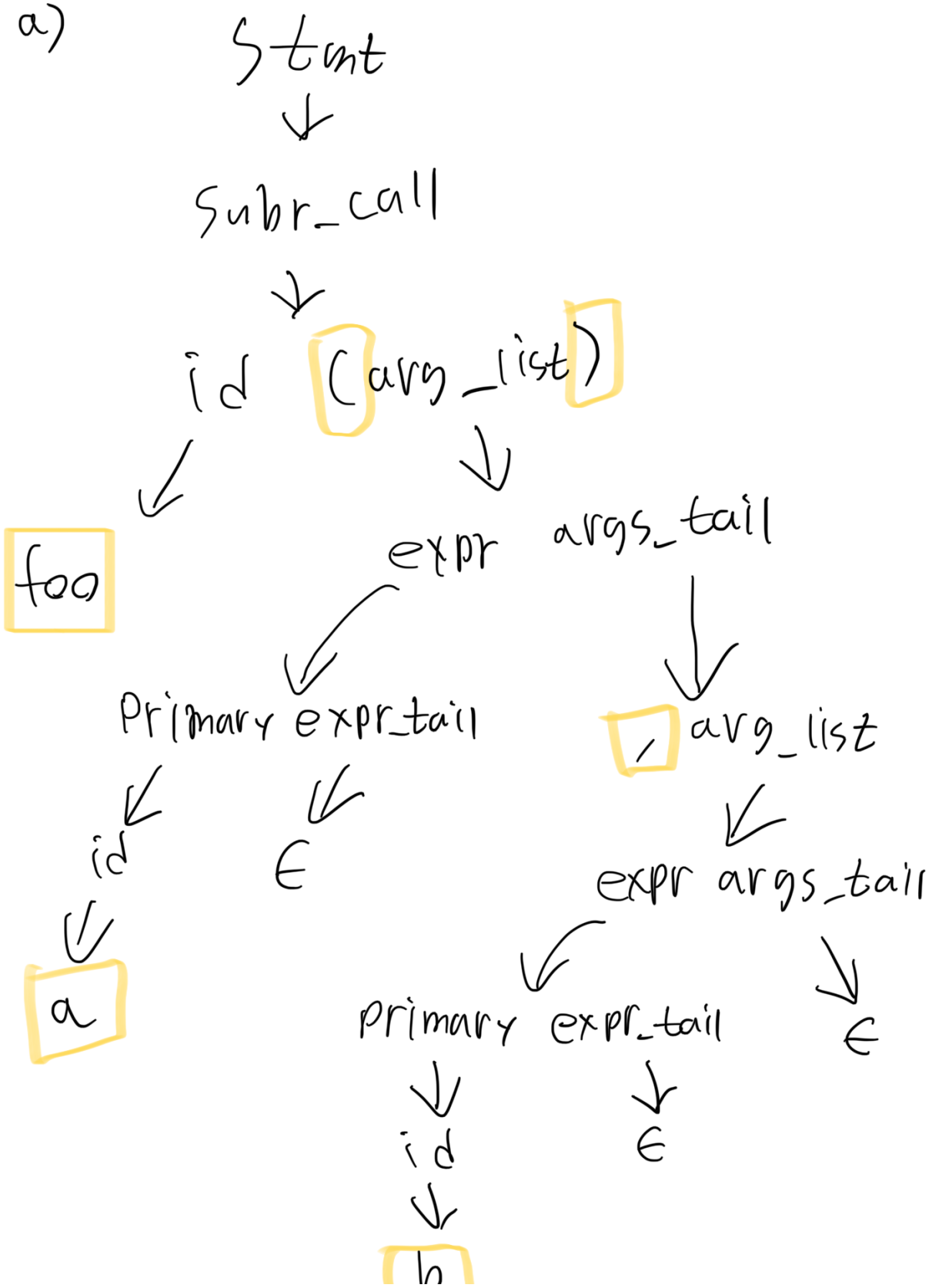
HW2

Page 108-112: 2.13, 2.17, 2.18 and 2.40 (use python)

