ACM/CS 114
Parallel algorithms for scientific applications

Michael A. G. Aïvázis

California Institute of Technology

Spring 2012

Overloading operators in classes

- *Don't!*
- most python operations involving instances can be intercepted and customized
- through methods that have special names

method	*purpose*	*method*	*purpose*
`__init__`	construction: `x = X()`	`__getattr__`	member access: `x.name`
`__del__`	destruction	`__getitem__`	indexing: `x[5]`
`__str__`	string coercion: `str(x)`	`__setitem__`	indexing: `x[5] = 0`
`__repr__`	representation: `repr(x)`	`__add__`	addition: `x + other`
`__len__`	size, truth tests: `len(x)`	`__radd__`	addition: `other + x`
`__cmp__`	comparisons: `cmp(x), x < other`	`__and__`	logic: `x and other`
`__call__`	function class: `x()`	`__or__`	logic: `x or other`

Namespace rules

- a more complete story
  - unqualified names are looked up in a chain of lexical namespaces
  - qualified names conduct a search in the indicated namespace
  - scopes initialize object namespaces: packages/modules, classes, instances
- unqualified names
  - are global on read
  - are local on write, unless explicitly marked `global`
- qualified names, e.g. `instance.name`, are looked up in the indicated namespace
  - module and package
  - instance, then class record, then ancestors as specified in the `__mro__`
- namespace dictionaries
  - `__dict__`
  - name qualification is a dictionary lookup

Exceptions

- a non-local, high level control flow device
- exceptions are used to signal
  - critical errors
  - but also recoverable runtime failures
- raised by the interpreter whenever an error is detected
  - there is an extensive exception class hierarchy that captures and documents a wide variety of errors
- user code can trigger an exception using the `raise` statement
- the `try ... except` statement sets up a net for catching exceptions
- proper exception hierarchies are an extremely important part of application design

Raising and catching exceptions

- exceptions are triggered by `raise`

```
raise <expression>
```

where `<expression>` must evaluate to an instance of a class that derives from `Exception`, the base class of all python exceptions

- exceptions may *chain* to other exceptions

```
raise <expression> from <expression>
```

this form is most useful after catching some other exception and you want to preserve the original error

- to catch an exception, you have to set up a `try` block; the full form is

```
try:
   <statements>
except <expression> as <name>:
   <statements>
else:
   <statements>
finally:
   <statements>
```

there may be as many `except` sections as you need; most parts of the statement are optional