

The Python Language (Part 4)

Copyright © 2025 by
Robert M. Dondero, Ph.D.
Princeton University

Objectives

- We will cover:
 - A subset of Python...
 - That is appropriate for COS 333...
 - Through example programs

Agenda

- **Object-oriented programming**
- Operator overloading
- Object identity and equality

Object-Oriented Programming

- See **fractionprelim.py**,
fractionprelimclient.py

```
$ python fractionprelimclient.py
Numerator 1: 1
Denominator 1: 2
Numerator 2: 3
Denominator 2: 4
frac1: 1/2
frac2: 3/4
frac1 does not equal frac2
frac1 is less than frac2
frac1 is less than or equal to frac2
-frac1: -1/2
frac1 + frac2: 5/4
frac1 - frac2: -1/4
frac1 * frac2: 3/8
frac1 / frac2: 2/3
$
```

Object-Oriented Programming

- See **fractionprelim.py**,
fractionprelimclient.py (cont.)

Suppose we add these statements to
the client. What is written?

```
f = fraction.Fraction(3, 4)
f._num = 6
print(f.to_string())
```

Object-Oriented Programming

- See **`fractionprelim.py`**,
`fractionprelimclient.py` (cont.)

What is the effect of executing these
statements in the client:

```
f = fraction.Fraction(3, 4)  
f.num = 6
```

Aside: Name Mangling

- Incidentally:
 - Use of leading **double** underscores causes *name mangling*
 - Example: In Fraction, compiler turns `__num` into `_Fraction__num`

Agenda

- Object-oriented programming
- **Operator overloading**
- Object identity and equality

Operator Overloading

- See **fraction.py**, **fractionclient.py**

```
$ python fractionclient.py
Numerator 1: 1
Denominator 1: 2
Numerator 2: 3
Denominator 2: 4
frac1: 1/2
frac2: 3/4
frac1 does not equal frac2
frac1 is less than frac2
frac1 is less than or equal to frac2
-frac1: -1/2
frac1 + frac2: 5/4
frac1 - frac2: -1/4
frac1 * frac2: 3/8
frac1 / frac2: 2/3
$
```

Operator Overloading

Special Method	Equivalent
<code>x.__neg__()</code>	<code>-x</code>
<code>x.__pos__()</code>	<code>+x</code>
<code>x.__add__(y)</code>	<code>x+y</code>
<code>x.__mod__(y)</code>	<code>x%y</code>
<code>x.__mul__(y)</code>	<code>x*y</code>
<code>x.__sub__(y)</code>	<code>x-y</code>
<code>x.__truediv__(y)</code>	<code>x/y</code>
<code>x.__floordiv__(y)</code>	<code>x//y</code>
<code>x.__pow__(y)</code>	<code>x**y</code>

Operator Overloading

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

Operator Overloading

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

Operator Overloading

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

And many more!

Agenda

- Object-oriented programming
- Operator overloading
- **Object identity and equality**

Object Identity and Equality

- See **EqualityStr.java**

```
$ javac EqualityStr.java
$ java EqualityStr
Enter the first string:
hi
Enter the second string:
hi
???
```

