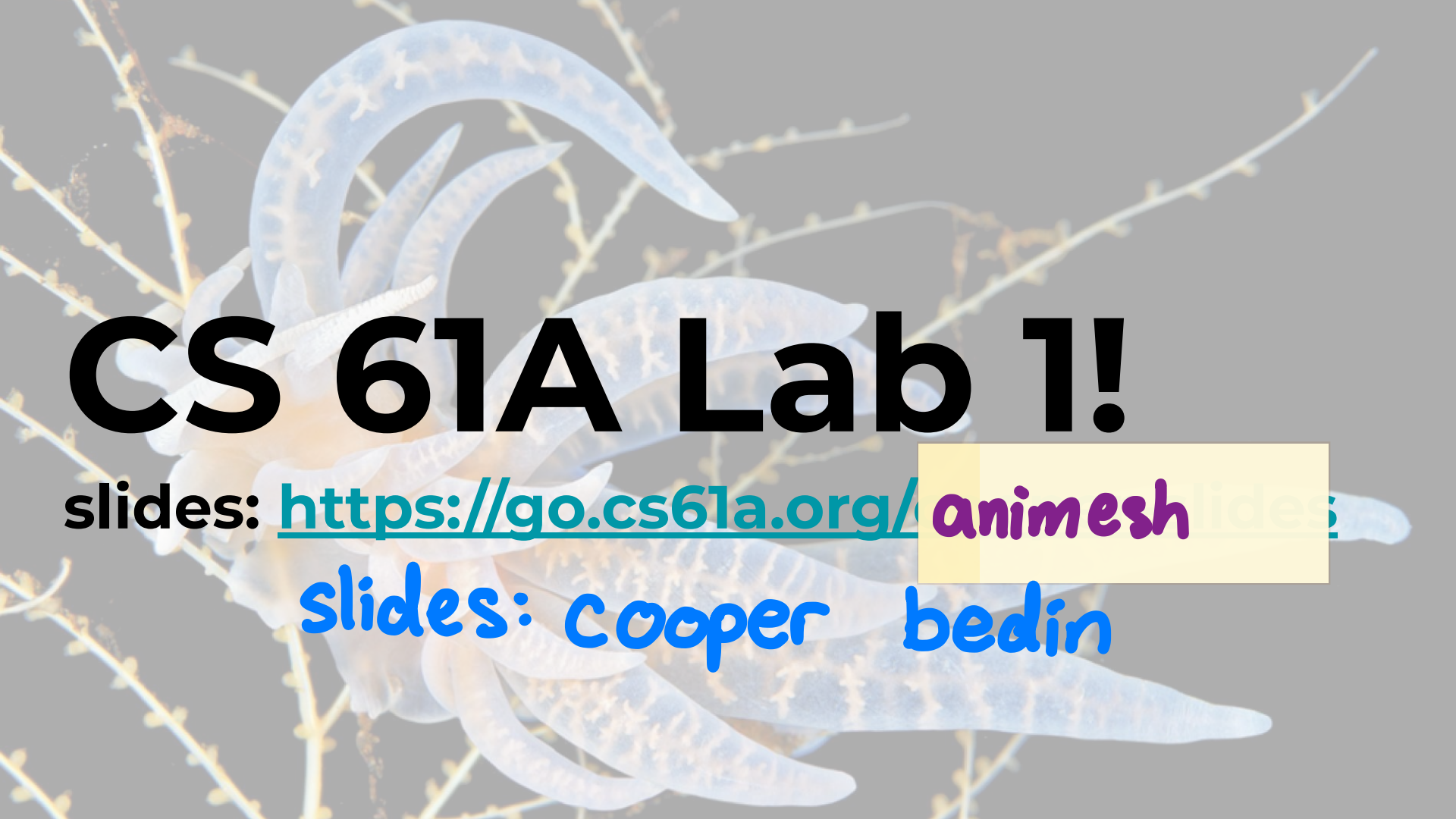


CS 61A Lab 1!

slides: https://go.cs61a.org/animesh

slides: cooper bedin

Announcements/assignments ⏰

- Lab 1 due ~~tomorrow~~ today's, Lab 0 due Thursday
- HW 1 due Thursday
- Fill out the study group matching form if you're interested in being put together with a study group for this class!
- Pls fill out the lab 0 setup survey (also linked in the lab)
- Upcoming staff panels
  - Thursday 5-6 (targeted at students from backgrounds underrepresented in CS)
  - Monday 1-2 (for everyone who wants tips on how to succeed in the class)
  - See Piazza for more details

