



THE UNIVERSITY OF
CHICAGO

**MASTERS IN
COMPUTATIONAL
SOCIAL SCIENCE**
THE UNIVERSITY OF CHICAGO

MACS 30111

List, Tuples, and Strings

Misc: Assignment deadlines

- ▶ SE2 due FRIDAY 1/17
- ▶ PA1 due 1/24
 - ▶ **START NOW!!!! Work on it in chunks!!**
- ▶ PA1 REFLECTION due 1/27
 - ▶ Grading about whether you critically engaged with where you are / the assignment
 - ▶ NO: 'it went ok'
 - ▶ Yes: it was OK but I struggled with part 3 because *reason*



KWL

Know (learned from readings)	Want to know (questions you have)	Learned (leave blank for now)

Topics:

- ❑ Pythonic
- ❑ Introduction
- ❑ List creation and basic usage
- ❑ List iteration
- ❑ Adding, removing elements from a list
- ❑ List slicing
- ❑ Lists of lists
- ❑ Tuples
- ❑ Strings
- ❑ List Comprehensions
- ❑ Lists in Memory (Probably Thursday)

Pythonic: what does it mean?

- ↳ Clean
- ↳ Beautiful
- ↳ Correct
- ↳ Alternative: *brute force*

Lists, Tuples, and Strings

Basic data types: integers, floats, strings, and booleans.

With these data types, a variable only contains a single value.

```
In [1]: n = 5
```

```
In [2]: n
```

```
Out[2]: 5
```

Lists, Tuples, and Strings

Construct more complex data structures from basic data types.

```
In [1]: for n in [1, 4, 8, 9, 11]:  
...:     print(n)  
...:
```

```
1  
4  
8  
9  
11
```

```
numbers = [1, 4, 8, 9, 11]
```

Variable *numbers* contains a list of five integers.

Topics:

- ❑ Introduction
- ❑ **List creation and basic usage**
- ❑ List iteration
- ❑ Adding, removing elements from a list
- ❑ List slicing
- ❑ Lists of lists
- ❑ Tuples
- ❑ Strings
- ❑ List Comprehensions
- ❑ Lists in Memory

Quiz

Which of the following is NOT a valid way to create a list?

- `lst = []`
- `lst = [1, 2, 3, 4]`
- `lst = 1 + 2 + 3 + 4`
- `lst = [0, 1] * 10`

True/False: In Python, all the elements of a list must be of the same type?

Creating Lists

```
lang = ["C", "C++", "Python", "Java"]
```

- ▶ Creating a literal list
- ▶ Creating an empty list
- ▶ Creation by concatenation
- ▶ Creation by multiplication

A pair of square brackets
Values separated by
comma

Coding practice: 2.1.1

Creating Lists

Coding practice: 2.1.1

- ▶ How are the following different, if at all?
 - ▶ `lst = [0, 1] * 10`
 - ▶ `lst = [0] * 10 + [1] * 10`
 - ▶ `lst = [0, 1, 0, 1, 0, 1, 0, 1, 0, 1]`

Basic Usage: code used

```
lang = ["C", "C++", "Python", "Java"]
```

- ▶ List length
- ▶ Accessing elements in a list
- ▶ Assigning a value to an element of the list
- ▶ Negative indices

Coding practice: 2.1.2

Basic Usage: code used

```
lang = ["C", "C++", "Python", "Java"]
```

- ▶ List length: `len(lang)`
- ▶ Accessing elements in a list: `lang[0]`
- ▶ Assigning a value to an element of the list: `lang[0] = "perl"`
- ▶ Negative indices `lang[-1]`

Coding practice: 2.1.2

Code snippet

```
lang = ['C', 'C++', 'Python 3', 'Java']
```

```
len(lang)
```

```
lang[2]
```

```
lang[5] Throws an error! But why?
```

```
lang[0] = "perl"
```

```
lang[-1]
```

Topics:

- ❑ Introduction
- ❑ List creation and basic usage
- ❑ **List iteration**
- ❑ Adding, removing elements from a list
- ❑ List slicing
- ❑ Lists of lists
- ❑ Tuples
- ❑ Strings
- ❑ List Comprehensions
- ❑ Lists in Memory

Quiz

How do I iterate over the values in a list?

- Using a “foreach” loop
- Using a “for” loop
- Using the built-in `iterate()` function

Which of the following loops is unpythonic?

- `for i in range(len(lst)):`
- `for x in lst:`
- `for i, x in enumerate(lst):`

List Iteration

iterate through the list and perform an action for each element in the list

```
In [1]: for n in [1, 4, 8, 9, 11]:  
...:     print(n)  
...:
```

```
1  
4  
8  
9  
11
```

enumerate()

Iterate the list over the indices **unpythonic**

```
for i in range(len(prices)):
    print("Item", i, "costs", prices[i], "dollars")
```

Python provides a way to iterate the list over the indices and values directly with the built-in **enumerate** function:

```
for i, p in enumerate(prices):
    print("Item", i, "costs", p, "dollars")
```

← unpack as (index, value) tuples

Coding practice: 2.1.3

Applied practice

- ▶ Create a list counting by three starting at 0 and going to 60 (inclusive)
 - ▶ `nums = list(range(0,61, 3))`

Topics:

- ❑ Introduction
- ❑ List creation and basic usage
- ❑ List iteration
- ❑ **Adding, removing elements from a list**
- ❑ List slicing
- ❑ Lists of lists
- ❑ Tuples
- ❑ Strings
- ❑ List Comprehensions
- ❑ Lists in Memory

Quiz

Does `append()` return a new list?

- No, it modifies the list in-place
- Yes, it leaves the list intact, and returns a new list with the appended value.

Which of the following functions will remove an element from a list?

- `extract`
- `pop`
- `excise`

Adding elements to a list

- ▶ *append()*
- ▶ *extend()*
- ▶ The + operator
- ▶ *insert()*
- ▶ In-place vs returning a new list (*id()*)

Coding practice: 2.1.4

Removing elements from a list

- *pop()* (remove by position)
- *remove()* (remove by value)
- Built-in operator *del*

Coding practice: 2.1.4

Topics:

- ❑ Introduction
- ❑ List creation and basic usage
- ❑ List iteration
- ❑ Adding, removing elements from a list
- ❑ List slicing
- ❑ Lists of lists
- ❑ Tuples
- ❑ Strings
- ❑ **List Comprehensions**
- ❑ Lists in Memory

List Comprehension

List comprehensions are more compact ways to generate a list

**<list name> = [<transformation expression> for
<variable name> in <list expression>]**

Note: for this, you can add conditionals but the formatting gets a little weird:

**<list name> = [<transformation expression> for <variable name> in
<list expression> if <condition>]**

BUT

**<list name> = [<transformation expression> if <condition> else
<transformation expression> for <variable name> in <list
expression>]**

List Comprehension

Given a list of integers, create a *new* list with those same numbers multiplied by 2.

```
lst = [1,2,3,4]
new_lst = []
for x in lst:
    new_val = x*2
    new_lst.append(new_val)
```

new_lst

A compact syntax using list comprehensions:

```
lst = [1,2,3,4]
new_lst2 = [x*2 for x in lst]
```

Coding practice: 2.1.5

List Comprehension

Given a list of integers, create a *new* list with those same numbers multiplied by 2.

```
lst = [1, 2, 3, 4]
```

```
new_lst = []
```

```
for x in lst:  
    new_val = x*2  
    new_lst.append(new_val)
```

```
<list name> = [ <transformation expression> for <variable name>  
                in <list expression> ]
```

```
new_lst = [x*2 for x in lst]
```

New List

Variable

Existing List

Expression

List Comprehension

List comprehensions are more compact ways to generate a list

**<list name> = [<transformation expression> for
<variable name> in <list expression>]**

Note: for this, you can add conditionals but the formatting gets a little weird:

**<list name> = [<transformation expression> for <variable name> in
<list expression> if <condition>]**

BUT

**<list name> = [<transformation expression> if <condition> else
<transformation expression> for <variable name> in <list expression>]**

List Comprehension

Create a new list from an existing list, but filtering elements based on some condition.

