Why Python?

• Because it’s easy and great fun!
  • only 15 years old, yet very popular now
    • a wide-range of applications, esp. in AI and Web
  • extremely easy to learn
    • many schools have shifted their intro-courses to Python
  • fast to write
    • much shorter code compared to C, C++, and Java
  • easy to read and maintain
    • more English-like syntax and a smaller semantic-gap
On to Python...
“Hello, World”

- **C**
  ```c
  #include <stdio.h>

  int main(int argc, char ** argv)
  {
    printf(“Hello, World!\n”);
  }
  ```

- **Java**
  ```java
  public class Hello
  {
    public static void main(String argv[])
    {
      System.out.println(“Hello, World!”);
    }
  }
  ```

- **now in Python**
  ```python
  print “Hello, World!”
  ```
void print_array(char* a[], int len) {
    int i;
    for (i = 0; i < len; i++) {
        printf("%s\n", a[i]);
    }
}

for element in list:
    print element
    only indentations
    no { ... } blocks!

or even simpler:
    print list

C

Python

for ... in ...:
    ...
    no C-style for-loops!

for (i = 0; i < 10; i++)
**Reversing an Array**

```java
static int[] reverse_array(int a[])
{
    int [] temp = new int[ a.length ];
    for (int i = 0; i < len; i++)
    {
        temp [i] = a [a.length - i - 1];
    }
    return temp;
}
```

```python
def rev(a):
    if a == []:
        return []
    else:
        return rev(a[1:]) + [a[0]]
```

**or even simpler:**

```python
a.reverse()  # built-in list-processing function
```

**Java**

**Python**

**no need to specify argument and return types!**

*python will figure it out. (dynamically typed)*
public void sort(int low, int high) {
    if (low >= high) return;
    int p = partition(low, high);
    sort(low, p);
    sort(p + 1, high);
}

void swap(int i, int j) {
    int temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}

int partition(int low, int high) {
    int pivot = a[low];
    int i = low - 1;
    int j = high + 1;
    while (i < j) {
        i++;
        while (a[i] < pivot) i++;
        j--;
        while (a[j] > pivot) j--;
        if (i < j) swap(i, j);
    }
    return j;
}

def sort(a):
    if a == []:
        return []
    else:
        pivot = a[0]
        left = [x for x in a if x < pivot]
        right = [x for x in a[1:] if x >= pivot]
        return sort(left) + [pivot] + sort(right)
Basic Python Syntax
Numbers and Strings

- like Java, Python has built-in (atomic) types
  - numbers (\texttt{int}, \texttt{float}), \texttt{bool}, \texttt{string}, \texttt{list}, etc.
  - numeric operators: + - * / ** %

```python
>>> a = 5
>>> b = 3
>>> type (5)
<type 'int'>
>>> a += 4
>>> a
9
>>> c = 1.5
>>> 5/2
2
>>> 5/2.
2.5
>>> 5 ** 2
25
```

no \texttt{i++} or \texttt{++i}

```python
>>> s = "hey"
>>> s + " guys"
'hey guys'
>>> len(s)
3
>>> s[0]
'h'
>>> s[-1]
'y'
```

```python
>>> from \_\_future\_\_ import division
>>> 5/2
2.5
```

recommended!
Assignments and Comparisons

```python
>>> a = b = 0
>>> a
0
>>> b
0

>>> a, b = 3, 5
>>> a + b
8
>>> (a, b) = (3, 5)
>>> a + b
8
>>> a, b = b, a
(swap)

>>> a = b = 0
>>> a == b
True
>>> type (3 == 5)
<type 'bool'>
>>> "my" == 'my'
True

>>> (1, 2) == (1, 2)
True
>>> 1, 2 == 1, 2
???
(1, False, 2)
```
for loops and `range()`

- `for` always iterates through a list or sequence

```python
>>> sum = 0
>>> for i in range(10):
...     sum += i
... print sum
45

>>> for word in ["welcome", "to", "python"]:...
...     print word,
...
welcome to python

>>> range(5), range(4,6), range(1,7,2)
([0, 1, 2, 3, 4], [4, 5], [1, 3, 5])
```

Java 1.5

```java
forEach (String word : words)
System.out.println(word)
```
while loops

