
HAPPY
NEW YEAR!

Lecture 1

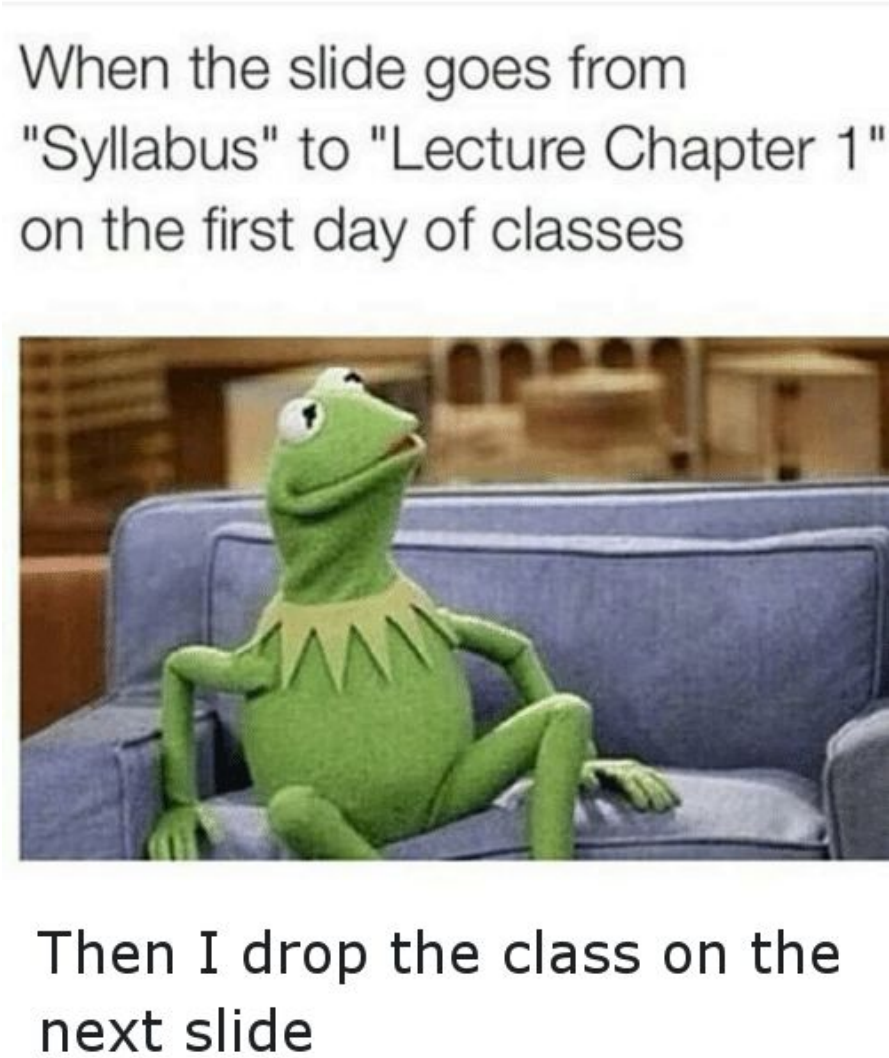
Introduction

Plan for today

- I will introduce myself
- Getting to know you
- Course Structure
- Flow of the class
- Short Break for questions (a few mins)

Plan for today

- I will introduce myself
- Getting to know you
- Course Structure
- Flow of the class
- Short Break
- Start an actual lecture.



About me

- Marina Langlois: Marina, Dr. Langlois, Prof. Langlois, Prof. Marina

- **Before UCSD:**
  - PhD from UIC (Chicago)
  - Lecturer at UIC
  - Lecturer at Yeshiva (NYC)

- **Research in theoretical AI**
  - Currently on hold

- **At UCSD:**
  - CSE 8A, 12, 20, 150
  - DSC 10, 20, 30 and 80

- **Not working:**
  - Having fun with my family

Waitlisted students

- Software was down, Margaret could not clear you yet.

A few questions about your background

Do you have any programming background (in any language)

A: Yes, took a few CSE classes (like cse8a, cse8b, cse12). Or something of this sort.

B: I'd say yes, I had AP classes that taught me a good amount of coding

C: I'd say yes, I was lucky to have an internship where I picked up a language

D: I only have DSC10 as my programming experience

E: Although I took DSC10 but feel that I did not learn much.

A few questions about your background

Did you start your hw/projects on time in DSC10?

A: Often yes

B: 50%is

C: 25%

D: Not as often as I should of

E: No, was always last minute

A few questions about your background

How long did it take you to complete the assignments per week

A: Less than 4 hours

B: 4 - 6

C: 6 - 8

D: more than 8

E: Never ending process.

A few questions about your background

How much did you rely on your partner?

A: I did not have one.

B: It was ~ equal effort. Very beneficial.

C: It was ~ equal effort. Did not feel that it was worth it.

D: Too much, I would not have survive without him/her.

E: I had a partner but did not use his/her help that much.

A few questions about your background

How much time do you expect to spend in this class?

A: About the same as in DSC10 (I know how to program)

B: About the same as in DSC10 (Almost new to programming)

C: A few hours more

D: A lot more.

E: Did not think about it yet.

Course Structure

Grade Components

Weekly labs	10%
Weekly homework assignments	25%
Project(s)	10%
Participation	5%
Performance	5%
Exams	
1 midterm exam in class	15%
Final exam	30%

Participation points: 5%

- You need to have an iClicker version 2 remote and **register** it on TritonEd by the end of week 1 if you have never done it before.
- You need to click on at least 75% of the questions each class in order to be receive a participation point for a particular class.
    - Today class does not count
    - You can miss 5 classes without warning me
    - Midterm above 93% gives you 8 more “free” classes
- Check the Syllabus on our webpage for more information

Performace points (5%)

- We will have iClicker questions regularly. (Almost) each question will be worth 1 point.
- At the end of the quarter you need to get ***at least*** 75% correct answers to get a full credit.
  - Midterm over 93% will give you 25% (all questions before the midterm)
  - Final over 93% will give you 50% (all questions after the midterm)
  - Discussion will have clicker points as well. Each correct question will be added to your total.
- Check the Syllabus on our webpage for more information.

Collaboration

Asking questions is highly encouraged

- *Discuss* all questions with each other (except exams)
- Submit lab assignments **individually**, but you can work with others in the same lab
- Submit homeworks **individually**, but feel free to discuss with others.
- You can work on the project(s) in pairs.

Collaboration

Asking questions is highly encouraged

- Discuss all questions with each other (except exams)
- Submit lab assignments individually, but you can work with others in the same lab
- Submit homeworks individually, but feel free to discuss with others

The limits of collaboration

- **Don't share solutions** with each other or **look at** someone's code
  - Including google
- **Academic integrity violations will result in failing the course**

Partners are not allowed
(except for the projects)

Not quite alone

- Team of tutors (their lab hours will be on the calendar).
- Some of them will be here during lecturers:
  - Quick questions about labs/homeworks before/after class.

