```
let a = 5,
    b = 10:
let f = lam(x): x + a end:
f(1)
```

a: 5 b: 10

```
lam(x): if false: a
        else: x + 1
        end
    end
```

PROVABILITY

```
def f(i): lambda: i
map(f, range(0, 10))
```

TRUTH

```
for i in range(0, 10):
    l.append(lambda: i)
```

interp(e) $\to v_1$ ↓ e_1 interp(swap(e, x, y)) $\to v_2$ ↓ e_2

$$\frac{v_1 == v_2}{e_1 == e_2}$$

$s_1 == s_2$

identical(v_1, v_2)

lam(x, y): x / y end v-clos(mte, [list: "x", "y"] _)

lam(y, x): y / x end v-clos(mte, [list: "y", "x"], _)

swap(e, x, y)

rename(rename(e, x, nn1), y, nn2) = e'

rn(rename(e', nn1, y), nn2, x)

lam(x): x end

idtype("x") → idtype("x")

a → a

let f = lam(x): x end:

f(1)

f("a")

idtype("f") = idtype("x") → idtype("x")

(1) (2)

idtype("f") =

Num → type(f(1))

(1) (2)

(1) Num = idtype("x")

(2) type(f(1)) = idtype("x")

Str = idtype("x")

Str = Num

```
fun ident<a>(x :: a) -> a:
  x
end
```

```
ident<Num>(1)
ident<Str>("a")

(lam(x :: Num) -> Num: x end)(1)
(lam(x :: Str) -> Str: x end)("a")
```

f<T>

```
fun map<a,b>(f :: (a -> b), l :: List<a>) -> List<b>:
  cases
    | empty => empty
    | link(fst, r) =>
      link(f(fst), map<a,b>(f, r))
  end
end
```

Name, List<Types> → Expr

"map" Num, Str → map
  ⋮
  map<Num,Str>(...)
end

```
fun ident<a>(x :: a) -> a:
  x
end
```

just x?

any value w/ x's type?

ident<Num>(2)

ident<Str>("a")

(lam(x :: Str) -> Str: x end)("a")

type-check :: Exps, TypeEnv, Types

data Type:
  | tid(x :: String)
end