For example:

```
lst = [1,2,3,4]
```

```
new_lst = []
```

```
for x in lst:
```

```
    if x % 2 == 0:
```

```
        new_lst.append(x)
```

```
new_lst
```

We can use a **list comprehension** for this too:

```
new_lst = [x for x in lst if x % 2 == 0]
```

```
new_lst
```

```
[ <transformation expression> for <variable name> in <list expression> if <boolean expression> ]
```

Applied practice

- ▶ Create a list counting by three starting at 0 and going to 60 (inclusive)
- ▶ Create a new list using this original list: square even numbers and make odd numbers negative
 - ▶ One partner does it the 'long' way and one try it with a list comprehension

Topics:

- ❑ Introduction
- ❑ List creation and basic usage
- ❑ List iteration
- ❑ Adding, removing elements from a list
- ❑ **List slicing**
- ❑ Lists of lists
- ❑ Tuples
- ❑ Strings
- ❑ List Comprehensions
- ❑ Lists in Memory

List slicing

Use the brackets operator to access individual elements of a list:

```
In [1]: lang = ['C', 'C++', 'Python', 'Java']
```

```
In [2]: lang[2]
```

```
Out[2]: 'Python'
```


List slicing

Specify a range of positions: specifying two indexes separated by a colon:

```
In [1]: lang = ['C', 'C++', 'Python', 'Java']
```

```
In [2]: lang[1:3]
```

```
Out[2]: ['C++', 'Python 3']
```

- A slice is a copy that doesn't refer back to the original list
- Omitting slice operands
- `[:]` as a way to copy lists
- Step through the list

Coding practice: 2.1.2.1

Other operations

↳ *[::]* to pull out based on index patterns

```
new_list = [x**2 for x in range(0,30)]
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225, 256, 289, 324, 361, 400, 441, 484, 529, 576, 625, 676, 729, 784, 841]
```

```
new_list[::2]
```

```
[0, 4, 16, 36, 64, 100, 144, 196, 256, 324, 400, 484, 576, 676, 784]
```

```
new_list[1::2]
```

```
[1, 9, 25, 49, 81, 121, 169, 225, 289, 361, 441, 529, 625, 729, 841]
```

KWL

Know (learned from readings)	Want to know (questions you have)	Learned (fill in now)

Topics:

- ❑ Introduction
- ❑ List creation and basic usage
- ❑ List iteration
- ❑ Adding, removing elements from a list
- ❑ List Comprehensions
- ❑ **List Operations**
- ❑ **Lists of lists**
- ❑ **Tuples**
- ❑ **Strings**
- ❑ **Lists in Memory**



Misc

- Workshop: Will try to end by 10:45
- Ed: Good use!
- Check in: How comfortable do you feel with the following:
 - pytest
 - Calling your code in an interpreter
 - Testing out code snippets
 - How often do you ****actually**** do testing, etc (be honest!)

Quiz

Which of the following specifies a **slice** of a list?

- `lst[4-7]`
- `lst[4..7]`
- `lst[4:7]`
- `lst[4,7]`

If I create a slice of a list, and then modify a value in the slice...

- The contents of the original list are unaffected
- The contents of the original list are changed as well

Other operations

- ↵ *min()*
- ↵ *max()*
- ↵ *sum()*
- ↵ *count()*
- ↵ *in*
- ↵ *reverse()*
- ↵ *sort()* VS *sorted()*

Operator comparison

- ↳ Start with a list: 0-100 by 5s.
- ↳ Provide the following:
 - ↳ Write two ways to find out if 15 is in the list
 - ↳ Write two ways to remove the number 10
 - ↳ Write two ways to reverse the list

Topics:

- ❑ Introduction
- ❑ List creation and basic usage
- ❑ List iteration
- ❑ Adding, removing elements from a list
- ❑ List Comprehensions
- ❑ List slicing
- ❑ **Lists of lists**
- ❑ Tuples
- ❑ Strings
- ❑ Lists in Memory

Quiz

True/False: A list can contain other lists, but I need to specify the sub-lists with curly braces (e.g., `m = [{1,2,3}, {4,5,6}]`)

If I want to treat a list of lists like a matrix...

- It is up to me to ensure it is a valid matrix. Python won't enforce matrix semantics.
- Python will enforce matrix semantics, as long as the variable name starts with the letter "m"
- Python will enforce matrix semantics automatically if all the lists are of the same length, and if they all contain a numeric type (integer or float)

Lists of lists

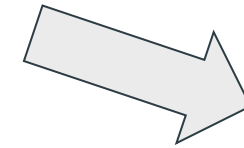
So far, we have seen lists containing simple values:

```
lst = [1, 2, 3, 4]
```

However, lists can also **contain other lists**:

```
m = [ [1,2,3,4], [5,6,7,8], [9,10,11,12] ]
```

This is a common way of representing matrix-like data.



1	2	3	4
5	6	7	8
9	10	11	12

Lists of lists

- ↩ access individual elements: use square brackets twice
- ↩ assign individual elements
- ↩ list-of-lists-of-lists

1	2	3	4
5	6	7	8
9	10	11	12

Matrix party

- ▶ Import random and set your seed to 4:
`random.seed(4)`
- ▶ Create a matrix: 3 x 3 matrix full of random odd integers that range from 0 to 11 (inclusive)
- ▶ What is the middle row?
- ▶ Check that the second element of the middle row is >5
- ▶ Replace ****in place**** all 5s with 7s

Lists: HELP!

⤵ Help me determine what is wrong here:

```
m = [[3, 3, 7], [7, 11, 11], [7, 1, 9]]
```

```
n = []
```

```
for i in m:  
    print(m)  
    if m[i] == 3:  
        print(i)  
    n[i] = 6
```

Topics:

- ❑ Introduction
- ❑ List creation and basic usage
- ❑ List iteration
- ❑ Adding, removing elements from a list
- ❑ List slicing
- ❑ Lists of lists
- ❑ **Tuples**
- ❑ Strings
- ❑ List Comprehensions
- ❑ Lists in Memory

Quiz

True/False: Tuples and lists are interchangeable types and behave exactly the same way. The only difference is we use parentheses instead of brackets.

Tuple

A tuple is very similar to a list, except it uses *parentheses* and is *immutable*. Once I create a tuple, I cannot change the values contained in the tuple.

```
tpl = (1, 2, 3, 4)
```

When iterating over a list of tuples, we can have a for loop automatically *unpack* the tuples

```
employees = [ ("Sam", "CEO"), ("Alex", "CTO"),  
              ("Pat", "VP") ]
```

```
for name, position in employees:  
    print(name, "is the", position)
```

Coding practice: 2.1.12

Tuple

- ↳ Can you tell by looking at something if it's a tuple?
- ↳ Why / when might we use them over lists?