Week 1 assignments

- Lab 0
  - Deadline is **Thursday 11:59 am**
  - Walks you through installing Python3 on your machines
  - No grade but you **must** submit. If not submitted your Lab 1 will not be graded.
- Lab 1
  - Deadline is **Thursday 11:59 am**
  - Covers expressions, if statements and loops
  - **Graded**
- Homework 1
  - Deadline is **Monday 11:59 am**
  - **Graded**
- **No tutors on Monday**

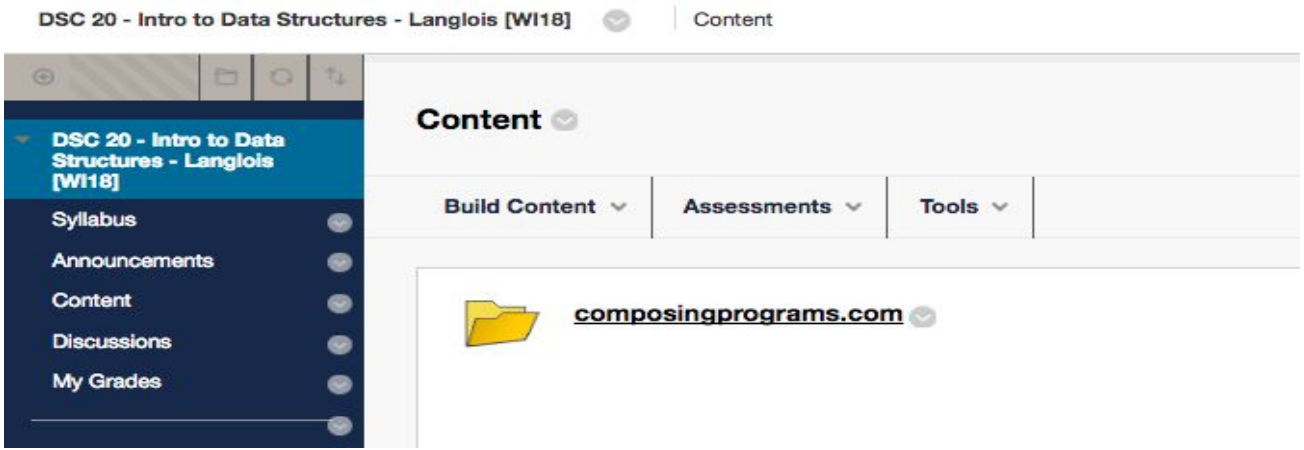
Late Submissions. Homework and project

- Less than 24 hours late: 20% penalty.
- 24 hours < SUBMISSION <= 48 hours late: 50% penalty.
- More than 48 hours late: 0 points.

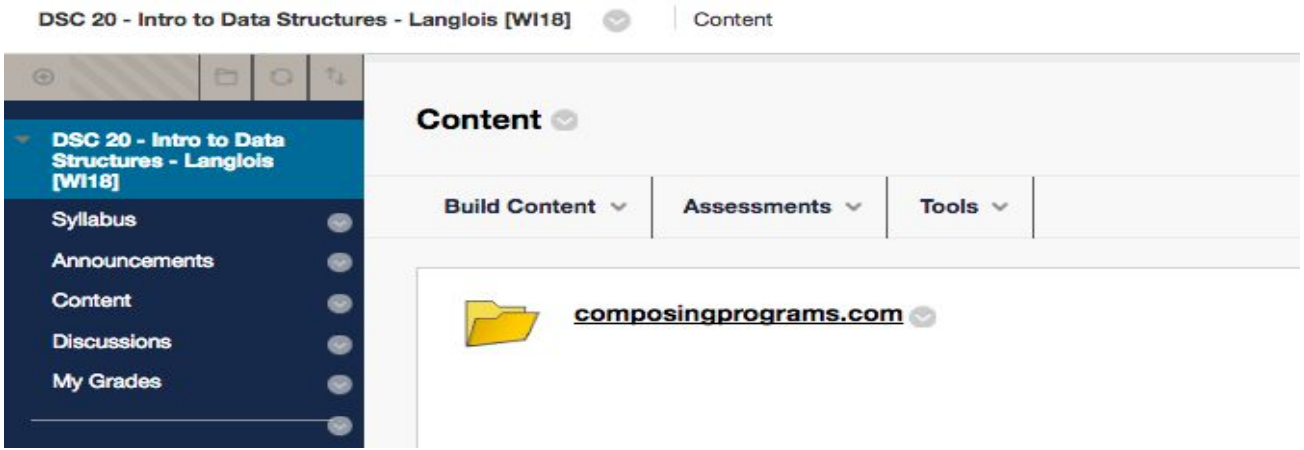
Labs deadline is set in stone. You must submit on time.

*Unless something exceptional had happened. Documentation will be required is some cases.

Textbook is on TritonEd



Textbook is on TritonEd

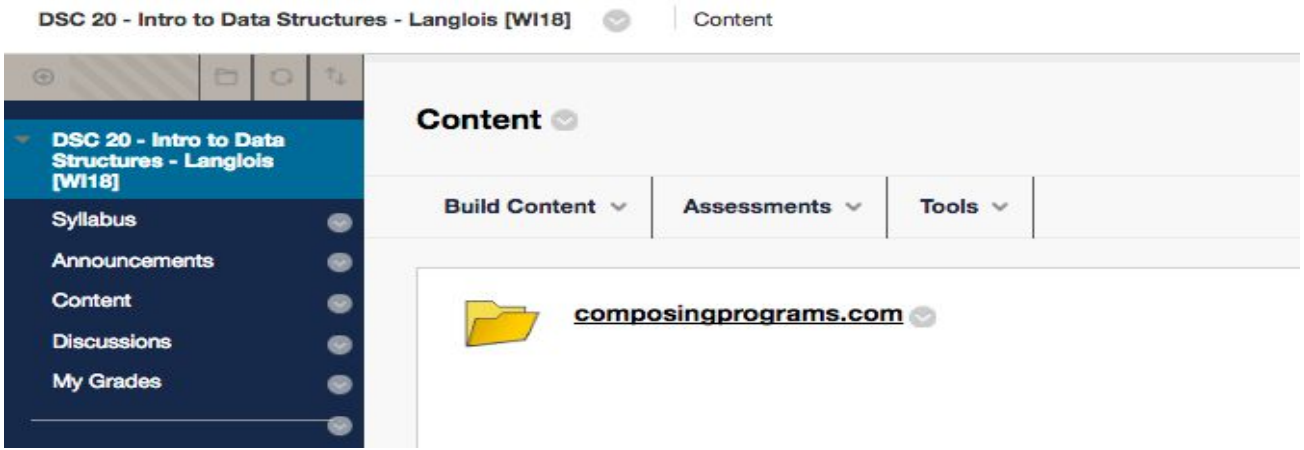


1.2.2 Call Expressions

Video: Show Hide

The most important kind of compound expression is a *call expression*, which applies a function to some arguments. Recall from algebra that the mathematical notion of a function is a mapping from some input

Textbook is on TritonEd



- Additional resources will be *posted* as well.