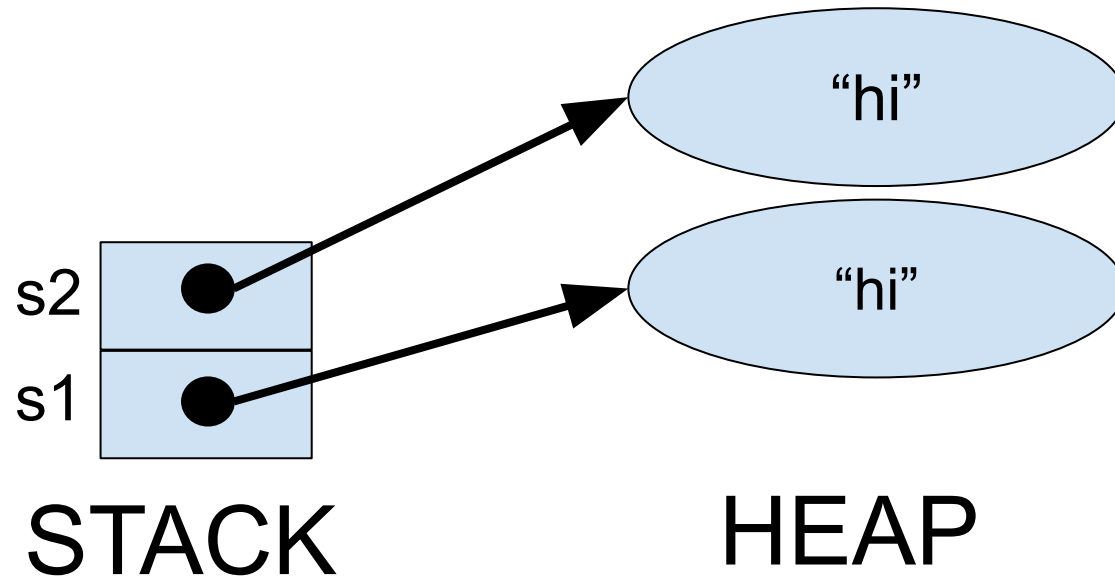
What is written?
(No fair looking at
the next slide.)

Object Identity and Equality

- See **EqualityStr.java**

```
$ javac EqualityStr.java
$ java EqualityStr
Enter the first string:
hi
Enter the second string:
hi
false
true
$
```

Object Identity and Equality

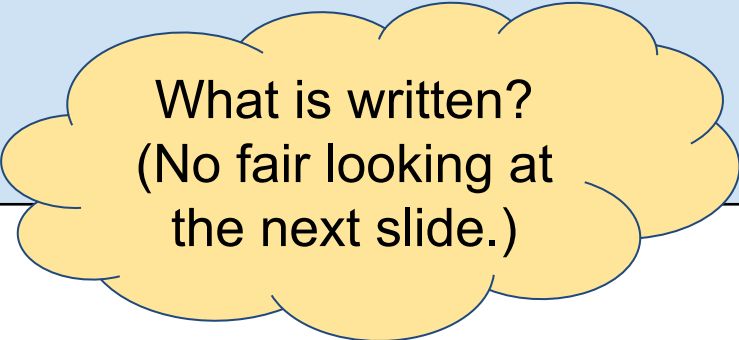


```
s1 == s2 => false  
s1.equals(s2) => true
```

Object Identity and Equality

- See **equalitystr.py**

```
$ python equalitystr.py
Enter the first string:
hi
Enter the second string:
hi
???
```



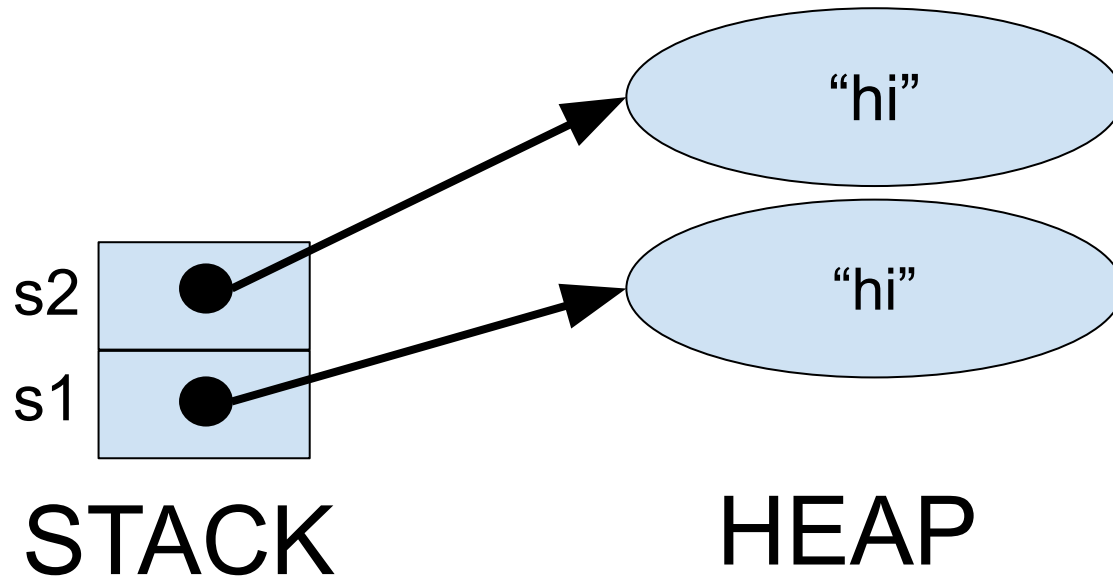
What is written?
(No fair looking at
the next slide.)