- very similar to `while` in Java and C
  - but be careful
    - `in` behaves differently in `for` and `while`
  - `break` statement, same as in Java/C

```python
>>> a, b = 0, 1
>>> while b <= 5:
...     print b
...     a, b = b, a+b
...
1
1
1
2
3
5
```

fibonacci series

• very similar to `while` in Java and C
• but be careful
  • `in` behaves differently in `for` and `while`
• `break` statement, same as in Java/C
Conditionals

```python
>>> if x < 10 and x >= 0:
...     print x, "is a digit"
...
>>> False and False or True
True
>>> not True
False
```
if ... elif ... else

```python
>>> a = "foo"
>>> if a in ["blue", "yellow", "red"]:
...     print a + " is a color"
... else:
...     print a + " is a country"
... else:
...     print "I don't know what", a, "is!"

I don't know what foo is!
```

```c
switch (a) {
    case "blue":
        print ...; break;
    case "US":
    case "China":
        print ...; break;
    else:
        print ...;
}
```
break, continue and else

- **break** and **continue** borrowed from C/Java
- special **else** in loops
  - when loop terminated *normally* (i.e., not by **break**)
  - very handy in testing a set of properties

```python
>>> for n in range(2, 10):
    ...    for x in range(2, n):
    ...        if n % x == 0:
    ...            break
    ...    else:
    ...        print n,
...```

**prime numbers**

```c
for (n=2; n<10; n++) {
    good = true;
    for (x=2; x<n; x++)
        if (n % x == 0) {
            good = false;
            break;
        }
    if (good)
        printf("%d ", n);
}```
Defining a Function  

- no type declarations needed! **wow!**
- Python will figure it out at run-time
  - you get a run-time error for type violation
  - well, Python does not have a compile-error at all

```python
>>> def fact(n):
...     if n == 0:
...         return 1
...     else:
...         return n * fact(n-1)

>>> fact(4)
24
```
Default Values

```python
>>> def add(a, L=[]):
...     return L + [a]
...

>>> add(1)
[1]

>>> add(1,1)
error!

>>> add(add(1))
[[1]]

>>> add(add(1), add(1))
???
[1, [1]]
```

lists are heterogenous!
Lists

heterogeneous variable-sized array

```
a = [1,'python', [2,'4']]```
Basic List Operations

- length, subscript, and slicing

```python
>>> a = [1, 'python', [2, '4']]
>>> len(a)
3
>>> a[2][1]
'4'
>>> a[3]
IndexError!
>>> a[-2]
'python'
>>> a[1:2]
['python']
>>> a[0:3:2]
[1, [2, '4']]
>>> a[0::2]
[1, [2, '4']]`
```
+ extend, +=, append

- extend (+=) and append mutates the list!

```python
>>> a = [1,'python', [2,'4']]
>>> a + [2]
[1, 'python', [2, '4'], 2]
>>> a.extend([2, 3])
>>> a
[1, 'python', [2, '4'], 2, 3]
same as  a += [2, 3]

>>> a.append('5')
>>> a
[1, 'python', [2, '4'], 2, 3, '5']
>>> a[2].append('xtra')
>>> a
[1, 'python', [2, '4', 'xtra'], 2, 3, '5']
```
Comparison and Reference

- as in Java, comparing built-in types is by value
- by contrast, comparing objects is by reference

```
>>> [1, '2'] == [1, '2']
True
>>> a = b = [1, '2']
>>> a == b
True
>>> a is b
True
>>> a[1] = 5
>>> a
[1, 5]
>>> a = 4
>>> b
[1, 5]
>>> a is b
False
>>> c = b[:]
>>> c
[1, 5]
>>> c == b
True
>>> c is b
False
>>> b[:0] = [2]
>>> b
[2, 1, 5]
>>> b[1:3] = []
>>> b
[2]
```

slicing gets a shallow copy
insertion
deletion

```
>>> a += b
>>> a
[2, 1, 5]
>>> a += b
>>> a
[2, 1, 5]
>>> a is b
True
```

a += b means a.extend(b)

NOT

a = a + b !!
List Comprehension

```python
>>> a = [1, 5, 2, 3, 4, 6]
>>> [x*2 for x in a]
[2, 10, 4, 6, 8, 12]

>>> [x for x in a if len([y for y in a if y < x]) == 3]
[4]

>>> a = range(2,10)
>>> [x*x for x in a if [y for y in a if y < x and (x % y == 0)] == []]
[4, 9, 25, 49]
```

4th smallest element

square of prime numbers
Strings

sequence of characters
String Literals

- single quotes and double quotes; escape chars
- strings are immutable!