  Posted WHERE?

Class website

- https://sites.google.com/a/eng.ucsd.edu/dsc20-winter-2019/

Websites

- Gradescope
  - *Instead of OK*
  - Regrades are there as well
  - Assignments and exams
- Piazza
  - For questions/answers etc
- TritonEd
  - Class textbook
  - Lecture slides
  - Participation/Performance points

Piazza tips

- Make sure to **search** for an answer before posting.

- **Do not post your code publicly.**
  - Violation of Academic Integrity
  - Warning for the first time
  - 0 for the assignment for the second time
  - Failed class for the third time

https://piazza.com/class/jqd2yk8a2xj400?cid=8

Class Flow

- Regular lecture with a lot of iClicker questions.
  - Answer, group discussion. Similar to DSC10.
- 2-3 minute break after ~ 40 min of lecture
- Finish the rest of the lecture

i - Clicker question

- Based on the syllabus and today's lecture, how many points will go towards your participation?

A: 0

B: 5

C: 10

D: 15

E: I do not remember

Questions?

Part 2. Let's get started

DSC 20 Goals

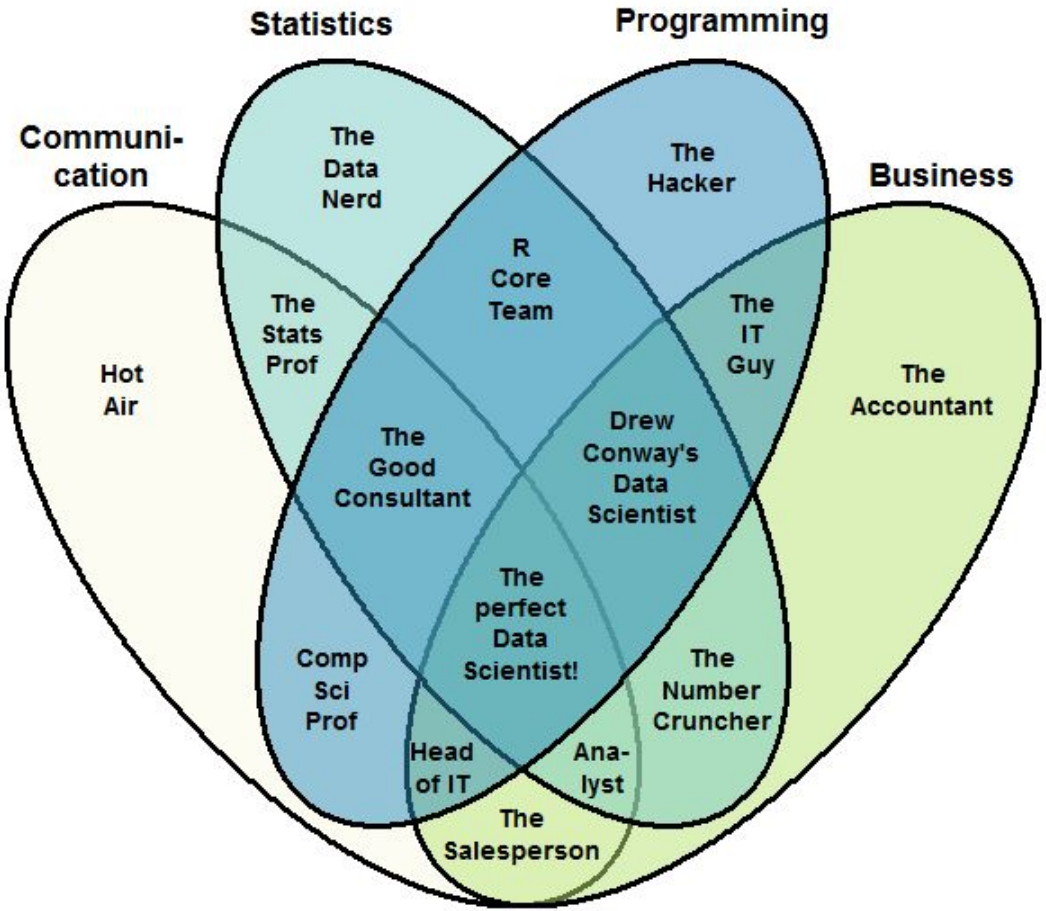
- Deeper understanding of the ***computing*** concepts introduced in DSC10
  - Hands-on experience => Foundational Concept

- Extend your understanding of the structure of *computation*.
  - What is involved in interpreting the code you write ?

- Deeper Computer Science Concepts
  - Higher-Order Functions, Recursion, Objects, Classes

- Managing complexity in creating larger software systems through *composition*

The Data Scientist Venn Diagram



What is this course about

- A course is about managing ***complexity***

https://www.techrepublic.com/article/how-complexity-is-killing-big-data-deployments/

What is this course about

- A course is about managing ***complexity***
  - Mastering abstractions

Abstraction

Details are removed

- "the act of representing essential features *without* including the background details or explanations"

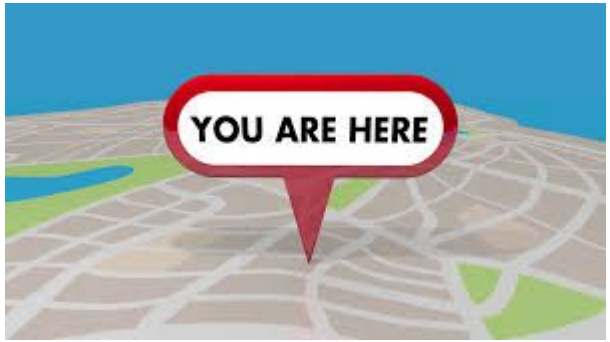
Henri Matisse *"Naked Blue IV"*

Experiment

Where are you from?

Possible answers:

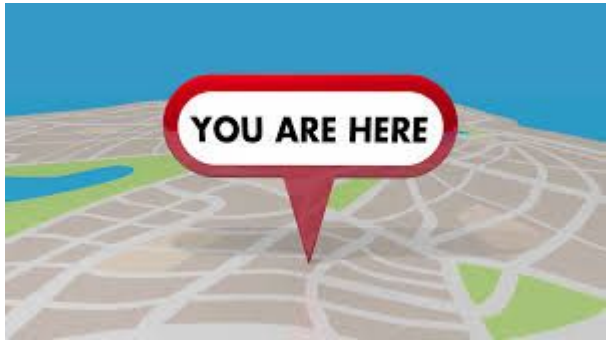
- Russia
- China
- California
- San Diego
- La Jolla
- Center Hall 119
- 32.877952, -117.237265



Where are you from?