Topics:

- ❑ Introduction
- ❑ List creation and basic usage
- ❑ List iteration
- ❑ Adding, removing elements from a list
- ❑ List Comprehensions
- ❑ List slicing
- ❑ Lists of lists
- ❑ Tuples
- ❑ **Strings**
- ❑ Lists in Memory

Quiz

True/False: After I create a string, I can use the brackets operator to change individual characters (e.g., `s[1] = "X"`)

Which of the following is a valid example of string formatting in Python?
(assuming that `x` contains an integer value)

- “The number is \$x”
- “The number is <int>”.format(x)
- “The number is {}”.format(x)

Strings

```
msg = "Hello, world!"
```

- Store text values
- A list of individual characters, most list operations are also available on strings.
- Methods we can invoke: *in*, *find*, *lower*, *upper*, *capitalize*, *replace*, *split*, *join*
- Python mechanisms for formatting strings

Strings: checking it out

- ↳ You can do a lot of the same operations on a string that you can with list (not totally, but a lot of similar ones!)
- ↳ Start with a string and try the following:
 - ↳ `upper()`
 - ↳ `lower()`
 - ↳ `+=`
 - ↳ `print("{} is my favorite".format(var))`
 - ↳ `print(f'{var} is my favorite.')`

Strings and f (formatted) strings

- ↳ To use formatted string literals, begin a string with f or F before the opening quotation mark or triple quotation mark. Inside this string, you can write a Python expression between { and } characters that can refer to variables or literal values.

- ↳ Ex:
- ↳ `year = 2016`
- ↳ `event = 'Referendum'`
- ↳ `f'Results of the {year} {event}'`
- ↳ `'Results of the 2016 Referendum'`

Strings: intermediate

↳ Advice on what to do?

```
s = "WhEn I wAlK in tHE roOm I cAn stILl mAkE  
the wHOle PLace SHIMMER. "
```


Strings: advanced (good OH question!)

↳ Consider the following:

```
s= ["Baby love, I think I've been a little too.  
Kind"], ["Didn't notice you walkin' all over my  
peace of mind"], ["In the shoes I gave you as a  
present"]
```

↳ How can we clean this up?

Topics:

- ❑ Introduction
- ❑ List creation and basic usage
- ❑ List iteration
- ❑ Adding, removing elements from a list
- ❑ List slicing
- ❑ Lists of lists
- ❑ Tuples
- ❑ Strings
- ❑ List Comprehensions
- ❑ **Lists in Memory**

MEMORY!

- ↳ Think about how things are stored - are they made somewhat easily accessible or are they harder to access?

Variables Revisited

Memory

a

42

```
In [1]: a = 42
```

Variables Revisited

Memory

a

42

b

42

```
In [1]: a = 42
```

```
In [2]: b = a
```

Variables Revisited

Memory

a

42

b

42

```
In [1]: a = 42
```

```
In [2]: b = a
```

```
In [3]: a = 37
```

Variables Revisited

Memory

a

37

b

42

```
In [1]: a = 42
```

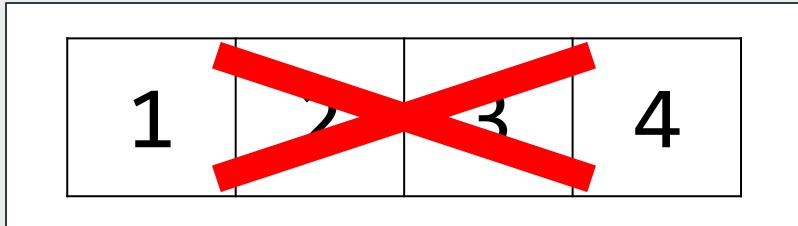
```
In [2]: b = a
```

```
In [3]: a = 37
```

