The University of North Carolina at Chapel Hill

COMP 144 Programming Language Concepts

Spring 2002

Lecture 7: Python's Built-in Types and Basic Statements

Felix Hernandez-Campos

Jan 25

Built-in Data Structures: Lists

- A list is an **ordered collection of objects**
- Lists can contain *any* type of object
- Lists are *mutable*
- Examples

`[]`	Empty list
`[1, ”2”, 3.0]`	Three-element list
`[1, [”2”, 4], 3.0]`	Nested list

Lists: Accessing Items

- Syntax: **`list[index]`**
  - Indexing from the left starts at 0
  - *E.g.*

```
>>> l = [1, ["2", 4], 3.0]
>>> l[0]
1
>>> l[2]
3.0
>>> l[1]
['2', 4]
>>> l[3] = 4
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in ?
    l[3] = 4
IndexError: list assignment index out of range
```

Lists: Accessing Items

- Syntax: **`list[-index]`**
  - Indexing from the right starts at -1
  - *E.g.*

```
>>> l = [1, ["2", 4], 3.0]
>>> l[-1]
3.0
>>> l[-3]
1
>>> l[-4]
Traceback (most recent call last):
  File "<pyshell#29>", line 1, in ?
    l[-4]
IndexError: list index out of range
```

Lists: Deleting Items

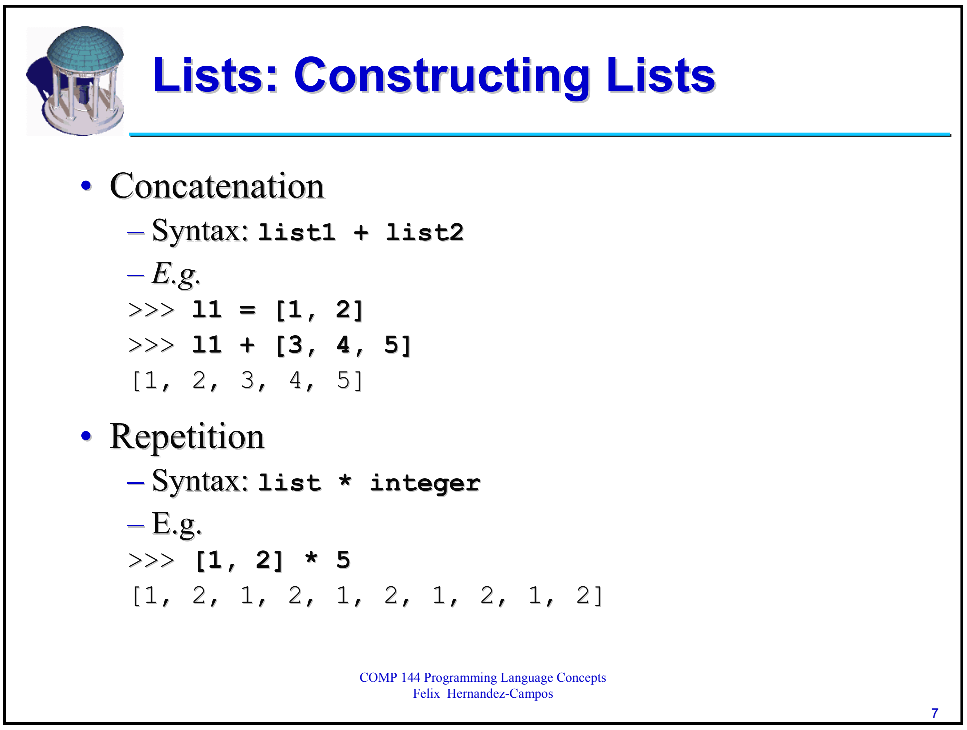
- Syntax: **`del list[index]`**
  - *E.g.*

```
>>> l = [1, ["2", 4], 3.0]
>>> del l[2]
>>> l
[1, ['2', 4]]
>>> del l[2]
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in ?
    del l[2]
IndexError: list assignment index out of range
```

