i116: Basic of Programming

9. Assignment calculator: interpreter

Kazuhiro Ogata, Canh Minh Do

Roadmap

- Interpreter for assignment calculator

Interpreter for assignment calculator

- Because expressions may have variables, the interpreter needs to use an environment, which is implemented with a dictionary.

Interpreter for assignment calculator

```
class ExpParseTree(object):
    …
    def interpret(self,env):
        pass
```

```
class NumParseTree(ExpParseTree):
    num = 0
    …
    def interpret(self,env):
        return self.num
```

The method interpret(…) takes one more argument *env*.

This is all we need to revise the existing parse tree classes for arithmetic expressions so that the method interpret(…) can be used for the assignment calculator.

Interpreter for assignment calculator

```
class UmiParseTree(ExpParseTree):
    exp = ExpParseTree()
    …
    def interpret(self,env):
        return -1 * self.exp.interpret(env)
```

```
class AddParseTree(ExpParseTree):
    exp1 = ExpParseTree()
    exp2 = ExpParseTree()
    …
    def interpret(self,env):
        return self.exp1.interpret(env) + self.exp2.interpret(env)
```

Interpreter for assignment calculator

```
class SubParseTree(ExpParseTree):
    exp1 = ExpParseTree()
    exp2 = ExpParseTree()
    …
    def interpret(self,env):
        return self.exp1.interpret(env) - self.exp2.interpret(env)
```

```
class MulParseTree(ExpParseTree):
    exp1 = ExpParseTree()
    exp2 = ExpParseTree()
    …
    def interpret(self,env):
        return self.exp1.interpret(env) * self.exp2.interpret(env)
```

Interpreter for assignment calculator

```
class QuoParseTree(ExpParseTree):
    exp1 = ExpParseTree()
    exp2 = ExpParseTree()
    …
    def interpret(self,env):
        if self.exp2.interpret(env) == 0:
            raise DivisionByZero('division by zero')
        else:
            return self.exp1.interpret(env) // self.exp2.interpret(env)
```

Interpreter for assignment calculator

```
class RemParseTree(ExpParseTree):
    exp1 = ExpParseTree()
    exp2 = ExpParseTree()
    …
    def interpret(self,env):
        if self.exp2.interpret(env) == 0:
            raise DivisionByZero('division by zero')
        else:
            return self.exp1.interpret(env) % self.exp2.interpret(env)
```

Interpreter for assignment calculator

```
class LTParseTree(ExpParseTree):
    exp1 = ExpParseTree()
    exp2 = ExpParseTree()
    …
 def interpret(self,env):
        if self.exp1.interpret(env) < self.exp2.interpret(env):
            return 1
        else:
            return 0
```

Interpreter for assignment calculator

```
class GTParseTree(ExpParseTree):
    exp1 = ExpParseTree()
    exp2 = ExpParseTree()
    …
 def interpret(self,env):
        if self.exp1.interpret(env) > self.exp2.interpret(env):
            return 1
        else:
            return 0
```

Interpreter for assignment calculator

```
class EQParseTree(ExpParseTree):
    exp1 = ExpParseTree()
    exp2 = ExpParseTree()
    …
 def interpret(self,env):
        if self.exp1.interpret(env) == self.exp2.interpretenv ():
            return 1
        else:
            return 0
```

Interpreter for assignment calculator

```
class NEQParseTree(ExpParseTree):
    exp1 = ExpParseTree()
    exp2 = ExpParseTree()
    …
 def interpret(self,env):
        if self.exp1.interpret(env) == self.exp2.interpret(env):
            return 0
        else:
            return 1
```

Interpreter for assignment calculator

```
class AndParseTree(ExpParseTree):
    exp1 = ExpParseTree()
    exp2 = ExpParseTree()
    …
 def interpret(self,env):
        if self.exp1.interpret(env) == 0 or self.exp2.interpret(env) == 0:
            return 0
        else:
            return 1
```

Interpreter for arithmetic expressions

```
class OrParseTree(ExpParseTree):
    exp1 = ExpParseTree()
    exp2 = ExpParseTree()
    …
 def interpret(self,env):
        if self.exp1.interpret(env) == 0 and self.exp2.interpret(env) == 0:
            return 0
        else:
            return 1
```

Interpreter for assignment calculator

```
class VarParseTree(ExpParseTree):
    name = ''
    ...
    def interpret(self,env):
        try:
            return env[self.name]
        except KeyError:
            raise UndefinedVar('undefined variable')
```

Var: 'x' .interpret(*env*)

env['x'] is returned.

If 'x' is not registered in *env*, then an exception called UndefinedVar is raised.

Interpreter for assignment calculator

`Var: 'x'` `.interpret({..., 'x':2, ...})` returns 2.

`Var: 'x'` `.interpret({..., 'x':0, ...})` returns 0.

`Var: 'x'` `.interpret({...})` raises an exception UndefinedVar.

Interpreter for assignment calculator

We should define method interpreter(…) for each class of statements.

```
class StmParseTree(object):
    …
    def interpret(self,env):
        pass
```

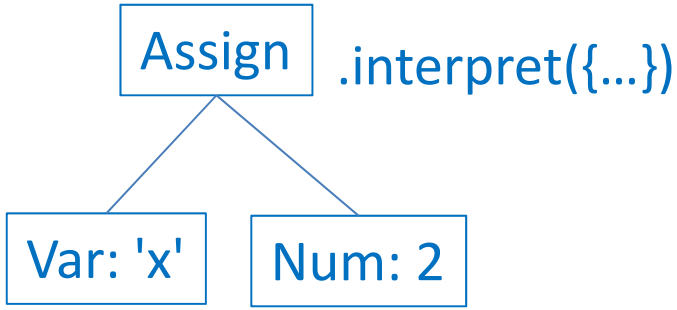
What should be returned by such a method?

