**Types of Expressions**

**Primitive expressions:**
- Number or Numeral
- Name
- String

**Call expressions:**
- `max` (2, 3)
- `add` ('hello')

An operand can also be a call expression:
- `max(min(pow(3, 5), -4), min(1, -2))`

**Discussion Question 1**

What is the value of the final expression in this sequence?

```python
>>> f = min
>>> g = max
>>> g, h = min, max
>>> max(f(2, g(h(1, 5), 3)), 4)
```
Defining Functions

Assignment is a simple means of abstraction: binds names to values
Function definition is a more powerful means of abstraction: binds names to expressions

```
>>> def <name>(<formal parameters>):
    return <return expression>
```

Execution procedure for def statements:
1. Create a function with signature <name>(<formal parameters>)
2. Set the body of that function to be everything indented after the first line
3. Bind <name> to that function in the current frame

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):
1. Add a local frame, forming a new environment
2. Bind the function’s formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

Looking Up Names In Environments

Every expression is evaluated in the context of an environment.
So far, the current environment is either:
- The global frame alone, or
- A local frame, followed by the global frame.

Most important two things I’ll say all day:
An environment is a sequence of frames.
A name evaluates to the value bound to that name in the earliest frame of the current environment in which that name is found.

E.g., to look up some name in the body of the square function:
- Look for that name in the local frame.
- If not found, look for it in the global frame.
  (Built-in names like “max” are in the global frame too, but we don’t draw them in environment diagrams.)

None Indicates that Nothing is Returned

The special value `None` represents nothing in Python.
A function that does not explicitly return a value will return `None`.

Careful: `None` is not displayed by the interpreter as the value of an expression
```
>>> def does_not_square(x):
    ... x * x ...
>>> does_not_square(4)
```
```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
```

None value is not displayed

Pure Functions & Non-Pure Functions

```
>>> def abs(i):
    ... return i
```
```
>>> abs(-2)
```
```
>>> def pow(i, j):
    ... return i ** j
```
```
>>> pow(2, 100)
```
```
1267650600228229401496703205376
```

Python displays the output `-2`

A side effect isn’t a value; it’s anything that happens as a consequence of calling a function.
Nested Expressions with Print

```python
>>> print(print(1), print(2))
None None
```

```
def print(...):
    display "None None"
```

```
def print(...):
    print(1):
    display "1"
```

```
def print(...):
    print(2):
    display "2"
```

```
def print(...):
    print(17):
```

Does not get displayed