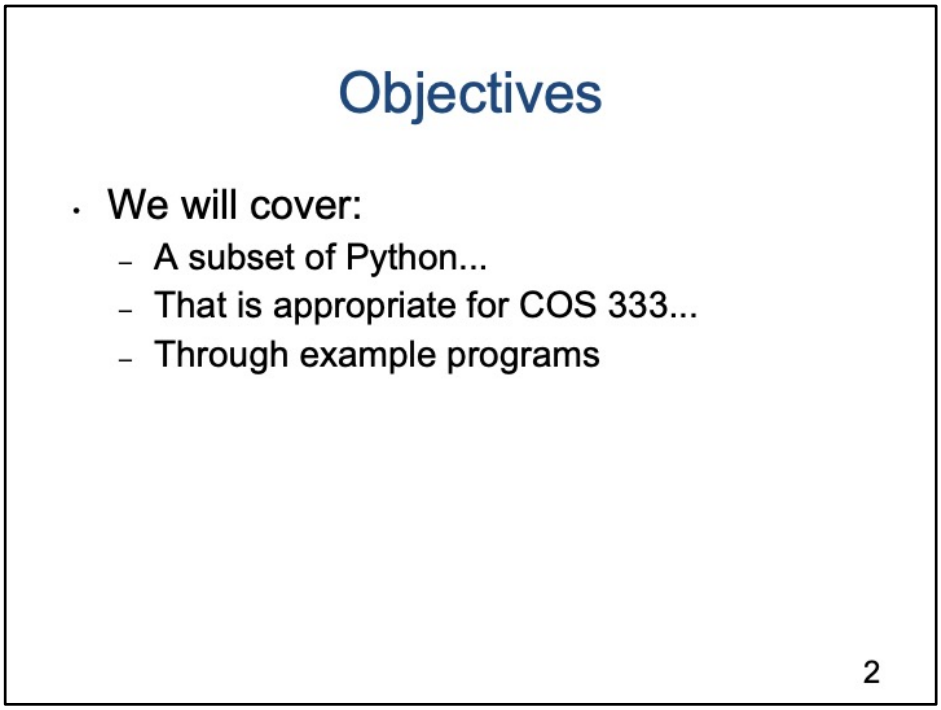
The Python Language (Part 4)

Robert M. Dondero, Ph.D.
Princeton University



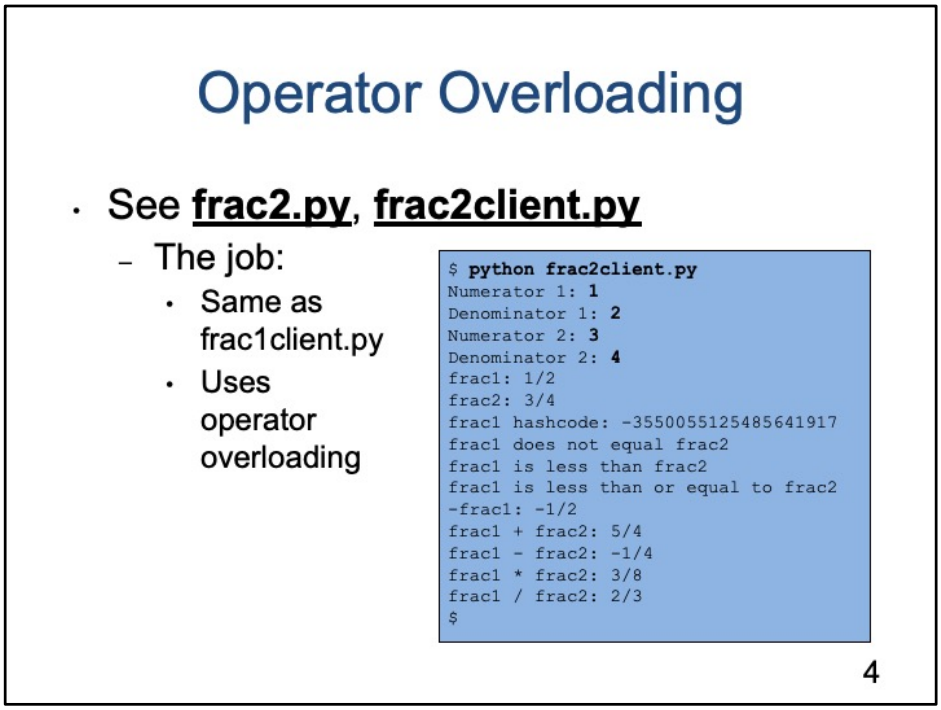
Objectives

Continued from Part 3…

[see slide]

