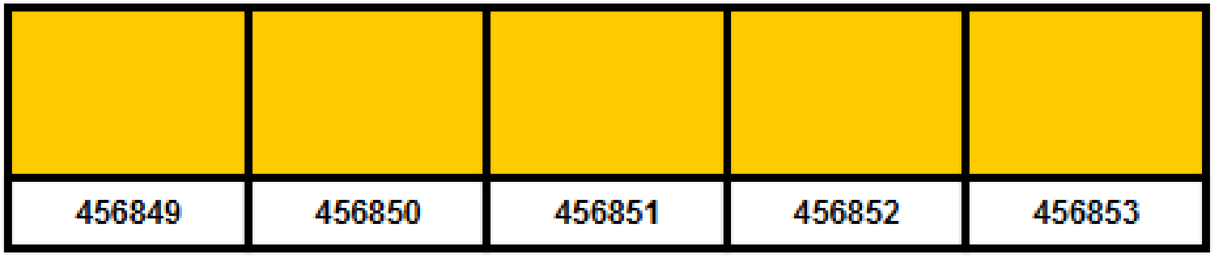
COMPSCI 107
Computer Science Fundamentals

Lecture 04 – Models of memory

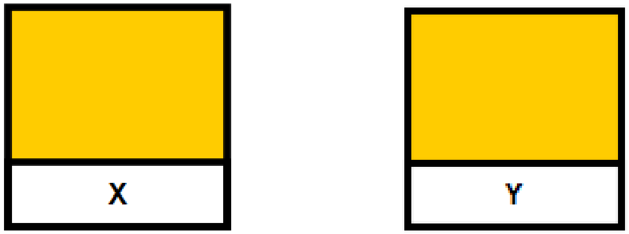
Mutable and immutable data

Variable identifiers

- Memory
  - Memory consists of boxes with numeric addresses
  - Each box holds a single number



- Variable identifiers
  - Labels for these boxes
  - Interpreter maintains tables linking label to address



Label	Address
X	456851
Y	456849

Exercise

- What is the output produced by each of the following programs?

```
x = '4'
y = x
y += '5'
print(y)
print(x)
```

```
x = [4]
y = x
y += [5]
print(y)
print(x)
```