Interpreter for assignment calculator

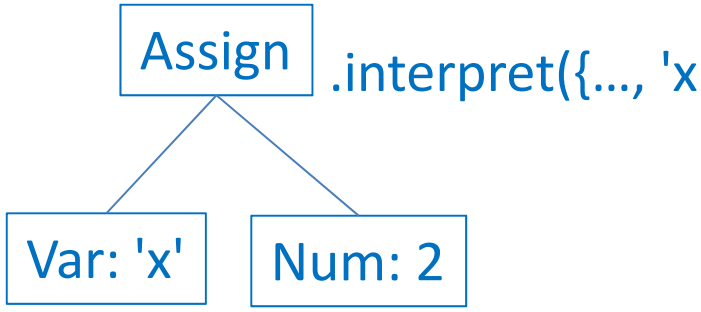
```
class AssignParseTree(StmParseTree):
    var = ExpParseTree()
    exp = ExpParseTree()
    ...
    def interpret(self,env):
        env[self.var.name] = self.exp.interpret(env)
        return env
```

1. The expression *exp* is calculated by using the environment *env*, which may cause an exception.
2. The value associated with the variable *var* is updated by the result obtained at step 1.
3. The updated environment *env* is returned.

Interpreter for assignment calculator

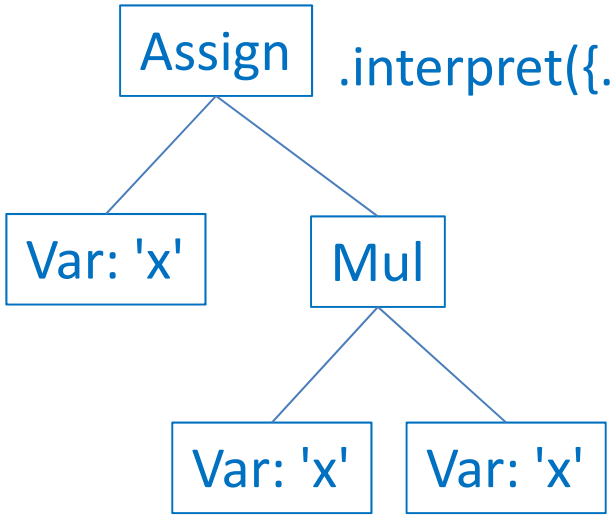


returns {..., 'x':2, ...}

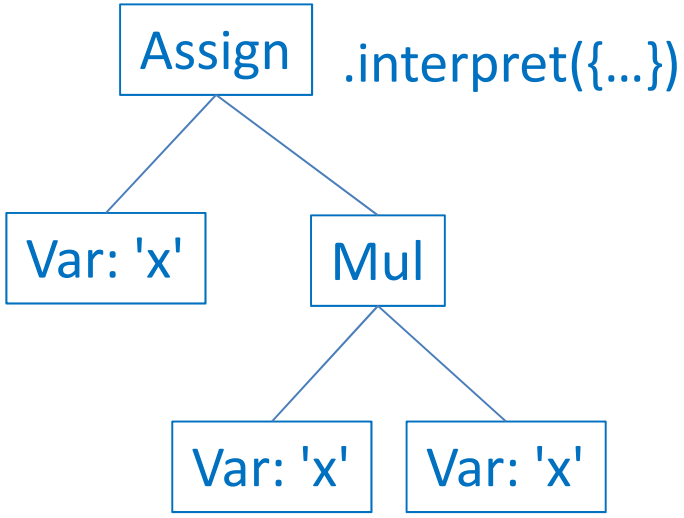


returns {..., 'x':2, ...}

Interpreter for assignment calculator



returns {..., 'x':4, ...}



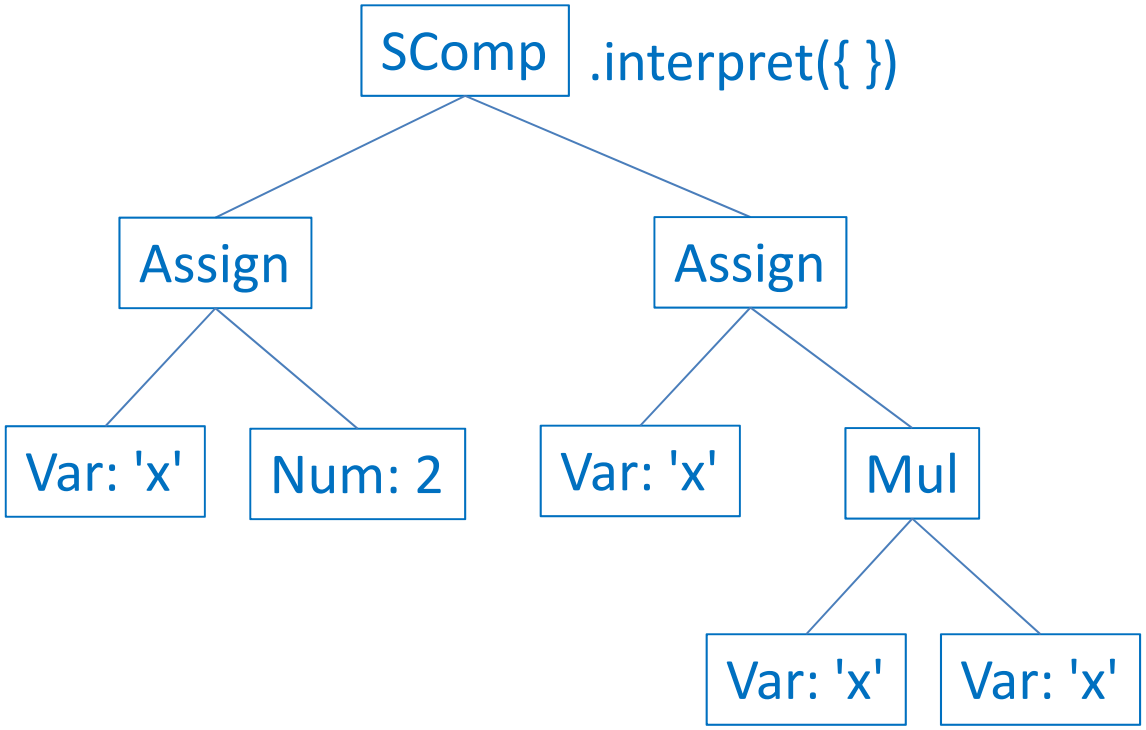
raises an exception UndefinedVar.