Agenda

- **Operator overloading**
- Object identity and equality
- Inheritance

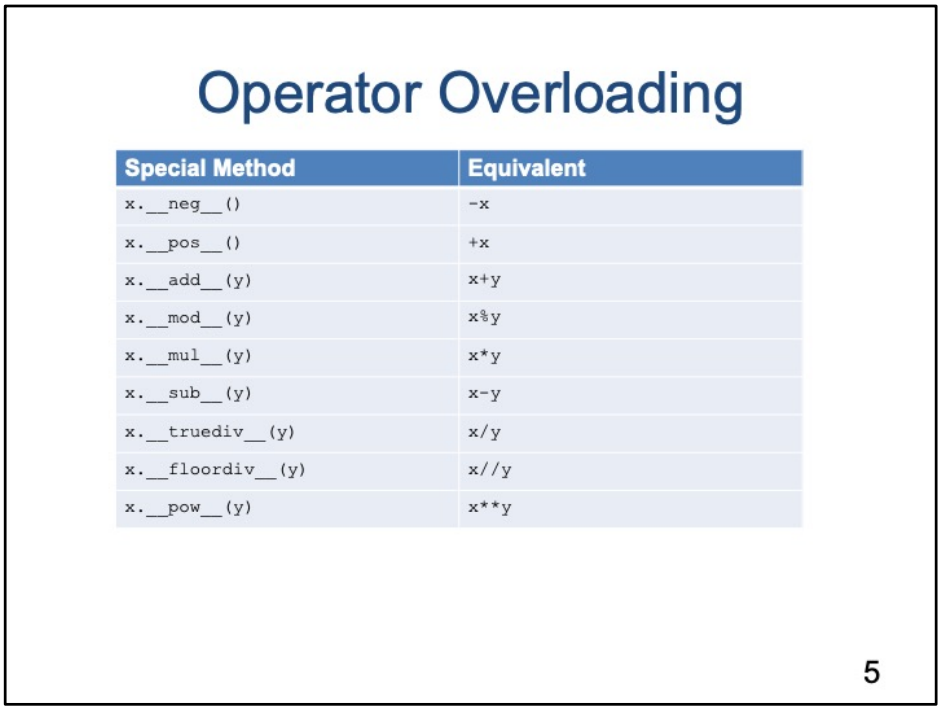


Operator Overloading

[see slide]

Code notes:

__add__() method
frac1 + frac2 same as frac1.__add__(frac2)
__eq__() method
f1 == f2 is same as f1.__eq__(f2)
__ne__() method
f1 != f2 is same as f1.__ne__(f2)
__lt__() method
f1 < f2 is same as f1.__lt__(f2)
__str__() method
Inherited from class object
str(f) is same as f.__str__()
Implicitly called by print function
__hash__() method
Inherited from class object
hash(frac1) is same as frac1.__hash__()
Fraction objects can be hashed
Fraction objects reasonably can be keys in a hash table



Operator Overloading

Very many operators can be overloaded

To overload an operator, define a corresponding special method

For reference…

[see slide]

Operator Overloading

Special Method	Equivalent
`x.__invert__()`	`~x`
`x.__and__(y)`	`x&y`
`x.__lshift__(y)`	`x<<y`
`x.__or__(y)`	`x|y`
`x.__rshift__(y)`	`x>>y`
`x.__xor__(y)`	`x^y`
`x.__float__()`	`float(x)`
`x.__int__()`	`int(x)`
`x.__str__()`	`str(x)`
`x.__hash__()`	`hash(x)`
`x.__abs__()`	`abs(x)`

