Working with Lists

```python
>> digits = [1, 8, 2, 8]  # digits = [2/2, 2+2-2, 2, 2+2/2]

The number of elements
>>> len(digits)
4

An element selected by its index
>>> digits[3]
8

Concatenation and repetition
>>> [2, 7] + digits + 2
[2, 7, 1, 8, 2, 8, 1, 8, 2, 8]

Nested lists
>>> pairs = [[18, 20], [38, 48]]
>>> pairs[1]
[38, 48]
>>> pairs[1][8]
18
```

Containers

Built-in operators for testing whether an element appears in a compound value

```python
>> digits = [1, 8, 2, 8]
>> 1 in digits
True
>> 8 in digits
True
>> 5 not in digits
True
>> not(5 in digits)
True
```

Sequence iteration

```python
def count(s, value):
    total = 0
    for element in s:
        if element == value:
            total = total + 1
    return total
```
For Statement Execution Procedure

```python
for <name> in <expression>:
    <suite>
```

1. Evaluate the header `<expression>`, which must yield an iterable value (a sequence)
2. For each element in that sequence, in order:
   A. Bind `<name>` to that element in the current frame
   B. Execute the `<suite>`

Sequence Unpacking in For Statements

```python
>>> pairs = [[1, 2], [2, 2], [3, 2], [4, 4]]
>>> same_count = 0
```

```python
for x, y in pairs:
    ...
    if x == y:
        ...
    same_count = same_count + 1
```

```python
>>> same_count
2
```

Ranges

A range is a sequence of consecutive integers.

```python
..., -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, ...
```

```python
range(-2, 2)
```

Length: ending value - starting value
Element selection: starting value + index

```python
list(range(-2, 2))
```

```python
[-2, -1, 0, 1]
```

```python
list(range(0, 4))
```

```python
[0, 1, 2, 3]
```

List Comprehensions

```python
[<map exp> for <name> in <iter exp> if <filter exp>]
```

Short version: ```[<map exp> for <name> in <iter exp>]```  

A combined expression that evaluates to a list using this evaluation procedure:
1. Add a new frame with the current frame as its parent
2. Create an empty result list that is the value of the expression
3. For each element in the iterable value of `<iter exp>`:
   A. Bind `<name>` to that element in the new frame from step 1
   B. If `<filter exp>` evaluates to a true value, then add the value of `<map exp>` to the result list

Strings

Representing data:

```python
'200'  '1.2e-5'  'False'  '[3, 2]'
```

```
"And, as imagination bodies forth
The forms of things to unknown, and the poet's pen
Turns them to shapes, and gives to airy nothing
A local habitation and a name."
```

Representing programs:

```python
'curry = lambda f: lambda x: lambda y: f(x, y)'
```

(Demo)

Strings are an Abstraction

Representing data:

```
'200'  '1.2e-5'  'False'  '[3, 2]'
```

```
"And, as imagination bodies forth
The forms of things to unknown, and the poet's pen
Turns them to shapes, and gives to airy nothing
A local habitation and a name."
```

Representing programs:

```python
'curry = lambda f: lambda x: lambda y: f(x, y)'
```

(Demo)
String Literals Have Three Forms

>>> 'I am string!'
'I am string!'

>>> "I've got an apostrophe"
"I've got an apostrophe"

>>> "'It's'

>>> '''The Zen of Python
claims, Readability counts.
Read more: import this.'"
'The Zen of Python
claims, Readability counts.
Read more: import this.'

Dictionaries

Dictionaries are unordered collections of key-value pairs.

Dictionary keys do have two restrictions:

• A key of a dictionary cannot be a list or a dictionary (or any mutable type)

• Two keys cannot be equal; There can be at most one value for a given key

This first restriction is tied to Python's underlying implementation of dictionaries

The second restriction is part of the dictionary abstraction

If you want to associate multiple values with a key, store them all in a sequence value

Line feed character
represents a new line

A backslash "escapes" the following character

"Line feed" character represents a new line

Single-quoted and double-quoted strings are equivalent

{'Dem': 0}