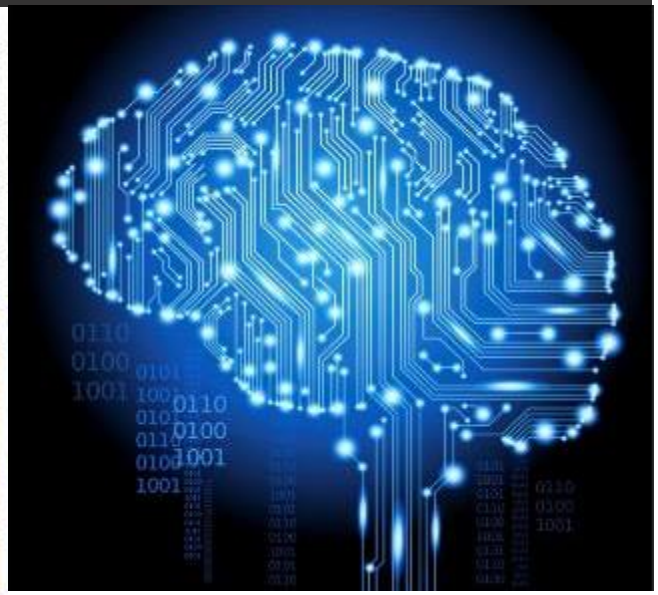
MONASH University

Information Technology

FIT1008 & 2085 Lecture 14
Python Variables and Scoping

Prepared by: M. Garcia de la Banda
based on D. Albrecht, J. Garcia



Where are we at?

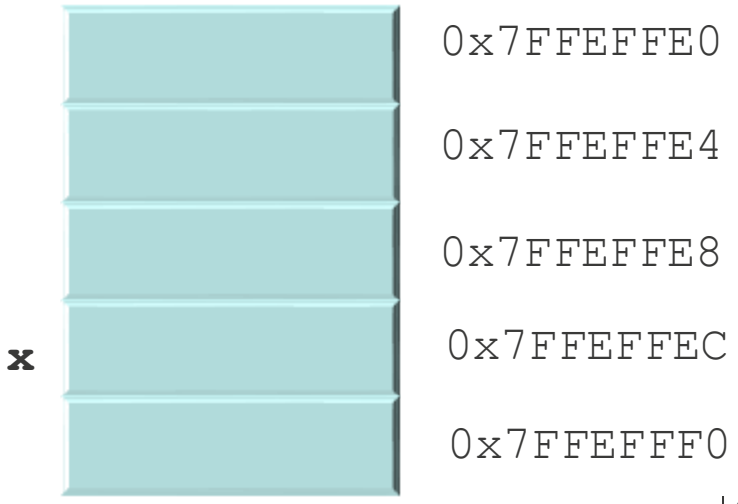
- **Have learnt how to implement in Python:**
  - Bubble Sort
  - Selection Sort
  - Insertion Sort
- **Have learned about runt time and Big O complexity**
- **Have started to become accustomed to think about:**
  - The use of invariants for improving our code
  - The properties of our algorithms (e.g., stable? incremental?)
  - Their Big O complexity

Objectives for this lecture

- **To learn about how variables and values are represented internally in Python**
- **To understand how this affects execution**
- **To understand the concept of mutable/immutable objects**
- **To be able to follow python code involving**
  - Variable assignments
  - Mutable types
  - Immutable types
  - Variable aliasing (assigning variables to other variables)

Variable representation

- **What is a program variable?**
  - A name (identifier) of some "something"
- **The name in almost all languages refers to a memory location**
  - The something depends on what you assign to it
- **As seen in MIPS, memory is divided into portions**
  - bits, bytes, words…
- **Each portion has an address**
- **Internally, a variable `x` is an address**
  - Say `0x7FFEFFEC`
- **That memory address contains…?**
  - The "something"?



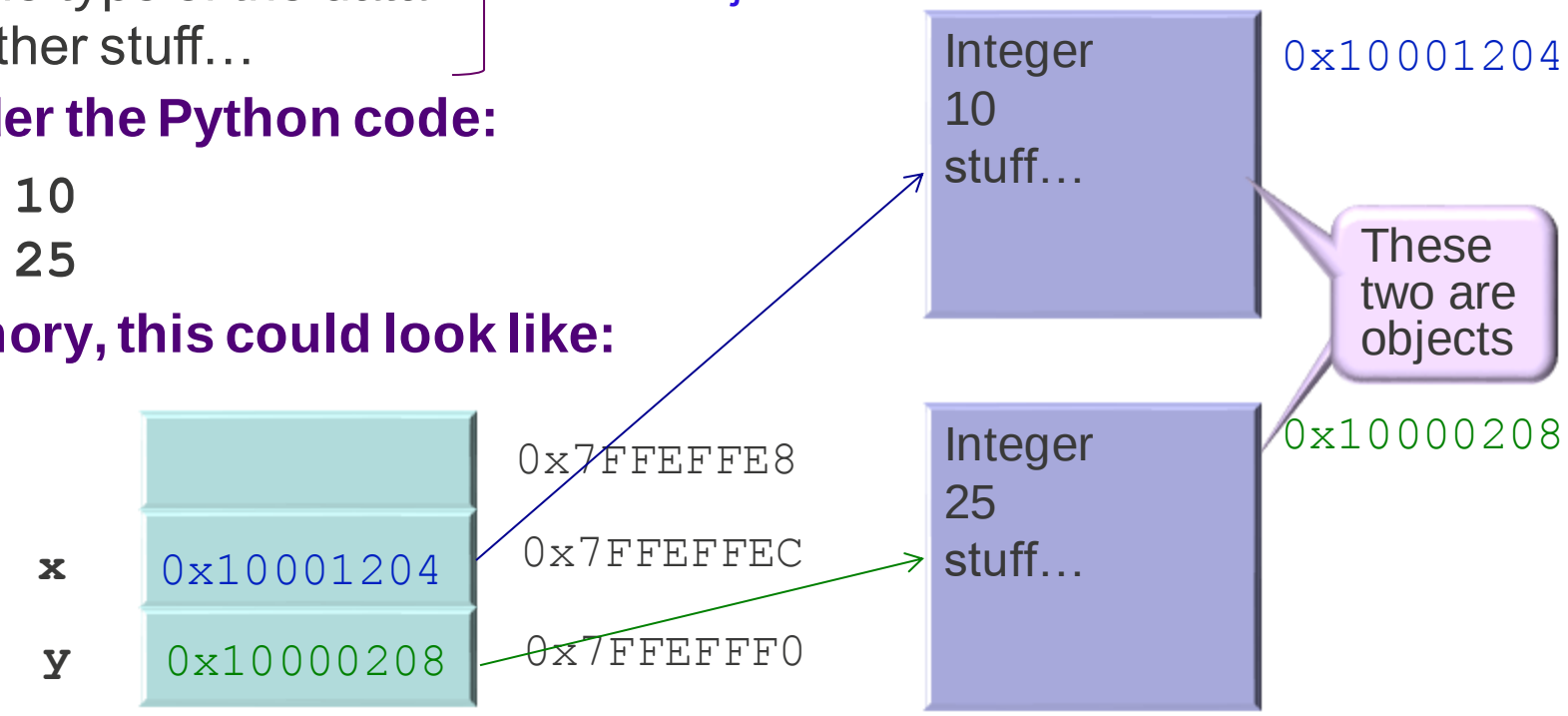
Variable representation in Python

- **The content depends … on the language!**
- **In Python: it is a reference to the memory location containing**
  - The data
  - The type of the data
  - Other stuff…