Operator Overloading

Special Method	Equivalent
`x.__iadd__(y)`	`x+=y`
`x.__ifloordiv__(y)`	`x//=y`
`x.__imod__(y)`	`x%=y`
`x.__imul__(y)`	`x*=y`
`x.__isub__(y)`	`x-=y`
`x.__itruediv__(y)`	`x//=y`
`x.__iand__(y)`	`x&=y`
`x.__ilshift__(y)`	`x<<=y`
`x.__ior__(y)`	`x|=y`
`x.__irshift__(y)`	`x>>=y`
`x.__ixor__(y)`	`x^=y`
`x.__ipow__(y)`	`x**=y`

Operator Overloading

Special Method	Equivalent
`x.__getitem__(y)`	`x[y]`
`x.__setitem__(y, z)`	`x[y] = z`
`x.__contains__(y)`	`y in x`
`x.__delitem__(y)`	`del(x[y])`
`x.__len__()`	`len(x)`

And many more!

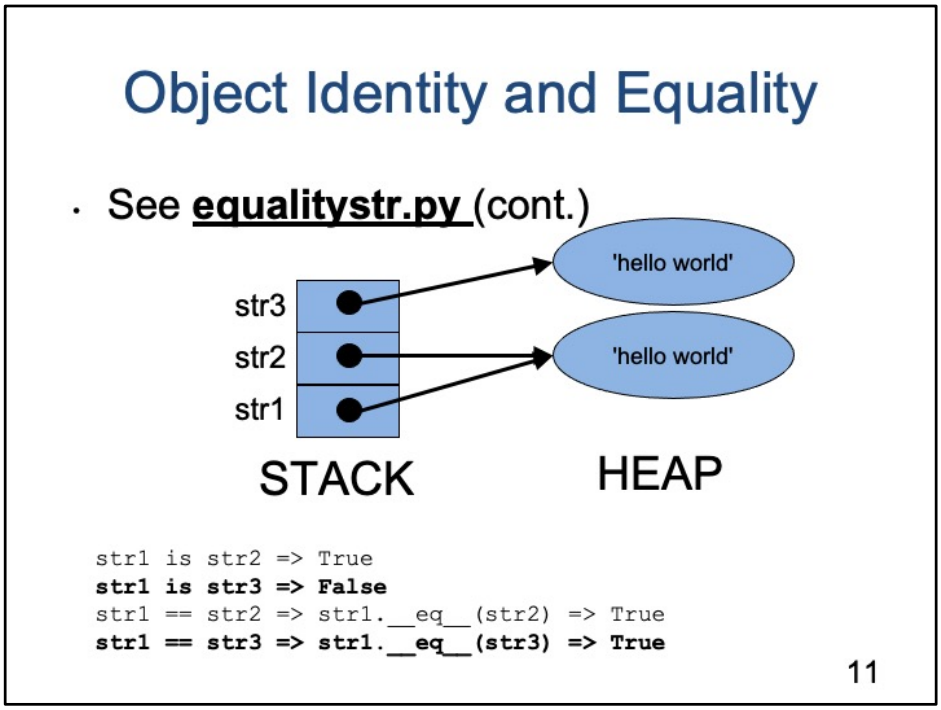
Agenda

- Operator overloading
- **Object identity and equality**
- Inheritance

Object Identity and Equality

- See **equalitystr.py**
  - The job:
    - Compare `str` object references and `str` objects

```
$ python equalitystr.py
str1 and str2 are identical
str1 and str3 are not identical
str1 and str2 are equal
str1 and str3 are equal
$
```



Object Identity and Equality

[see slide]

Code notes:

Strings are objects

str1 is str3

Tests **object references** for **equality**

Or, if you prefer…

Tests **objects** for **identity**

str1 == str3

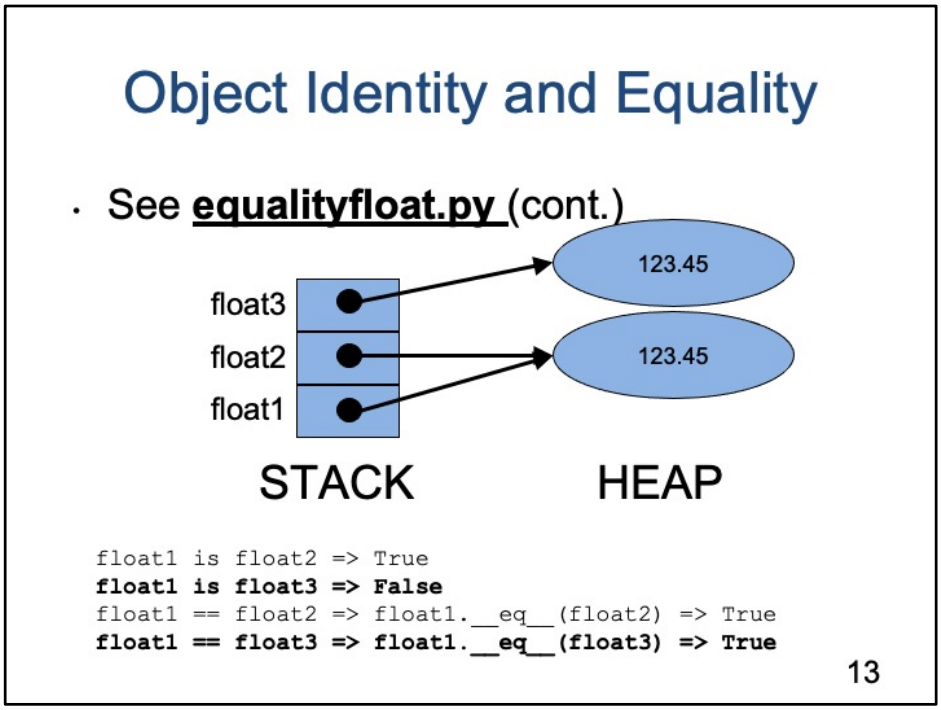
Abbreviation for str1.__eq__(str3)

Tests **objects** for **equality**

Object Identity and Equality

- See **equalityfloat.py**
  - The job:
    - Compare float object references and float objects

```
$ python equalityfloat.py
float1 and float2 are identical
float1 and float3 are not identical
float1 and float2 are equal
float1 and float3 are equal
$
```



Object Identity and Equality

[see slide]

Code notes:

Floats are objects!!!