Lab plan + policies

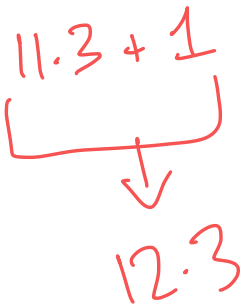
- I'll start each lab with a quick presentation on the topics on the lab (which will hopefully help you out a bit)
- Then the rest of the time is yours—feel free to work by yourself or in groups and I'll be available for questions office-hours style
- I'll take attendance in lab the same way I do in discussion—I'll project a link and QR code when I'm done presenting and then you fill out a quick google form to get the point; there will also be a password given out in lab that you should not share with anyone who isn't in the room
- If you finish the lab early, you're welcome to leave early!
- If you finish the lab assignment before lab section, please still show up to get your attendance point—you're welcome to use this space as a study hall, or dip out once you get your point
- I am totally fine with you asking questions in lab that are not about the lab (homeworks, projects, etc.), but if there are a lot of questions I may prioritize the folks who are actively working on the lab

Expressions and Evaluation Rules

Expressions vs Values

Expressions

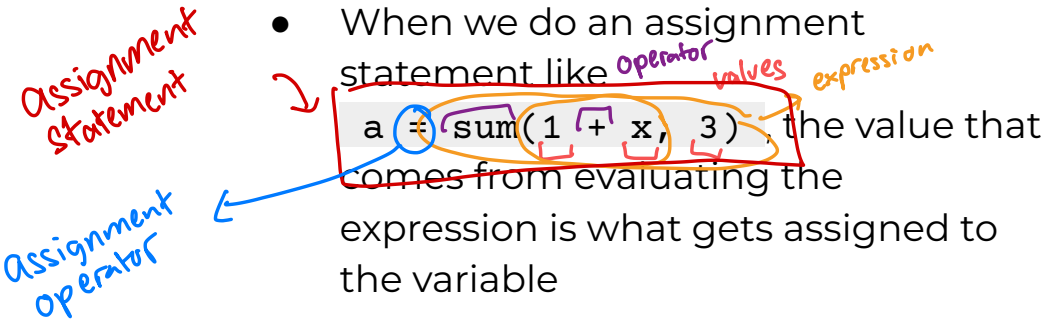
- Some piece of code that can be *evaluated* to a value
- E.g. `sum(1 + x, 3)`



Values

- Some piece of data that probably came from an expression
- E.g. `12.3` , `True`

- When we do an assignment statement like `a = sum(1 + x, 3)` , the value that comes from evaluating the expression is what gets assigned to the variable



Function evaluation

sum(1, 1/0)

1. Evaluate operator → thing that does the work
2. Evaluate operands → things that the operator operates on
3. Apply operator to operands

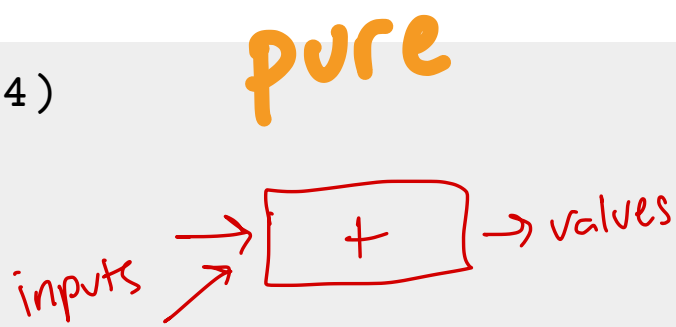
sum(1,3)
~~qqq(1,3)~~

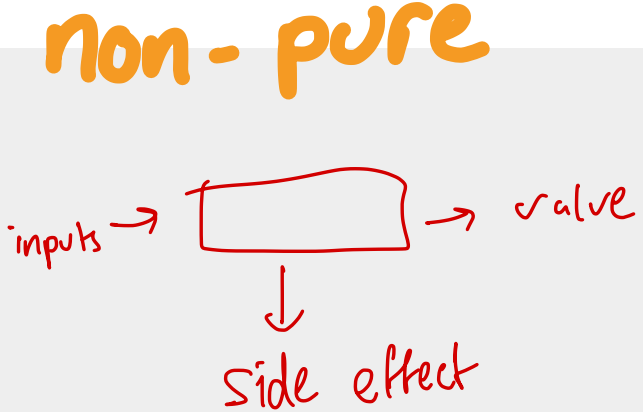
- Any one of these steps may itself involve a function call
- Looking up the binding of a name is a kind of a evaluation