  } The "object"
- **Consider the Python code:**

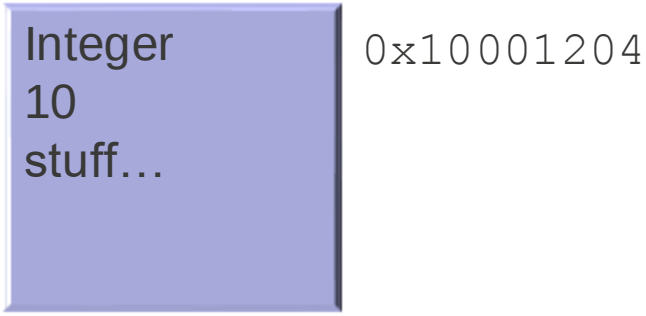
```
x = 10
y = 25
```

- **In memory, this could look like:**



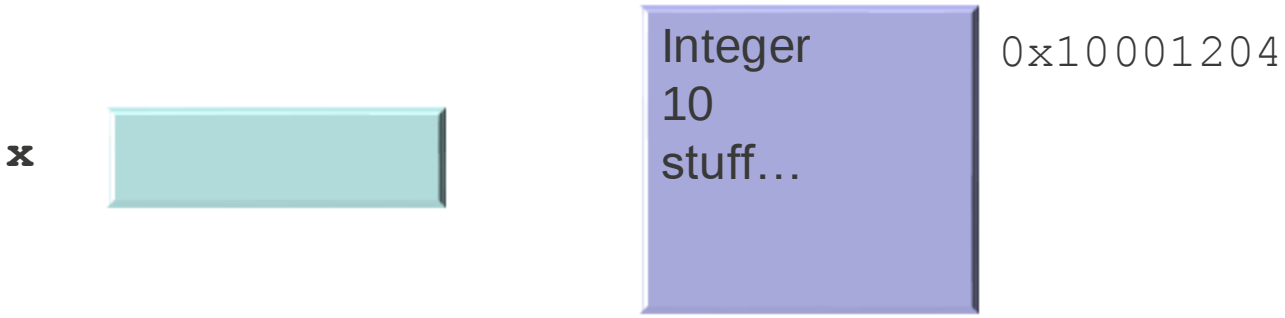
Creating variables in Python

- **A variable is created when you first assign it a value**
  - In many other languages (like Js), variables can be *declared*
    - This means created *without* a value (or given a default)
- **So when you say `x = 10` in Python, it:**
  1. Creates an object to represent `10`, starting at some address



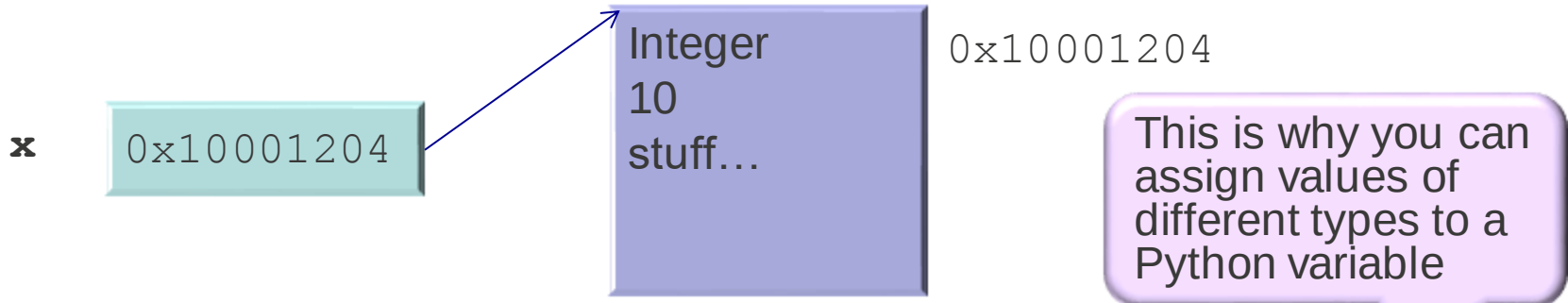
Creating variables in Python

- **A variable is created when you first assign it a value**
  - In many other languages, variables can be *declared*
    - This means created *without* a value
- **So when you say `x = 10` in Python, it:**
  1. Creates an object to represent `10`, starting at some address
  2. Creates the variable `x` if it does not exist



Creating variables in Python

- **A variable is created when you first assign it a value**
  - In many other languages, variables can be *declared*
    - This means created *without* a value
- **So when you say `x = 10` in Python, it:**
  1. Creates an object to represent `10`, starting at some address
  2. Creates the variable `x` if it does not exist
  3. Links it with the object created (assigns the address to `x`)



- **Important consequence: Python variables do not have a type**
  - Types are associated with values (i.e., with objects)

Our visualisation of objects in Python

- **To simplify things, we will:**
  - Only display values within the object
    - The type and other stuff will be ignored for now
  - Ignore the exact value of the references (i.e., the address)
    - We will use arrows to represent them
- **So `x = 10` will look like:**

- **I will not put `x` inside the box to emphasize the content is the reference, not the name**

Assigning variables to other values

- **We said: variables are always references to objects**
- **Changing the assignment does not alter the object itself**
- **It only alters the reference:**
  - The variable will refer (point) to a different object
- **Consider the code:**

```
x = 10
x = x + 3
```

- **Lets see how it executes:**
  1. Creates object `10` somewhere

Assigning variables to other values

- **We said: variables are always references to objects**
- **Changing the assignment does not alter the object itself**
- **It only alters the reference:**
  - The variable will refer (point) to a different object
- **Consider the code:**

```
x = 10
x = x + 3
```

- **Lets see how it executes:**
  1. Creates object `10` somewhere
  2. Creates variable `x`

Assigning variables to other values

- **We said: variables are always references to objects**
- **Changing the assignment does not alter the object itself**
- **It only alters the reference:**
  - The variable will refer (point) to a different object
- **Consider the code:**

```
x = 10
x = x + 3
```

- **Lets see how it executes:**
  1. Creates object `10` somewhere
  2. Creates variable `x`
  3. Links `x` to `10`

Assigning variables to other values

- **We said: variables are always references to objects**
- **Changing the assignment does not alter the object itself**
- **It only alters the reference:**
  - The variable will refer (point) to a different object
- **Consider the code:**

```
x = 10
x = x + 3
```

- **Lets see how it executes:**
  1. Creates object `10` somewhere
  2. Creates variable `x`
  3. Links `x` to `10`
  4. Evaluates `x+3`

This is why you MUST assign a value to a variable before using it

A variable in an expression is immediately replaced with the object it currently refers to. Then the expression is evaluated.