float1 is float3

Tests **object references** for **equality**

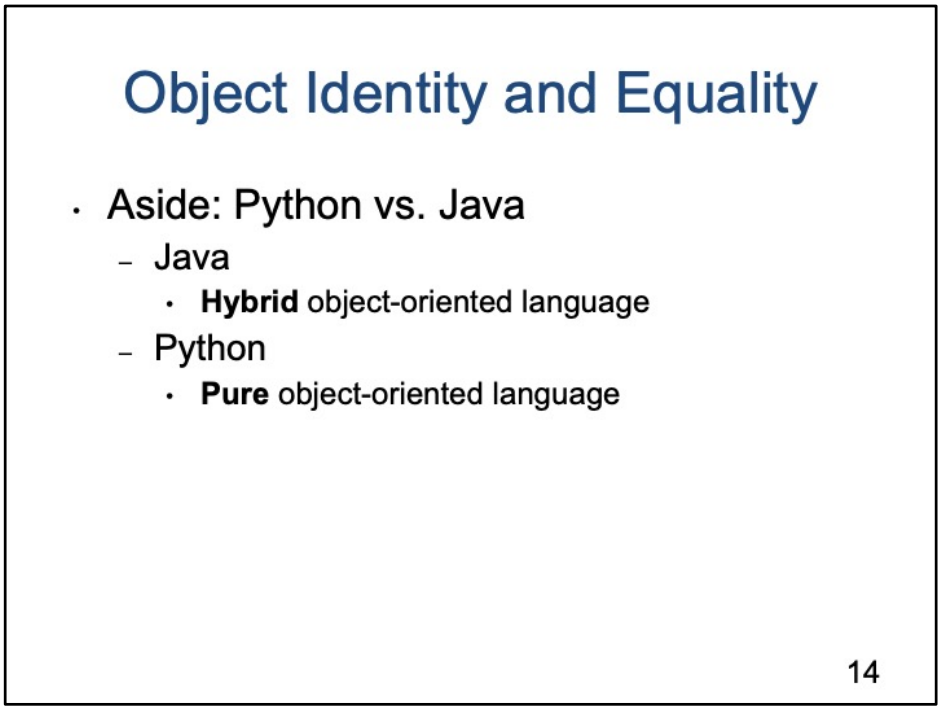
Or, if you prefer…

Tests **objects** for **identity**

float1 == float3

Abbreviation for float1.__eq__(float3)

Tests **objects** for **equality**



Object Identity and Equality

Aside: Python vs. Java

Java

Hybrid object-oriented language

Some data – data of primitive types (int, doubles, etc.) – are not objects

Python

Pure object-oriented language

All data – data of all types (str, float, int, bool, …) – are objects

Object Identity and Equality

Java:

Expression	Compares
`x == y`	(sometimes) Values (sometimes) Object references
`x.equals(y)`	Objects

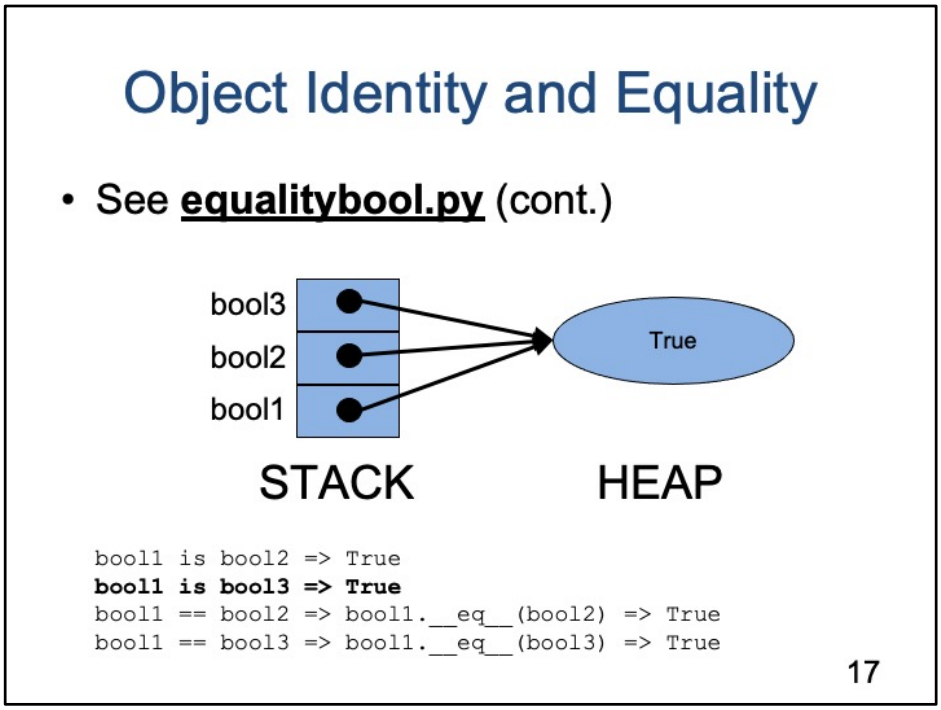
Python:

Expression	Compares
`x is y`	Object references
`x == y` `x.__eq__(y)`	Objects

Object Identity and Equality

- See **equalitybool.py**
  - The job:
    - Compare `bool` object references and `bool` objects

```
$ python equalitybool.py
bool1 and bool2 are identical
bool1 and bool3 are identical
bool1 and bool2 are equal
bool1 and bool3 are equal
$
```



Object Identity and Equality

[see slide]