List variables

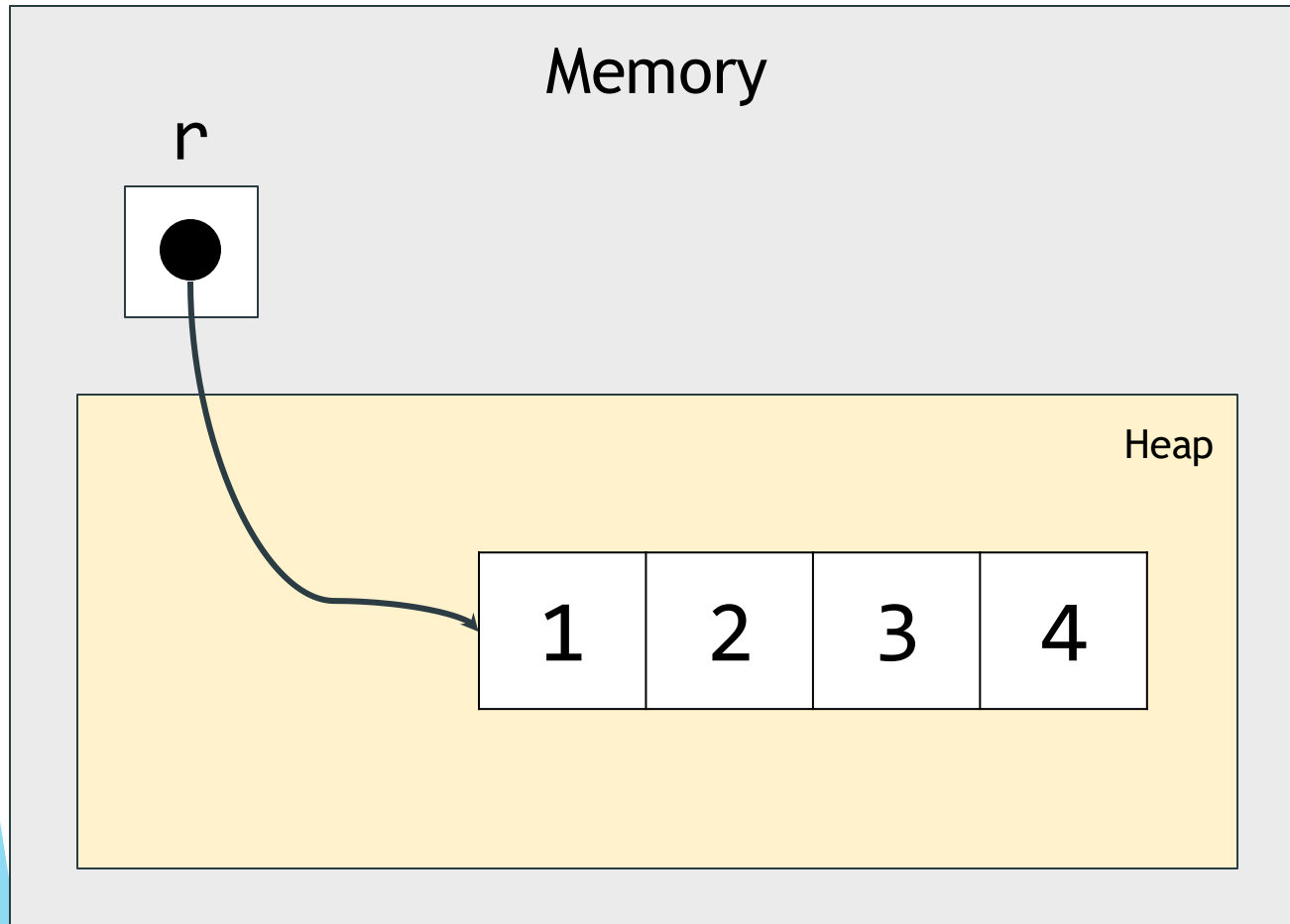
Memory

r



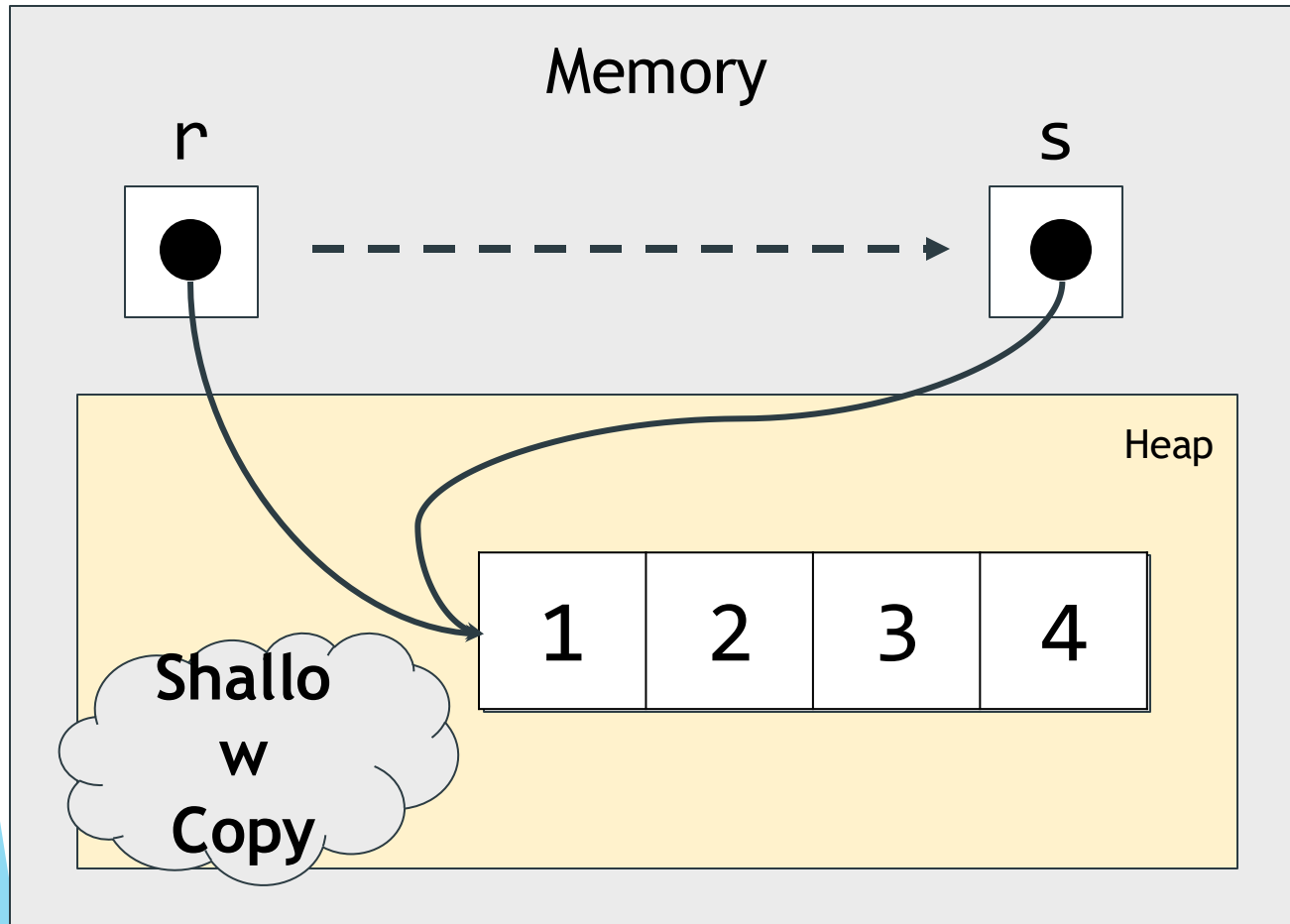
```
In [1]: r = [1, 2, 3, 4]
```


List variables



```
In [1]: r = [1, 2, 3, 4]
```

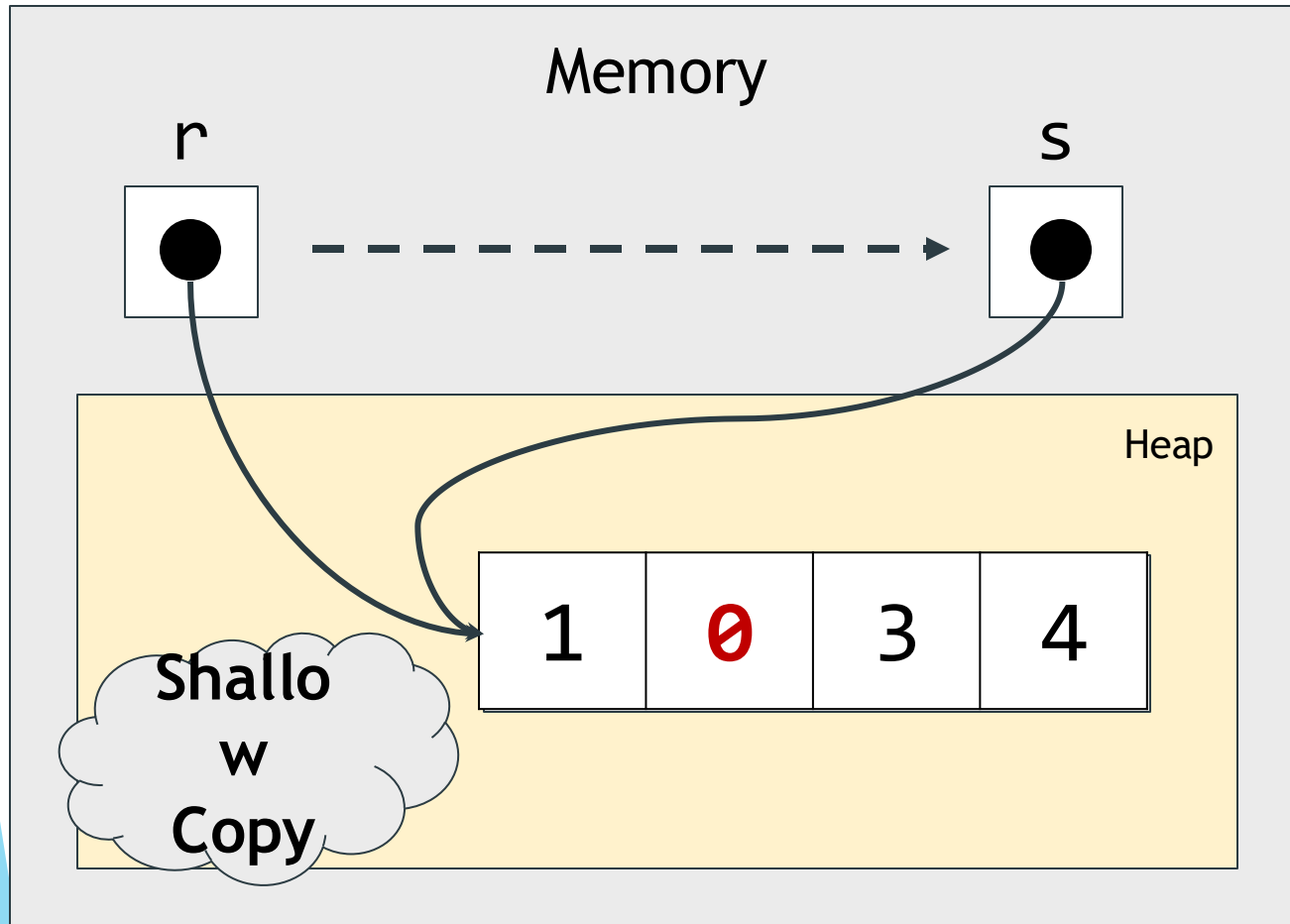
List variables



```
In [1]: r = [1, 2, 3, 4]
```

```
In [2]: s = r
```