Assigning variables to other values

- **We said: variables are always references to objects**
- **Changing the assignment does not alter the object itself**
- **It only alters the reference:**
  - The variable will refer (point) to a different object
- **Consider the code:**

```
x = 10
x = x + 3
```

- **Lets see how it executes:**
  1. Creates object `10` somewhere
  2. Creates variable `x`
  3. Links `x` to `10`
  4. Evaluates `x+3`
  5. Creates object `13`

This is why you MUST assign a value to a variable before using it

13

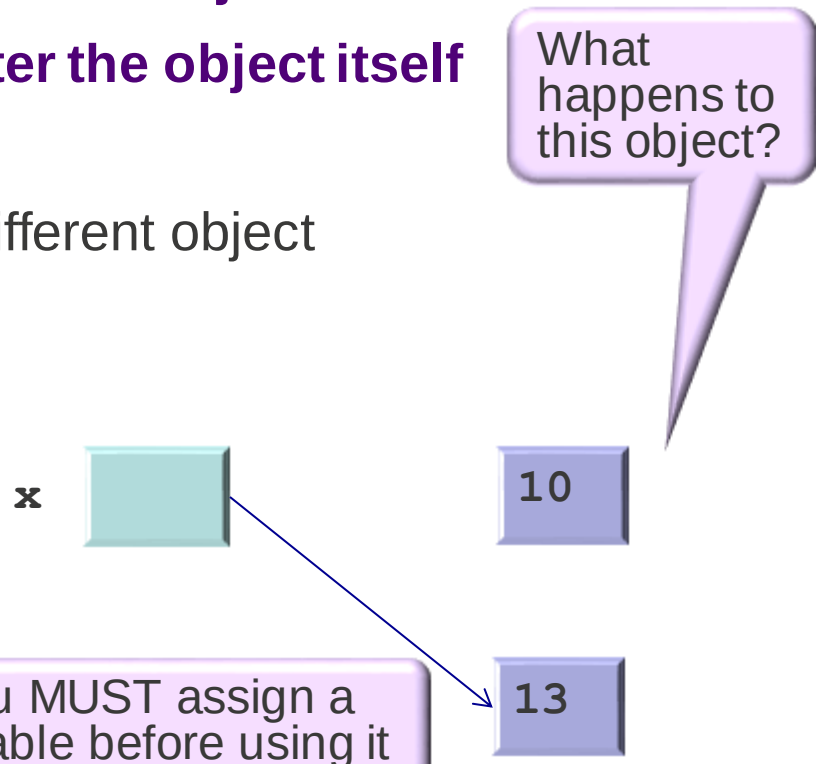
A variable in an expression is immediately replaced with the object it currently refers to. Then the expression is evaluated.

Assigning variables to other values

- **We said: variables are always references to objects**
- **Changing the assignment does not alter the object itself**
- **It only alters the reference:**
  - The variable will refer (point) to a different object
- **Consider the code:**

```
x = 10
x = x + 3
```

- **Lets see how it executes:**
  1. Creates object `10` somewhere
  2. Creates variable `x`
  3. Links `x` to `10`
  4. Evaluates `x+3`
  5. Creates object `13`
  6. Links `x` to `13`



Assigning variables to other variables

- **So conceptually, every time a new value is created:**
  - Python creates a new object (a chunk of memory) to represent it
- **What about assigning a variable to another variable? Consider:**

```
x = 10
y = x
x = 'hi'
```

If I print `x` and `y` after this, what would it say?

- **Lets see how it executes:**
  1. Creates object `10` somewhere

Assigning variables to other variables

- **So conceptually, every time a new value is created:**
  - Python creates a new object (a chunk of memory) to represent it
- **What about assigning a variable to another variable? Consider:**

```
x = 10
y = x
x = 'hi'
```

If I print `x` and `y` after this, what would it say?

- **Lets see how it executes:**
  1. Creates object `10` somewhere
  2. Creates variable `x`

Assigning variables to other variables

- **So conceptually, every time a new value is created:**
  - Python creates a new object (a chunk of memory) to represent it
- **What about assigning a variable to another variable? Consider:**

```
x = 10
y = x
x = 'hi'
```

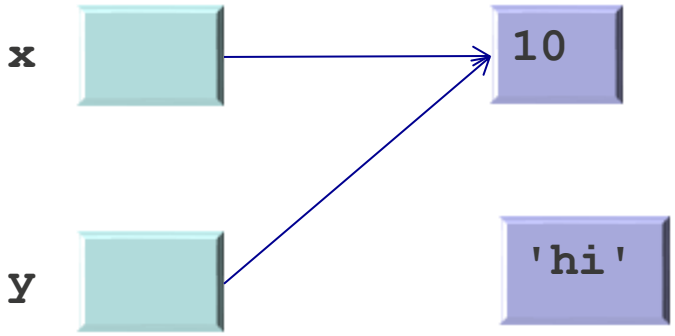
If I print `x` and `y` after this, what would it say?

- **Lets see how it executes:**
  1. Creates object `10` somewhere
  2. Creates variable `x`
  3. Links it to the object

Assigning variables to other variables

- **Now what? Time to execute `y = x`**
- **Variable `x` already exists, so no need to create it. So:**
    4. Creates variable `y`
    5. Links it to the object pointed to by `x`
- **Now time to execute `x = 'hi'`**
    4. Creates object `'hi'`
    5. Links `x` to this object

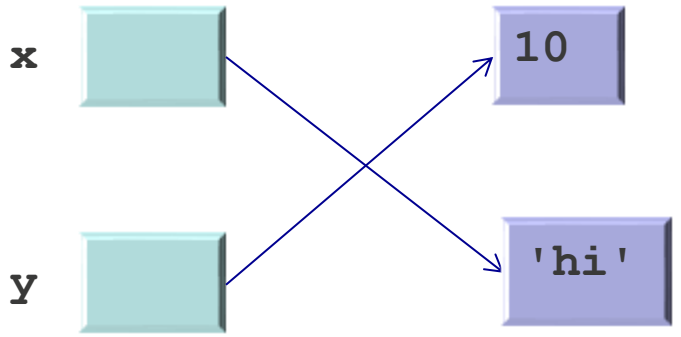
In MIPS: the code would load the value of x (which is an address) into, say, $t0 and then store it into y. So they have the same address and thus, the same link



