Operator Overloading

You can overload definitions of all of Python's operators to apply to newly defined classes. Each operator has a corresponding method name assigned to it. For example, + uses __add__, - uses __sub__, etc.

```python
Given
class Triple:
def __init__(self, A=0, B=0, C=0):
    self.a = A
    self.b = B
    self.c = C
def __str__(self):
    return "(%d, %d, %d)" % (self.a, self.b, self.c)
def __add__(self, other):
    return Triple(self.a + other.a, self.b + other.b, self.c + other.c)
the following code
t1=Triple(1,2,3)
t2=Triple(4,5,6)
print t1+t2
produces
(5, 7, 9)
```

Exceptions

Python provides an exception mechanism that's quite similar to the one used by Java. You “throw” an exception by using a raise statement:

```python
raise exceptionValue
```

There are numerous predefined exceptions, including OverflowError (arithmetic overflow), EOFError (when end-of-file is hit), NameError (when an undeclared identifier is referenced), etc.

```python
Exceptions
You may define your own exceptions as subclasses of the predefined class Exception:
class badValue(Exception):
    def __init__(self, val):
        self.value = val
You catch exceptions in Python's version of a try statement:
```
the except clause. If the raised exception is not matched by any except clause, the next enclosing try is considered, or the exception is reraised at the point of call. For example, using our badValue exception class,

```python
def sqrt(val):
    if val < 0.0:
        raise badValue(val)
    else:
        return cmath.sqrt(val)

try:
    print "Ans =", sqrt(-123.0)
except badValue, b:
    print "Can’t take sqrt of", b.value
```

When executed, we get

```
Ans = Can’t take sqrt of -123.0
```

**Modules**

Python contains a module feature that allows you to access Python code stored in files or libraries. If you have a source file `mydefs.py` the command

```python
import mydefs
```

will read in all the definitions stored in the file. What’s read in can be seen by executing

```python
dir(mydefs)
```

To access an imported definition, you qualify it with the name of the module. For example,

```python
mydefs.fct
```

accesses `fct` which is defined in module `mydefs`.

To avoid explicit qualification you can use the command

```python
from modulename import id1, id2, ...
```

This makes `id1`, `id2`, ... available without qualification. For example,

```python
>>> from test import sqrt
>>> sqrt(123)
(11.0905365064+0j)
```

You can use the command

```python
from modulename import *
```

to import (without qualification) all the definitions in `modulename`.

**The Python Library**

One of the great strengths of Python is that it contains a vast number of modules (at least several hundred) known collectively as the Python Library. What makes Python really useful is the range of prewritten modules you can access. Included are network access modules, multimedia utilities, data base access, and much more.

See

`www.python.org/doc/lib`

for an up-to-date listing of what’s available.