2.13

a)


Stmt
subr_call
id
(arg_list)
foo
expr
args_tail
Primary
expr_tail
,
arg_list
id
ϵ
expr
args_tail
a
primary
expr_tail
ϵ
id
ϵ
b

(b)
foo(a, b)

stmt → subr_call
→ id (arg_list)
→ id (expr args_tail)
→ id (expr , arg_list)
→ id (expr , expr args_tail)
→ id (expr , expr ε)
→ id (expr , primary expr_tail
ε)
→ id (expr , primary ε ε)
→ id (expr , id ε ε)
→ id (primary expr_tail , id ε ε)
→ id (id ε , id ε ε)

(c)
It can’t be LL(1). Because there is an ambiguity in the "expr" non-terminal to ”id”. We cannot determine whether to expand "expr" into "primary" followed by an "id" or a "subr call”.

(d)

stmt → *id stmt_tail*
stmt_tail → id := *expr*
→ (*arg list*)
expr → primary expr_tail
expr_tail → op expr
→ ε
primary → id primary_tail
primary → (arg_list)
→ (*expr*)
primary_tail → ε
op → + | - | * | /
arg_list → *expr args_tail*
args_tail → , *arg list*
→ ε

2.17

program → stmt list $$
stmt_list → stmt list_stmt | ε
stmt → id := expr
| read id
| write expr
| if condition then stmt_list fi

```
                | while condition do stmt_list od
condition   → expr comp_op expr
comp_op     → < | <= | > | >= | == | !=
expr        → term expr_tail
expr_tail   → add_op term expr_tail | ε
term        → factor term_tail
term_tail   → mult_op factor term_tail | ε
factor      → ( expr )
                | id
                | number
add_op → + | -
mult_op     → * | /
```

2.18

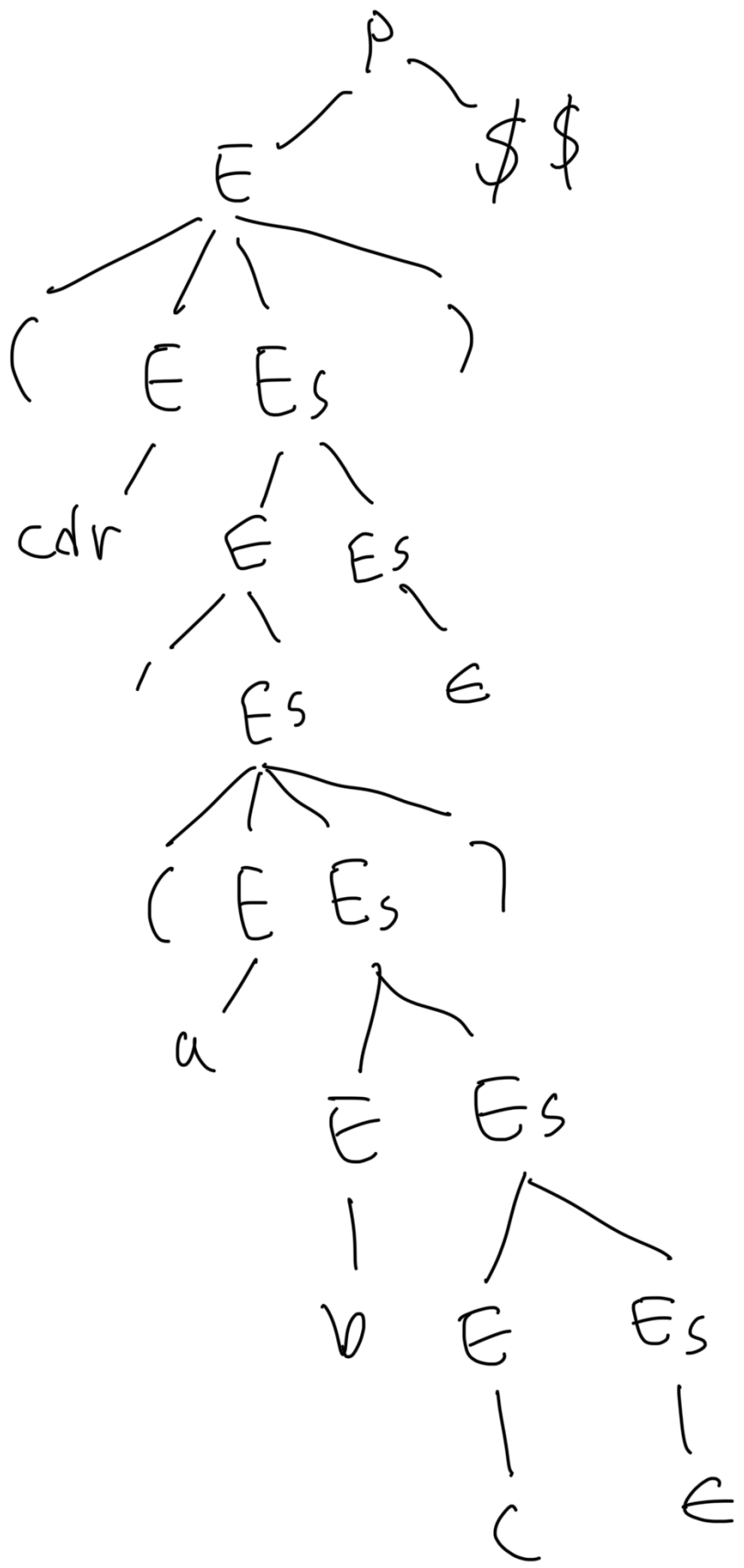
(a)