List variables



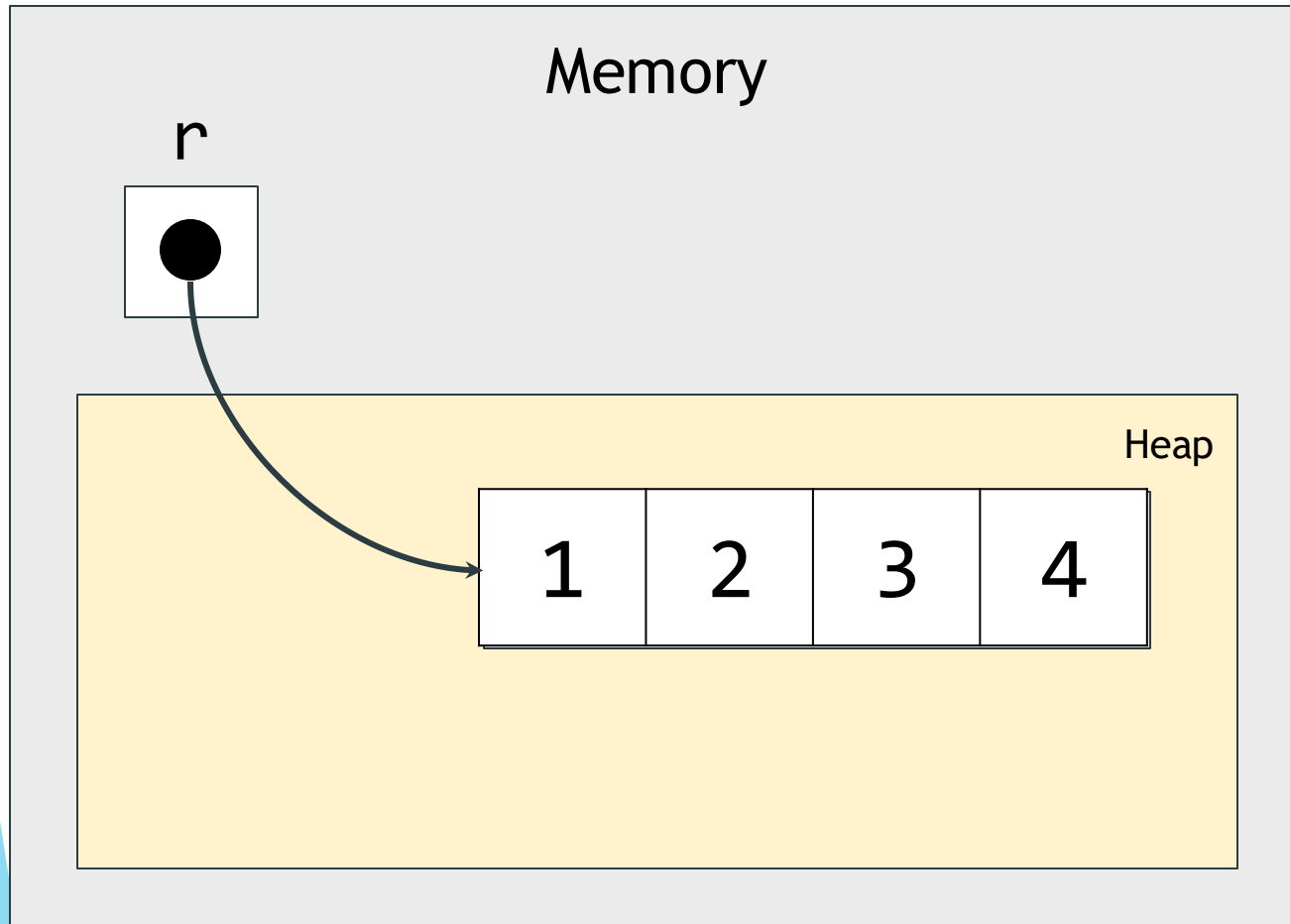
```
In [1]: r = [1, 2, 3, 4]
```

```
In [2]: s = r
```

```
r[1] = 0
```

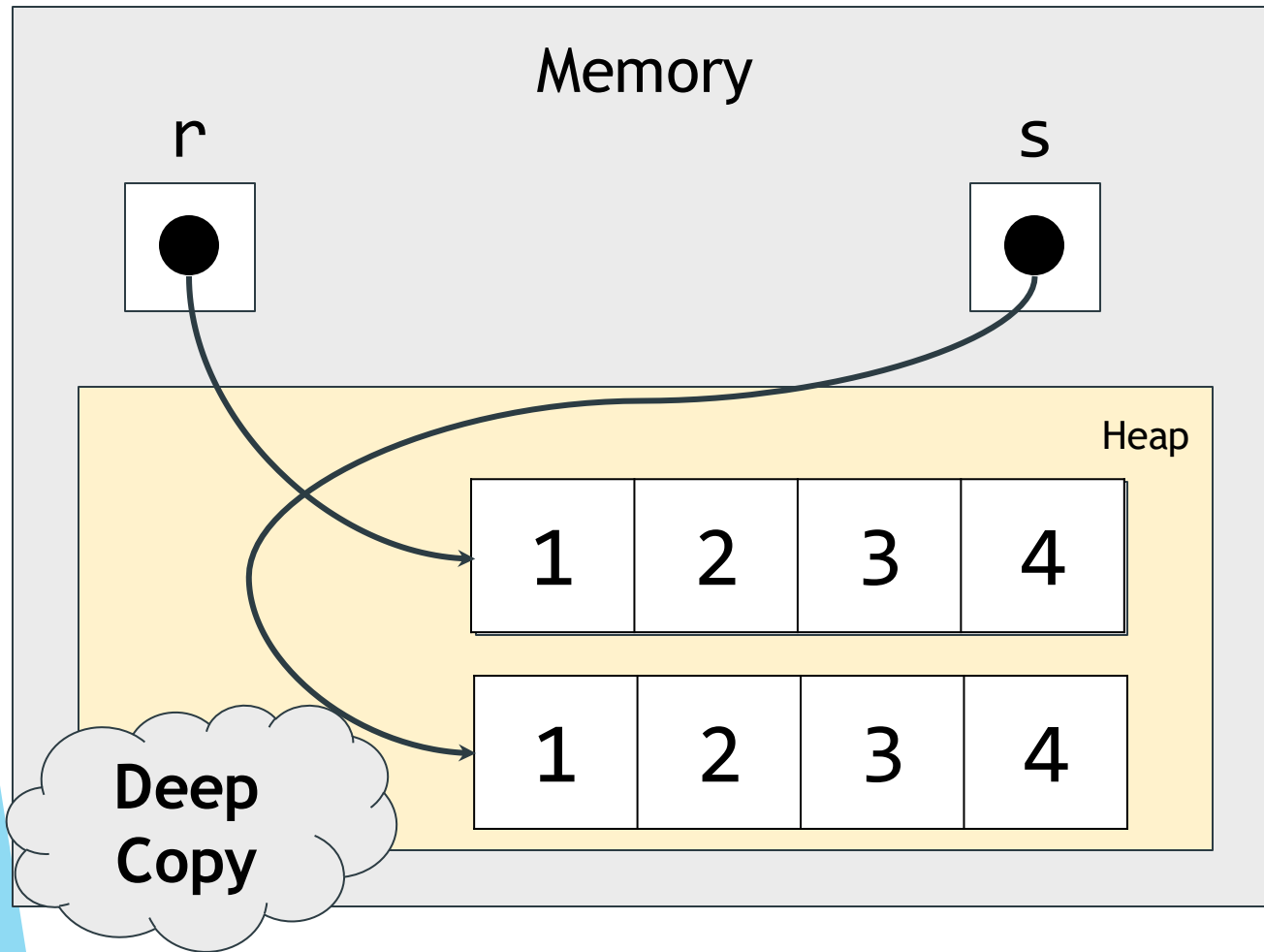
1	r
[1, 0, 3, 4]	
1	s
[1, 0, 3, 4]	