Assigning variables to other variables

- **Now what? Time to execute `y = x`**
- **Variable `x` already exists, so no need to create it. So:**
    4. Creates variable `y`
    5. Links it to the object pointed to by `x`
- **Now time to execute `x = 'hi'`**
    4. Creates object `'hi'`
    5. Links `x` to this object
- **If we now print their values…**

```
>>> x = 10
>>> y = x
>>> x = 'hi'
>>> x;y
'hi'
10
```



And another example

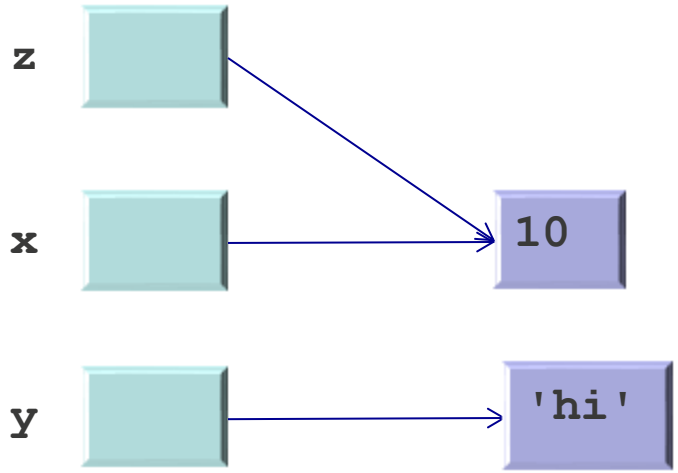
- **Consider the code**

```
>>> x = 10
>>> y = 'hi'
>>> z = x
>>> x = y
>>> x;y;z
```

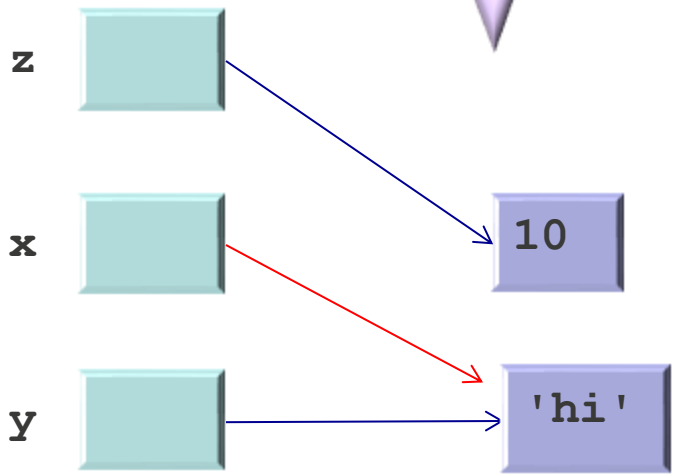
So it prints:

```
'hi'
'hi'
10
```

- **What will it print?**
- **After the first 3 lines it looks like:**



After the 4th line, it looks like this (only change in red):



Mixing assignments and evaluations

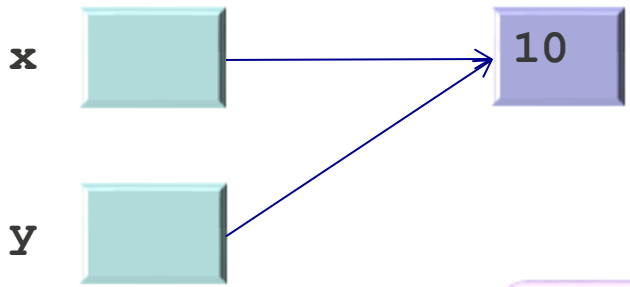
- **Consider the code**

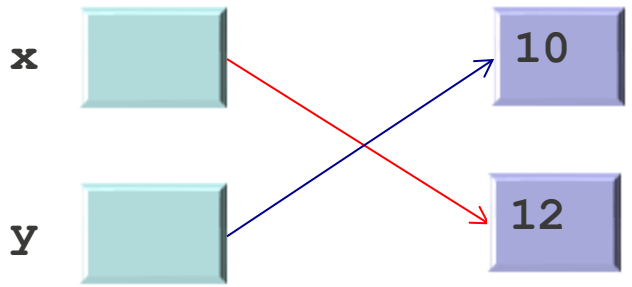
```
>>> x = 10
>>> y = x
>>> x = x+2
>>> x;y
```

Creates an object for the value resulting from evaluating the expression

After the 3rd , it looks like this :

- **What will it print?**
- **After the first 2 lines it looks like:**





So it prints:

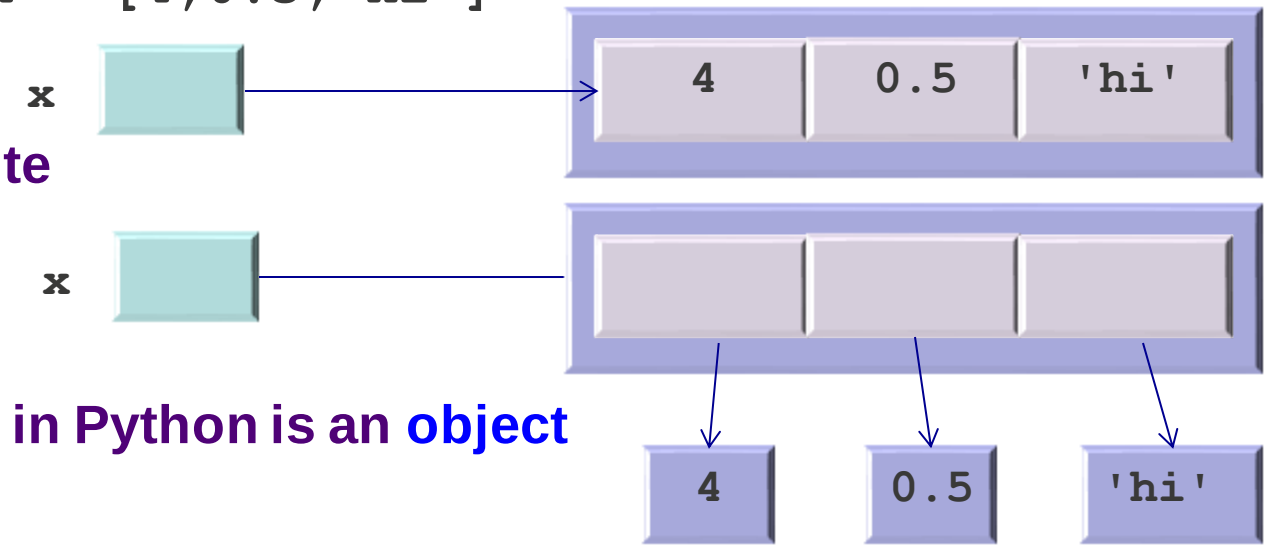
12

10

What about Python lists?