Interpreter for assignment calculator

```
class SCompParseTree(StmParseTree):
    stm1 = StmParseTree()
    stm2 = StmParseTree()
    ...
    def interpret(self,env):
        return self.stm2.interpret(self.stm1.interpret(env))
```

1. The statement (program) *stm1* is interpreted by using the environment *env*, which may cause an exception.
2. The statement (program) *stm2* is then interpreted by using the updated environment at step1, which may cause an exception.
3. The updated environment at step 2 is returned.

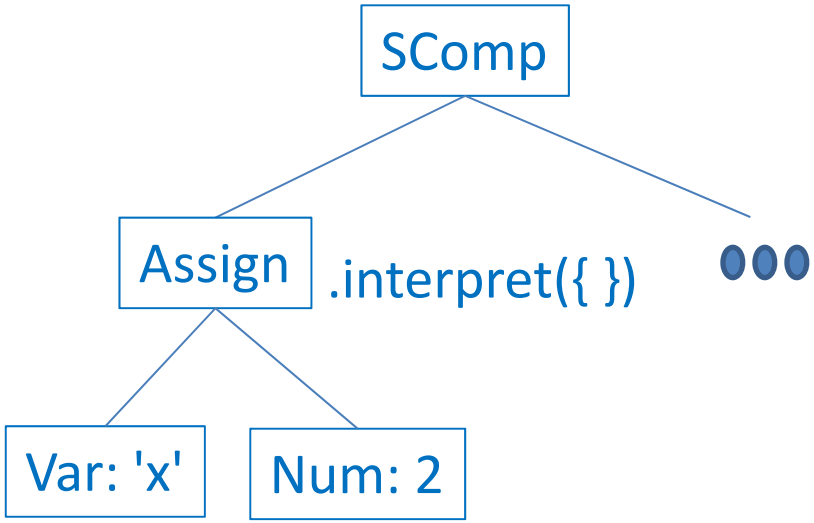
Interpreter for assignment calculator



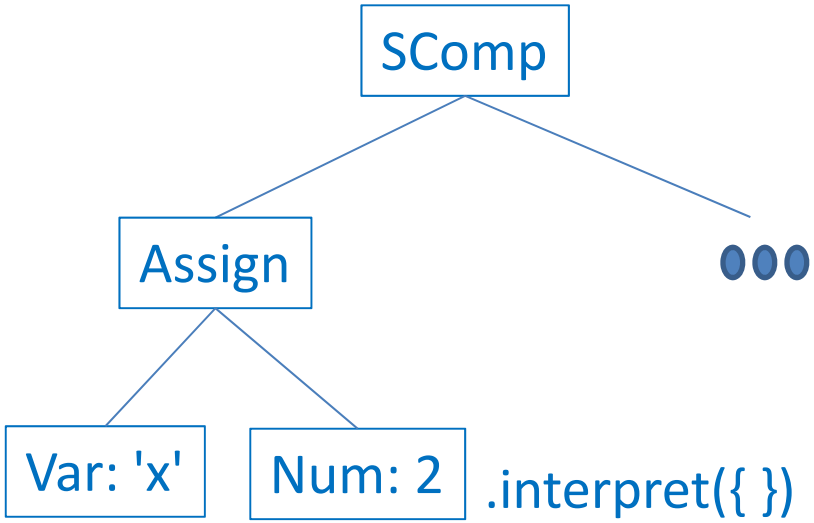
x := 2 ;

x := *x* * *x* ;