```python
>>> 'spam eggs'
'spam eggs'
>>> 'doesn\'t'
SyntaxError!
>>> 'doesn\'t'
"doesn\'t"
>>> "doesn\'t"
"doesn\'t"
>>> "doesn\"t"
"doesn\"t"
>>> "\"Yes,\" he said."
"Yes," he said.'
>>> s = "Isn\'t," she said.'
>>> s
"Isn\'t," she said.'
>>> s[0] = 'b'
TypeError!
>>> s = "a\nb"
>>> s
'a\nb'
>>> print s
a
b
>>> "\"Yes,\" he said."
"Yes," he said.'
>>> s = "Isn\'t," she said.'
>>> s
"Isn\'t," she said.'
>>> print s
"Isn\'t," she said.
```
Basic String Operations

- `join`, `split`, `strip`
- `upper()`, `lower()`

```python
>>> s = " this is a python course. \n"
>>> words = s.split()
>>> words
['this', 'is', 'a', 'python', 'course. ']
>>> s.strip()
'this is a python course.'
>>> " ".join(words)
'this is a python course.'
>>> " ; ".join(words).split(" ; ")
['this', 'is', 'a', 'python', 'course. ']
>>> s.upper()
' THIS IS A PYTHON COURSE. \n'
```

[http://docs.python.org/lib/string-methods.html](http://docs.python.org/lib/string-methods.html)
Basic Search/Replace in String

```python
>>> "this is a course".find("is")
2
>>> "this is a course".find("is a")
5
>>> "this is a course".find("is at")
-1

>>> "this is a course".replace("is", "was")
'thwas was a course'
>>> "this is a course".replace(" is", " was")
'this was a course'
>>> "this is a course".replace("was", "were")
'this is a course'
```

these operations are much faster than regexps!
>>> print "%.2f%%" % 97.2363
97.24%

>>> s = '%s has %03d quote types.' % ("Python", 2)
>>> print s
Python has 002 quote types.
## Pythonic Styles

- **do not write ...**

<table>
<thead>
<tr>
<th>Python Code</th>
<th>Pythonic Way</th>
</tr>
</thead>
<tbody>
<tr>
<td>for key in d.keys():</td>
<td>for key in d:</td>
</tr>
<tr>
<td>if d.has_key(key):</td>
<td>if key in d:</td>
</tr>
</tbody>
</table>
| i = 0 for x in a: ...
  i += 1 | for i, x in enumerate(a): |
| a[0:len(a) - i] | a[:-i] |
| for line in \sys.stdin.readlines(): | for line in sys.stdin: |
| for x in a: print x, print | print " ".join(map(str, a)) |
| s = "" for i in range(lev): s += " 
  print s | print " " * lev |
Tuples

immutable lists
Tuples and Equality

- caveat: singleton tuple
- `==`, `is`, `is not`
enumerate

```python
>>> words = ['this', 'is', 'python']
>>> i = 0
>>> for word in words:
...     i += 1
...     print i, word
...     print i+1, word
...     print i+2, word
1 this
2 is
3 python

>>> for i, word in enumerate(words):
...     print i+1, word
...     print i+2, word
...     print i+3, word
1 this
2 is
3 python
```

- how to enumerate two lists/tuples simultaneously?
>>> a = [1, 2]
>>> b = ['a', 'b']

>>> zip (a,b)
[(1, 'a'), (2, 'b')]

>>> zip(a,b,a)
[(1, 'a', 1), (2, 'b', 2)]

>>> zip ([1], b)
[(1, 'a')]

>>> a = ['p', 'q']; b = [[2, 3], [5, 6]]
>>> for i, (x, [_, y]) in enumerate(zip(a, b)):
...    print i, x, y

... 0 p 3
1 q 6
Dictionaries

(heterogeneous) hash maps
Constructing Dicts

- **key : value pairs**