- **How are Python lists represented internally?**
  - They are implemented as arrays
  - But they are also objects
    - Like any object, they will have type, value and stuff…
    - And we will visualise it as before:

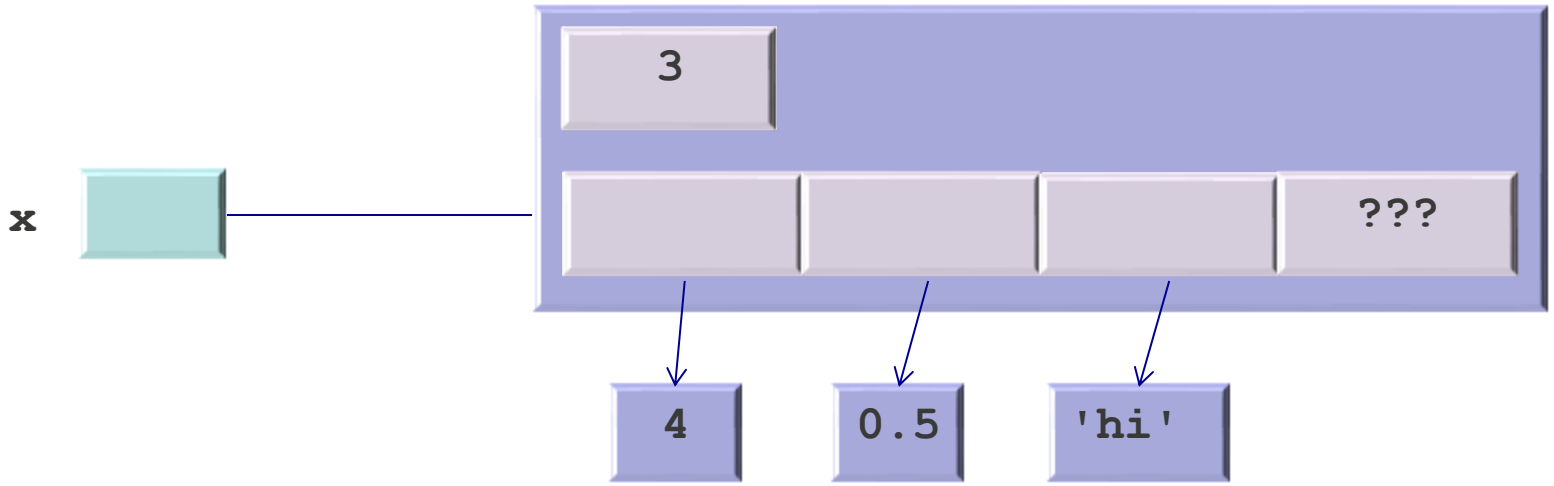
  - Question is: how is the value of a Python list represented?
- **Consider the list `x = [4,0.5,'hi']`**
- **Like this?**
- **Close, but not quite**
- **More like this:**



- **Since every value in Python is an object**

What about Python lists? (cont)

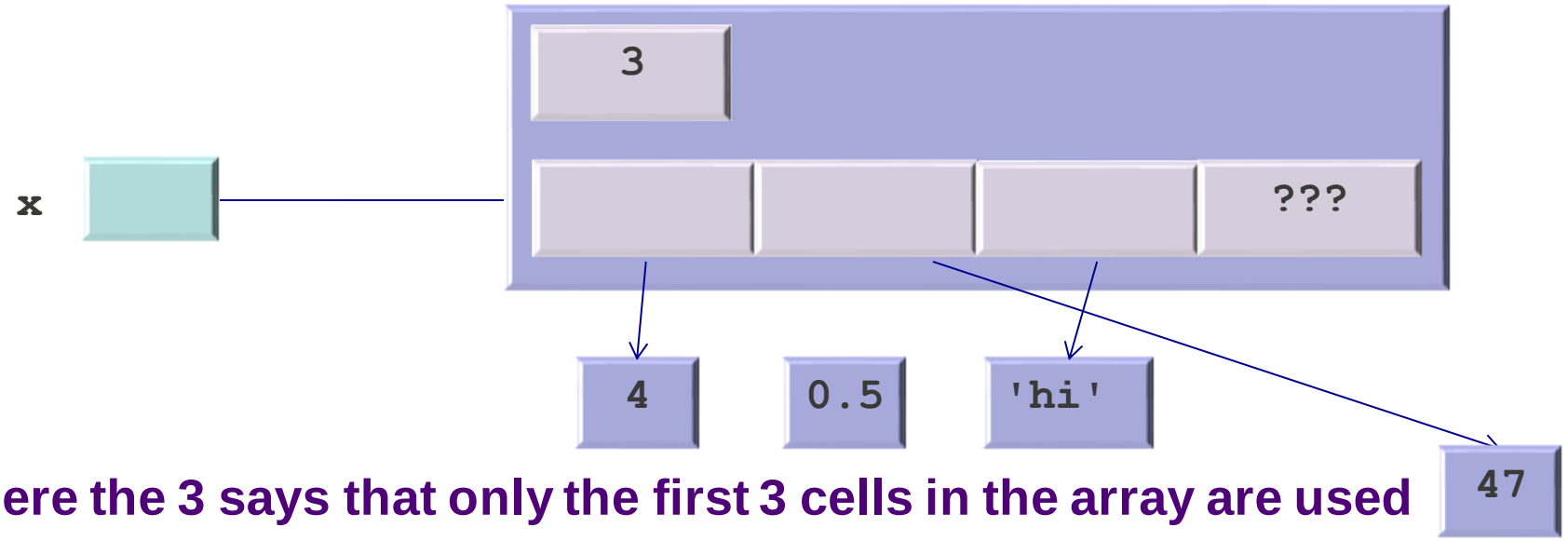
- In fact, their implementation is closer to this:



- Where the 3 says that only the first 3 cells in the array are used
- The important point is that they are arrays of references
- What does `x[1] = 47` do?

What about Python lists? (cont)

- **In fact, their implementation is closer to what we saw last week:**



- **Where the 3 says that only the first 3 cells in the array are used**
- **The important point is that they are arrays of references**
- **What does `x[1] = 47` do?**
  - Simply changes the reference, nothing else!
  - This modifies the object referred to by `x`, not `x`

Assigning variables versus list elements

- **Let's compare the two pieces of code:**

```
>>> x = 10
>>> y = x
>>> x = 'hi'
>>> x;y
'hi'
10
```

```
>>> x = [4,0.5,'hi']
>>> y = x
>>> x[1] = 47
>>> x;y
[4,47,'hi']
[4,47,'hi']
```