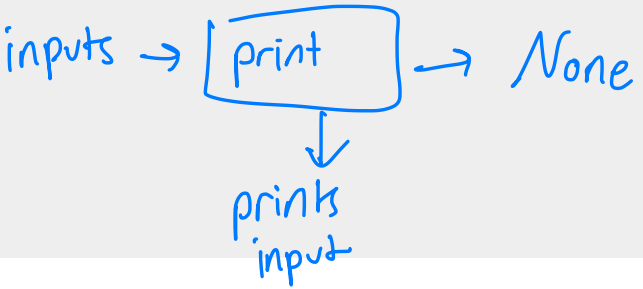
`sum(`	`1 + 4`	`3)`
operator	operand 1	operand 2
⇒ func sum(...) [p=G]	⇒ 5	⇒ 3

So what's up with this?

```
>>>x=print(4)
4
>>>y=4
4
>>> x
None
>>> y
4
```





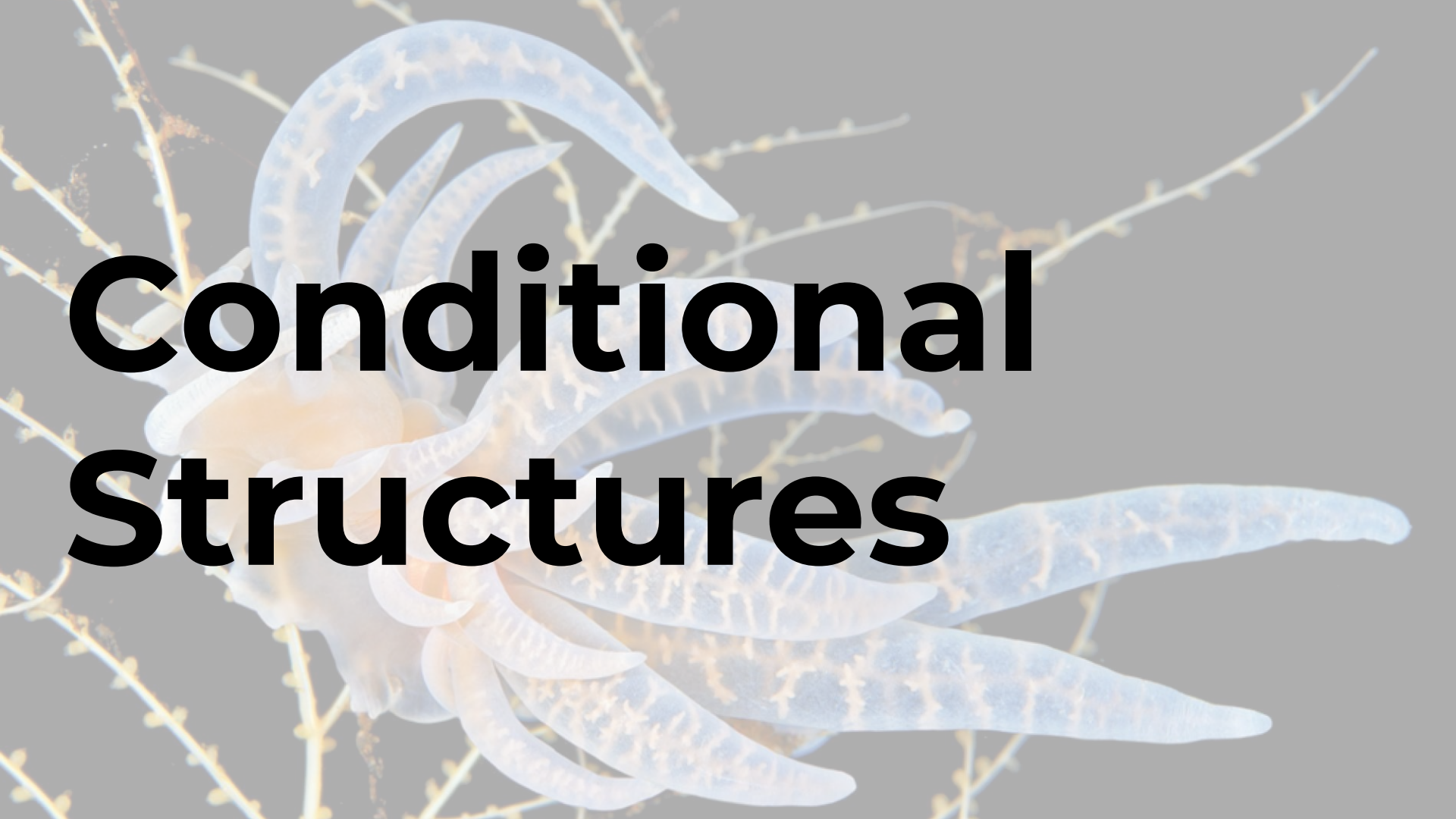


So what's up with this? (cont'd)

```
def foo(x):
    print(x)
    return x + 4

def bar(x):
    x
    return x + 4
```

```
>>> foo(3)
3
7
>>> bar(3)
7
```

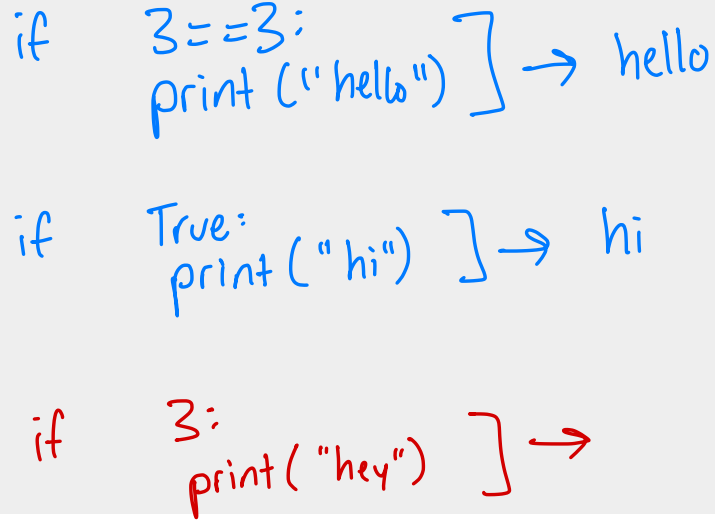


Conditional Structures

If and while

```
if <predicate>:
    <a bunch of statements>
```

```
while <predicate>:
    <a bunch of statements>
```



Bunch of ifs vs using elif

```
if 2 < 3:
    print(3)
elif 2 < 4:
    print(4)
# output
3
```

```
if 2 < 3:
    print(3)
if 2 < 4:
    print(4)
# output
3
4
```

While loop that executes k times

```
count = 1
while count <= k:
    <whatever you need to happen>
    count += 1 # equivalent to count = count + 1

# this is a very useful template!
```

Truthiness and Falsiness

"hello" → T
"" → F

Truthy values

- Will be treated as practically “True” in boolean contexts (if statements, and statements, or statements, etc.)
- Nonzero numbers, non-empty strings, non-empty lists, etc. are all truthy

3 → "True"
4 → "True"

Falsy values

- Will be treated as practically “False” in boolean contexts
- 0, “” (empty string), [] (empty list), None, etc. are all falsy

3 == True
↓
false

C

While loop that executes k times, using this

```
while k:
    <whatever you need to happen>
    k = k - 1

# this is a very useful template!
```

Short-circuiting

```
True or 1 / 0  # will short-circuit
```

1) Evaluate left
- T → Return left
- F → evaluate right
  - T → Return right
  - F → Return right

