

# CSC111 Lecture 12: Abstract Syntax Trees, Continued

Hisbaan Noorani

February 24, 2021

## Contents

1	Exercise 1: If statements	1
2	Exercise 2: For loops (over ranges)	3
3	Additional exercises	5

**Note:** if you are working in PyCharm, you can use the starter file `lecture12.py` on Quercus for this exercise.

## 1 Exercise 1: If statements

Now that you've seen the `Module` class in lecture, you are ready to implement *compound statements* whose bodies consist of multiple statements. In this exercise, we'll cover if statements.

First, review this new class to represent if statements.

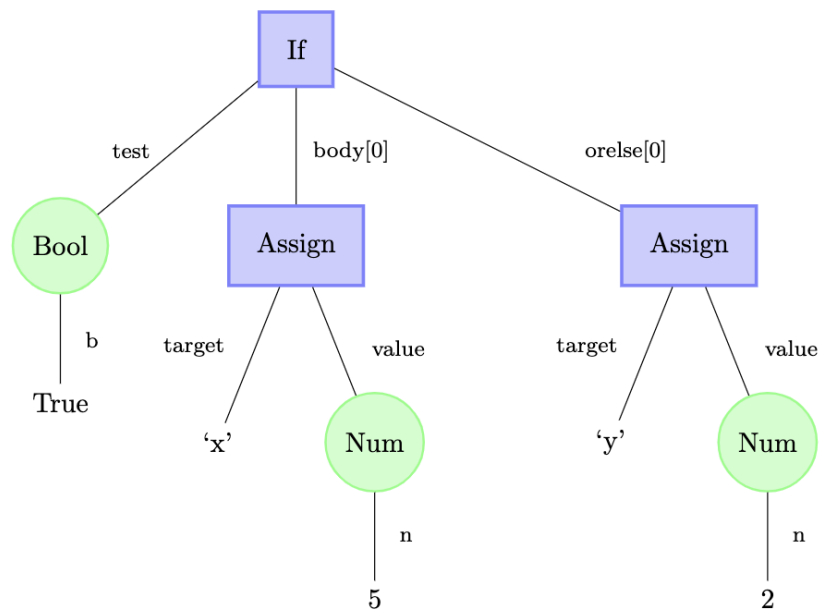
```
1 class If(Statement):
2     """An if statement.
3
4     This is a statement of the form:
5
6         if <test>:
7             <body>
8         else:
9             <orelse>
10
11     Instance Attributes:
12     - test: The condition expression of this if statement.
13     - body: A sequence of statements to evaluate if the condition is True.
14     - orelse: A sequence of statements to evaluate if the condition is False.
15     """
16     test: Expr
17     body: list[Statement]
18     orelse: list[Statement]
19
20     def __init__(self, test: Expr, body: list[Statement],
21                   orelse: list[Statement]) -> None:
```

```

22     self.test = test
23     self.body = body
24     self.orelse = orelse

```

1. Write an AST expression using If (and other AST types) to represent the following diagram.



```

1 >>> If(Bool(b),
2 ...     [Assign('x', Num(1))],
3 ...     [Assign('y', Num(0))])

```

2. Write an AST expression using If (and other AST types) to represent the following Python code.

```

1 if x < 100:
2     print(x)
3 else:
4     y = x + 2
5     x = 1

```

```

1 >>> If(Compare(Name('x'), [('<', Num(100))]),
2 ...     [Print(Name('x'))],
3 ...     [Assign('y', BinOp(Name('x'), '+', Num(2))), Assign('x', Num(1))])
4 ... )

```

3. In Python, the else branch is optional. How would we represent an if statement with no else branch using our If class?

We can use an empty list for the orelse branch.