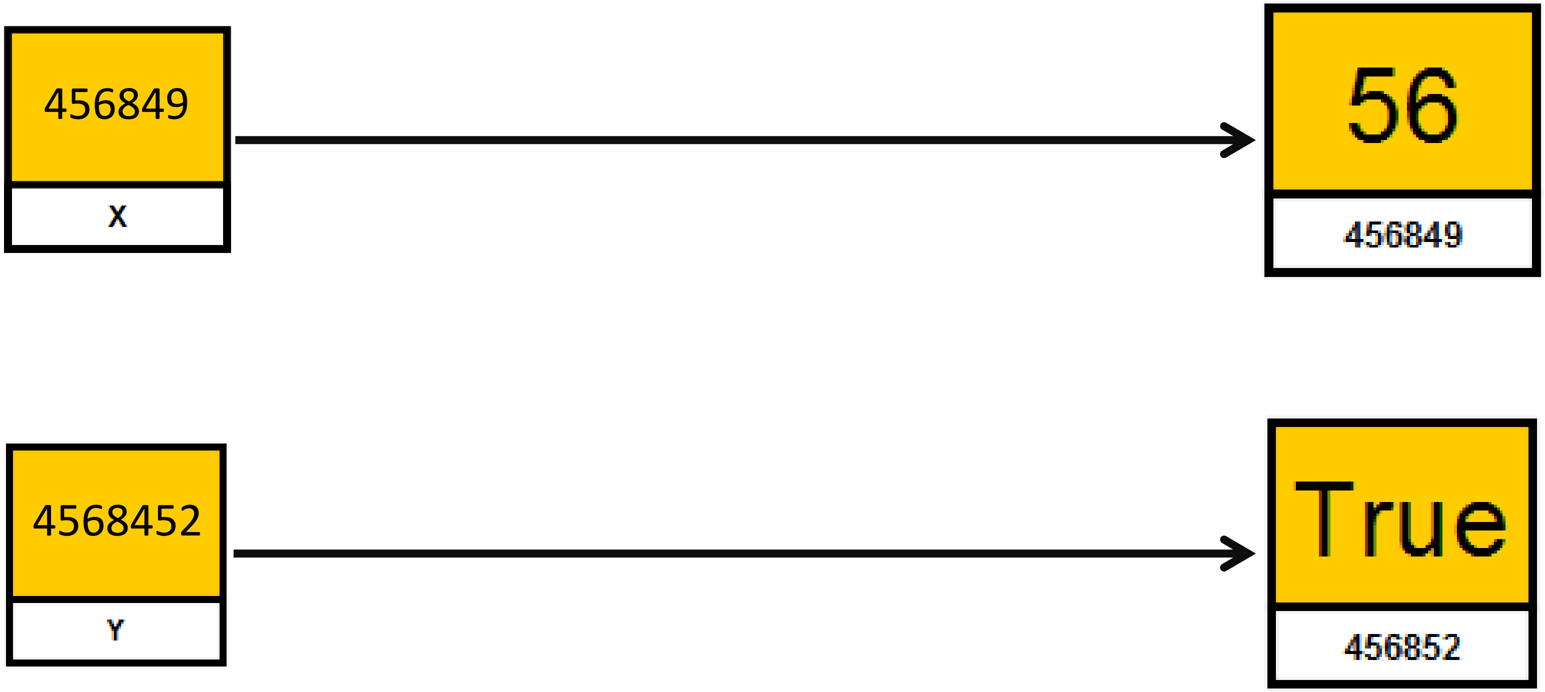
- Discuss your answers

What do we store in the box?

- Data
  - data is stored directly in the box
  - good model for simple data
  - need a more complex model for more complex data

Data

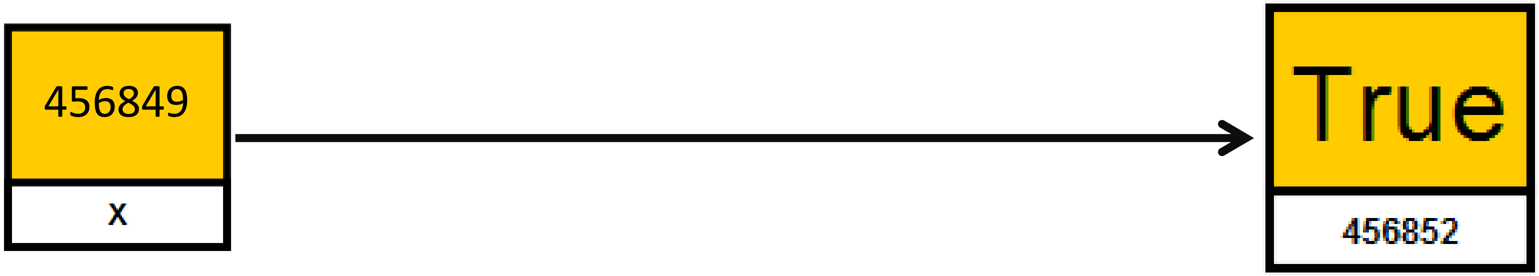
- Linking variables with data
  - Data is stored in the memory
  - Variables hold a reference to the location of the data