```python
>>> d = {'a': 1, 'b': 2, 'c': 1}
>>> d['b']
2
>>> d['b'] = 3
>>> d['b']
3
>>> d['e']
KeyError!
>>> d.has_key('a')
True
>>> 'a' in d
True
>>> d.keys()
['a', 'c', 'b']
>>> d.values()
[1, 1, 3]
```
default values

• counting frequencies

```python
>>> def incr(d, key):
...     if key not in d:
...         d[key] = 1
...     else:
...         d[key] += 1
...     
...     def incr(d, key):
...         d[key] = d.get(key, 0) + 1
...     
...     incr(d, 'z')
...     d
...     {'a': 1, 'c': 1, 'b': 2, 'z': 1}
...     incr(d, 'b')
...     d
...     {'a': 1, 'c': 1, 'b': 3, 'z': 1}
```
defaultdict

• best feature introduced in Python 2.5

```python
>>> from collections import defaultdict
>>> d = defaultdict(int)
>>> d['a']
0
>>> d['b'] += 1
>>> d
{'a': 0, 'b': 1}

>>> d = defaultdict(list)
>>> d['b'] += [1]
>>> d
{'b': [1]}

>>> d = defaultdict(lambda : <expr>)
```
Basic import and I/O
import and I/O

• similar to import in Java
• File I/O much easier than Java

```python
demo
import sys
for line in sys.stdin:
    print line.split()
```

```python
or
from sys import *
for line in stdin:
    print line.split()
```

```
import System;
Java
import System.*;
```

```python
g = open("my.out", "wt")
g.close()
```

```python
to read a line:
    line = f.readline()
to read all the lines:
    lines = f.readlines()
```

```
file copy
note this comma!
```
import and __main__

- multiple source files (modules)
  - C: `#include "my.h"`
  - Java: `import My`
- demo
- handy for debugging

```python
def pp(a):
    print " " . join(a)

if __name__ == "__main__":
    from sys import *
    a = stdin.readline()
    pp (a.split())
```

```python
>>> import foo
>>> pp([1,2,3])
1 2 3
```
Functional Programming
lambda

• map/filter in one line for custom functions?
  • “anonymous inline function”
  • borrowed from LISP, Scheme, ML, OCaml

>>> f = lambda x: x*2
>>> f(1)
2
>>> map (lambda x: x**2, [1, 2])
[1, 4]
>>> filter (lambda x: x > 0, [-1, 1])
[1]
>>> g = lambda x,y : x+y
>>> g(5,6)
11
>>> map (lambda (x,y): x+y, [(1,2), (3,4)])
[3, 7]
Object-Oriented Programming
Classes

```
class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def norm(self):
        return self.x ** 2 + self.y ** 2
```

- “self” is like “this” pointer in C++/Java/C#/PHP
- constructor `__init__(self, ...)`
- every (new-style) class is a subclass of Object like Java
  - we will only use new-style classes in this course

```python
>>> p = Point(3, 4)
>>> p.x
3
>>> p.norm()
25
```
Member variables

- each instance has its own hashmap for variables!
- you can add new fields on the fly (weird... but handy...)

```python
class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return "(%s, %s)" % (self.x, self.y)

>>> p = Point(5, 6)
>>> p.z = 7
>>> print p
(5, 6)
>>> p.z
7
>>> print p.w
AttributeError - no attribute 'w'
>>> p["x"] = 1
AttributeError - no attribute 'setitem'
```
More efficient: `__slots__`

- like C++/Java: fixed list of member variables
- class-global hash: all instances of this class share this hash
- can’t add new variables to an instance on the fly

```python
class Point(object):
    __slots__ = "x", "y"

    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        "like toString() in Java"
        return "(%s, %s)" % (self.x, self.y)

>>> p = Point(5, 6)
>>> p.z = 7
AttributeError!
```
Special function `__str__`

```python
class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def norm(self):
        return self.x ** 2 + self.y ** 2

    def __str__(self):
        return "(%s, %s)" % (self.x, self.y)
```

```text
>>> P = Point(3, 4)
>>> p.__str__()
'(3, 4)
>>> Point.__str__(p)
'(3, 4)
>>> str(p)
'(3, 4)
>>> print p
(3, 4)
```
class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return "(%s, %s)" % (self.x, self.y)

>>> p = Point(3,4)
>>> print p
(3, 4)
>>> p
<__main__.Point instance at 0x38be18>
Special functions: str vs repr

