

Discussion 0

Working Effectively in CS

CS61A Sections 13/26

Dickson Tsai

dickson.tsai+cs61a@berkeley.edu

dicksontsai.com/cs61a

OH: M 4-5pm, Th 2-4pm Garbarini Lounge

Objectives

- * Understand the importance of groupwork
- * Icebreaker that doesn't take too long yet effective and physically active
- * Understanding statements vs. expressions
- * Understand the power of names
- * Understand what issues names can raise
- * Final remarks

Anuncios

- * Please put all electronic devices away. You will not need your laptop for discussion ever.
 - * Please ☺
- * Join the Piazza
- * Look out for HW1 this weekend
- * Midterm is in 3 weeks so...
- * ... Let's get rolling!

Lost on the mooooooooo...on

- * Take 10 minutes to complete this answer alone
- * Now, without changing your answers in the first column, get into a group of 3 and come up with a new list.
 - * The more debate, the better
 - * Get to know each other
- * Drumroll, please...

Lost on the moon [Solutions]

- * Evaluate your thought process. Apart from copying down the answers, don't fixate too much on the results.
 - * If your answer is right, what method led you to the right answer?
 - * If your answer is wrong, how should you change your approach for next time?

This worksheet seemed random???

- * Show of hands: how many people's group score was better than their individual score?
- * Personal anecdote: chess with TA's over summer
- * Groups are powerful!
- * CS application: It's better to have clusters of computers rather than one super-expensive computer.

Brief discussion

- * All Berkeley students are distinguished students. What made you successful?
- * [Discuss]: Share your tips for success. Personal anecdotes welcome!
- * Result: Self-confidence and motivation from peers are important
 - * Lab is designed to foster this positive feeling
 - * Take the time to encourage your friends as well!

A little tangent... Mastering CS

- * There are millions of people learning CS right now.
 - * Q: How do you stand out?
- * A:
 - * Everyone has access to same documentation
 - * What matters is how well you can combine pieces together!
- * My high school Spanish anecdote

A little tangent... Mastering CS

- * Therefore:
- * Everything has a purpose. Try to figure the purpose out with **simple examples! Master the basics!**
- * Then, play with the pieces, then practice putting them together.
- * With proper reinforcement, you'll build a strong intuition!

Breaktime!

- * Get in a circle! We'll play “Have you ever?”
- * At the end of each turn:
 - * If you have a new neighbor, introduce yourselves
 - * If you have an old neighbor, learn something now about them

Elements of Programming Languages

- * Primitive expressions and statements
- * Means of combination
- * Means of abstraction
- * Examples
 - * Numbers, booleans, strings, lists, ...
 - * Infix (+, *), call expression (a(b, c))
 - * Assignment (a = 5), functions (def love(x)), objects

Expressions vs. Statements

- * An expression describes a computation and evaluates to a value.
 - * Expression ----gets evaluated to---> a value
- * A statement makes a change, gets executed

Expressions are expressive

- * `buy_groceries(['eggs', 'milk', 'juice', 'strawberries'])`
- * [Discuss]: What collection of procedures might this computation entail?
- * Write list on phone
- * Drive to Safeway -> (open car door, ignite engine, ...)
- * For each item in the list, locate it and put it in cart
- * ...