Aliasing

- Assignment statements copy the value on the right to the variable on the left

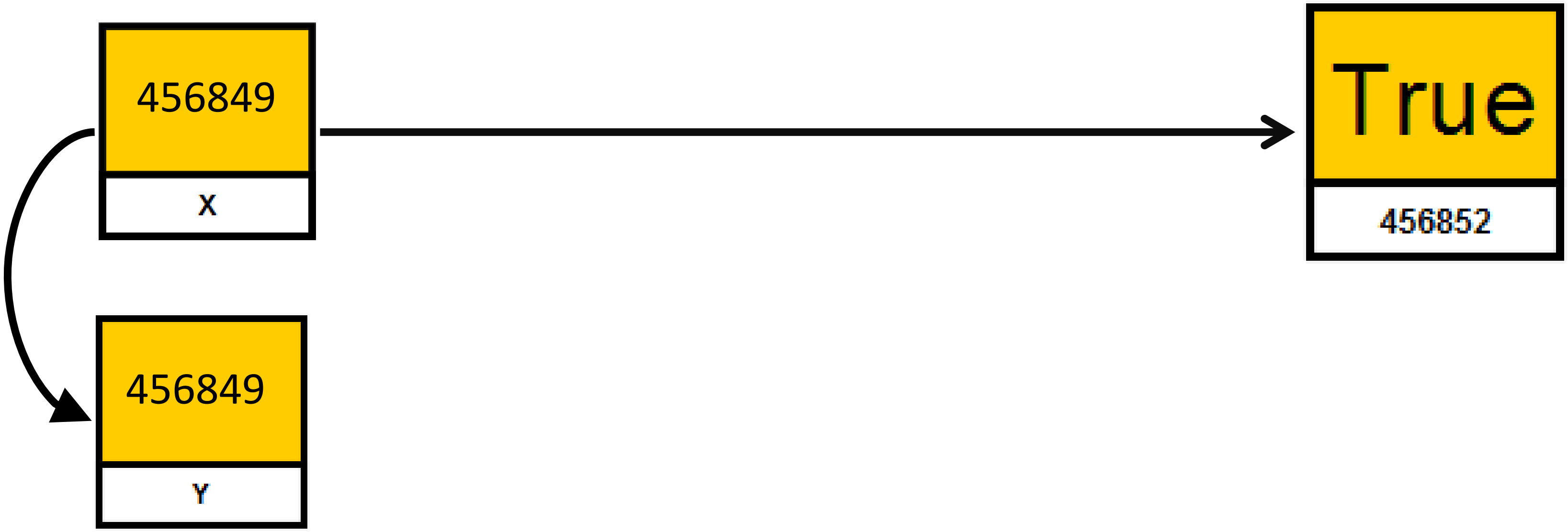
$$y = x$$



Aliasing

- Assignment statements copy the value on the right to the variable on the left

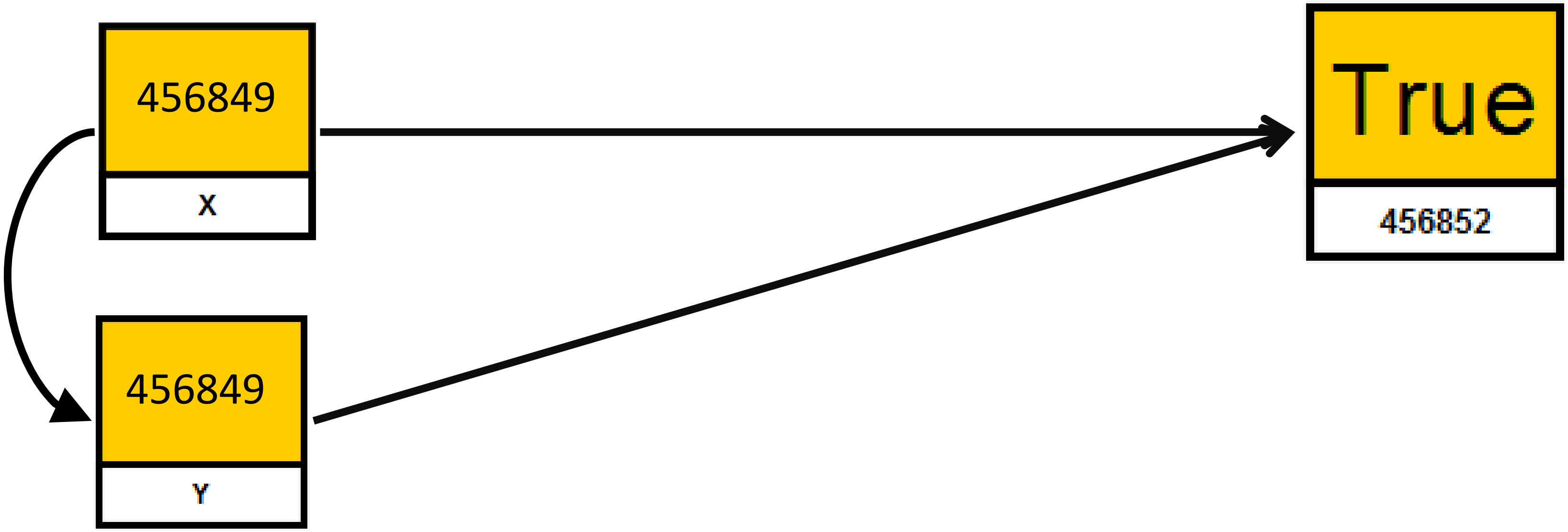
```
y = x
```



Aliasing

- Assignment statements copy the value on the right to the variable on the left

y = x



Exercise

- Does that explain the behaviour of this code?
  - What would you expect to see?

```
x = '4'
y = x
y += '5'
print(y)
print(x)
```

```
x = [4]
y = x
y += [5]
print(y)
print(x)
```

Mutable and immutable types

- Mutable
  - A type of variable in which the contents can be changed
  - lists, dictionaries, most complex data types
- Immutable
  - A type of variable in which the contents cannot be changed
  - int, float, boolean, string, tuple

Subtle distinctions

- In place operators use different code to normal operators

```
x = x + 4
x += 4
```

These are not the same operator

- With immutable types, they both perform the same function
- With mutable types, the in place operator modifies the contents referred to by x, but the normal operator + creates a new object.

Exercise

