CSE 231: Assignments and Blocks

Ranjit Jhala *April 16, 2026*

Assignments and Blocks

Next, we add **imperative** features, starting with

- **assignments** that let us *update* values of variables,
- **blocks** that let us *sequence* multiple expressions.

These are a stepping stone towards then adding **loops**.

Concrete Syntax

```
<expr> := ...
        | (set! <ident> <expr>)   -- update variable
        | (block <expr>+)         -- sequence exprs
```

QUIZ: Abstract Syntax

First, lets fill in the cases for `set!` and `block`

```
enum Expr {
  // ...


}
```

QUIZ: Semantics of `set!` and `block`

Program *Result*

```
(let (x 5)
  (set! x 10)
)
```

```
(let (x 10)
  (let (y (set! x (+ x 5)))
    (+ x y))
```

```
(let (x 5)
  (block
    (set! x (+ x 100))
    x
  )
)
```

QUIZ: Evaluator

Lets fill in the cases for the `eval` function.

How to
- *Update* a variable?
- *Sequence* expressions?

```
fn eval(expr: &Expr, env: &Env) -> i64 {
  match expr {
    // ...
    Expr::Set(x, e) => {

      ____________________________________________

      ____________________________________________

      ____________________________________________
    }
    Expr::Block() => {

      ____________________________________________

      ____________________________________________

      ____________________________________________

      ____________________________________________

      ____________________________________________
    }
  }
}
```

QUIZ: Assembly for `set!` and `block`

Complete the assembly code for

Program

```
(let (x 10)
  (let (y (set! x (+ x 1)))
    x)
```

Assembly

```
mov rax, 20
mov [rbp - 8.1], rax
mov rax, [rbp - 8.1]
add rax, 2
_____________________
_____________________
_____________________
```

```
(let (x 10)
  (block
    (set! x (+ x 1))
    x
  )
)
```

```
mov rax, 20
mov [rbp - 8.1], rax
mov rax, [rbp - 8.1]
add rax, 2
_____________________
_____________________
_____________________
```

QUIZ: Strategy for `set!` and `block`

```
(set! x e)
```

```
(block
  e0
  e1
  e2
)
```

Compilation Code

Lets fill in the cases for `compile_expr` for `set!` and `block`

```
fn compile_expr(expr: &Expr, env: &Env) -> String {
  match expr {
    // other cases ...
    Expr::Set(x, e) => {









    }

    Expr::Block(es) => {









    }
}
```