```python
class Point (object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return "(%s, %s)" % (self.x, self.y)

    def __repr__(self):
        return self.__str__()

>>> p = Point(3,4)
>>> print p
(3, 4)
>>> p
(3, 4)
>>> repr(p)
(3, 4)
```

<__main__.Point instance at 0x38be18>
class Point (object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

when __str__ is not defined, __repr__ is used
if __repr__ is not defined, Object.__repr__ is used

>>> p = Point(3,4)
>>> print p
<_main__.Point instance at 0x38be18>
>>> p
<_main__.Point instance at 0x38be18>
by default,
Python class object comparison is by pointer! define \_\_cmp\_
!

```python
class Point (object):
    def \_\_init\_(self, x, y):
        self.x = x
        self.y = y

    def \_\_str\_(self):
        return "(%s, %s)" % (self.x, self.y)

>>> p = Point(3,4)
>>> Point (3,4) == Point (3,4)
False
```
class Point (object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return "(%s, %s)" % (self.x, self.y)

    def __cmp__(self, other):
        if self.x == other.x:
            return self.y - other.y
        return self.x - other.x

if __eq__ is not defined, __cmp__ is used;
if __cmp__ is not defined, Object.__cmp__ is used (by reference)

>>> cmp(Point(3,4), Point(4,3))
-1

>>> p = Point(3,4)

>>> p
<Point instance at 0x38be18>

>>> Point (3,4) == Point (3,4)
False

>>> Point (3,4) == Point (3,4)
True
unique signature for each method

```python
class Point(object):
    def __init__(self, x, y):
        self.x, self.y = x, y

    def __init__(self, x, y, z):
        self.x, self.y, self.z = x, y, z

    def __init__(self, (x, y)):
        self.x, self.y = x, y
```

- no polymorphic functions (earlier defs will be shadowed)
  - ==> only one constructor (and no destructor)
- each function can have only one signature
  - because Python is dynamically typed
class Point (object):
    ...
    def __str__(self):
        return str(self.x) + ', ' + str(self.y)
    ...

class Point3D (Point):
    "A 3D point"
    def __init__(self, x, y, z):
        Point.__init__(self, x, y)
        self.z = z

    def __str__(self):
        return Point.__str__(self) + ', ' + str(self.z)

    def __cmp__(self, other):
        tmp = Point.__cmp__(self, other)
        return tmp if tmp != 0 else self.z - other.z

super-class, like C++
(multiple inheritance allowed)
__slots__ in inheritance

- like C++/Java: fixed list of member variables
- class-global hash: can’t add new field on the fly

```python
class Point(object):
    __slots__ = "x", "y"

def __init__(self, x, y):
    self.x, self.y = x, y

class Point3D(Point):
    __slots__ = "z"

def __init__(self, x, y, z):
    Point.__init__(self, x, y)
    self.z = z

>>> p = Point3D(5, 6, 7)
>>> p.z = 7
```
class Tree (object):
    __slots__ = "node", "children"
    def __init__(self, node, children=[]):
        self.node = node
        self.children = children

def total(self):
    if self == None:
        return 0
    return self.node + sum([x.total() for x in self.children])

def pp(self, dep=0):
    print " |" * dep, self.node
    for child in self.children:
        child.pp(dep+1)

def __str__(self):
    return "(%s)" % " ".join(map(str, 
        [self.node] + self.children))

left = Tree(2)
right = Tree(3)

>>> t = Tree(1, [Tree(2), Tree(3)])
>>> total(t)
6

>>> t.pp()
1
 2
 3
>>> print t
(1 (2) (3))
numpy

• numeric/scientific computations

```python
>>> from numpy import *
>>> a = arange(15).reshape(3, 5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
>>> a.shape
(3, 5)
>>> a.ndim
2
>>> a.dtype.name
'int32'
>>> a.itemsize
4
>>> a.size
15
>>> type(a)
numpy.ndarray
>>> b = array([6, 7, 8])
>>> b
array([6, 7, 8])
>>> type(b)
numpy.ndarray
```

