CS 61A
Spring 2018

Introduction and Functions

Discussion 0: June 19, 2018

1 Lost on the Moon

Your spaceship has just crashed on the moon. You were scheduled to rendezvous with a mother ship 200 miles away on the lighted surface of the moon, but the rough landing has ruined your ship and destroyed all the equipment on board except for the 15 items listed below.

Your crew's survival depends on reaching the mother ship, so you must choose the most critical items available for the 200-mile trip. Your task is to rank the 15 items in terms of their importance for survival. Place a number 1 by the most important item, number 2 by the second most important, and so on, through number 15, the least important.

Item	Your Rank (1)	Group's Rank (2)	NASA's Rank (3)	$\lvert(3)-(1)\rvert$	$\lvert(3)-(2)\rvert$
Box of matches					
Food concentrate					
50 feet of nylon rope					
Parachute silk					
Solar-powered portable heating unit					
Two .45 caliber pistols					
One case of dehydrated milk					
Two 100-pound tanks of oxygen					
Stellar map (of the moon's constellations)					
Self-inflating life raft					
Magnetic compass					
5 gallons of water					
Signal flares					
First-aid kit containing injection needles					
Solar-powered FM receiver-transmitter					
Total					

2 Python basics

Primitive expressions

A **primitive expression** requires only a single evaluation step. Literals, such as numbers and strings, evaluate to themselves. Names require a single lookup step (see the *Assignment statements* section below).

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

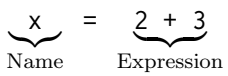
Arithmetic expressions

Arithmetic expressions in Python are very similar to ones we've seen in other math contexts. They involve binary arithmetic operators (`+`, `-`, `*`, `/`, `//`, `%`, and `**`) and follow PEMDAS rules.

```
>>> 6 + 2 * 5
16
>>> 9 // 2     # Floor division (rounding down)
4
>>> 9 % 2      # Modulus (remainder of 9 // 2)
1
>>> (3 + 2) * 4 // 3
6
>>> 4 ** 3     # Exponent
64
```

Assignment statements

An assignment statement assigns a certain value to a variable name.



To execute an assignment statement:

1. Evaluate the expression on the right-hand-side of the statement to obtain a value.
2. Bind the value to the name on the left-hand-side of the statement.

Let's try to assign the primitive value 6 to the name `a`, and subsequently do a lookup on `a`.

```
>>> a = 6
>>> a
6
```

Now, let's reassign `a` to another value. This time, let's use a more complex expression. Note that the name is bound to the value, not the expression!

```
>>> a = (3 + 5) // 2
>>> a
4
```

Questions

2.1 What would Python display?

```
>>> 3 + 4 ** 2

>>> a = 6 + 2 * 4
>>> a

>>> b = (2 + 2) * 2 + 3 % 2
>>> b

>>> a + 2 * b

>>> b += a      # Equivalent to b = b + a
>>> a

>>> b
```

3 Functions

Writing and calling functions is a crucial part in manage complexity when programming.

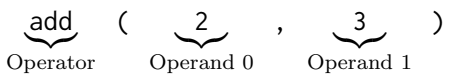
Functions allow us to apply a series of statements to some arguments to produce a return value. For, example, let's take a look at the built-in `abs` function:

```
>>> abs(123)
123
>>> abs(-2000)
2000
```

This function takes in one argument, a number, and returns its absolute value. Since this is a built-in function, we don't know what statements were executed to obtain this output. The expressions above are known as call expressions.

Calling functions

A **call expression** applies a function to 0 or more arguments and evaluates to the function's return value.



To evaluate a function call:

1. Evaluate the operator, which should evaluate to a function.
2. Evaluate the operands from left to right to get the values of the arguments.
3. Apply the function (the value of the operator) to the arguments (the values of the operands) to obtain the return value.

If the operator or an operand is itself a call expression, then these steps are applied in order to evaluate it.

Defining functions

The `def` statement defines functions.

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

When a `def` statement is executed, Python creates a binding from the name (e.g. `square`) to a function. The names in parentheses are the function's **parameters** (in this case, `x` is the only parameter). When the function is called, the **body** of the function is executed (in this case, `return x * x`).

Questions

3.1 Consider the function below.

```
def foo(x, y):
    y = y + 4
    x = y / 2
    return x * y
```

How many arguments does this function take in, and what type should they be?

Consider the call expression `foo(5, 1 + 3)`. What are `x` and `y` bound to inside the body of the function during this call?

What does the call expression `foo(5, 4)` return?

What about `foo(10, 4)`?

3.2 Now, let's practice defining functions. In the space below, write a function called `square_diff` that takes in two integer arguments and returns the result of squaring the difference between the arguments. You may name the parameters whatever you'd like. Be sure to use proper Python syntax!

```
>>> square_diff(4, 1)
9
>>> square_diff(2, 7)
25
```