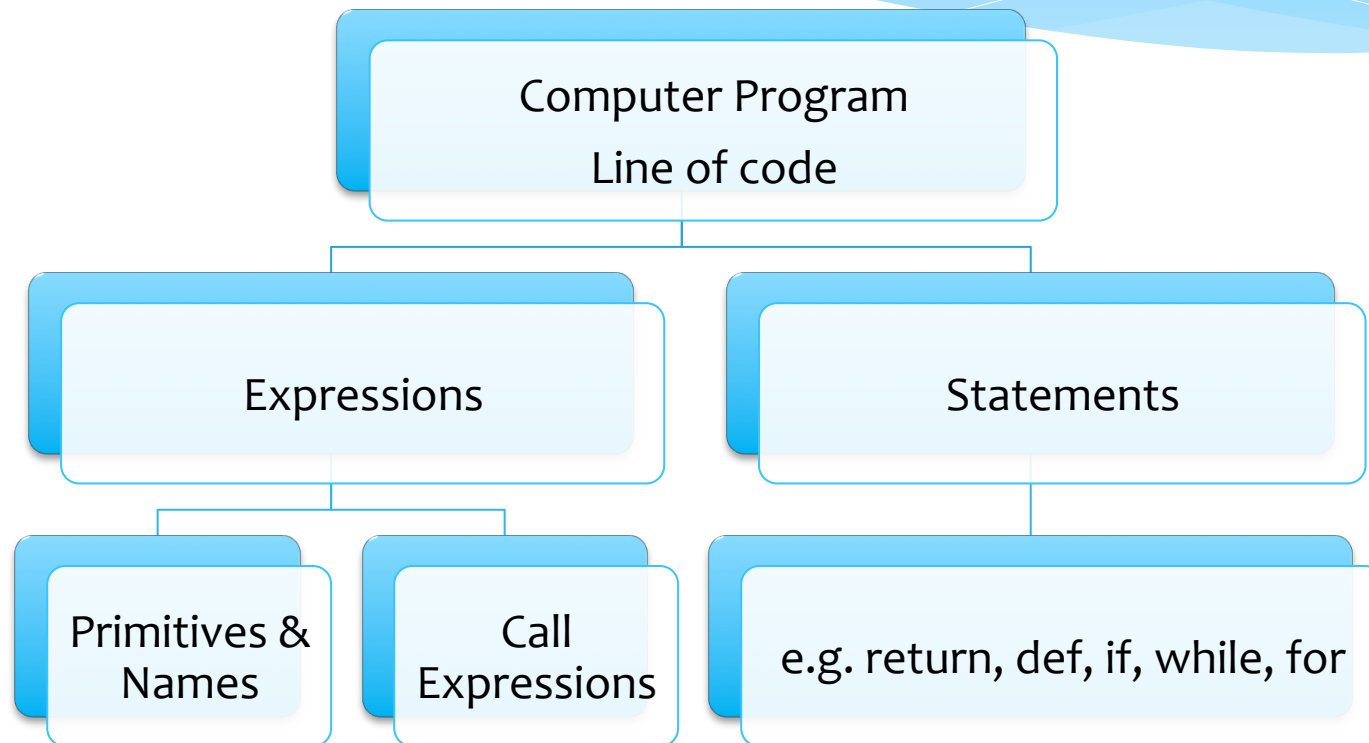
Abstraction

- * The property that expressions exploit is abstraction, our main tool for dealing with complexity.
- * Example: Dining hall food is just there. I don't usually think about where it comes from. It's a huge process.
 - * The interface is the dining hall serving area
- * [Discuss]: Where has abstraction appeared in your field of study?

Abstraction

- * Q: What does abstraction do for us?
- * Allow us to manipulate massive amounts of data under simpler names
- * Categorize similar things together, more organized
- * Allow us to have one name stand for a collection of instructions, so we can code more quickly (instructions are data too!)
- * The list goes on...

Expressions vs. Statements



Each type of code has its own procedure. E.g. names have a lookup procedure, assignment has rules

Example Evaluation Rule

- * To evaluate a call expression, Python will do the following:
 - * Evaluate the **operator** and **operand** subexpressions
 - * Apply the **function** that is the value of the operator subexpression to the **arguments** that are the values of the operand subexpressions

Practice evaluating call expressions

`add(1, 2)`

Practice walking through the rules step-by-step instead of making assumptions

`good_number = mul(add(1, 2), 3)`

`mul(2, good_number)`

Errata: It turns out that `mul` can only take two arguments

`add(add(mul(5, 2), 5), mul(sub(1000, 900), 2, 10))`

Practice evaluating call expressions

`add(1, 2)`

`good_number = mul(add(1, 2), 3)`

`mul(2, good_number)`

Here is the corrected version

`add(add(mul(5, 2), 5), mul(sub(1000, 900), 20))`

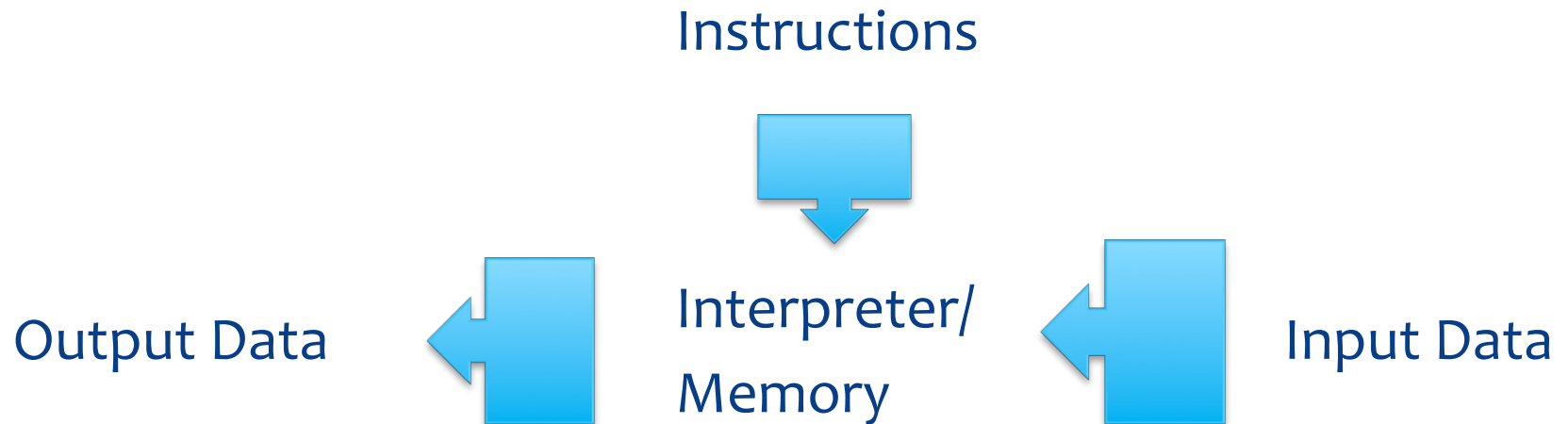
A note on infix operators

Disclaimer: You are not responsible for how infix operators work exactly in Python! Sorry for bringing this up in section...