Object Identity and Equality

- See **equalitystr.py**

```
$ python equalitystr.py
Enter the first string:
hi
Enter the second string:
hi
False
True
$
```

Object Identity and Equality



`s1 is s2 => False`

`s1 == s2 => s1.__eq__(s2) => True`

Object Identity and Equality

Java, for type String:

Expression	Compares
<code>s1 == s2</code>	Object references
<code>s1.equals(s2)</code>	Objects

Python, for type str:

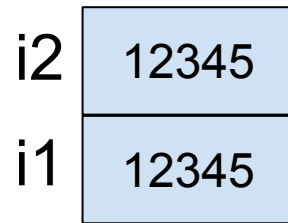
Expression	Compares
<code>s1 is s2</code>	Object references
<code>s1 == s2</code> <code>s1.__eq__(s2)</code>	Objects

Object Identity and Equality

- See **EqualityInt.java**

```
$ javac EqualityInt.java
$ java EqualityInt
Enter the first int:
12345
Enter the second int:
12345
true
$
```

Object Identity and Equality



STACK

```
i1 == i2 => true
```

```
i1.equals(i2) => Illegal
```

Object Identity and Equality

- See **equalityint.py**

```
$ python equalityint.py
Enter the first int:
12345
Enter the second int:
12345
???
```

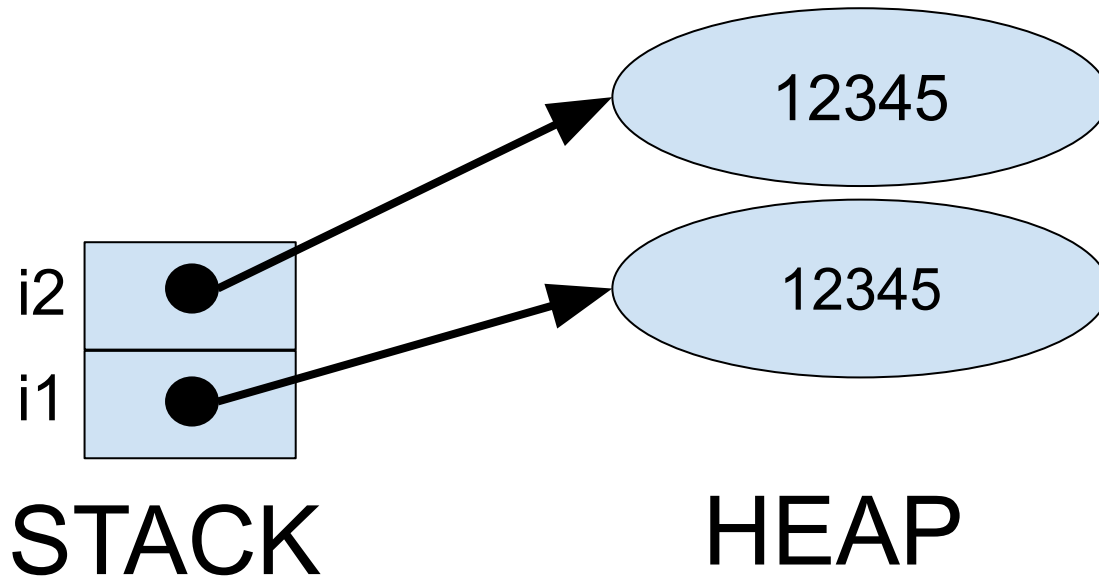
What is written?
(No fair looking at
the next slide.)