Interpreter for assignment calculator



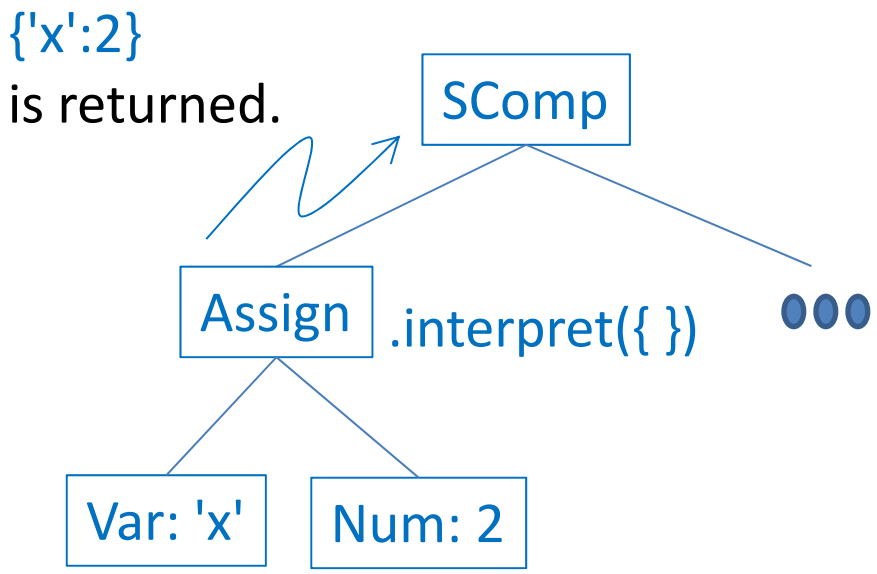
Interpreter for assignment calculator



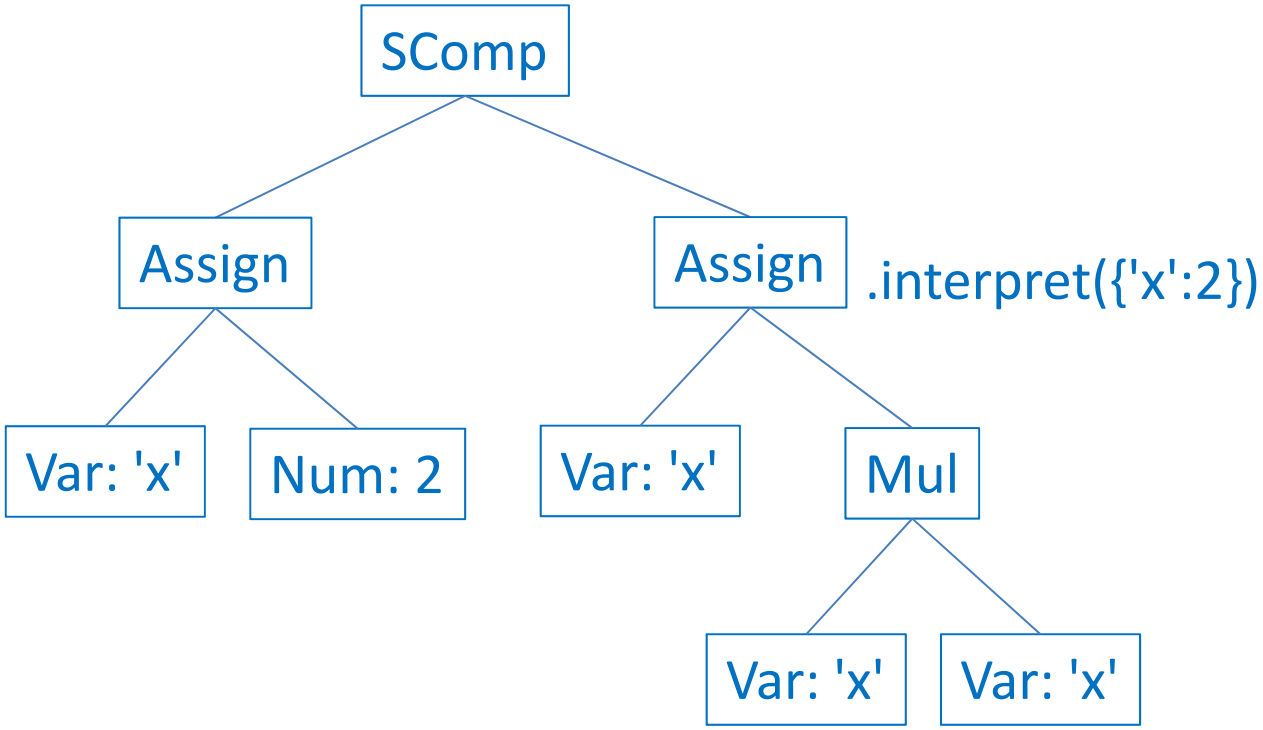
2 is returned.

'x', together with 2, is registered in { },
generating {'x':2}.