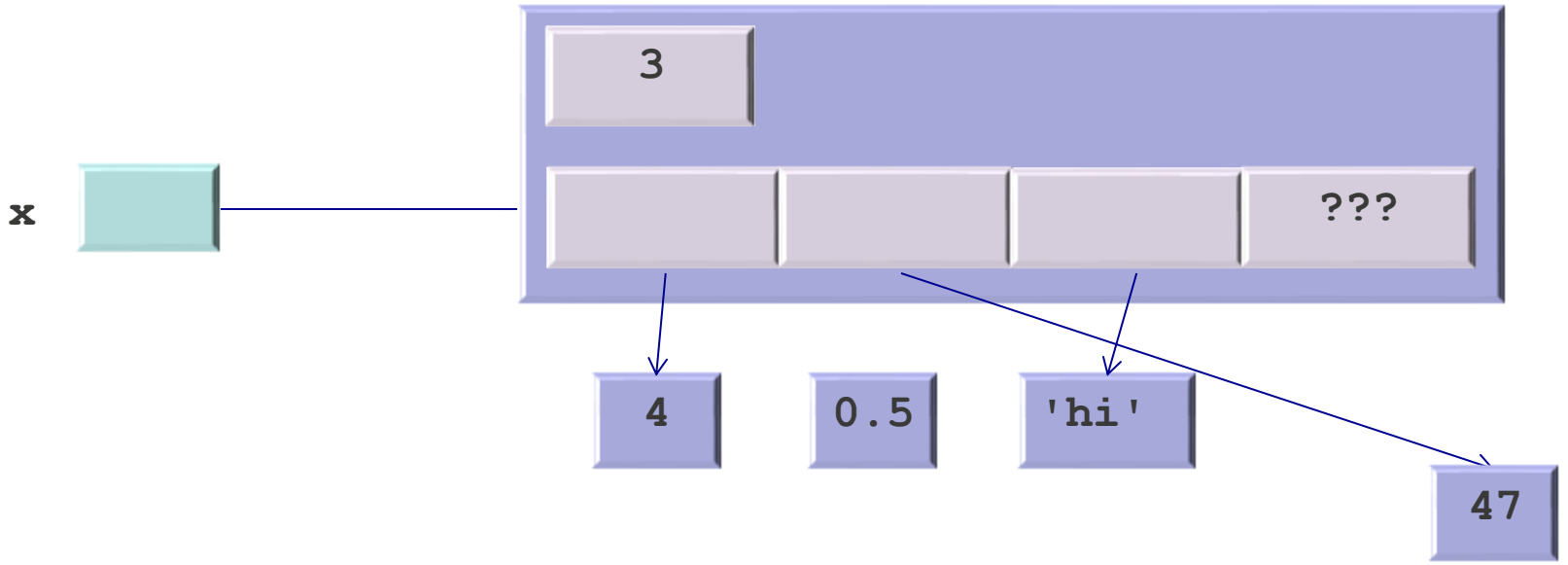
I was re-assigning an element in the list, not x

- **They behave differently!**
- **Not really. If I do:**
- **It prints:**
- **So the same as before**

```
>>> x = [4,0.5,'hi']
>>> y = x
>>> x = [9,-1]
>>> x;y
[9,-1]
[4,0.5,'hi']
```

Conclusion: Python lists are mutable

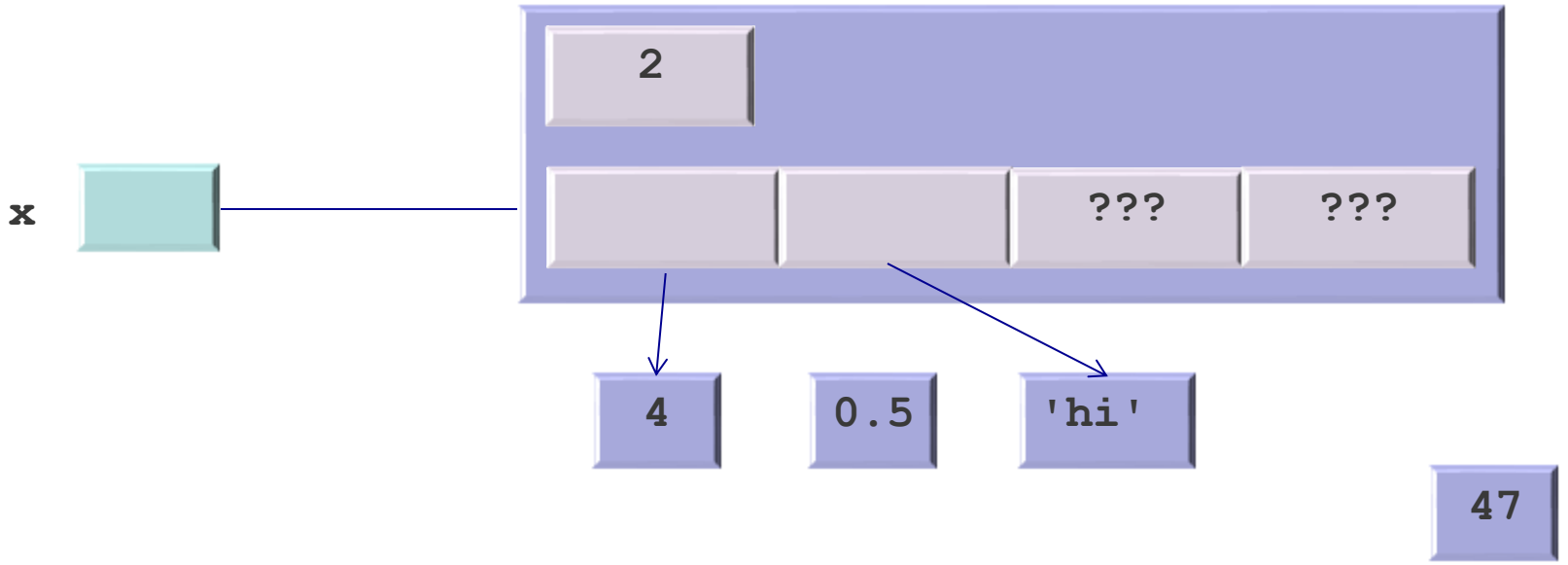
- What happens if given our old list:



- I delete an element from the list, say 47

Conclusion: Python lists are mutable

- As we will see next week, that I get:



- Again, it modifies the object referred to by `x`, not `x`
- Which means Python list are mutable:
  - In other words: objects of type list in Python can be changed

Can I modify an integer object then?