vs

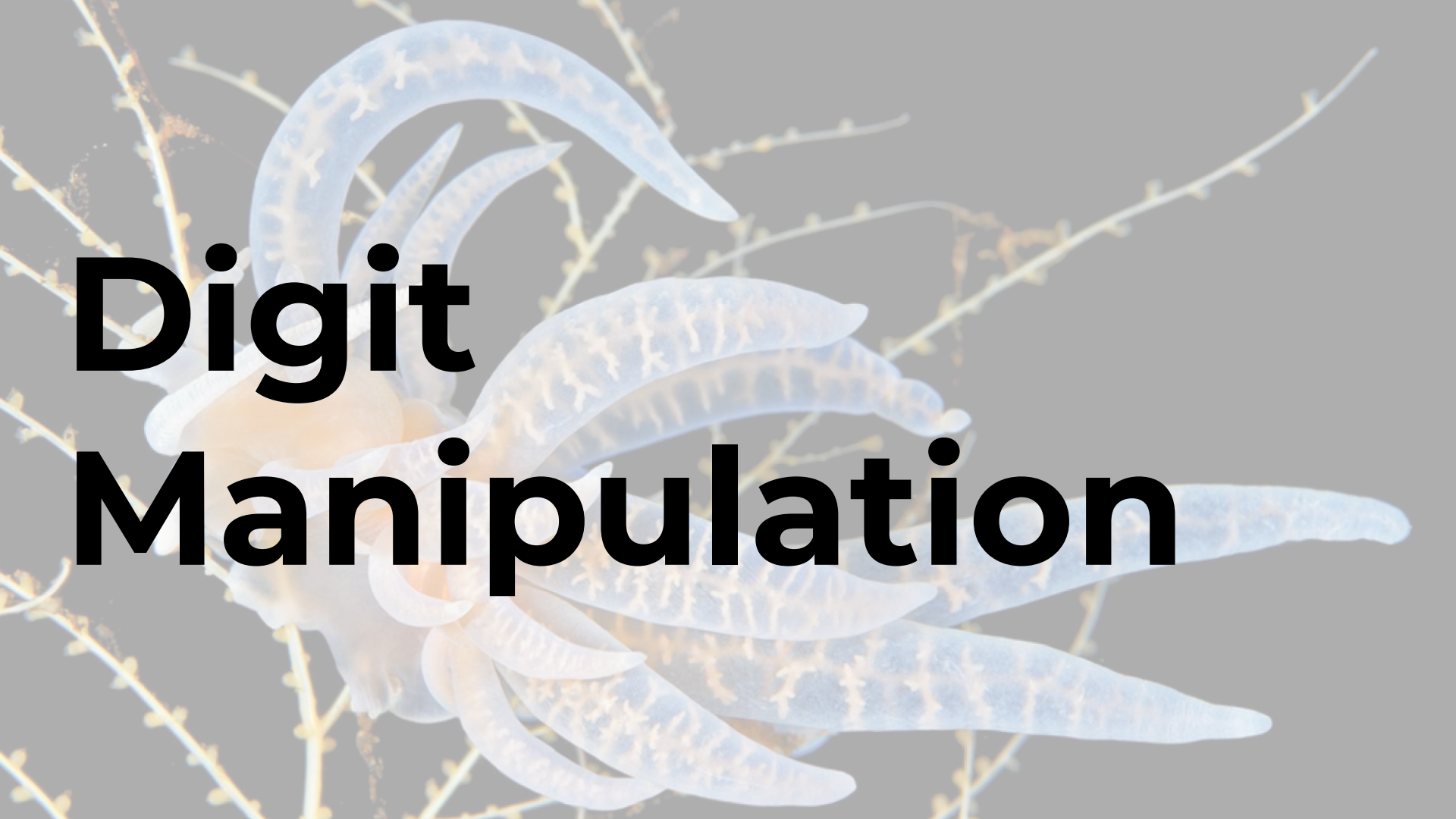
```
False or 1 / 0  # will not short-circuit
```

```
True and 1 / 0  # will not short-circuit
```

vs

```
False and 1 / 0  # will short-circuit
```

evaluate left
- T → evaluate right
  - T → return right
  - F → return right
- F → return left



Digit Manipulation

// vs % ????

```
>>> 1024 // 10
102
```

```
>>> 1024 % 10
4
```

Iteratively prints the digits of a number

```
temp = n
while n:
    print(n % 10) # isolates the last digit
    n //= 10 # reduces the number
    # equivalent to n = n // 10
    # n = n + 1 ⇒ n += 1
n = temp

# this is good stuff! use this template as much as you can!
```

Attendance! https://go.cs61a.org/animesh

11 rooms

↳ join your room for problem

Ex: Q4 → room 4

1 and 3 and 5 → 5
↳ True

if 3 and 5:

Eval L
"Truthy" "Falsy"

(1 and 3) + (3 and 5)

Eval R
"Truthy" "Falsey"

3 + 5

8

True + True

Error

return R

return R

n % 10

923 % 10

↓

3

n // 10

923 // 10

↓

92

92 // 10

↓

9