- What is the output of the following code?

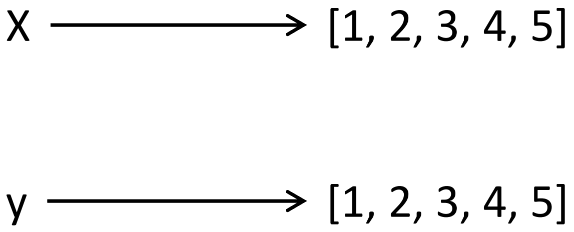
```
data = [1, 2, 3, 4]
backup = data

while len(data) > 0:
    element = data.pop()
    print(element, data)

print(data)
print(backup)
```

Modeling objects in memory

- Value equality

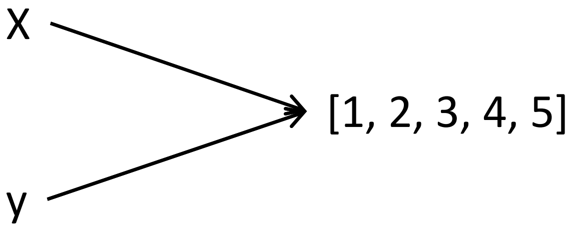


X → [1, 2, 3, 4, 5]

y → [1, 2, 3, 4, 5]

Two different objects that store the same information.

- Reference equality



X

[1, 2, 3, 4, 5]

y

Two different references / names for the same object.

Different ways to compare equality

- ==
  - Calls a method of the object
  - Programmer who defined the object decides how to determine equality
  - Typically involves checking the contents of the objects
  - We should always use this kind of equality unless you need to check references

- is
  - Checks the references of the objects
  - Evaluates to True if they are the same object

Exercise

- What is the output from each of the examples below? Explain.

```
x = 100
y = 100
print(x == y, x is y)
```

```
x = 500
y = 500
print(x == y, x is y)
```

```
x = 2.5
y = 2.5
print(x == y, x is y)
```

```
x = 'Hello World'
y = 'Hello World'
print(x == y, x is y)
```