- **No, they are immutable, I can only create integer objects:**
  - Once created they cannot be changed
  - I can create a new one, but not modify an already created one
- **Is this because they are "atomic" (indivisible)?**
  - No: tuples are not atomic and they are also immutable
- **Really?**

Lets see:

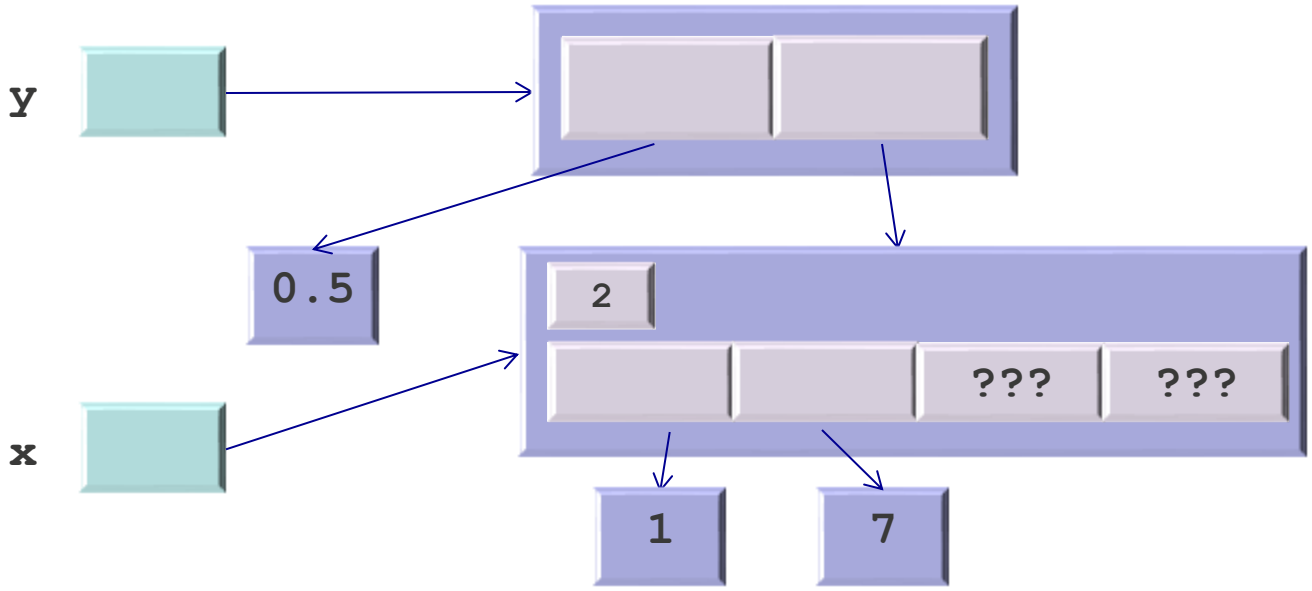
```
>>> y = (0.5, 7, 31)
>>> y
(0.5, 7, 31)
>>> y[0]
0.5
>>> y[0]=3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Numbers, strings and tuples are immutable

Yes, tuples really are immutable

- You might think you have modified tuples before
- Lets see:

```
>>> x = [1,7]
>>> y =
(0.5,x)
>>> x[0]=10
```

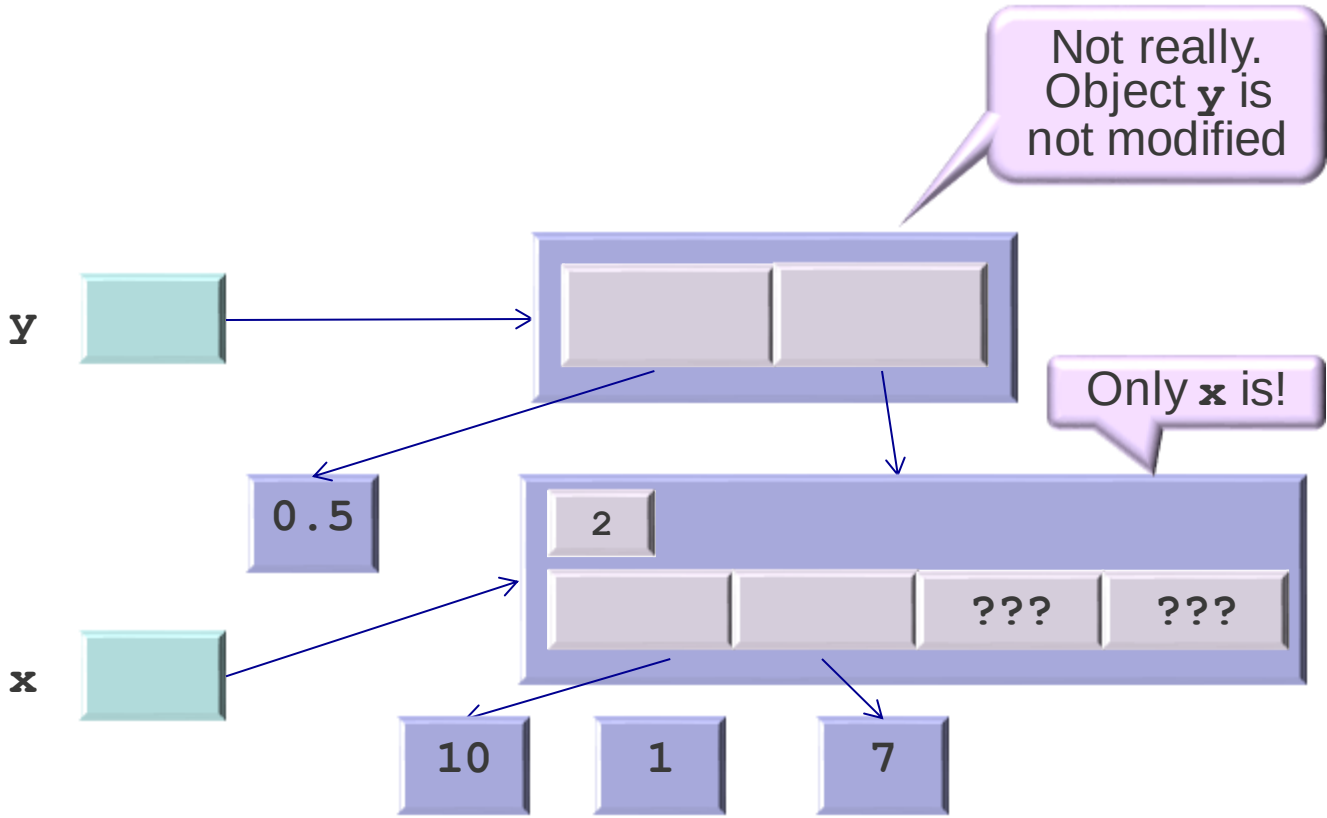


Yes, tuples really are immutable

- **You might think you have modified tuples before**
- **Lets see:**

