Basic Python Syntax
Numbers and Strings

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

```python
>>> a = 5
>>> b = 3
>>> type (5)
<type 'int'>
>>> a += 4
>>> a
9
no i++ or ++i
>>> c = 1.5
>>> 5/2
2
>>> 5/2.
2.5
>>> 5 ** 2
25
>>> 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

>>> 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 \texttt{range()} \\

\begin{itemize}
  \item \texttt{for} always iterates through a list or sequence
\end{itemize}

\begin{verbatim}
>>> 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])
\end{verbatim}

\texttt{Java 1.5}

\texttt{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
>>> if x < 10 and x >= 0:
...     print x, "is a digit"
...
>>> False and False or True
True
>>> not True
False

>>> if 4 > 5:
...     print "foo"
... else:
...     print "bar"
...
bar

>>> print "foo" if 4 > 5 else "bar"
...
conditional expr since Python 2.5

>>> bar

C/Java printf( (4>5)? "foo" : "bar");
```python
>>> a = "foo"
>>> if a in ["blue", "yellow", "red"]: ...
...     print a + " is a color"
... else:
...     if a in ["US", "China"]: ...
...         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 ...;
}
C/Java
```
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 Example

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

``` prime numbers

### C/Java Example

```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);
}
```

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

```python
def fib(n):
    if n <= 1:
        return n
    else:
        return fib(n-1) + fib(n-2)
```  
conceptually cleaner, but much slower!

>>> fib(5)
5
>>> fib(6)
8
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!
**Approaches to Typing**

- **strongly typed**: types are strictly enforced, no implicit type conversion
- **weakly typed**: not strictly enforced
- **statically typed**: type-checking done at compile-time
- **dynamically typed**: types are inferred at runtime

<table>
<thead>
<tr>
<th></th>
<th>weak</th>
<th>strong</th>
</tr>
</thead>
<tbody>
<tr>
<td>static</td>
<td>C, C++</td>
<td>Java, Pascal</td>
</tr>
<tr>
<td>dynamic</td>
<td>Perl, VB</td>
<td>Python, OCaml</td>
</tr>
</tbody>
</table>
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[0:3:2]
[1, [2, '4']]
>>> a[0::2]
[1, [2, '4']]
>>> a[:]
[1, 'python', [2, '4']]
>>> a[0:3:2]
[1, [2, '4']]
>>> a[0:-1]
[1, 'python']
>>> a[0:3:]
[1, 'python', [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

```python
>>> [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]
>>> a += b
>>> a
[2]
>>> a is b
>>> False
>>> b += [1]
>>> b
[2, 1]
>>> a is b
>>> True
```
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
List Comprehensions

```python
>>> vec = [2, 4, 6]
>>>=[[x, x**2] for x in vec]
[[2, 4], [4, 16], [6, 36]]

>>>=[x, x**2 for x in vec]
SyntaxError: invalid syntax

>>>=[(x, x**2) for x in vec]
[(2, 4), (4, 16), (6, 36)]

>>> vec1 = [2, 4, 6]
>>> vec2 = [4, 3, -9]
>>>=[x*y for x in vec1 for y in vec2]
[8, 6, -18, 16, 12, -36, 24, 18, -54]

>>>=[x+y for x in vec1 for y in vec2]
[6, 5, -7, 8, 7, -5, 10, 9, -3]

should use zip instead!

>>>=[vec1[i]*vec2[i] for i in range(len(vec1))]
[8, 12, -54]
```

(cross product)

(dot product)
Strings

sequence of characters
Basic String Operations

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

```python
>>> s = " this is a python course. 
" >>> 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. 
'
```

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'
```

despite these operations are much faster than regexps!
String Formatting

>>> print "%.2f%%" % 97.2363
97.24%

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