Announcements

No lab this week, final prep and lab next week.

Please check the Assignment 2 FAQ on the course forum.

Extra A2 office hour schedule to be posted today!
This Friday: Research Guest Speakers

Joseph Jay Williams

Fanny Chevalier
Abstract Syntax Trees

CSC148, INTRODUCTION TO COMPUTER SCIENCE
DAVID LIU
Programs as data

An expression tree is a structured way of modeling (“simple”) Python code.
Why bother?

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 more general: evaluating a statement can produce a value, and/or has some other effect.

Every expression is a statement, but not vice-versa!
Examples of statements

```python
x = 5

if x > 5:
    y = 10
else:
    y = 15

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

return 10

break
Variable bindings

How do we model variables in an abstract syntax tree?
A variable name: the Name class

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

    Attributes
    id: The variable name.
    """

    id: str

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}
```
Passing in the environment

class Expr:
    def evaluate(self, env: Dict[str, Any]) -> Any:
        """Return the *value* of this expression, in the given environment."""
An example

```python
>>> expr = Name('x')
>>> expr.evaluate({'x': 10})
```
Creating bindings: the **Assign** class

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

    Attributes
    target: the variable name
    value: the expression
    """
```
Evaluating an `Assign` *mutates* the `env`

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

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

Assign.evaluate
Add a new variable binding to the current environment.
Module: blocks of code

A Module is a class that represents an entire Python program. Its **body** is a list of statements.
Implementing control flow structures

if <condition>:
    <body>
else:
    <orelse>

for <target> in range(<start>, <stop>):
    <body>