```
>>> x = [1,7]
>>> y =
(0.5,x)
>>> x[0]=10
>>> x;y
[10, 7]
(0.5, [10, 7])
```

Modified…

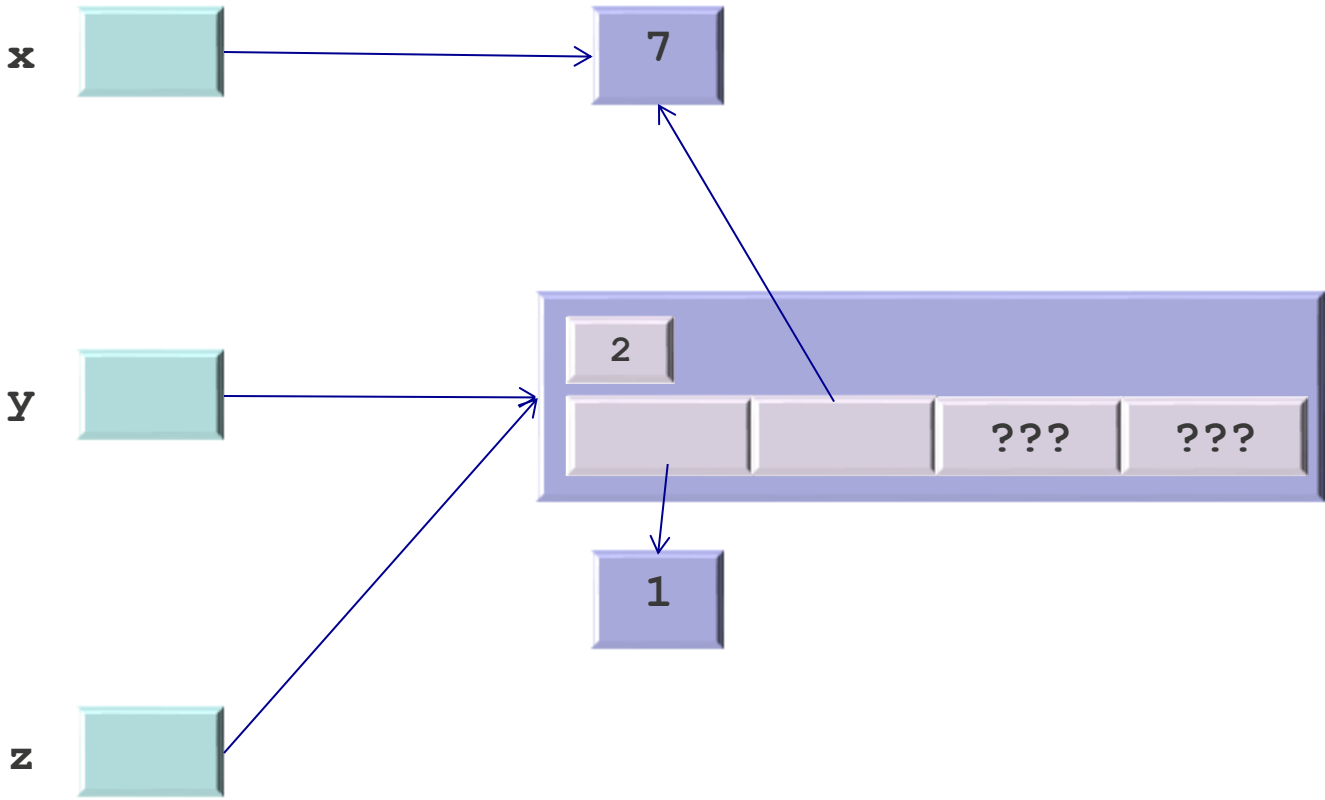


Your turn

- **What does the following code print?**

```
>>> x = 7
>>> y = [1,x]
>>> z = y
>>> z[1] = 9
>>> x;y;z
```

After the first 3 lines it looks like

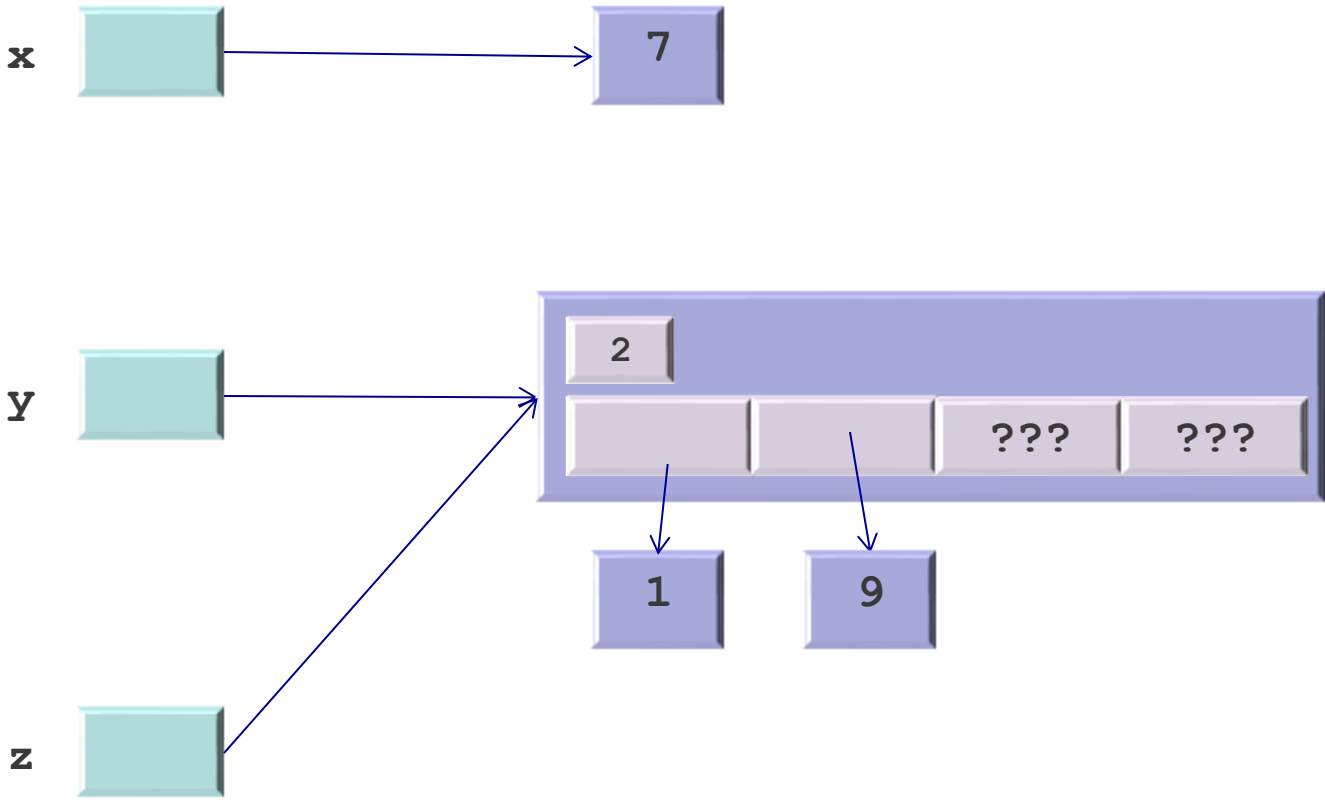


Your turn

- **What does the following code print?**

```
>>> x = 7
>>> y = [1,x]
>>> z = y
>>> z[1] = 9
>>> x;y;z
```

After the 4th line it looks like



Summary

- **We have seen how to draw memory diagrams for code involving:**
  - Variable assignments
  - Mutable types
  - Immutable types
  - Variable aliasing (assigning variables to other variables)