Object Identity and Equality

- See **equalityint.py**

```
$ python equalityint.py
Enter the first int:
12345
Enter the second int:
12345
False
True
$
```

Object Identity and Equality



```
i1 is i2 => False
```

```
i1 == i2 => i1.__eq__(i2) => True
```

Object Identity and Equality

Java, for type int:

Expression	Compares
<code>i1 == i2</code>	Values
<code>i1.equals(i2)</code>	Illegal

Python, for type int:

Expression	Compares
<code>i1 is i2</code>	Object references
<code>i1 == i2</code> <code>i1.__eq__(i2)</code>	Objects

Object Identity and Equality

Language	Characteristics
Java	Hybrid OO language
Python	Pure OO language

Java:

Expression	Compares
<code>x == y</code>	(sometimes) Values (sometimes) Object references
<code>x.equals(y)</code>	Objects

Python:

Expression	Compares
<code>x is y</code>	Object references
<code>x == y</code> <code>x.__eq__(y)</code>	Objects

Lecture Summary

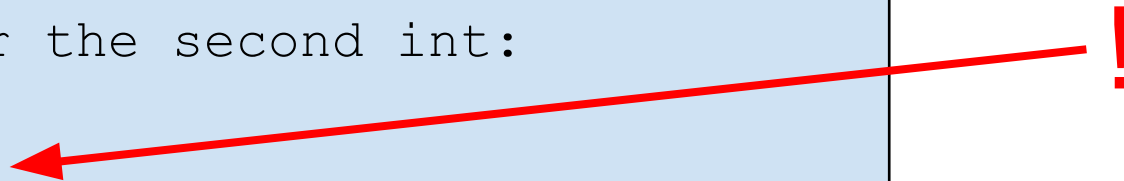
- In this lecture we covered these aspects of Python:
 - Object-oriented programming
 - Operator overloading
 - Object identity and equality
- See also:
 - **Appendix 1:** Predefined Objects
 - **Appendix 2:** Objects as Arguments and Parameters
 - **Appendix 3:** Inheritance

Appendix 1: Predefined Objects

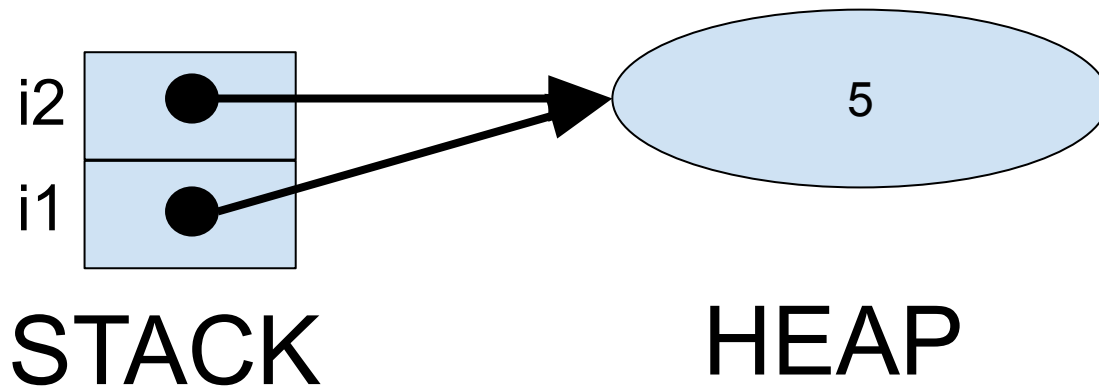
Predefined Objects

- Recall equalityint.py

```
$ python equalityint.py
Enter the first int:
5
Enter the second int:
5
True
True
$
```



Predefined Objects



```
i1 is i2 => True
```

```
i1 == i2 => i1.__eq__(i2) => True
```

Predefined Objects

- Explanation:
 - For efficiency, cPython creates `int` objects `-5...256` at process startup
 - At any time:
 - Only one `-5 int` object exists
 - Only one `-4 int` object exists
 - ...
 - Only one `256 int` object exists

Predefined Objects

- And, incidentally...
 - For efficiency, cPython creates `bool` objects `True` and `False` at process startup
 - At any time:
 - Only one `True` object exists
 - Only one `False` object exists