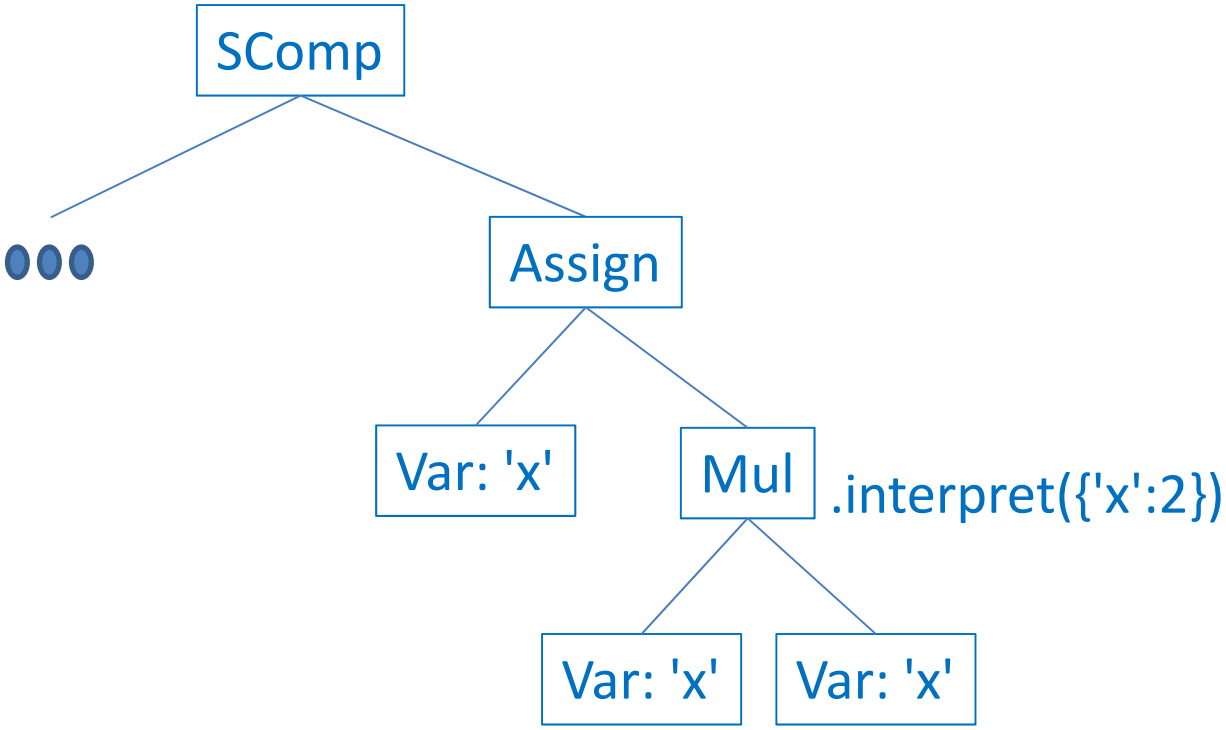
Interpreter for assignment calculator



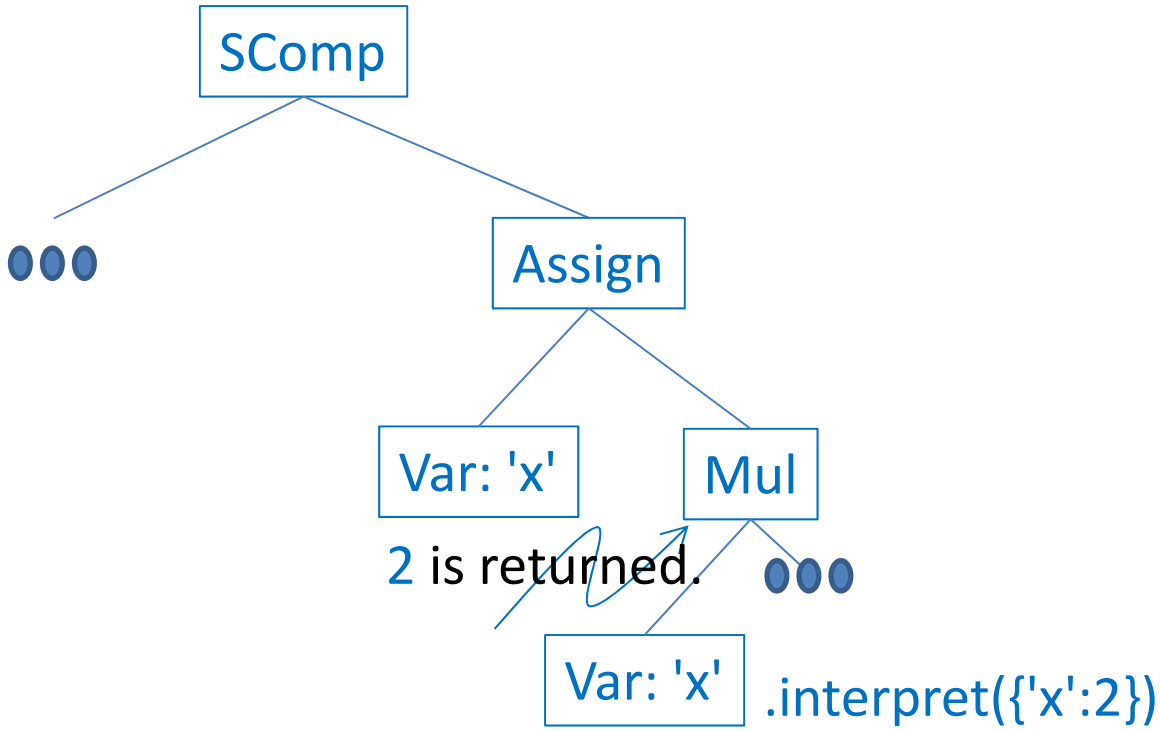
Interpreter for assignment calculator



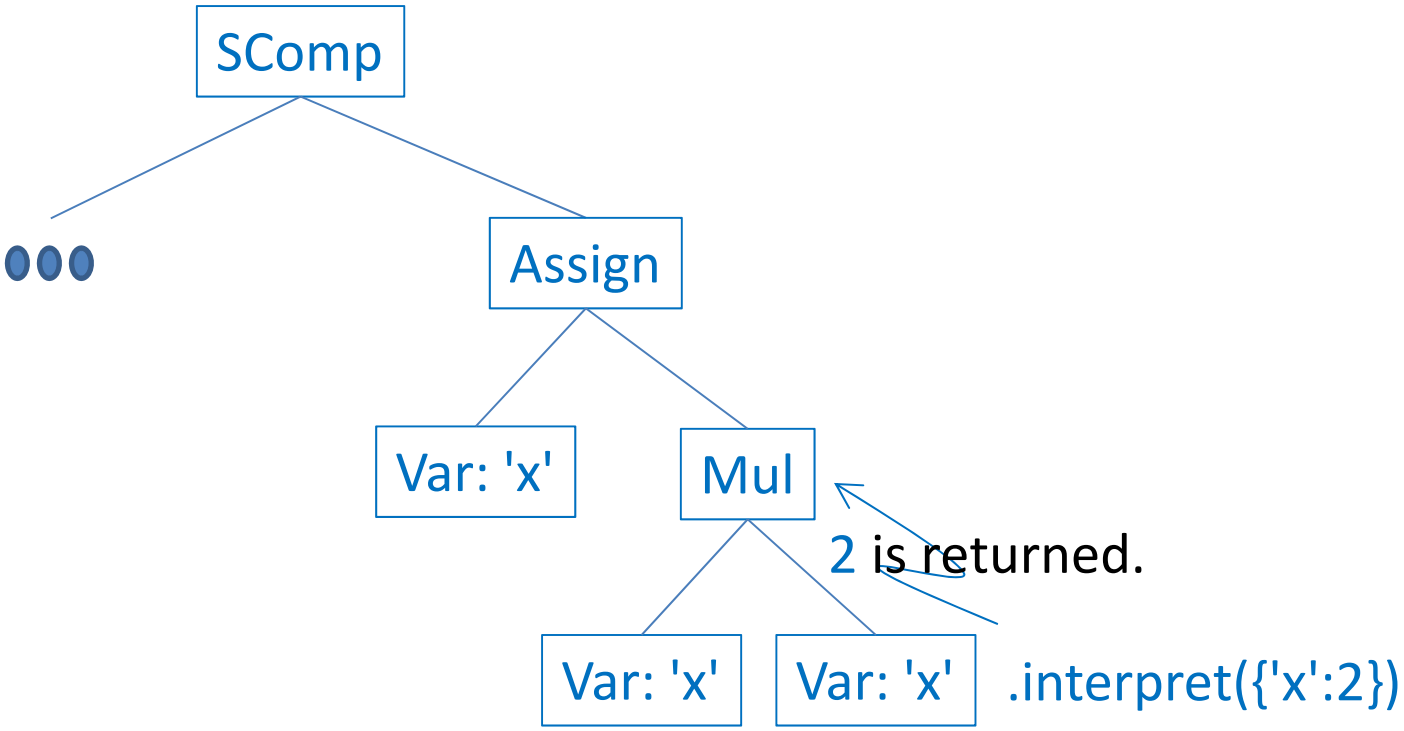
Interpreter for assignment calculator



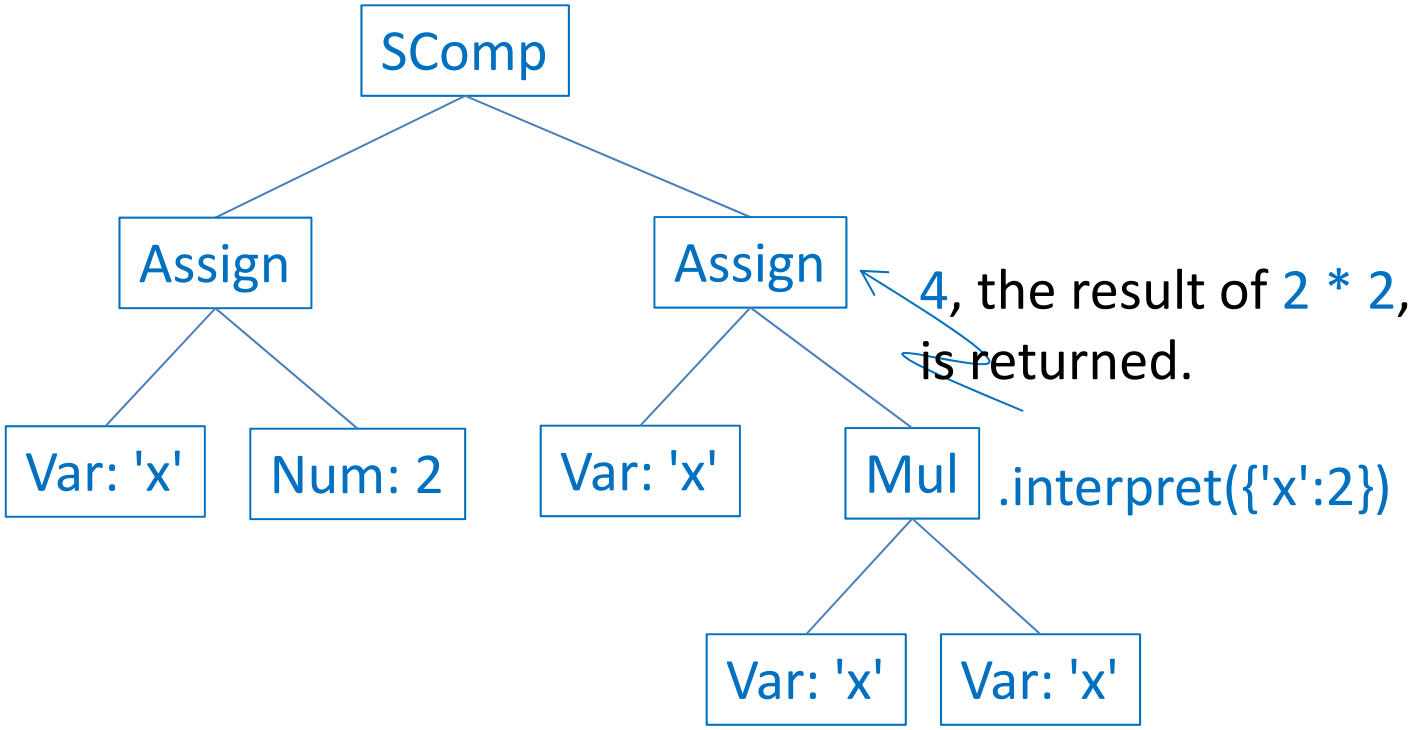
Interpreter for assignment calculator



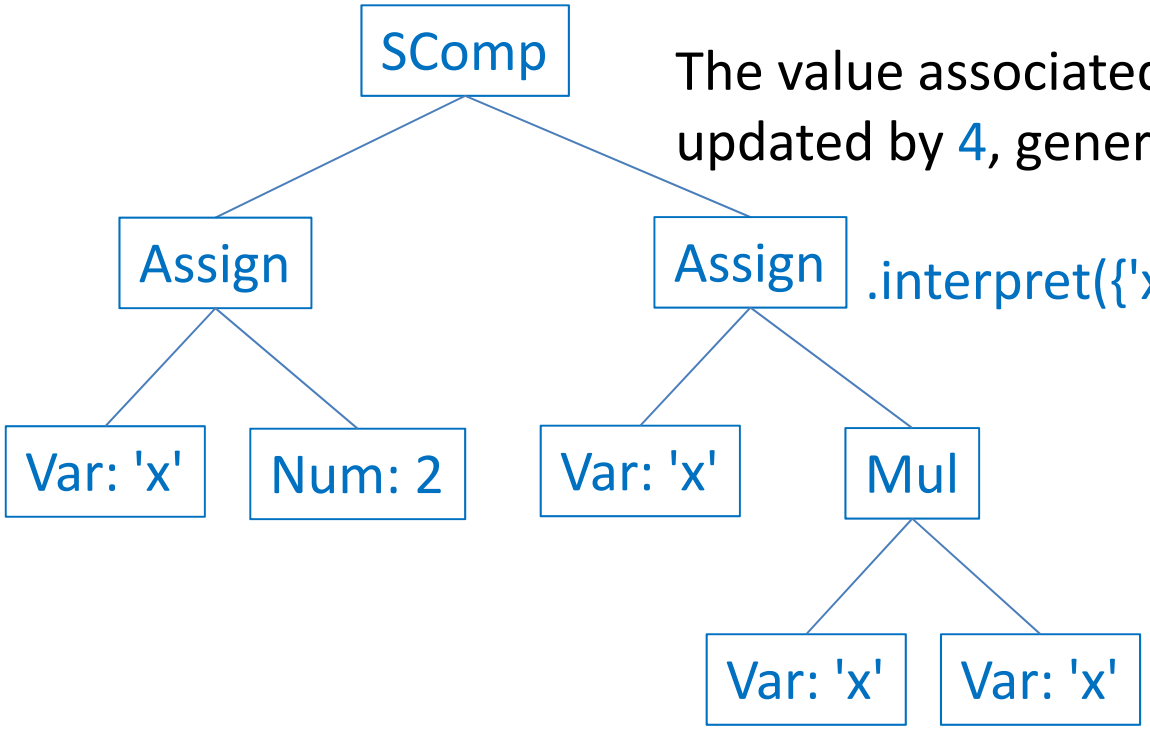
Interpreter for assignment calculator



Interpreter for assignment calculator