Possible answers:

- Russia
- China
- California
- San Diego
- La Jolla
- Center Hall 119
- 32.877952, -117.237265



All correct answers but different levels of *abstraction*!

Abstraction

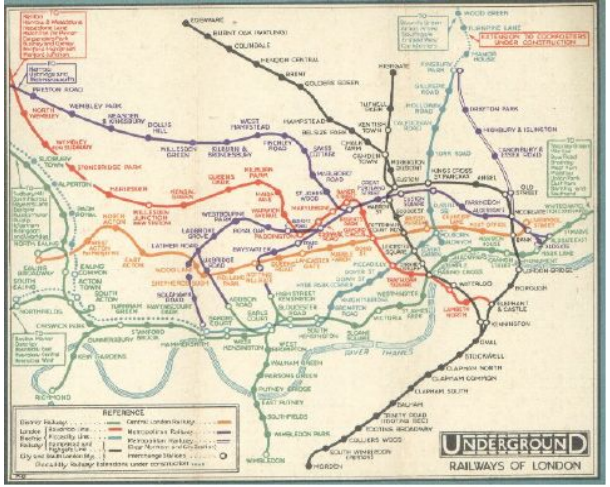
gone

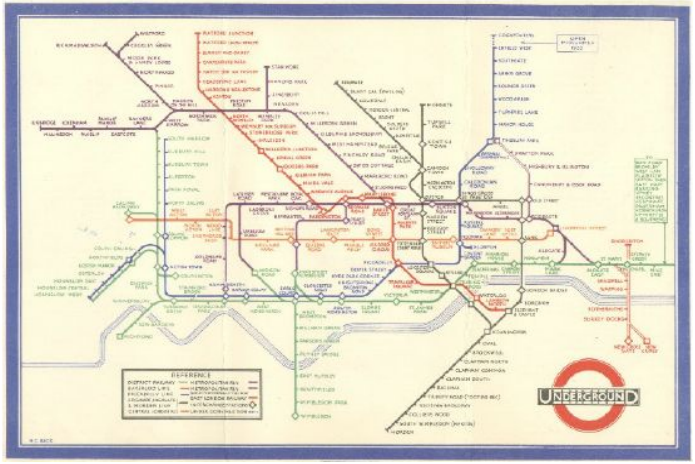
wrong.

Where?

Detail Removal (in Data Science)

- You'll want to look at only the interesting data, leave out the details, zoom in/out…
- Abstraction is the idea that you focus on the essence, the cleanest way to map the messy real world to one you can build
- Experts are often brought in to know what to remove and what to keep!





The London Underground 1928 Map & the 1933 map by Harry Beck.

The Power of Abstraction, Everywhere!

Abstraction is the act of representing essential features without including the background details or explanations.

- World map is an **abstraction** of the Earth in terms of *longitude* and *latitude*

Abstractions

Abstraction is the act of representing essential features without including the background details or explanations.

- World map is an abstraction of the Earth in terms of longitude and latitude

- Cell phones

Abstractions

Abstraction is the act of representing essential features without including the background details or explanations.

- World map is an abstraction of the Earth in terms of longitude and latitude

- Cell phones

- In math, we generalize formulas in terms of variables instead of numbers so we can *use* them to solve problems involving different values

- Can you think of one abstraction from DSC10?

The Power of Abstraction, Everywhere!

We only need to worry about the interface, or specification, or contract NOT how (or by whom) it's built

Above the abstraction line

Abstraction Barrier (Interface)
(the interface, or specification, or contract)

Below the abstraction line

This is where / how / when / by whom it is actually built, which is done according to the interface, specification, or contract.

What is this course about

- A course is about managing complexity
  - Mastering abstractions

- An introduction to Python
  - Full understanding of language fundamentals
  - Learning through implementation

Python

- Guido van Rossum

- Released in 1991
- General Purpose: build anything
- Open Source (free to use)
- Python Packages (for Data Science as well)

python™

Algorithm

- An algorithm is a procedure or formula to solve a problem.
- An algorithm is a sequence of instructions to change the state of a system. For example:
  - A computer's memory,
  - your brain (math), or
  - the ingredients to prepare food (cooking recipe).

Algorithm: Properties

- An algorithm is a description that can be expressed within a finite amount of space and time.

- Executing the algorithm may take *infinite* space and/or time, e.g. ``calculate all prime numbers”.

- In CS and math, we prefer to use well-defined *formal languages* for defining an algorithm.

i - clicker question

Calculate: 6 / 2 * (1 + 2) = ?