FIRST(Es) : atom, ′, (,), ε

FOLLOW (E): atom, ′, (,), $$

PREDICT (Es →ε):)

(b)



(c)

P → E $$
→ (E Es) $$
→ (cdr E s) $$
→ (cdr E Es) $$
→ (cdr ‘ E E s) $$
→ (cdr ‘ (E Es) Es) $$
→ (cdr ‘ (a Es) Es) $$
→ (cdr ‘ (a E Es) Es) $$
→ (cdr ‘ (a b Es) Es) $$
→ (cdr ‘ (a b E Es) Es) $$
→ (cdr ‘ (a b c Es) Es) $$
→ (cdr ‘ (a b c) Es) $$
→ (cdr ‘ (a b c)) $$

(d)

Parse stack	**Input stream**	**Comment**
P	(cdr ’ (a b c)) $$	
E$$	(cdr ’ (a b c)) $$	Predict P → E $$
(E Es) $$	(cdr ’ (a b c)) $$	Predict E → (E Es)

E Es) $$	cdr ′ (a b c)) $$	Match (
atom Es) $$	cdr ′ (a b c)) $$	Predict E → atom
Es) $$	′ (a b c)) $$	Match atom
E Es) $$	′ (a b c)) $$	Predict Es →E Es
′ E Es) $$	′ (a b c)) $$	Predict E → ′ E
E Es) $$	(a b c)) $$	Match ′
(E Es) Es) $$	(a b c)) $$	Predict E → (E Es)
E Es) Es) $ $	a b c)) $$	Match (
atom Es) Es) $$	a b c)) $$	Predict E → atom
Es) Es) $$	b c)) $$	Match atom
E Es) Es) $ $	b c)) $$	Predict Es → E Es
atom Es) Es) $$	b c)) $$	Predict E → atom
Es) Es) $$	c)) $$	Match atom

E Es) Es) $ $	c)) $$	Predict Es → E Es
atom Es) Es) $$	c)) $$	Predict E → atom
Es) Es) $$	)) $$	Match atom
) Es) $$	)) $$	Predict Es → ε
Es) $$	) $$	Match)
) $$	) $$	Predict Es → ε
$$	$$	Match)

(e)
“P”, “E”, “Es”, “match”

2.40
Easier. I think that Python’s amazing readability is the main reason why Python is so popular. The whitespaces makes the code neat and the lack of braces and parentacies

mekes the code cleaner.