Appendix 2: Objects as Arguments and Parameters

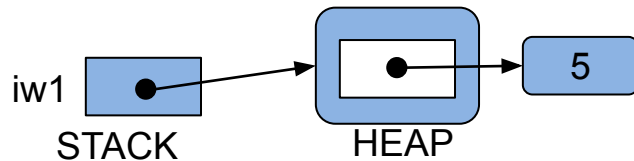
Objects as Args and Params

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

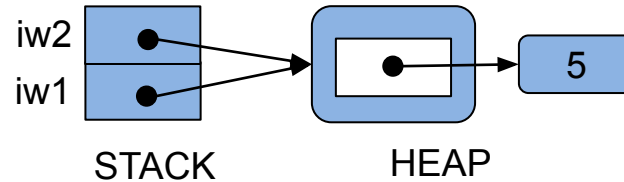
Objects as Args and Params

See [objectparam1.py](#) (cont.)

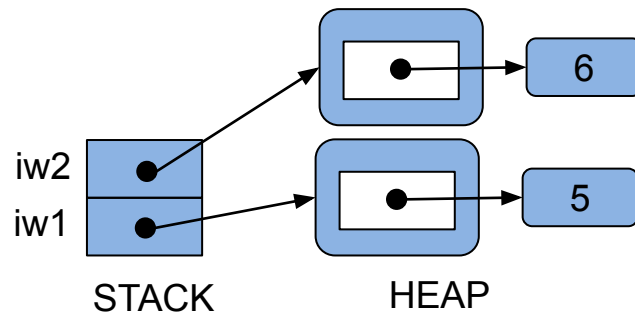
(1) Before call of my_func()



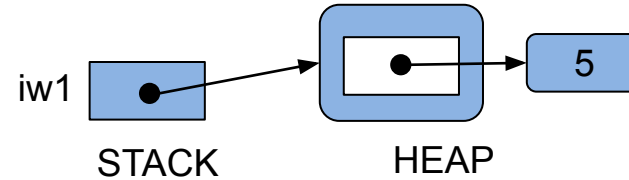
(2) After call of my_func()



(3) Before return from my_func()



(4) After return from my_func()



Writes 5

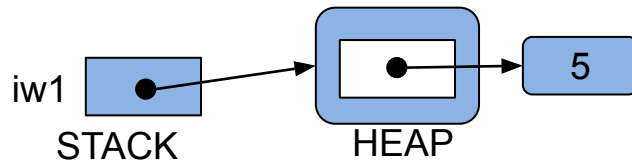
Objects as Args and Params

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

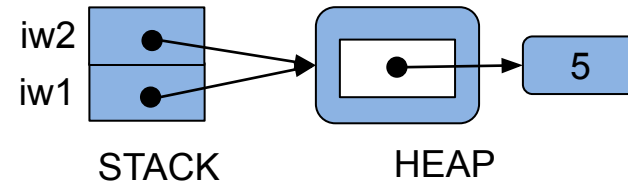
Objects as Args and Params

See [objectparam2.py](#) (cont.)

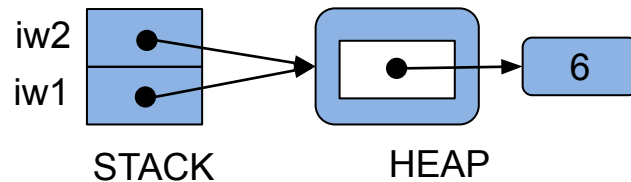
(1) Before call of my_func()



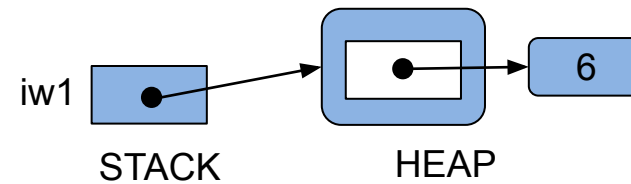
(2) After call of my_func()



(3) Before return from my_func()



(4) After return from my_func()



Writes 6

Objects as Args and Params

- In Python
 - As in Java...
 - Objects are passed by reference
 - More precisely...
 - Object references are passed by value

Appendix 3: Inheritance

Inheritance

- See [queue.py](#), [priorityqueue.py](#), [priorityqueueclient.py](#)

```
$ python priorityqueueclient.py
Enter non-negative ints, one per line.
Enter a negative int to stop.
4
8
5
6
-1
8
6
5
4
$
```