List variables



```
In [1]: r = [1, 2, 3, 4]
```

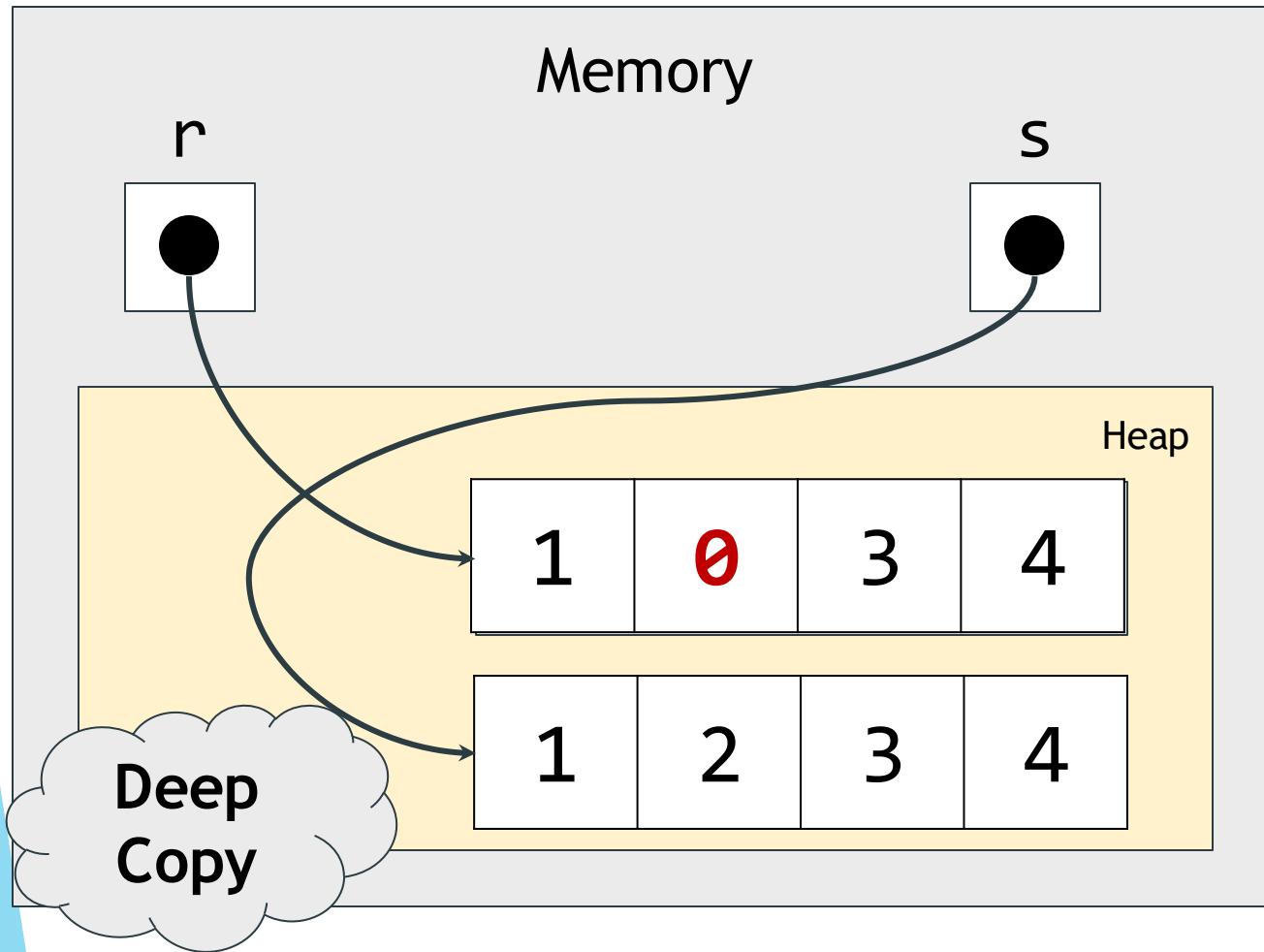
List variables



```
In [1]: r = [1, 2, 3, 4]  
In [2]: s = r[:]
```

Can think of this as a 'clone' if that's helpful!

List variables



```
In [1]: r = [1, 2, 3, 4]
```

```
In [2]: s = r[:]
```

```
r[1] = 0
```

1	r
[1, 0, 3, 4]	

1	s
[1, 2, 3, 4]	

Lists of lists: where it gets weird

↓ Test out the following:

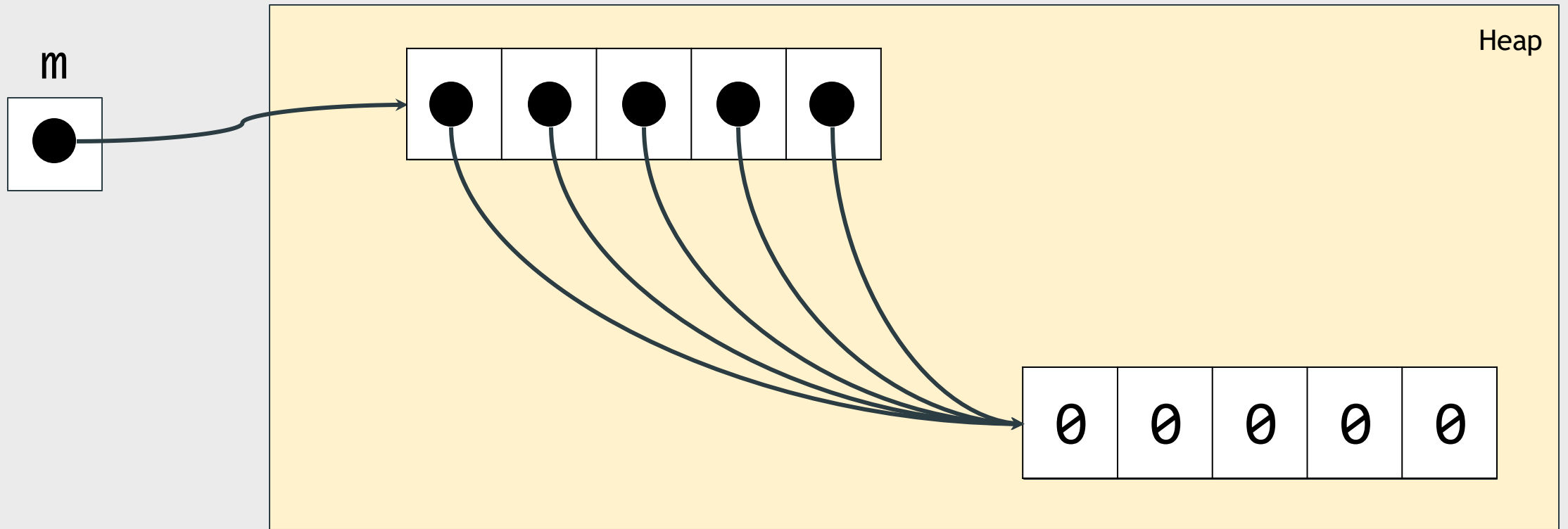
↩ `m = [[0]*5]*5`

↩ `m1 = [[0]*5 for i in range(5)]`

↓ Now, try: `m1[2][3] = 1` vs `m[2][3] = 1`

Lists of lists

Memory



```
m = [[0]*5]*5
```

```
[0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0],  
[0, 0, 0, 0, 0]]
```