```python
>>> from numpy import *
>>> a = array([2,3,4])
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int32')
>>> b = array([1.2, 3.5, 5.1])
>>> b.dtype
dtype('float64')
>>> b = array(([(1.5, 2), (4, 5, 6)])
>>> b
array([[ 1.5,  2. ,  3. ],
       [ 4. ,  5. ,  6. ]])
```

```python
>>> arange(10, 30, 5)
arange([10, 15, 20, 25])
>>> arange(0, 2, 0.3)  # accepts floats
array([ 0. ,  0.3,  0.6,  0.9,
        1.2,  1.5,  1.8])
```
```python
>>> a = arange(6)  # 1d array
>>> print a
[0 1 2 3 4 5]

>>> b = arange(12).reshape(4,3)  # 2d array
>>> print b
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

>>> c = arange(24).reshape(2,3,4)  # 3d array
>>> print c
[[[ 0  1  2  3]
  [ 4  5  6  7]
  [ 8  9 10 11]]

[[12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]]

>>> print arange(10000)
[ 0  1  2 ..., 9997 9998 9999]

>>> print arange(10000).reshape(100,100)

[[ 0  1  2 ..., 97 98  99]
 [100 101 102 ..., 197 198 199]
 [200 201 202 ..., 297 298 299]
 ..., [9700 9701 9702 ..., 9797 9798 9799]
 [9800 9801 9802 ..., 9897 9898 9899]
[9900 9901 9902 ..., 9997 9998 9999]]
```
array operations

```python
>>> A = array( [[1,1],
...             [0,1]] )
>>> B = array( [[2,0],
...             [3,4]] )

>>> A*B
array([[2, 0],
       [0, 4]]))

>>> C = dot(A,B)

>>> C
array([[5, 4],
       [3, 4]])

>>> C.transpose()
array([[5, 3],
       [4, 4]])

>>> C.T
array([[5, 3],
       [4, 4]])
```
array operations

```python
>>> a = array([20, 30, 40, 50])
>>> b = arange(4)
>>> b
array([0, 1, 2, 3])
>>> c = arange(0, 2, 0.5)  # arange supports float step size
>>> c
array([0, 0.5, 1, 1.5])
>>> c = a-b
>>> c
array([20, 29, 38, 47])
>>> b**2
array([0, 1, 4, 9])
>>> 10*sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
>>> a<35
array([[True, True, False, False],
       [True, True, False, False]], dtype=bool)
```
in-place operations

```python
>>> a = ones((2,3), dtype=int)
>>> b = random.random((2,3))
>>> a *= 3
>>> a
array([[3, 3, 3],
       [3, 3, 3]])
>>> b += a
>>> b
array([[ 3.69092703,  3.8324276 ,  3.0114541 ],
       [ 3.18679111,  3.3039349 ,  3.37600289]])
>>> a += b  # b is converted to integer type
>>> a
array([[6, 6, 6],
       [6, 6, 6]])
```
indexing and slicing

```python
>>> a = range(10)**3
>>> a
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
>>> a[:6:2] = -1000 # equivalent to a[0:6:2] = -1000;
>>> a
array([-1000,  1, -1000,  27, -1000,  125,  216,  343,  512,  729])
>>> a[:, ::-1]
# reversed a
array([ 729,  512,  343,  216,  125, -1000,  27, -1000,    1, -1000])
>>> for i in a:
...     print i**(1/3.),
...
nan 1.0 nan 3.0 nan 5.0 6.0 7.0 8.0 9.0

>>> a = array( [20,30,40,50] )
>>> a<35
array([ True,  True, False, False], dtype=bool)
>>> a[a<35] = 35 # indexing with boolean array
>>> a
array([ 35, 35, 40, 50])
```
>> def f(x,y):
...   return 10*x+y
...

>>> b = fromfunction(f,(5,4),dtype=int)
>>> b
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])

>>> b[2,3]
23

>>> b[0:5, 1]
array([ 1, 11, 21, 31, 41])

# each row in the second column of b

# equivalent to the previous example

>>> b[:,1]
array([ 1, 11, 21, 31, 41])

# each column in the second and third row of b

>>> b[1:3, :]
array([[10, 11, 12, 13],
       [20, 21, 22, 23]])