A: 1

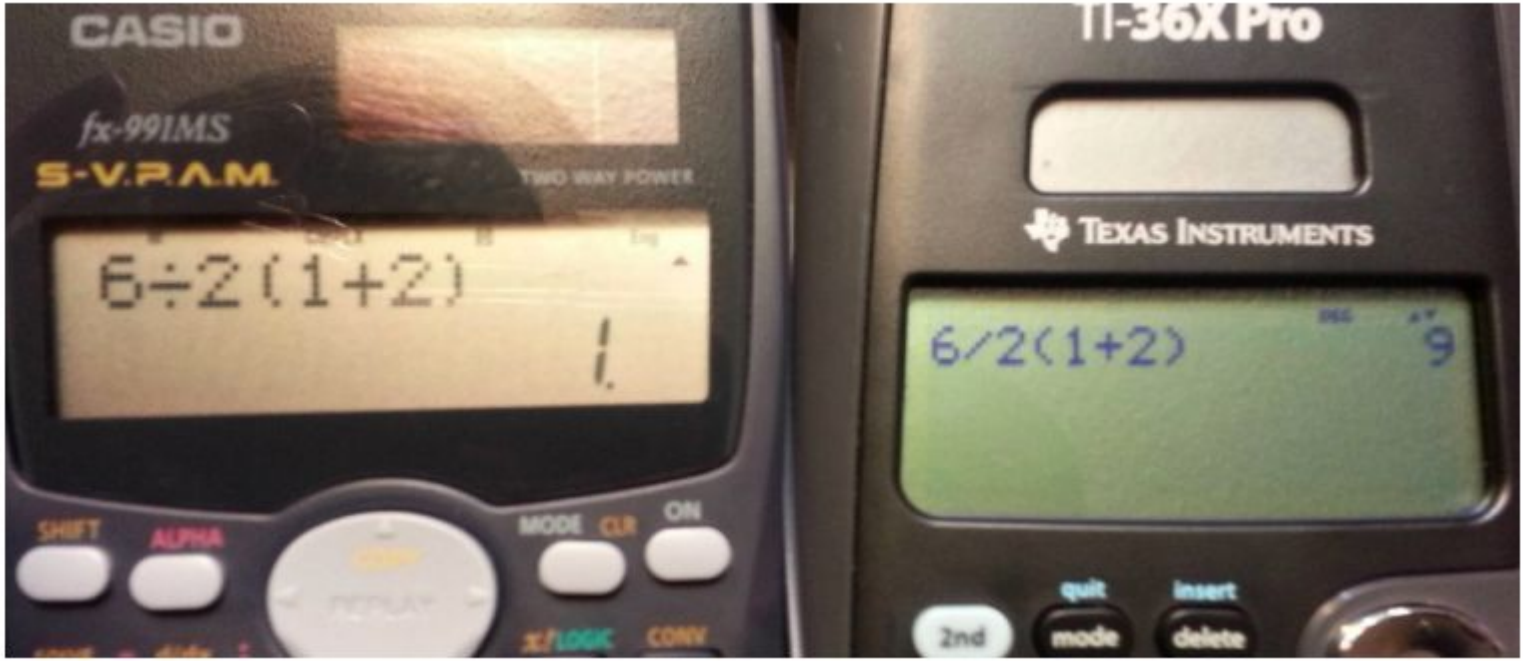
B: 9

C: 5

D: Can’t do it :(

Algorithm: Well-definition

Must be well - defined (unambiguous)



Code

Human-readable code (programming language)

```
def add5(x):
    return x+5

def dotwrite(ast):
    nodename = getNodename()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print ' %s [label="%s' % (nodename, label),
    if isinstance(ast[1], str):
        if ast[1].strip():
            print '= %s"];' % ast[1]
        else:
            print '"]'
    else:
        print '"];'
        children = []
        for n, child in enumerate(ast[1:]):
            children.append(dotwrite(child))
        print ' %s -> {' % nodename,
        for name in children:
            print '%s' % name,
```

Machine-executable instructions (byte code)

Compiler or Interpreter

Here: Python

Basic Language Structures (Python)

Question

An expression describes a computation and evaluates to a value

17	17 + 32	-17 - -32	| - 17 |	√17	17/32
1	2	3	4	5	6

Select all *expressions* from the table above:

A: 1 and 2
B: 1, 2 and 6
C: 1, 2, 5 and 6
D: 1, 2, 4, 5 and 6
E: All of them are expressions

Question

An expression describes a computation and evaluates to a value

17	17 + 32	-17 - -32	| - 17 |	√17	17/32
1	2	3	4	5	6

Select all *expressions* from the table above:

A: 1 and 2
B: 1, 2 and 6
C: 1, 2, 5 and 6
D: 1, 2, 4, 5 and 6
E: All of them are expressions

Call Expressions (demo)

All expressions can use function call notation: *call expressions*

- `max(17, 33, -2, 20+30)`
- `max(min(1, -2), min(pow(3, 5), -4))`
-
```
from operator import add, sub, mul
sub(100, mul(2, add(17, 33)))
```

Question

```
from operator import add, sub, mul
mul(add (2, mul(6, 8)), add(3, 5))
```

Above expression evaluates to:

A: 128
B: 400
C: 750
D: Can’t be evaluated to a number
E: None of the above

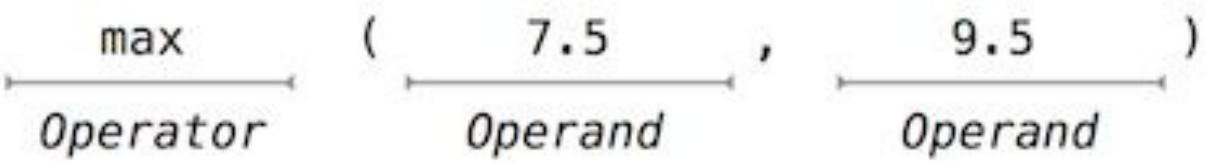
Question

```
from operator import add, sub, mul
mul(add (2, mul(6, 8)), add(3, 5))
```

Above expression evaluates to:

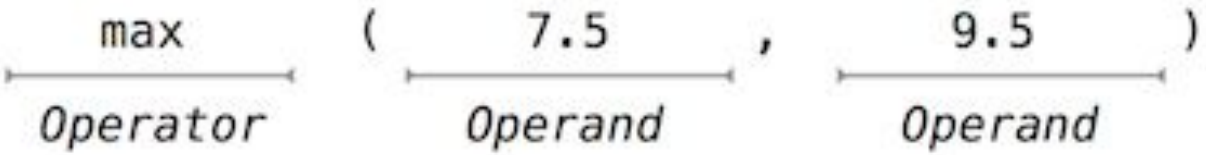
A: 128
B: 400
C: 750
D: Can’t be evaluated to a number
E: None of the above

Terminology



Operators and Operands are also Expressions

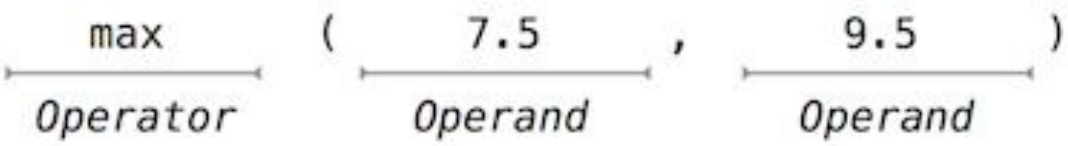
Terminology



Operators and Operands are also Expressions

They evaluate to values

Terminology



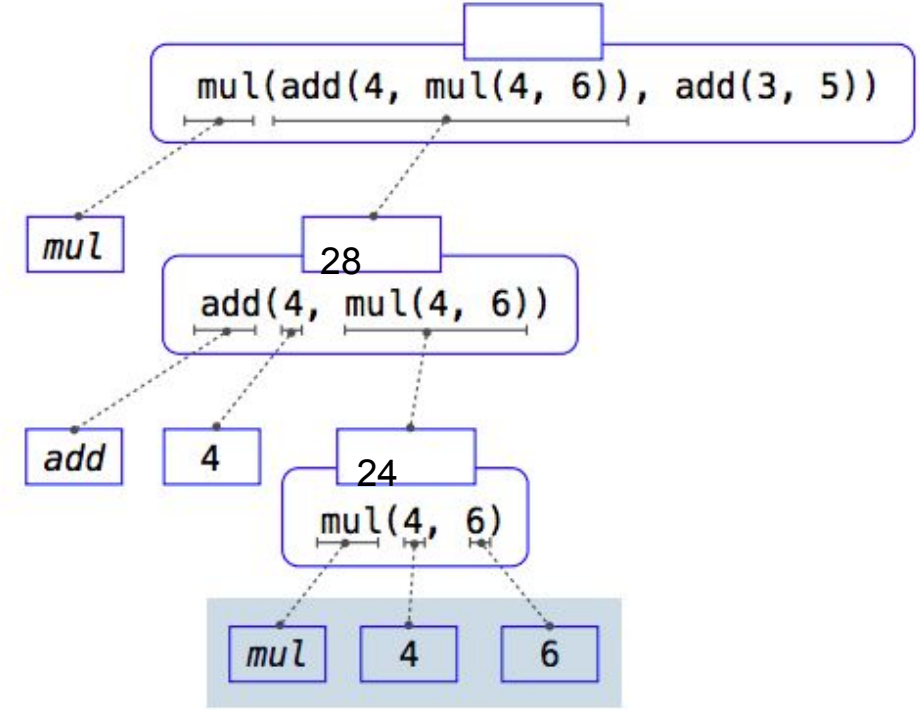
Operators and Operands are also Expressions
They evaluate to values