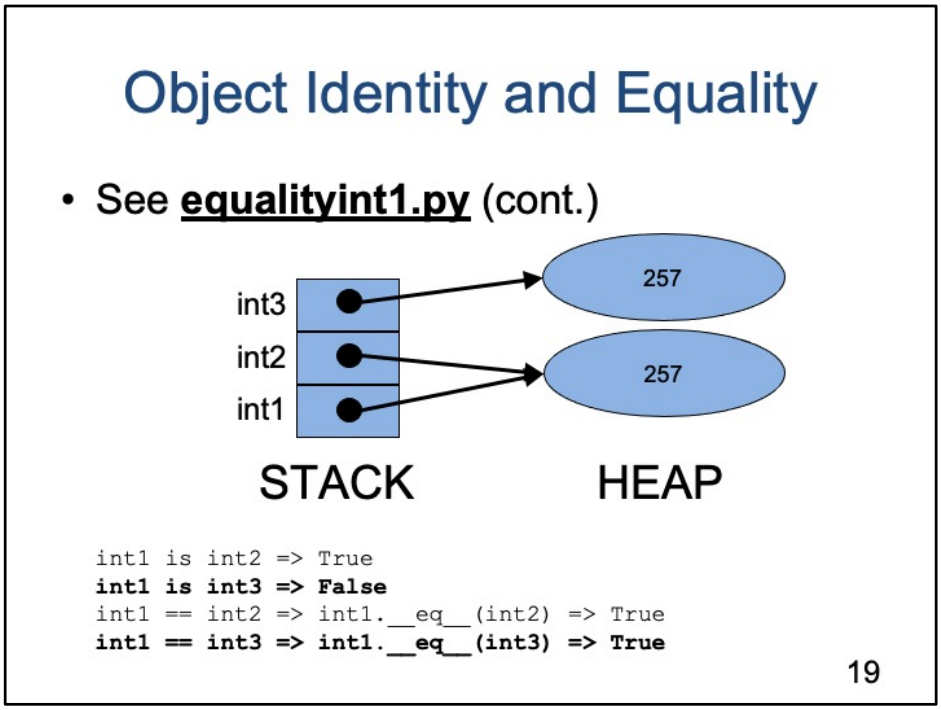
Code notes:

Booleans are objects
A program has exactly one True object
A program has exactly one False object

Object Identity and Equality

- See **equalityint1.py**
  - The job:
    - Compare `int` object references and `int` objects

```
$ python equalityint1.py
int1 and int2 are identical
int1 and int3 are not identical
int1 and int2 are equal
int1 and int3 are equal
$
```



Object Identity and Equality

Code note:

Integers are objects

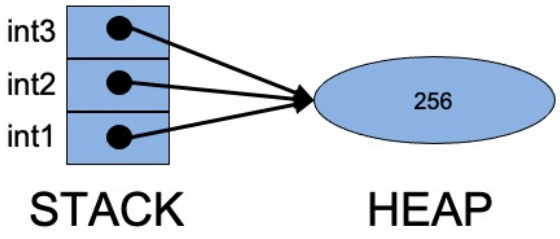
Object Identity and Equality

- See **equalityint2.py**
  - The job:
    - Compare `int` object references and `int` objects

```
$ python equalityint2.py
int1 and int2 are identical
int1 and int3 are identical
int1 and int2 are equal
int1 and int3 are equal
$
```

Object Identity and Equality

- See **equalityint2.py** (cont.)




```
int1 is int2 => True
int1 is int3 => True
int1 == int2 => int1.__eq__(int2) => True
int1 == int3 => int1.__eq__(int3) => True
```