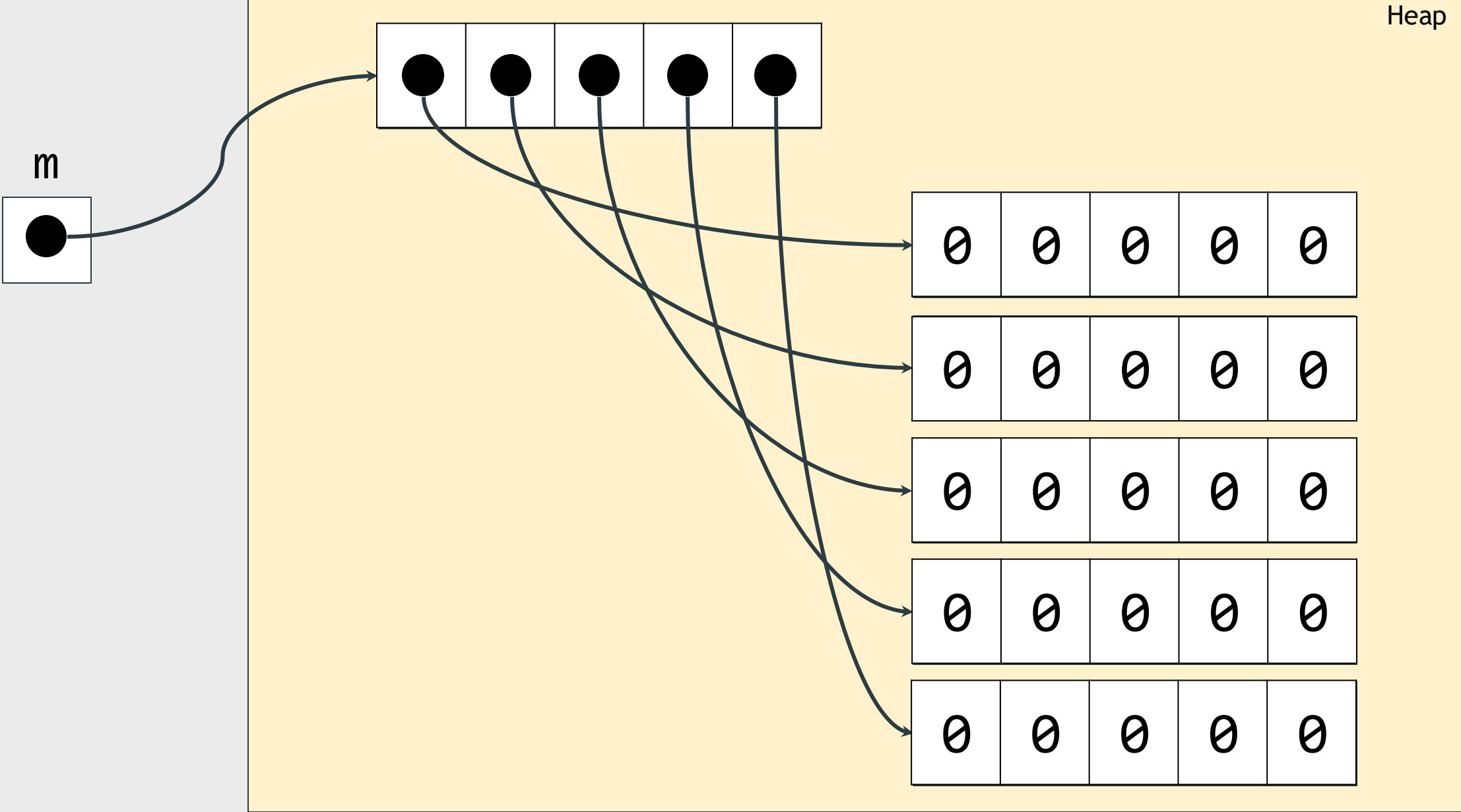
```
m[2][3] = 1
```

```
[0, 0, 0, 1, 0],  
[0, 0, 0, 1, 0],  
[0, 0, 0, 1, 0],  
[0, 0, 0, 1, 0],  
[0, 0, 0, 1, 0]]
```



```
m = [[0]*5 for i in range(5)]
```

Memory

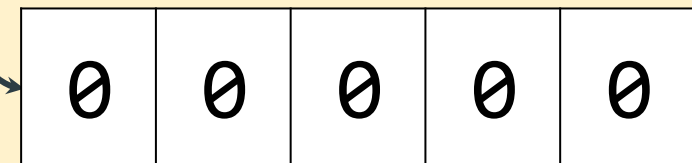
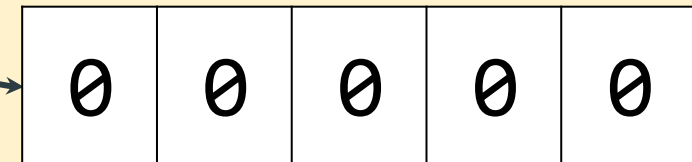
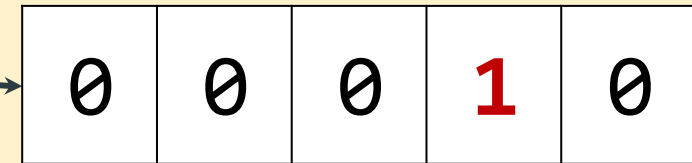
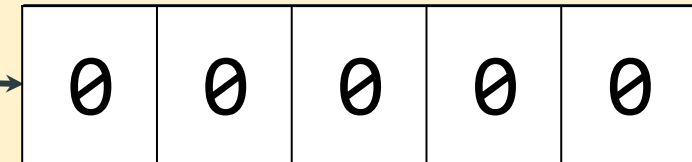
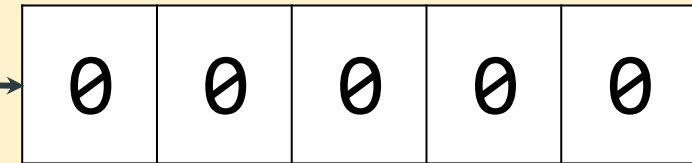
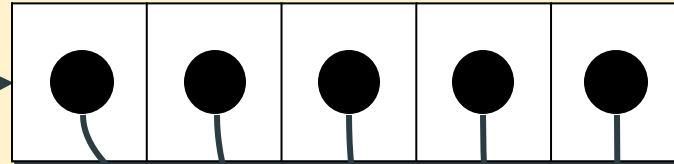
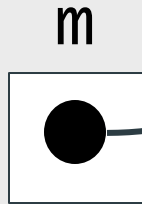


```
m = [[0]*5 for i in range(5)]
```