Evaluation procedure for call expressions:

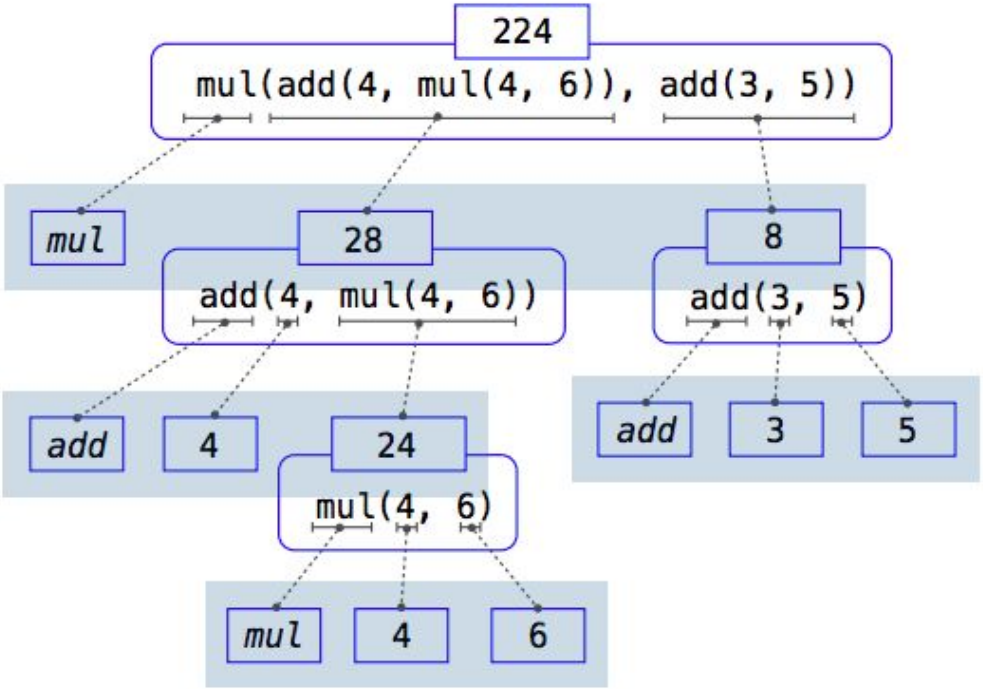
1. Evaluate the *operator* and then the *operand subexpressions*

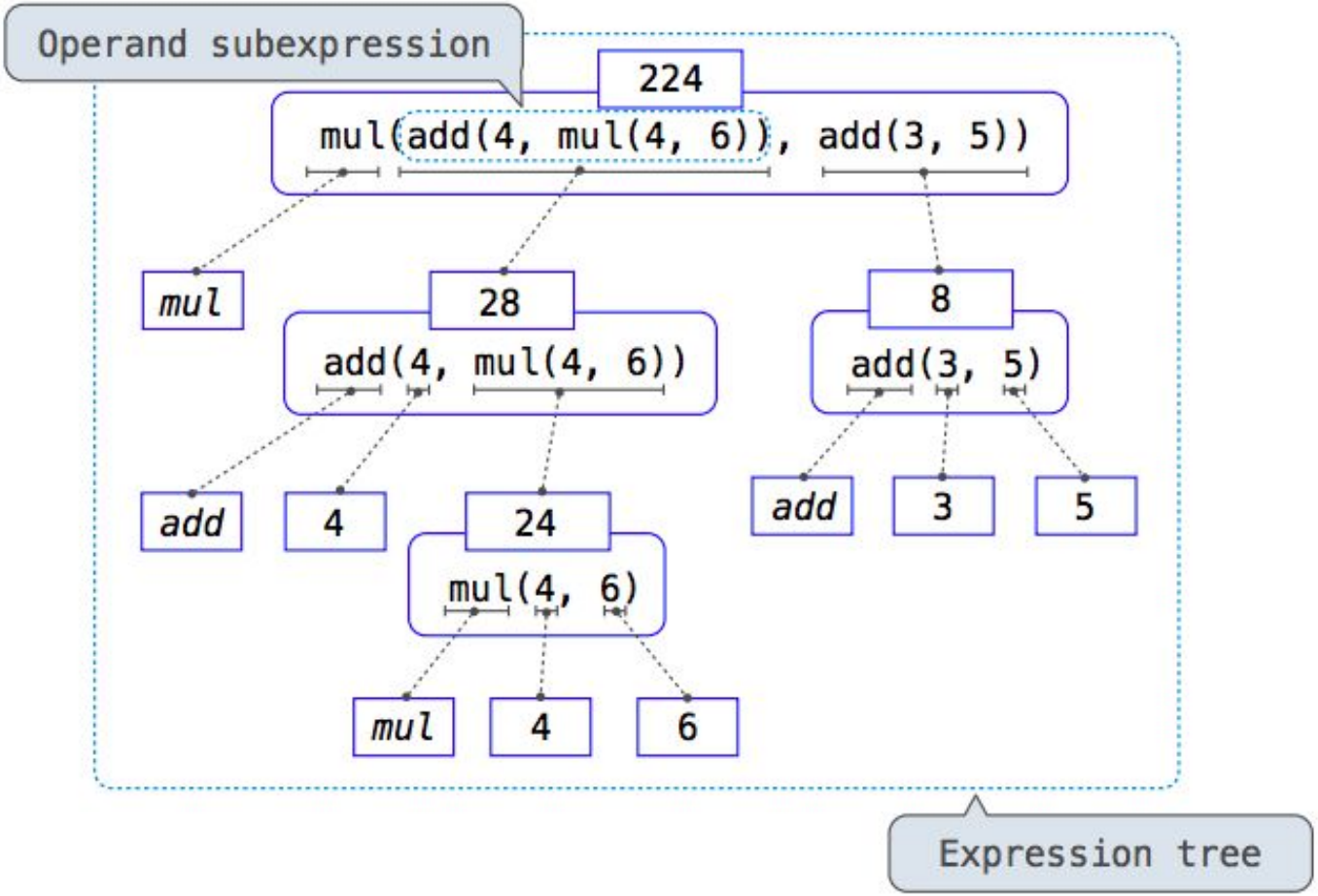
1. Apply the function that is the value of the operator subexpression to the arguments that are the values of the operand subexpression

Evaluating Nested Expressions



Evaluating Nested Expressions




Operand subexpression
224
mul(add(4, mul(4, 6)), add(3, 5))
mul
28
add(4, mul(4, 6))
8
add(3, 5)
add
4
24
mul(4, 6)
add
3
5
mul
4
6
Expression tree

The non-pure print function (demo)

Question

```
print(print("Hello"), print("World"))
```

What does the code above print?