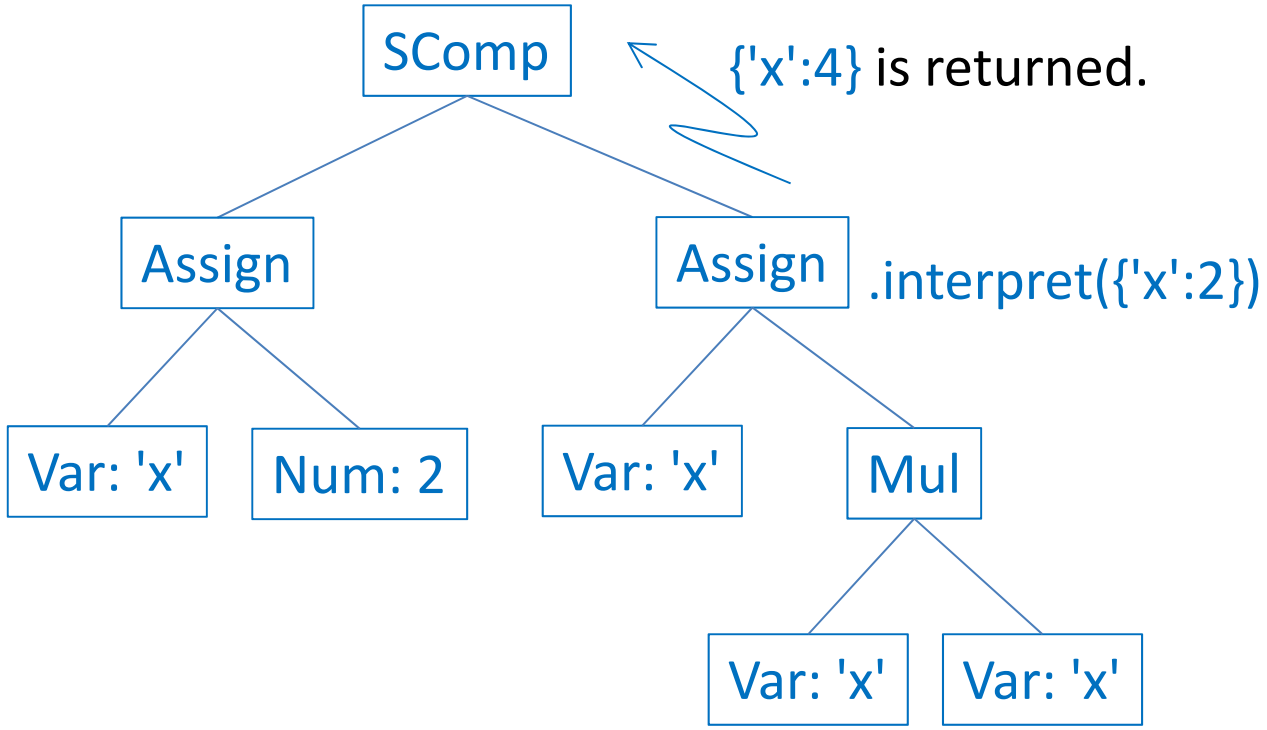
Interpreter for assignment calculator



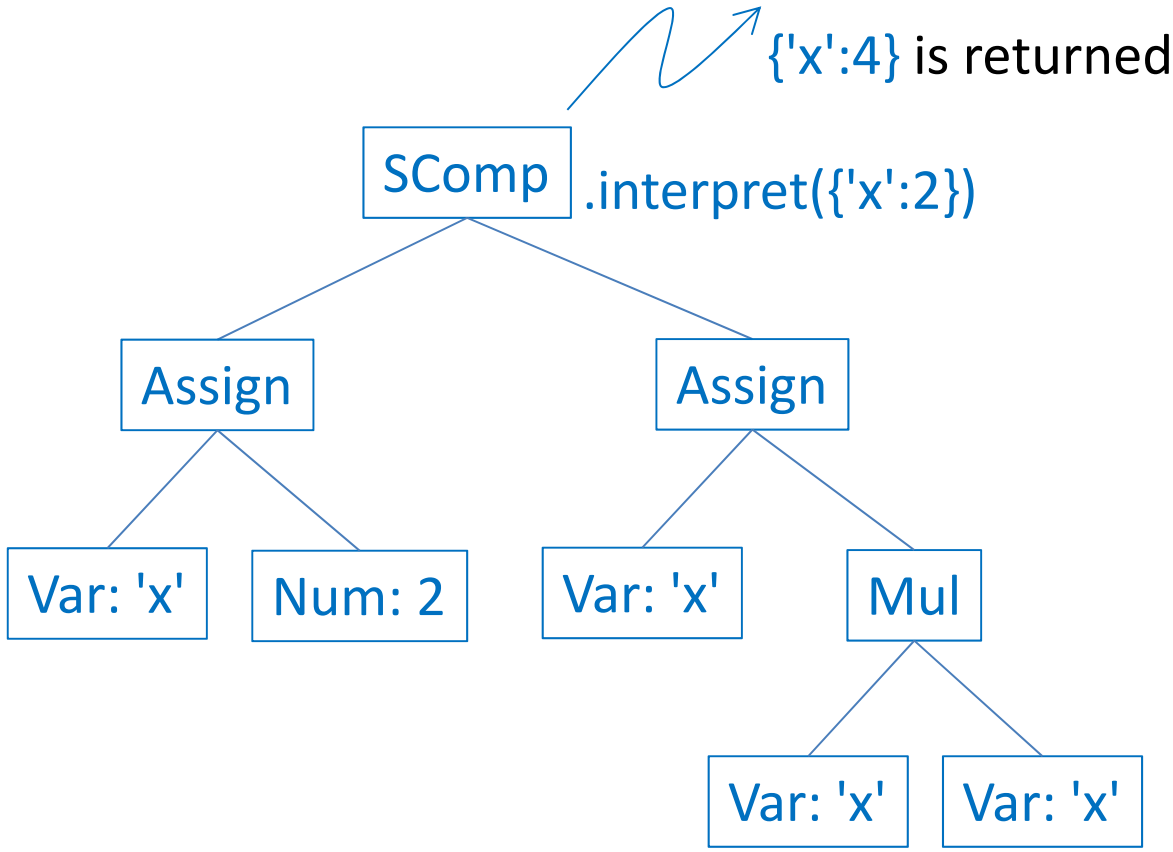
The value associated with 'x' is updated by 4, generating {'x':4}.

.interpret({'x':2})

Interpreter for assignment calculator



Interpreter for assignment calculator



Interpreter for assignment calculator

```
varX = VarParseTree('x')
two = NumParseTree(2)
as1 = AssignParseTree(varX,two)
