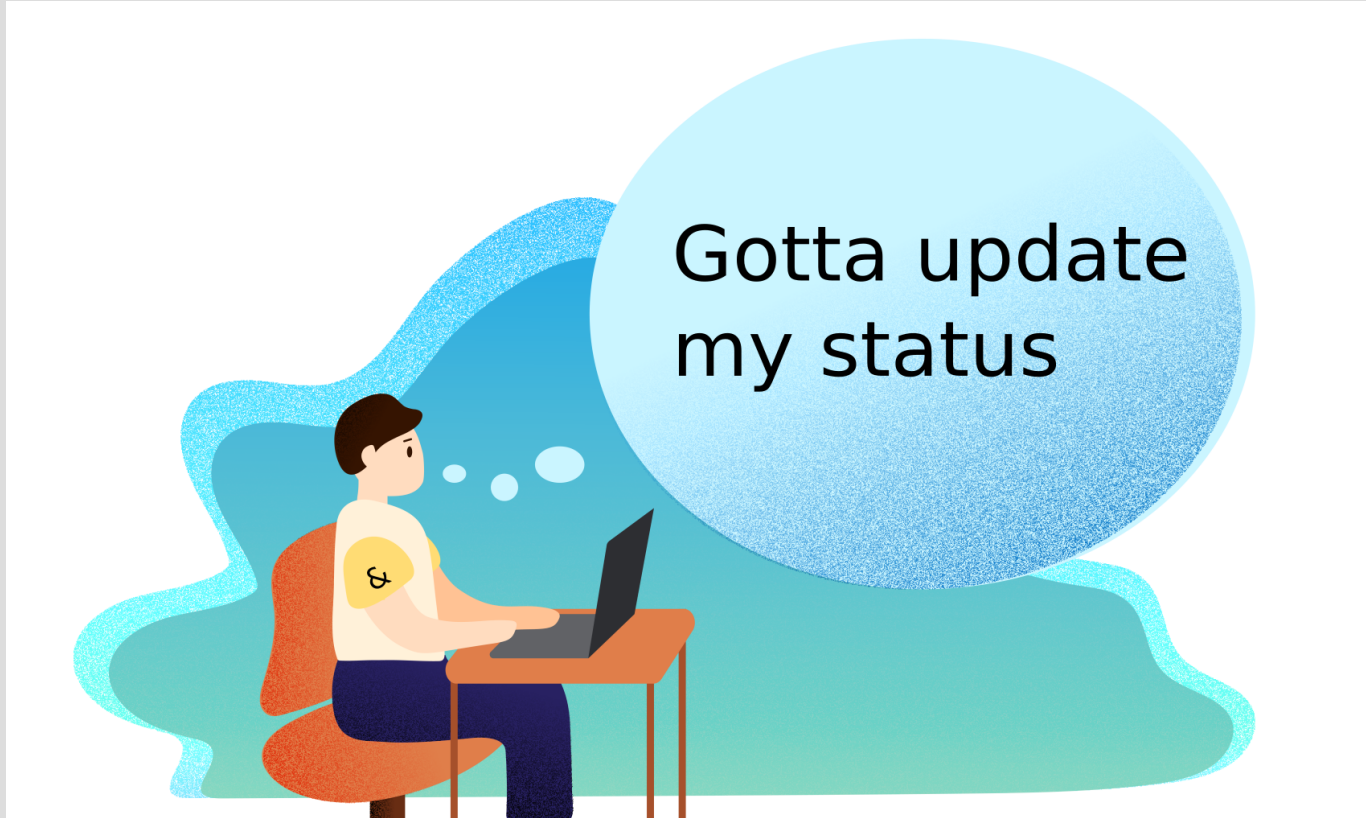
Imperative Programming

Having an effect

15-150 M21

Lecture 0802
02 August 2021

0 Effects in SML



Effect: A change to the state of the computer or the world



Value: A piece of data which is “fully calculated” or “fully simplified” – the kind of thing that can serve as an answer to a computational question (need to specify what this means)

Recall we had the notion of **purity**:

- An expression is pure if evaluating it causes no side-effects
- A function is pure if applications of it never cause side-effects
- A language is pure if it doesn't have any kind of effects

So far, we've been working with just the pure fragment of SML. But SML is not completely pure...

Demonstration:

`print`

```
print : string -> unit
```

We're generally not interested in pure `unit`-returning functions: for each type `t`, there's only one pure, total function of type `t -> unit`.

Usually, functions which return a `unit` are *impure*: we're executing them for their effect.

Demonstration:

before, ignore, and ;

1 Refs

SML has a built-in datatype called `ref`

```
datatype 'a ref = ref of 'a
```

But `refs` are special: the data inside a `ref` cell is **mutable**, using the reassignment operator, `:=`.

Ref methods

`(op :=) : 'a ref * 'a -> unit`

`! : 'a ref -> 'a`

Demonstration:

The Ref structure

Now with fewer bugs!

5 minute break

Warning #1:

Keep it fresh

A simple example

```
val r1 = ref 0
val r2 = r1
val r3 = ref 0
```

`r2` is just an *alias* for `r1`, whereas `r3` is an independent ref cell.

Key Point:

One ref cell is created for every
use of the `ref` constructor

Warning #2:
Stage carefully

0802.0 (refs.sml)

```
2 fun f1 x y =  
3 let  
4   val r = ref x  
5 in  
6   if y < (!r)  
7   then  
8     !r + y before r := y  
9   else !r + y  
10 end
```

0802.1 (refs.sml)

```
13 fun f2 x =  
14 let  
15   val r = ref x  
16 in  
17   fn y =>  
18     if y < (!r)  
19     then  
20       !r + y before r := y  
21     else !r + y  
22 end
```

When `ref`s and effects are involved, we find ourselves in an annoying situation: evaluating the same code at different times can give different results.

Our old definition of extensional equivalence is inadequate to guarantee **referential transparency**: if e_1 and e_2 are impure, then $e_1 \cong e_2$ does not mean e_1 and e_2 are interchangeable, because they might have different side effects.

2 Other Effects

Colors!

Standard colors								High-intensity colors																											
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15																				
216 colors																																			
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51
52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87
88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123
124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195
196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231
Grayscale colors																																			
232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255												

```
(* YY, with foreground color 0<=XX<=255 *)
```

```
"\^[[38;5;XXmYY\^[[0m"
```

```
(* YY, with background color 0<=XX<=255 *)
```

```
"\^[[48;5;XXmYY\^[[0m"
```

Demonstration:

OS.Process.system

Demonstration:

File I/O

Module:

Timing

- SML is not a pure functional language: there are ways to cause effects
- We can use ref cells to have mutable data
- Effects introduce extra headache when reasoning about code

Review!

Thank you!