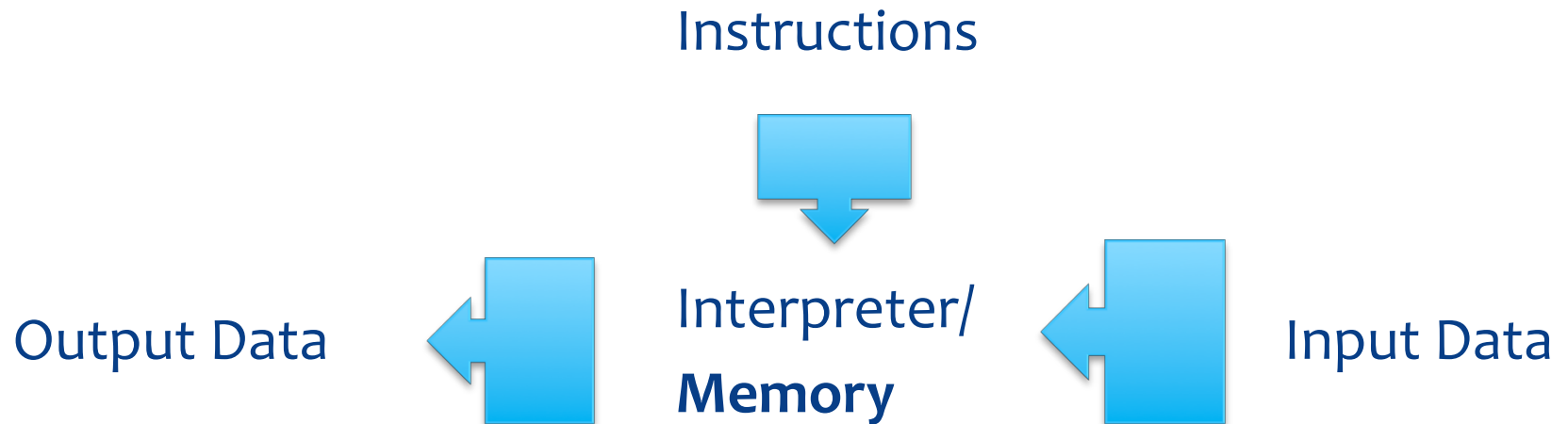
- * Infix operators are governed by order of operations.
 - * $a + b * c$ False or True and 4
- * Just trust the order of operations, but remember to evaluate the expressions on both sides before applying them.
- * If you are curious, you can write a function like:

```
def num(x):  
    print(x)  
    return x
```
- * And then try it out like $\text{num}(1) + \text{num}(2) * \text{num}(3)$

Basic Computer Program



Basic Computer Program



Names

- * [Discuss]: What makes a calculator useful?
 - * The interpreter keeps track of things by associating a **name** with a **value**
 - * $a = 5$
 - * $a = a * (a + 1)$
- The difference comes when a does not represent a simple value:
- * $a = \text{some}(\text{very}(\text{expensive}, \text{fancy}, \text{computation}))$

Usefulness of names

- * One advantage: you shouldn't have to recompute something if you've seen it before

Without names	With names
<code>tau = 2 * 3.14159</code>	<code>pi = 3.14159</code>
<code>circ = 2 * 3.14158 * 5</code>	<code>radius = 5</code>
	<code>circ = 2 * pi * radius</code>
<code>area = 3.14159 * 5 * 5</code>	<code>area = pi * radius ** 2</code>


- * But are there problems associated with using names?

Too many names!

- * What if two different functions want to use the same name!
- * Interpreter takes care of this by using frames. Each function has its own separate frame/scope to play with.
 - * [Board demo]
- * You will learn more about environments and control flow next time.


Final Remarks

- * I'm not all-knowing, and my explanations certainly won't write your exam answers for you.
 - * You have to put in the effort to learn!
- * Everything has a purpose. Try to figure the purpose out with **simple examples! Master the basics!**
- * Then, play with the pieces, then practice putting them together.
- * With proper reinforcement, you'll build a strong intuition!



Simple Examples

Master the basics



Simple Examples

Master the basics

When you're stuck...



Keep these in mind!

Simple Examples
Master the basics



... and use
Google +
StackOverflow

FIN

Let's have a fun
semester!