EXPRESSIONS, STATEMENTS, AND FUNCTIONS 1

COMPUTER SCIENCE 61A

September 4, 2014

0.1 Warmup — What Would Python Do?

```
>>> x = 6
>>> def square(x):
...     return x * x
>>> square(x)
```

```
>>> max(pow(2, 3), square(-5)) - square(4)
```

1 Expressions

An expression describes a computation and evaluates to a value.

1.1 Primitive Expressions

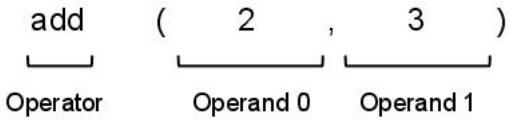
A *primitive expression* requires only a single evaluation step: you either look up the value of a name, or use the literal value directly. For example, numbers, names, and strings are all primitive expressions.

```
>>> 2
2
>>> 'Hello World!'
'Hello World!'
```

1.2 Call Expressions

A call expression applies a function, which may or may not accept arguments. The call expression evaluates to the function's return value.

The syntax of a function call:



Every call expression requires a set of parentheses delimiting its comma-separated operands.

To evaluate a function call:

1. First evaluate the operator, and then the operands (from left to right).
2. Apply the function (the value of the operator) to the arguments (the values of the operands).

If an operand is a nested call expression, then these two steps are applied to that operand in order to evaluate it.

1.3 Questions

1. Determine the result of evaluating the following functions in the Python interpreter:

```
>>> from operator import add
>>> def double(x):
...     return x + x
>>> def square(y):
...     return y * y
>>> def f(z):
...     add(square(double(z)), 1)
>>> f(4)
```

CS 61A Fall 2014: John DeNero, with
Soumya Basu, Matthew Chow, Ajeya Cotra, Brian Hou, Andrew Huang, Robert Huang, Michelle Hwang, Mehdi Jamei, Joy Jeng, Chloe Lischinsky, Kaylee Mann, Beth Marrone, Allen Nguyen, Youri Park, Jack Qiao, Sumukh Sridhara, Steven Tang, Michael Tao, Dickson Tsai, Iris Wang, Albert Wu, Chenyang Yuan, Marvin Zhang

2. What will Python print?

```
>>> def welcome():
...     print('welcome to')
...     return 'hello'
>>> def cs61a():
...     print('cs61a')
...     return 'world'
>>> print(welcome(), cs61a())
```

3. What will Python print?

```
>>> from operator import add, mul
>>> def f(x, y):
...     print('h', y)
...     return add(add(x, y), 1)
>>> def g(x, y):
...     print('u', x)
...     return mul(mul(x, y), 2)
>>> f(1, g(2, f(500, 2)))
```

2 Statements

A statement in Python is executed by the interpreter to achieve an effect.

For example, we can talk about an assignment statement. In an assignment statement, we ask Python to assign a certain value to a variable name:

```
>>> x = 6
```

Here, we ask Python to assign the value of the expression `6` to the name `x`. Since `6` is a primitive expression (just a number!), its value is the integer 6. Therefore, Python will create a binding from the name `x` to the value 6.

Statements may range from one line, like the assignment statement, to multiple lines. A good example of a multiple line statement is the `def` statement:

```
>>> def <name>(<parameters>):
...     <suite>
```

When a `def` statement is executed, Python will create a binding from the `name` to a function that has certain `parameters`. When the function is applied in a call expression, the `suite` will be executed. We'll talk more about bindings in Python environments in the next few weeks as we learn more about environment diagrams.

What is a *suite*? A suite is a sequence of statements or expressions associated with a header. The suite will always have one more level of indentation than its header. To execute a suite, we execute its sequence of statements or expressions in order.

Let's take a closer look at the `def` statement with our favorite function:

```
>>> def square(x):
...     return x * x
```

`square` is the name of our function, which has one parameter, `x`. Because it has one parameter, it is always called with one argument. Here, the suite consists of just one statement: `return x * x`.

In short, the `def` statement achieves the effect of creating a function, but does not perform a computation like an expression.

2.1 Questions

1. Determine the result of evaluating `foo(5)` in the Python interpreter if the following functions are defined in the order below:

```
>>> def bar(param):
...     return param
>>> bar = 6
>>> def foo(bar):
...     return bar(bar)
>>> bar = 7
>>> foo(5)
```

2. What would Python print?

```
>>> from operator import add
>>> def sub(a, b):
...     sub = add
...     return a - b
>>> add = sub
>>> sub = min
>>> print(add(2, sub(2, 3)))
```

3 Pure and Non-Pure Functions

1. Pure functions have no side effects – they only produce a return value. They will always evaluate to the same result, given the same argument value(s).
2. Non-pure functions produce side effects, such as printing to your terminal.

Later in the semester, we will expand on the notion of a pure function versus a non-pure function.

3.1 Moar Questions

1. What do you think Python will print for the following?

```
>>> def om(foo):
...     return -foo

>>> def nom(foo):
...     print(foo)

>>> nom(4)

>>> om(-4)

>>> brian = nom(4)
>>> brian + 1

>>> michelle = om(-4)
>>> michelle + 1
```

2. What do you think Python will print for the following?

```
>>> x = 6
>>> def beep(x):
...     print(x)

>>> def boop(x):
...     y = x
...     x = 7
...     print(x)

>>> y = beep(x)

>>> boop(x)

>>> y + beep(8)
```

4 Environment Diagrams

1. Draw the environment diagram that results from running the following code.

```
>>> a = 1
>>> def b(b):
...     return a + b
>>> a = b(a)
>>> a = b(a)
```

2. Here's the the practice problem from above. Draw the environment diagram so we can visualize exactly how Python evaluates the code.

```
>>> from operator import add
>>> def sub(a, b):
...     sub = add
...     return a - b
>>> add = sub
>>> sub = min
>>> print(add(2, sub(2, 3)))
```

5 Secrets to Success in CS 61A

CS 61A is definitely a challenge, but we all want you to learn and succeed, so here are a few tips that might help:

- Ask questions. When you encounter something you dont know, *ask*. That is what we are here for. This is not to say you should raise your hand impulsively; some usage of the brain first is preferred. You are going to see a lot of challenging stuff in this class, and you can always come to us for help.
- Study in groups. Again, this class is not trivial; you might feel overwhelmed going at it alone. Work with someone, either on homework, on lab, or for midterms, as long as you don't violate the cheating policy!
- Go to office hours. Office hours give you time with the instructor or TAs by themselves, and you will be able to get some (nearly) one-on-one instruction to clear up confusion. You are *not* intruding; the instructors and TAs *like* to teach! Remember that, if you cannot make office hours, you can always make separate appointments with us!
- Do (or at least attempt seriously) all the homework. We do not give many homework problems, but those we do give are challenging, time-consuming, and rewarding. The fact that homework is graded on effort does not imply that you should ignore it: it will be one of your primary sources of preparation and understanding.
- Do all the lab exercises. Most of them are simple and take no more than an hour or two. This is a great time to get acquainted with new material. If you do not finish, work on it at home, and come to office hours if you need more guidance!
- Do the readings before lecture! There is a reason why they are assigned. And it is not because we are evil; that is only partially true.
- Most importantly, *have fun!*