CSC 148: Introduction to Computer Science

Week 10

Tree applications: Abstract Syntax Trees

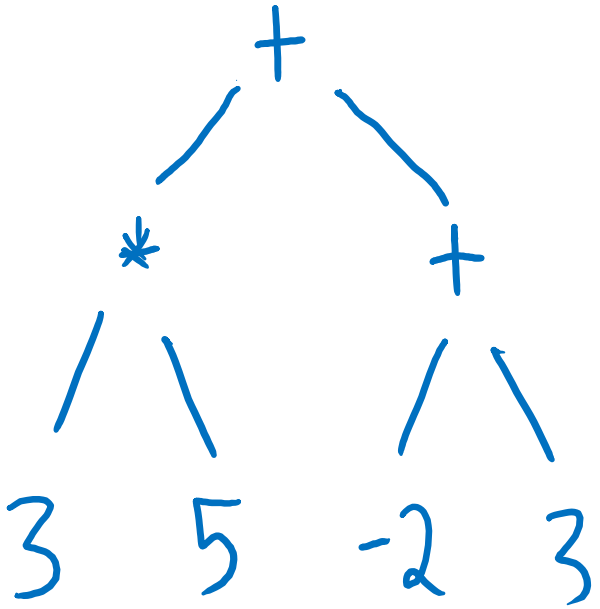
Applied example: Modeling Python code

University of Toronto Mississauga,

Department of Mathematical and Computational Sciences

Programs as Data

- An expression tree is a structured way of modeling ("simple") Python code.



Why?

By modeling programs as data, we can start thinking about writing programs that *operate on other programs*.

- A Python interpreter is a program that runs Python code.
- A Java compiler is a program that turns Java code into a sequence of "primitive instructions"
- PyCharm and PythonTA are programs that analyse Python code and report potential problems.

From Expressions to Statements

- An *expression* is a unit of code that, when evaluated, produces a single value.

- A *statement* is a command that often has *side effects*.
  - Evaluating a statement may cause an object to be *printed*
  - … or a value to be *assigned to a variable*
  - … or for control to be directed to a particular block of code.

- Every expression is a statement but not vice-versa!

Examples of Statements

```
1) x = 5
```

```
2) return 10
```

```
3) break
```

```
4) if x > 5:
       y = 10
   else:
       y = 15
```

```
5) for i in range(10):
       print(i)
```

Variable Bindings

How do we model variables in an abstract syntax tree?

University of Toronto Mississauga,

Department of Mathematical and Computational Sciences

A Variable Name: the `Name` class

```
class Name(Expr):
    """A variable name.

    === Attributes ===
    id: The variable name.
    """
    id: str
```

- e.g., Name('x'), Name('y'), Name('student_name'), etc.
- But how do we evaluate it?

Mapping Variables to Values

- A variable environment is a map from variable names to values.

  We'll implement this using a Python `dict`:

- `{'x': 1, 'y': True}`

- `Name('x').evaluate({'x': 10})`

Passing in the Environment

```
class Statement:
    def evaluate(self, env: dict[str, Any]) -> Any:
        """Return the *value* of this expression,
        in the given environment.
        """
```

Example

```
>>> expr = Name('x')
>>> expr.evaluate({'x': 10})
10
```

Creating Bindings: the `Assign` class

```
class Assign(Statement):
    """An assignment statement. <target> = <value>

    === Attributes ===
    target: the variable name
    value: the expression
    """
```

- e.g., x = 42 + 148

Evaluating an `Assign` mutates the `env`

```
>>> stmt = Assign('x', Num(10))
>>> env = {}
>>> stmt.evaluate(env)
```

```
>>> env
{'x': 10}
```

Consolidate!

- `Name.evaluate`
- Look up the variable name in the current environment.

- `Assign.evaluate`
- Add a new variable binding to the current environment.

(mutates env!)

As usual, practice ...

- Worksheet: the variable environment and statements