4. Implement the If.evaluate method. Note that you can use if statements in your implementation (similar to how we used the + and \* operators to implement BinOp).

```
1 class If:
2     def evaluate(self, env: Dict[str, Any]) -> None:
3         """Evaluate this statement.
4
5         Preconditions:
6             - self.test evaluates to a boolean
7
8         >>> stmt = If(Bool(True),
9                     [Assign('x', Num(1))],
10                    [Assign('y', Num(0))])
11                    ...
12        >>> env = {}
13        >>> stmt.evaluate(env)
14        >>> env
15        {'x': 1}
16        """
17        test_val = self.test.evaluate(env)
18
19        if test_val:
20            for statement in self.body:
21                statement.evaluate(env)
22        else:
23            for statement in self.orelse:
24                statement.evaluate(env)
```

## 2 Exercise 2: For loops (over ranges)

Please take a moment to review the ForRange class we introduced in lecture.

```
1 class ForRange(Statement):
2     """A for loop that loops over a range of numbers.
3
4     for <target> in range(<start>, <stop>):
5         <body>
6
7     Instance Attributes:
8         - target: The loop variable.
9         - start: The start for the range (inclusive).
10        - stop: The end of the range (this is *exclusive*, so <stop> is not included
11              in the loop).
```

```

12     - body: The statements to execute in the loop body.
13     """
14     target: str
15     start: Expr
16     stop: Expr
17     body: list[Statement]
18
19     def __init__(self, target: str, start: Expr, stop: Expr,
20                 body: list[Statement]) -> None:
21         """Initialize a new ForRange node."""
22         self.target = target
23         self.start = start
24         self.stop = stop
25         self.body = body

```

1. Write the Python code that corresponds to the following AST expression.

```

1 >>> ForRange('x',
2 ...         Num(1),
3 ...         BinOp(Num(2), '+', Num(3)),
4 ...         [Print(Name('x'))])

```

```

1 for x in range(1, 2 + 3):
2     print(x)

```

2. Write the Module AST expression that corresponds to the following Python code:

```

1 sum_so_far = 0
2
3 for n in range(1, 10):
4     sum_so_far = sum_so_far + n
5
6 print(sum_so_far)

```

*Hint:* Your Module body should have three elements (an Assign, a ForRange, and a Print).

```

1 >>> Module([
2 ...     Assign('sum_so_far', Num(0)),
3 ...     ForRange('n', Num(1), Num(10),
4 ...             [Assign('sum_so_far', BinOp(Name('sum_so_far'), '+', Name('n')))]),
5 ...     Print(Name('x'))
6 ... ])

```

3. Finally, implement the `ForRange.evaluate` method. Think carefully about how you make the loop variable accessible when you evaluate the statements in the loop body—you'll need to update the variable environment. Note that you can use `for` loops in your implementation!

```

1 class ForRange:
2     def evaluate(self, env: Dict[str, Any]) -> None:
3         """Evaluate this statement.
4
5         Preconditions:
6             - self.start and self.stop evaluate to integers
7
8         >>> statement = ForRange('x', Num(1), BinOp(Num(2), '+', Num(3)),
9           ...               [Print(Name('x'))])
10        >>> statement.evaluate({})
11        1
12        2
13        3
14        4
15        """
16        start_val = self.start.evaluate(env)
17        stop_val = self.stop.evaluate(env)
18
19        for i in range(start_val, stop_val):
20            # Need to assign self.target to have value i
21            # NOTE: only use one of the two versions
22
23            # Version 1:
24            env[self.target] = i
25
26            # Version 2:
27            Assign(self.target, Num(i)).evaluate(env)
28
29        for statement in self.body:
30            statement.evaluate(env)

```

### 3 Additional exercises

1. Let's add some *type-checking* to the methods we implemented in this exercise.
  - (a) Modify the `If.evaluate` method to raise a `TypeError` if its condition doesn't evaluate to a boolean. (This is technically stricter than real Python.)
  - (b) Modify the `ForRange.evaluate` method to raise a `TypeError` if its start or stop expressions don't evaluate to an integer.
2. Modify the `If` class to support an arbitrary number of `elif` branches. You may add additional instance attributes.
3. Design and implement a new AST `Statement` subclass to represent a while loop.
4. Design and implement a new AST `Expr` subclass to represent a list comprehension over a range of numbers.