

The Python Language (Part 4)

Copyright © 2022 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

2

Objectives

Continued from Part 3...

[see slide]

Agenda

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

Operator Overloading

- See **frac2.py**, **frac2client.py**

- The job:
 - Same as `frac1client.py`
 - Uses operator overloading

```
$ python frac2client.py
Numerator 1: 1
Denominator 1: 2
Numerator 2: 3
Denominator 2: 4
frac1: 1/2
frac2: 3/4
frac1 hashCode: -3550055125485641917
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
$
```

4

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

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>

5

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
<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

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

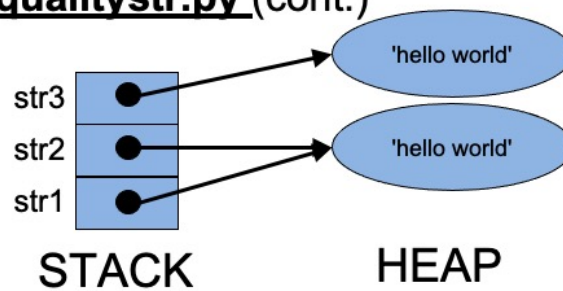
Object Identity and Equality

- 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 [equalitystr.py](#) (cont.)



```
str1 is str2 => True
str1 is str3 => False
str1 == str2 => str1.__eq__(str2) => True
str1 == str3 => str1.__eq__(str3) => True
```

11

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**

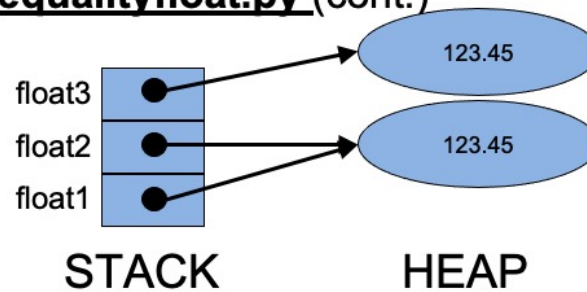
Object Identity and Equality

- 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 **equalityfloat.py** (cont.)



```
float1 is float2 => True
float1 is float3 => False
float1 == float2 => float1.__eq__(float2) => True
float1 == float3 => float1.__eq__(float3) => True
```

13

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
 - Python
 - **Pure** object-oriented language

14

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
<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

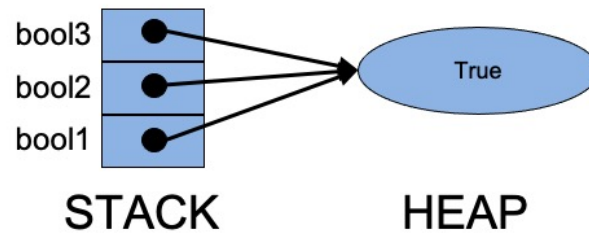
Object Identity and Equality

- 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 [equalitybool.py](#) (cont.)



```
bool1 is bool2 => True
bool1 is bool3 => True
bool1 == bool2 => bool1.__eq__(bool2) => True
bool1 == bool3 => bool1.__eq__(bool3) => True
```

17

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

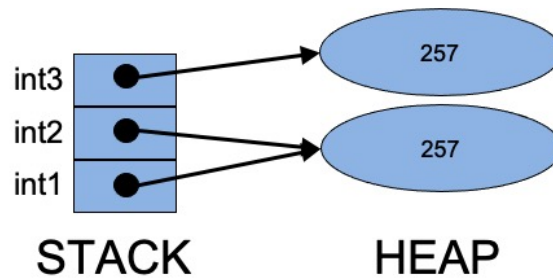
Object Identity and Equality

- 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

- See **equalityint1.py** (cont.)



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

19

Object Identity and Equality

Code note:

Integers are objects

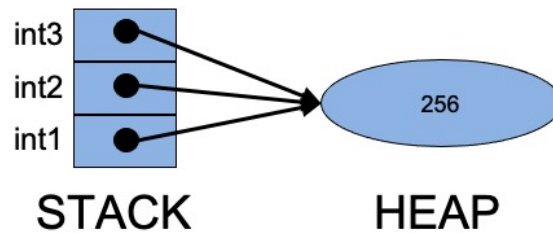
Object Identity and Equality

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

- The job:

- Read non-neg integers from stdin
- Write them in descending order to stdout

```
$ 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
$
```

23

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

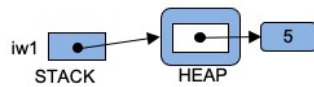
Objects as Args and Params

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

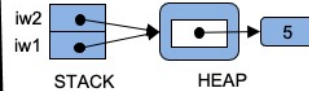
Objects as Args and Params

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

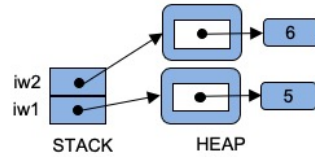
(1) Before call of f()



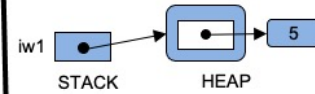
(2) After call of f()



(3) Before return from f()



(4) After return from f()



Writes 5

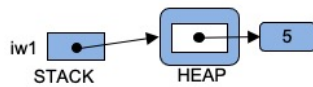
Objects as Args and Params

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

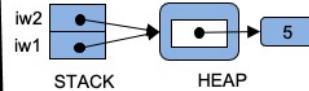
Objects as Args and Params

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

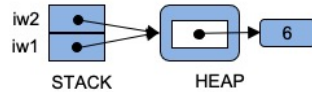
(1) Before call of f()



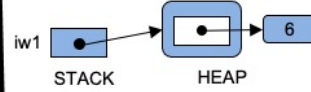
(2) After call of f()



(3) Before return from f()



(4) After return from f()



Writes 6

Objects as Args and Params

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

30

Objects as Arguments and Parameters

[see slide]

As in Java