21

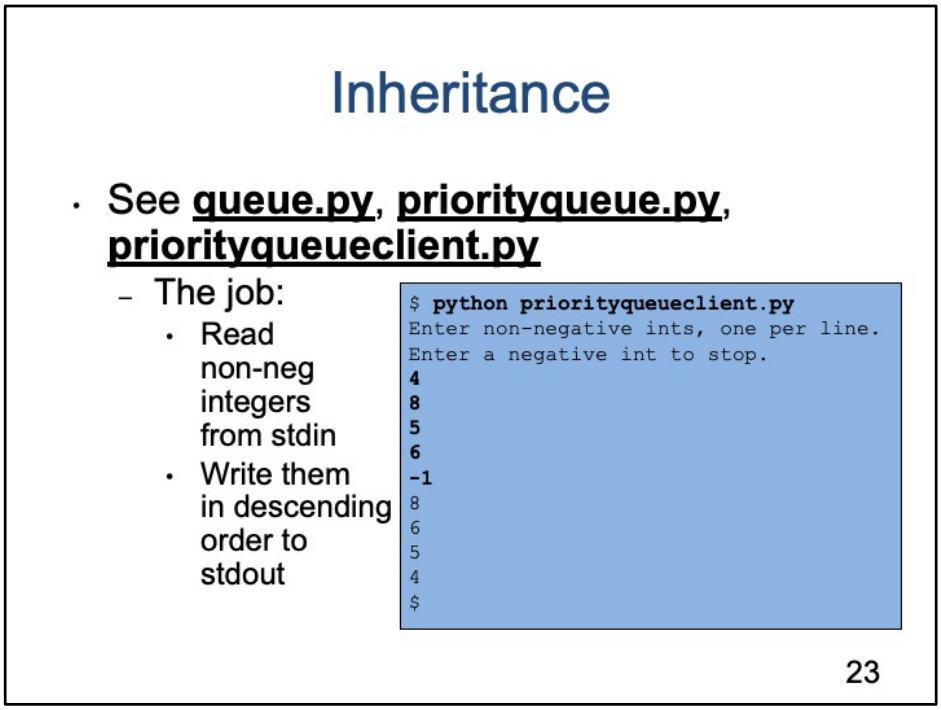
Object Identity and Equality

Code notes:

On our system (cpython 2.8)…
int objects -5 through 256 are instantiated at process startup
A program has exactly one -5 object
A program has exactly one -4 object
…
A program has exactly one 255 object
A program has exactly one 256 object

Agenda

- Operator overloading
- Object identity and equality
- Objects as arguments and parameters
- **Inheritance**



Inheritance

[see slide]

Code notes: queue.py:

An object of class Queue is implemented as a **linked list**

Suggestion : Trace

Class Queue implicitly inherits from class object

An object of class Queue has:

Constructor

put() method

Puts given item at end

get() method

Removes and returns item at front

is_empty() method

Returns True iff the Queue object contains no objects

Additional methods inherited from class object

Inner class definition

Class _Node is defined within class Queue

None keyword

self.item = None

self.item refers to no object

Java: null; C: NULL
Suggestion: trace

Code notes: priorityqueue.py:
Class PriorityQueue explicitly inherits from class Queue
An object of class PriorityQueue has
put() method
Overrides def of put() inherited from Queue
Puts given item into queue **at proper place**
get() method
Inherited from class Queue
isEmpty() method
Inherited from class Queue
Additional methods inherited from class object
Suggestion: trace

Code notes: priorityqueue.py:put()
Recall duck typing…
Python doesn't care about object classes
Python cares only that objects respond to messages sent to them
Objects in a PriorityQueue object can be of any class, with one constraint…
curr.item < item => curr.item.__lt__(item)
Objects in a PriorityQueue object must be comparable via <
Any object in a PriorityQueue object must have __lt__() method

Code notes: priorityqueueclient.py:
Instantiates an object of class PriorityQueue
priorityQueue.put()
Calls put() defined in class PriorityQueue
priorityQueue.is_empty()
Calls is_empty() defined in class Queue
priorityQueue.get()
Calls get() defined in class Queue

Summary

- We have covered these aspects of Python:
  - Operator overloading
  - Object identity and equality
  - Inheritance
- See also:
  - **Appendix 1**: Objects as Arguments and Parameters

Appendix 1:
Objects as Arguments and Parameters

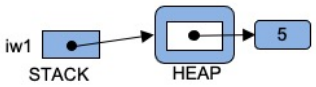
Objects as Args and Params

- See **objectparam1.py**
  - What does it write?