Shallow copies

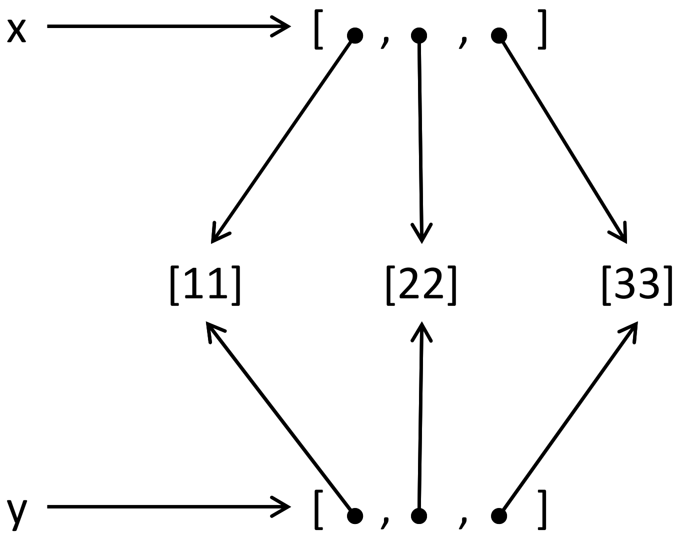
- Lists and dictionaries have a copy method
  - data.copy()

```
x = [1, 2, 3, 4, 5]
y = x.copy()
print( x is y )

a = [ [11], [22], [33] ]
b = a.copy()
print( a is b )
print( a[0] is b[0] )
```

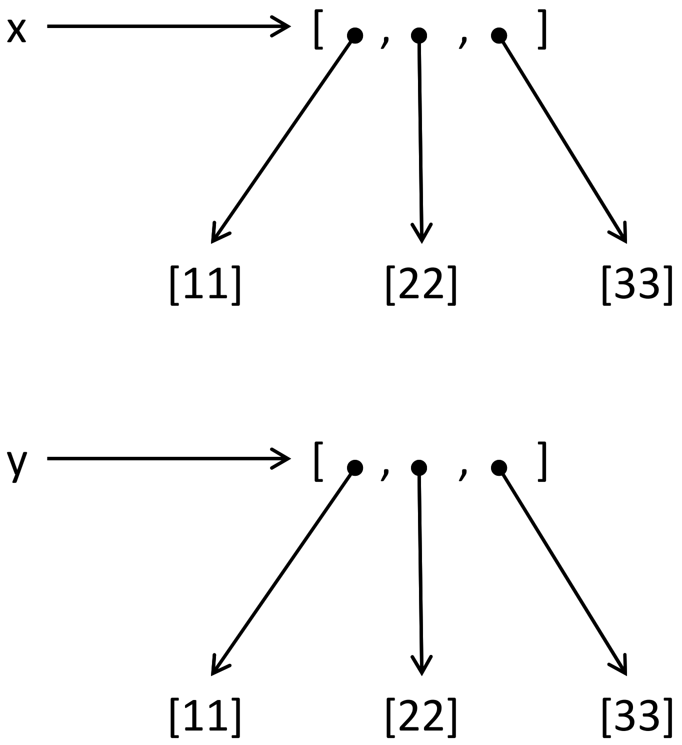
Shallow copy

- New object created
  - Contents of the original object are copied
  - If the contents are references, then the *references* are copied



Deep copies

- New object created
  - Contents of the original object are copied
  - If the contents are references, then the copy the objects referred to



Deep copies

- Use the module *copy*
  - new_copy = copy.copy(original)
  - new_copy = copy.deepcopy(original)

```
import copy

a = [ [11], [22], [33] ]
b = copy.deepcopy(a)
print( a is b )
print( a[0] is b[0] )
```

Summary

- Variables store references to the objects, not the actual objects
  - When you assign a variable, a reference is copied, not the object

- There are two kinds of equality
  - Equality of content (value equality) can be tested with **==**
  - Equality of identity (reference equality) can be tested with **is**

- When a copy is created, it can be a shallow or deep copy
  - A shallow copy copies the references
  - A deep copy recursively copies the objects referred to