A: Hello
World

B: Hello World

C: Hello World None

D: Hello
World
None None

E: None of the above

The non-pure functions

- None indicates that *Nothing* is returned

- The special value None represents nothing in Python
- A function that does not explicitly return will return None
- Note: None is *not* displayed by the interpreter as the value of an expression

The non-pure functions

None is *not* displayed by the interpreter as the value of an expression

```
def add_me_not (a, b):
      a+b
```

```
add_me_not (1, 2)
```

```
three = add_me_not (1, 2)
three + 17
```

A:	None	20
B:	None	None
C:	nothing is printed	20
D:	3	20
E:	None of the above	

The non-pure print function

None is **not** displayed by the interpreter as the value of an expression

```
>>> def add_me_not (a, b):
...     a+b
...
>>> add_me_not(1, 2)
>>>
```

```
>>> three = add_me_not(1, 2)
>>> three+4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'NoneType' and 'int'
>>>
```

Pure functions and non-pure functions

Pure functions

Just return the values

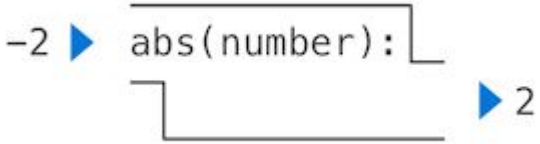
Non - pure functions

Have side effects

Pure functions and non-pure functions

Pure functions

Just return the values



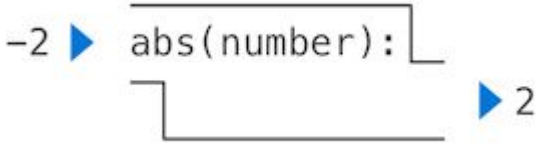
Non - pure functions

Have side effects

Pure functions and non-pure functions

Pure functions

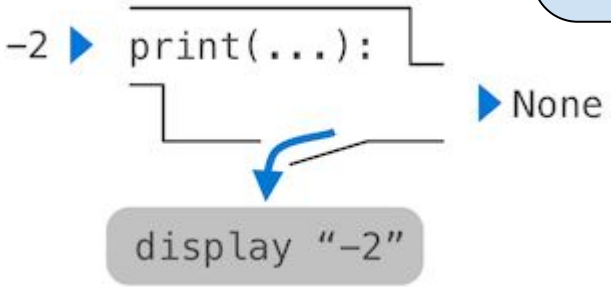
Just return the values



A side effect is not a value. It's anything that happens as a consequence of calling a function.

