Python
Part I
Hello World

-start Python in interactive mode
-at the >>> prompt, type
   print 'Hello, World!'
-bask in glory
Numbers

• type these expressions

  \[3 + 3\quad 10 \times (5 + 1)\]

• (# starts comments, don't type)

  \[5 - 2\quad 2 ** 10\]

  \[12 / 6\quad 2 ** 1000\]

  \[10 \times 5 + 1\quad 22 / 7 \quad \text{integer division!}\]

  \[22 ./ 7\]

  \[22 \% 7\]
Variables

• type:

282 / 7.695
miles = 282
gallons = 7.695
miles / gallons
Strings

- delimiters
  - single-quote ( ' ), or
  - double-quotes ( " )

- type:
  - print "Arthur "No Sheds" Jackson"
  - print 'Arthur "No Sheds" Jackson'
  - print "Arthur \"No Sheds\" Jackson"
  - print """Arthur "No Sheds" Jackson""

- triple quotes only match triple quotes
String Concatenation

- type:
  
  'miles' + 'per gallon'
  'mpg=' + 282./7.7    #FAIL
  'mpg=' + str(282./7.7)
  'ni ' * 100
String Parsing

- type:
  cheese='Red Windsor'
  cheese[0]
  cheese[1]
  len(cheese)
  cheese[len(cheese)-1]
  cheese[-1]

- negative index means offset from end of string, -1 is last char, -2 is next-to-last, etc
Slicing Strings

- a:b refers to the substring starting at index a and ending before index b.
- number after optional second ':' is the step value, e.g., 0:10:2 is every other char from first ten
- type

  - `cheese[2:5]`
  - `cheese[0:2]`
  - `cheese[:4]`
  - `cheese[4:]`
  - `cheese[::2]`
  - `cheese[1::2]`
  - `cheese[:4:2]`
  - `cheese[:::-1]`
String Methods

cheese.upper()

cheese.lower()

cheese.capitalize()

'I want to buy some cheese.' .title()

cheese .title()

cheese[0] = 'W'

cheese

• strings are immutable - can't be changed
  • methods return new strings
Lists

wants=[]
wants.append('Stilton')
wants.append('Emmenthal')
wants
wants.append('Gruyere')
wants.append('Jarlsburg')
wants.append('Liptauer')
wants[3]='Norwegian Jarlsburg'
wants
del wants[-1]
wants[2]
wants[:2]
wants[2:]
more=['Lancashire', 'White Stilton']
wants += more
wants
List Methods

wants.sort()
wants
wants.pop(0)
wants
help([])
Tuples

- Lists are mutable, tuples are not
  
  ```python
tup=('Jackie','6')
tup[0]
tup[0]='John'  # won't work
  ```

- unpacking (also for lists):
  
  ```python
name,age = tup
name
age
  ```

- creating empty or one-item tuples different from lists in older python versions
  
  ```python
tup=() # creates an empty tuple in 2.5.4
tup=('foo') # creates a string, parens for grouping in this case
tup=('foo',) # creates a one-item tuple
  ```

- concatenation
  
  ```python
tup = (1,2,3)
tup+(4,5,6)
tup+wants # fail
tup+tuple(wants) # now concatenating tuples
list(tup) + wants # now concatenating lists
  ```
Dictionaries

• lists and tuples are sequences
• dictionaries are mappings

```python
dict={'Louis' : 16, 'George' : 3}
dict['Louis']
dict['George']
dict['Benedict']=2
dict
```
Loops

- while loop

```python
>>> count=1
>>> while count <= 5 :
...     print count
...     count += 1
...
1
2
3
4
5
```
For Loops

• for loops use iterables to get a value for each loop pass

```python
>>> for parrot in range(1,6):
    print parrot

1
2
3
4
5
```

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

• use xrange() for large ranges, won’t create a list in memory, but iterates anyway

```python
>>> xrange(1,6)
xrange(1, 6)
```

```python
>>> for i in xrange(1,6):
    print i

1
2
3
4
5
```

```python
>>> range(1,6)
[1,2,3,4,5,6]
```
Iterating Containers

- lists
  ```python
  >>> for cheese in wants:
  ...     print cheese
  ...
  Gruyere
  Lancashire
  Norwegian Jarlsburg
  Stilton
  White Stilton
  ```

- tuples
  ```python
  >>> for i in (0,1,2,3):
  ...     print 2**i
  ...
  1
  2
  4
  8
  ```

- dictionaries
  ```python
  >>> for king in dict:
  ...     print king, dict[king]
  ...
  Benedict 2
  Louis 16
  George 3
  ```

Tuesday, April 7, 2009
Example: shopping

>>> wants
['Gruyere', 'Lancashire', 'Norwegian Jarlsburg',
'Silton', 'White Stilton']

>>> prices={'Gruyere':3.20, 'Lancashire':12.50, 'Norwegian
Jarlsburg':6.75, 'Silton':14.25 , 'White Stilton':17.50}

>>> total=0.0

>>> for item in wants:
...     total += prices[item]
...

>>> total
54.200000000000003