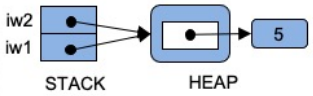
Objects as Args and Params

See **objectparam1.py** (cont.)

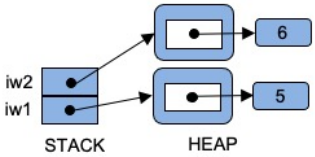
(1) Before call of f()



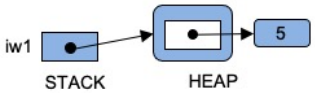
(2) After call of f()



(3) Before return from f()



(4) After return from f()



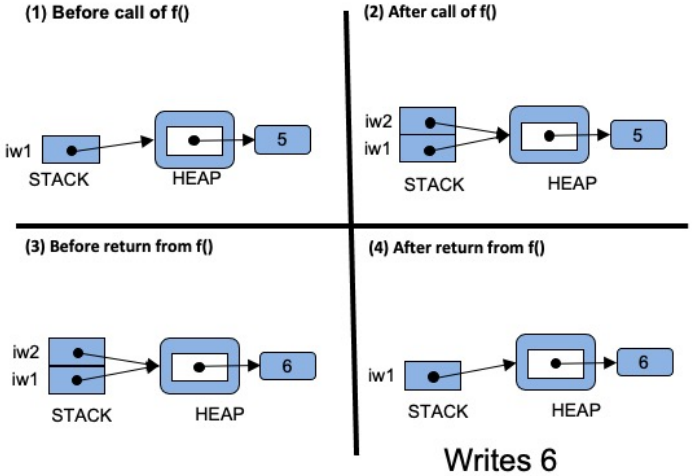
Writes 5

Objects as Args and Params

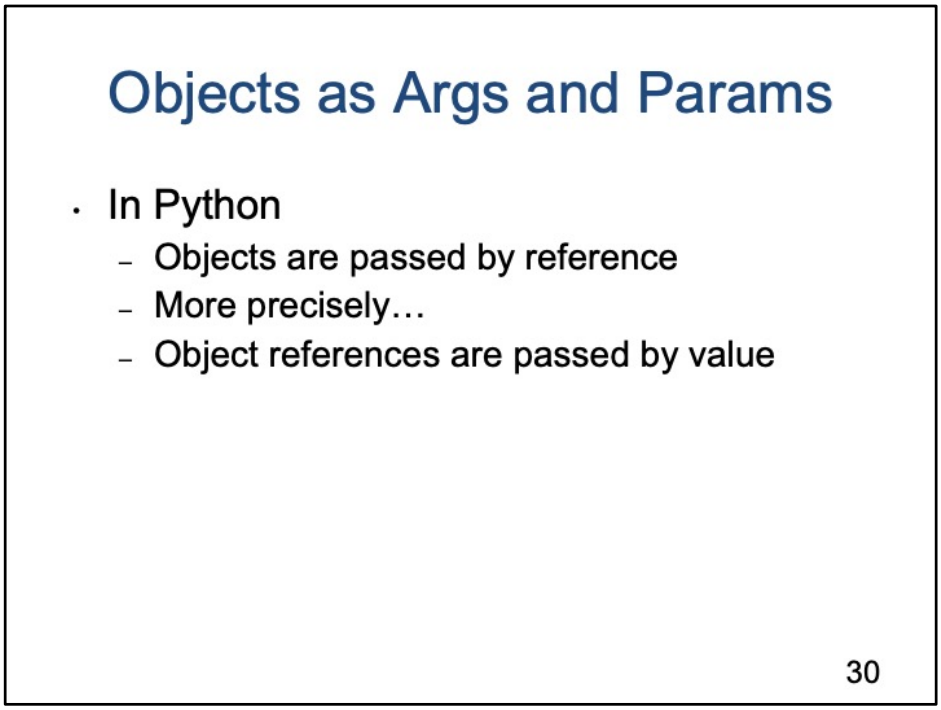
- See **objectparam2.py**
  - What does it write?

Objects as Args and Params

See **objectparam2.py** (cont.)



Writes 6



Objects as Arguments and Parameters

[see slide]

As in Java