e1 = MulParseTree(varX,varX)
as2 = AssignParseTree(varX,e1)
pgm1 = SCompParseTree(as1,as2)
print(pgm1)
print(pgm1.interpret({}))
```

$x := 2 ;$

$x := x * x ;$

Interpreter for assignment calculator

```
varX = VarParseTree('x')
varY = VarParseTree('y')
two = NumParseTree(2)
as1 = AssignParseTree(varX,two)
as2 = AssignParseTree(varY,two)
e1 = MulParseTree(varX,varX)
e2 = MulParseTree(varY,varY)
as3 = AssignParseTree(varX,e1)
as4 = AssignParseTree(varY,e2)
e3 = MulParseTree(varX,varY)
as5 = AssignParseTree(varX,e3)
pgm1 = SCompParseTree(as1,as3)
pgm2 = SCompParseTree(pgm1,as3)
pgm3 = SCompParseTree(pgm2,as3)
pgm4 = SCompParseTree(pgm3,as2)
pgm5 = SCompParseTree(pgm4,as4)
pgm6 = SCompParseTree(pgm5,as5)
print(pgm6)
print(pgm6.interpret({}))
```

$$x := 2 ;$$
$$x := x * x ;$$
$$x := x * x ;$$
$$x := x * x ;$$
$$y := 2 ;$$
$$y := y * y ;$$
$$x := x * y ;$$