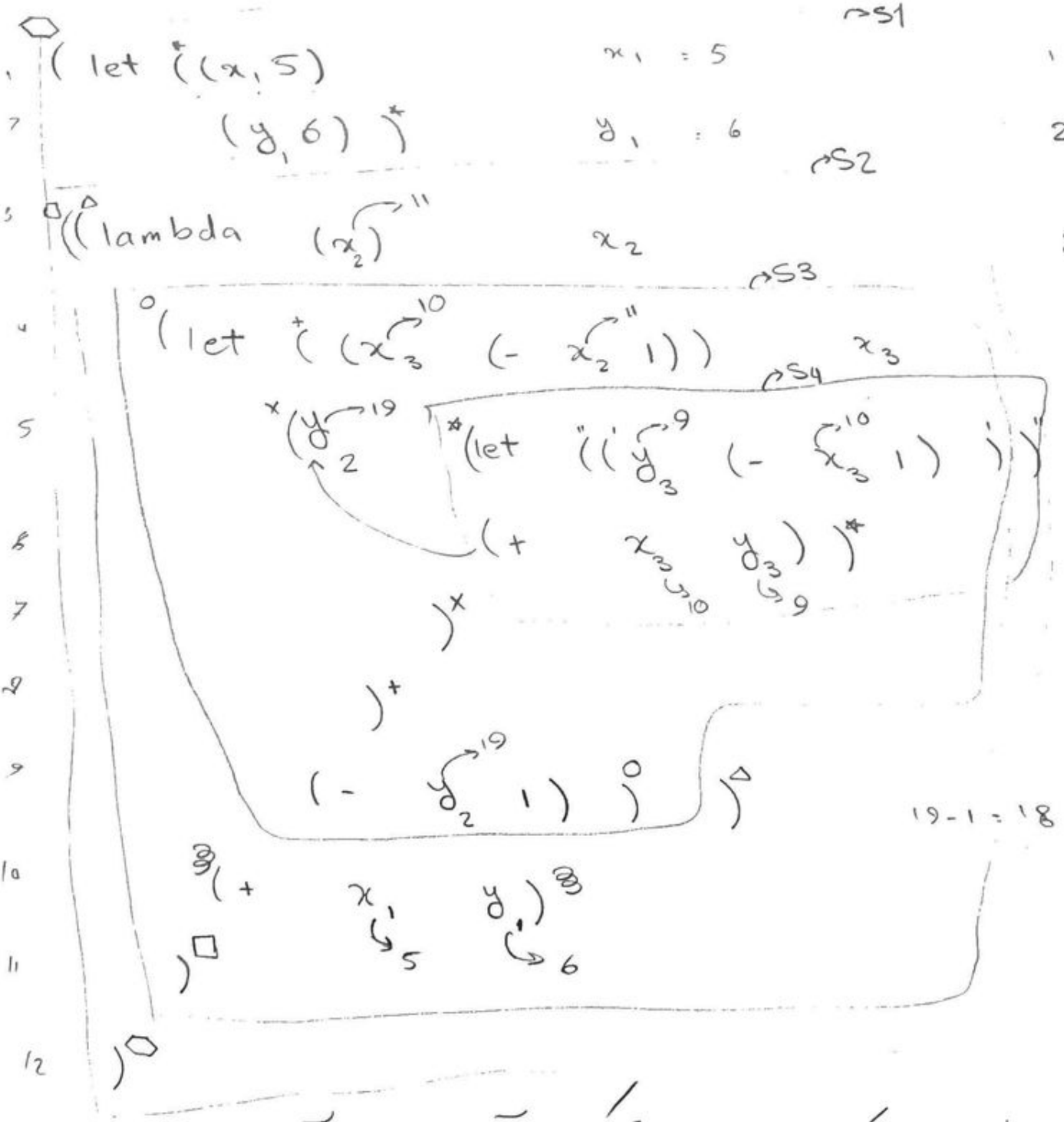

عبارت درون پرانتز ⬡ ، یک let-exp است که خروجی آن ، body آن یعنی همان عبارت درون پرانتز □ است . پرانتز □ به این صورت است : (ورودی تابع) .

عبارت درون پرانتز △ تابعی از است . عبارت درون ⬢ هم ورودی آن است که چون خارج از بدنه تابع lambda یا همان پرانتز △ است ، x و y آن از عبارت let بیرونی می آید . یعنی $x = x_1 = 5$ و $y = y_1 = 6$ ، (+ x y) = 11 . (ورودی تابع)

ادامه‌ی سؤال یک

حالا درون تابع لامبدا (Δ) ، ورودی x_2 است (که برای ما وقتی تابع کال می‌شود ۱۱ است) و خروجی تابع ، خروجی let-exp درون آن است (پرانتز ○) که خروجی آن هم همان خروجی body آن یعنی (- y 1) است. حال باید y را بدست آوریم.

در پرانتز ○ ، یک x جدید برابر با $x_{\text{ورودی}} - 1$ می‌شود که برای ما می‌شود ۱۱ - ۱ = ۱۰. سپس یک y تعریف می‌کند که خودش خروجی عبارت let درونی (پرانتز ☆) است که این صورت است:

☆(let
((y_3 (- x_3 1)))
☆((+ x_3 y_3))

که این، مقدار y خارج از پرانتز ☆ و داخل پرانتز x می‌شود.

این داریم $x_3 = x_{\text{ورودی}} - 1 = 10$. پس y_3 می‌شود ۹ = ۱ - ۱۰. پس y_2 می‌شود

$= x_3 + y_3 = 10 + 9 = 19$.

حال خروجی تابع (- y_2 1) است که می‌شود ۱۸ = ۱ - ۱۹.

پس خروجی کل عبارت ما می‌شود ۱۸. (یعنی خروجی عبارت پرانتز □ که همان خروجی body پرانتز ○ است.

$(value\text{-}of\ (const\text{-}exp\ 2)\ \rho\ \sigma) = (2\ \sigma)$

$(value\text{-}of\ exp_1\ \rho\ \sigma_0) = (val_1, \sigma_1)$ ② 다음

$(value\text{-}of\ exp_2\ \rho\ \sigma_1) = (val_2, \sigma_2)$

$(value\text{-}of\ (sd\text{-}exp\text{-}exp_1\ exp_2)\ \rho\ \sigma_0)$

$= (\ \lceil \lfloor val_1 \rfloor * \lfloor val_1 \rfloor + \lfloor val_2 \rfloor * \lfloor val_2 \rfloor$

$- 2 * \lfloor val_1 \rfloor * \lfloor val_2 \rfloor \rceil\ )$

سوال 3) - این شکل پیاده سازی می کنیم که می گوییم while یک if است، اما بدون else

باید اگر شرط درست بود، جلو از اجرای عبارت مربوط به true بودن آن، دوباره شرط را چک کند و اگر درست بود... و بنابراین باید خودش را صدا بزند.

```
(while-exp ( exp1 exp2)
  (value-of/k exp1 env
    (while-test-cont exp2 env cont ) ) )

(apply-cont (while-test-cont exp2 env cont) val)
  = (if (expval->bool val)
      (value-of/k exp2 env
        (whl-jump-cont val cont))
      (no-op (هیچ کاری نکند) یا null)
    )

(apply-cont (whl-jump-cont val cont) exp2)
  = (value-of/k val env
      (while-test-cont exp2 env cont))
```

که برگردد اول while.