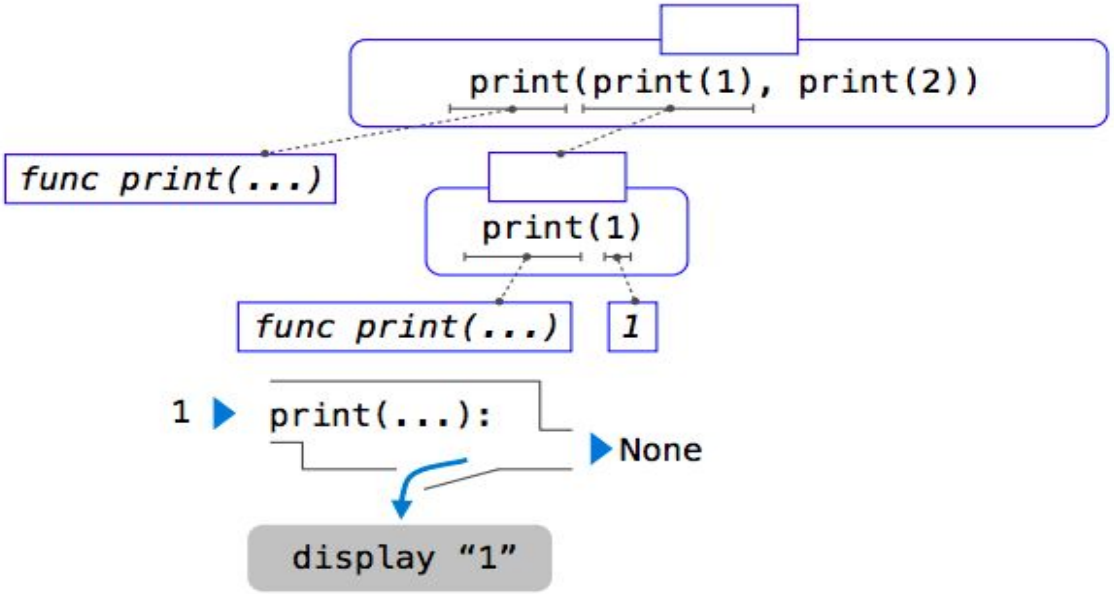
Non - pure functions

Have side effects

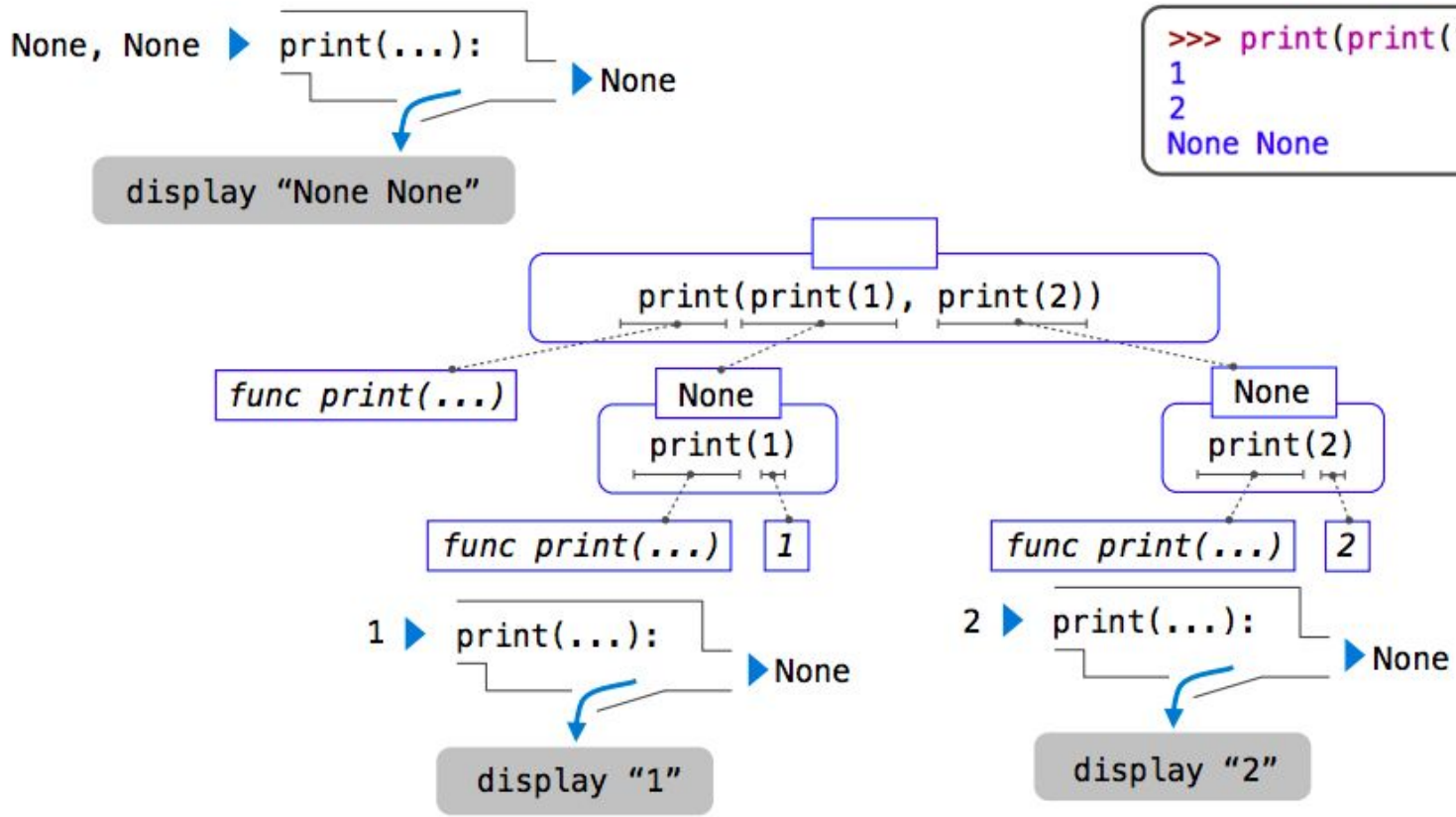


Expression Tree for nested print

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

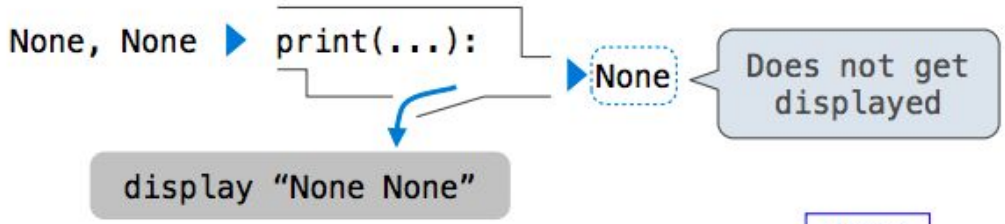


Expression Tree for nested print

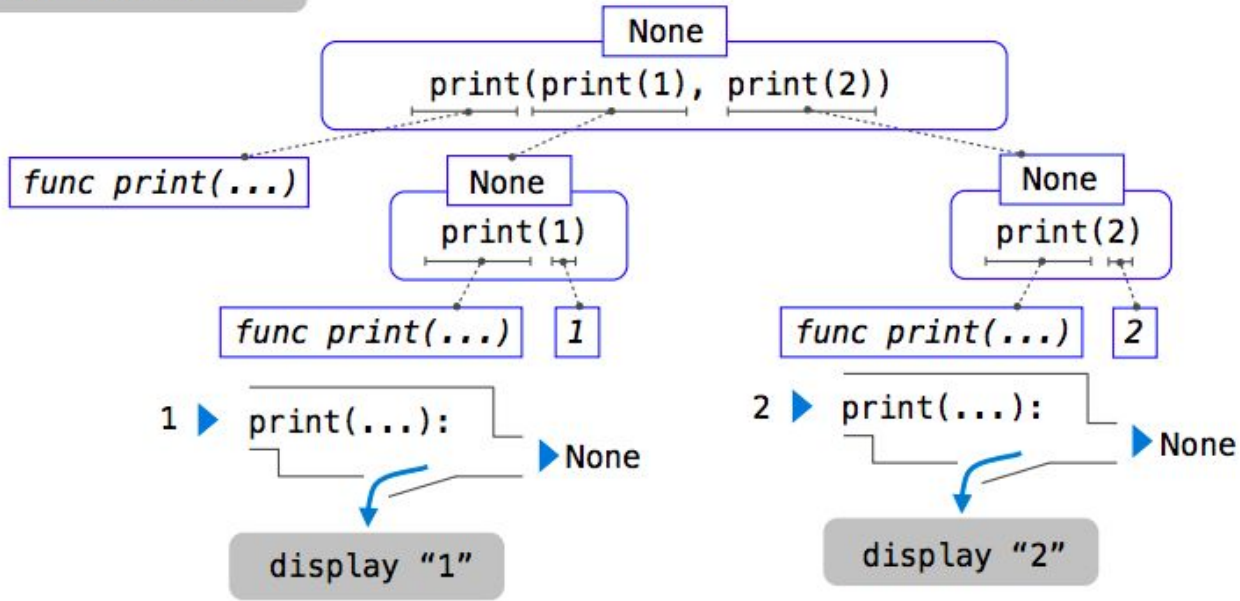



```
>>> print(print(1), print(2))
1
2
None None
```

Expression Tree for nested print




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



i-clicker question

```
def question(n):
    if n > 0:
        return print(n)
    else:
        return


t = question(10)
print(t)
```

What would Python display?

```
A:    10

B:     None

C:    Error of some sort

D:    10
      None

E:     None
       10
```

i-clicker question

```
def question(n):
    if n > 0:
        return print(n)
    else:
        return


t = question(-10)
print(t)
```

What would Python display?

```
A:    -10

B:     None

C:    Error of some sort

D:    -10
      None

E:     None
       -10
```

Short-circuiting

Short-circuiting

```
>>> False  and 1/0
```

A: False

B: True

E: Error

Short-circuiting

```
>>> False  and 1/0
```

A: False

B: True

E: Error

```
>>> (5 and True) or (1/0 or True)
```

A: 5

B: True

C: False

D: Error