#### Python:

$$x = 3$$
  
return x + (10 \* x)

Racket:

(let ((x 3)) (+ x (\* 10 x)))

#### x += 5 and x++ are

#### syntactic abstractions

#### of x = y + z

## convenient but not necessary

#### In Racket,

## cond

#### is a syntactic abstraction of

if

#### In Racket,

### let

#### is a syntactic abstraction of

## applying a function

[list comprehensions in Python]

.

[Local variables bind a value to a name.]

.

The syntax of Racket's let expression:

This:

.

٠

is equivalent to:

```
(let ((op (first exp))
      (arg1 (second exp))
      (arg2 (third exp)))
   (list arg1 op arg2))
```

is equivalent to:

((lambda (op arg1 arg2)
 (list arg1 op arg2))
 (first exp) (second exp) (third exp))

```
(let ((op (first exp))
      (arg1 (second exp))
      (arg2 (third exp)))
   (list arg1 op arg2))
```

is equivalent to:

((lambda (op arg1 arg2)
 (list arg1 op arg2))
 (first exp) (second exp) (third exp))

# (let ((op (first exp)) (arg1 (second exp)) (arg2 (third exp))) (list arg1 op arg2))

is equivalent to:

((lambda (op arg1 arg2)
 (list arg1 op arg2))
(first exp) (second exp) (third exp))

```
(let ((op (first exp))
        (arg1 (second exp))
        (arg2 (third exp)))
        (list arg1 op arg2))
```

is equivalent to:

```
((lambda (op arg1 arg2)
    (list arg1 op arg2))
  (first exp) (second exp) (third exp))
```

## translational semantics

[ images showing compilation with and without preprocess ]

.