Memory

```
m[2][3] = 1
```

Heap

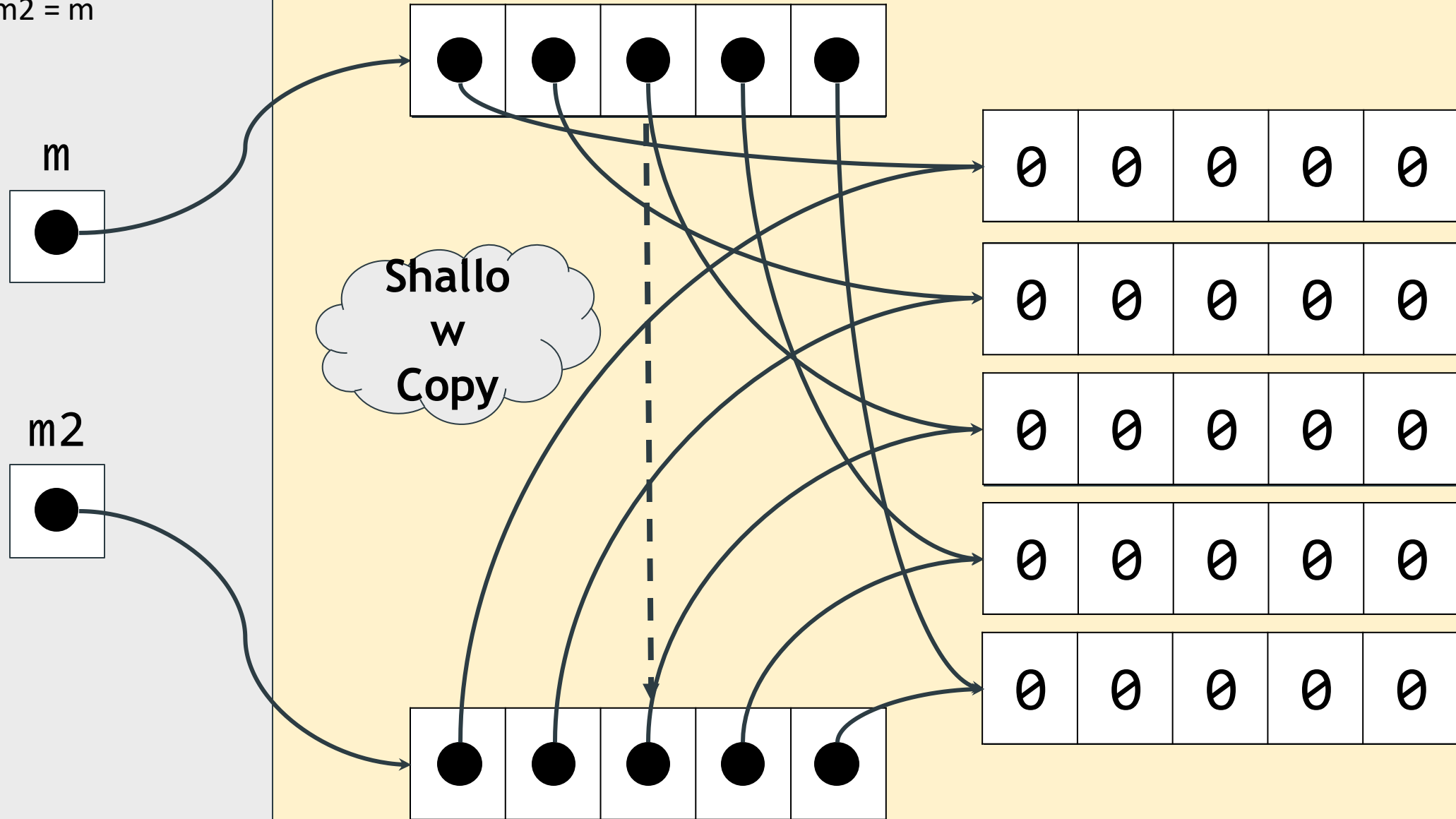


```
m= [[0]*5 for i in range(5)]
```

```
m2 = m
```

Memory

Heap

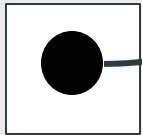


```
m = [[0]*5 for i in range(5)]
```

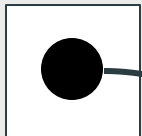
```
m2 = m
```

```
m[2][3] = 1
```

m

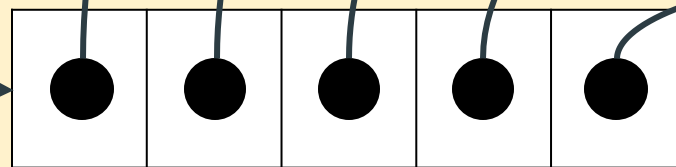
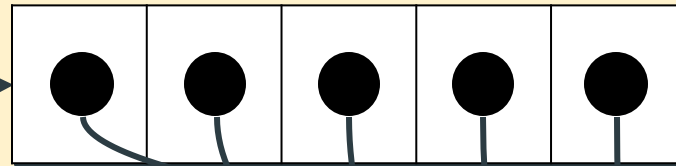


m2

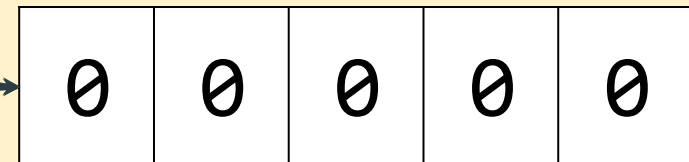
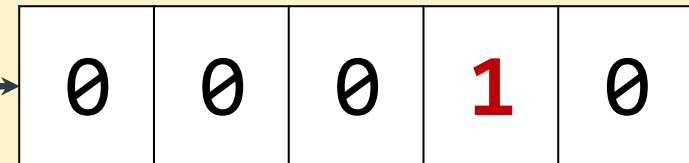
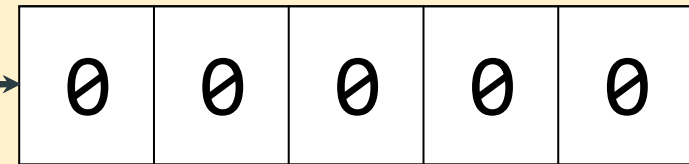
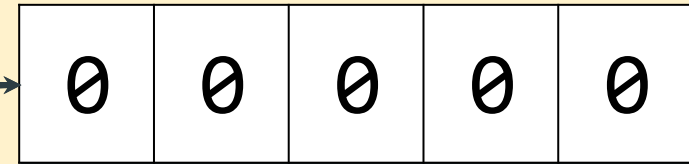


Memory

Shallow Copy



Heap



Memory: TL;DR

- ↳ SHALLOW VS DEEP
- ↳ Modify with new vs in-place
- ↳ Challenge q: can I use `m[:]` to make a deep copy of `m`?

TT2: anatomy of a project

- ↩ Good to start working through TT to see what questions are coming up
- ↩ Items:
 - ↩ TT files (linked on Canvas)
 - ↓ geometry
 - ↓ list_exercises
 - ↓ min_max
 - ↩ Walkthrough (on course page):
<https://classes.ssd.uchicago.edu/macss/mac30121/modules/tt/tt2.html>

TT2: anatomy of a project

↳ Suggested workflow:

- ↓ Follow along with page
- ↓ Try it all on your own (pretend solutions do not exist! Resist temptation!)
- ↓ Read through and compare your solution to the published solution - how are they different? What does that mean about your code?

Example function

keyword name parameters

```
def multiply(a, b):  
    '''  
    Compute the product of two values.  
  
    Inputs:  
    a, b: the values to be multiplied.  
  
    Returns: the product of the inputs  
    '''  
  
    n = a * b  
    return n
```

Function header


docstring

body

Coding practice: 1.4.1

Function Call Control Flow

Calling a function alters the control flow of a program.



```
1  def multiply(a, b):
2      print("Start of multiply(a, b) function")
3      rv = a * b
4      print("End of multiply(a, b) function")
5      return rv
6
7  def main():
8      x = 5
9      y = 4
10     print("calling multiply(x, y)...")
11     z = multiply(x, y)
12     print("Returned from multiply(x, y)")
13     print("The value of z is", z)
```

return: specify the value to be returned to the caller and to transfer control back to the call site.

PA 1: IT'S COMING!!!

- ↳ MULTIPLE STEPS
- ↳ TIME CONSUMING
- ↳ NEED GOOD WORKFLOW
- ↳ START NOW!!!
 - ↳ I suggest doing the first four tasks in groups of two
 - ↳ Task 5 will likely take awhile to go back and ensure everything comes together

Looking ahead: next class /

deadlines

10/11 Friday: SE 2

↵ 10/18 Friday PA 1

↵ 10/21 Monday PA 1 reflection

↵ **Content: Read up on functions Ch 1.4**