Lists: Length

- Syntax: **`len(list)`**
  - *E.g.*

```
>>> l = [1, ["2", 4], 3.0]
>>> len(l)
3
>>> l = []
>>> len(l)
0
```



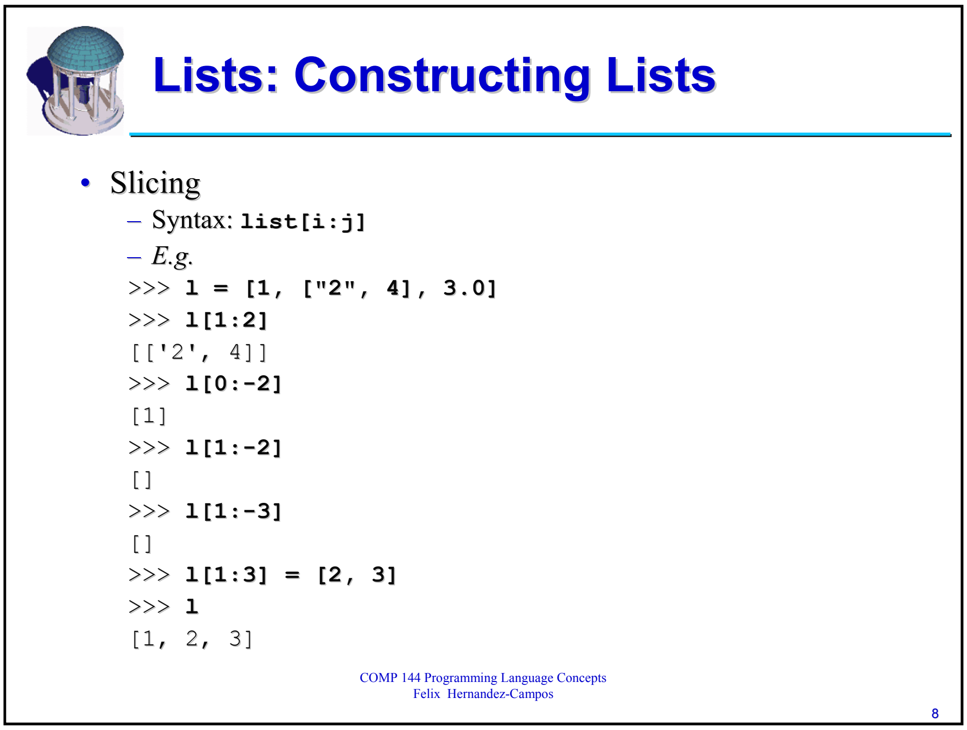
Lists: Constructing Lists

- Concatenation
  - Syntax: `list1 + list2`
  - *E.g.*

```
>>> l1 = [1, 2]
>>> l1 + [3, 4, 5]
[1, 2, 3, 4, 5]
```

- Repetition
  - Syntax: `list * integer`
  - E.g.

```
>>> [1, 2] * 5
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```



Lists: Constructing Lists

- Slicing
  - Syntax: `list[i:j]`
  - *E.g.*

```
>>> l = [1, ["2", 4], 3.0]
>>> l[1:2]
[['2', 4]]
>>> l[0:-2]
[1]
>>> l[1:-2]
[]
>>> l[1:-3]
[]
>>> l[1:3] = [2, 3]
>>> l
[1, 2, 3]
```

Lists: Constructing Lists

- Ranges
  - Syntax: **`range(start, end, step)`**
  - Default values for start (0) and step (1)
  - *E.g.*

```
>>> range(1,100,10)
[1, 11, 21, 31, 41, 51, 61, 71, 81, 91]
>>> range(1,13)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13]
>>> range(3)
[0, 1, 2]
```

Lists: Methods

- Inserting an item at a given position
  - Syntax: **`list.insert[index, item]`**
  - *E.g.*

```
>>> l = [1, ["2", 4], 3.0]
>>> l.insert(0, 8.3)
>>> l
[8.3, 1, ['2', 4], 3.0]
```

- Adding an item at the end of the list
  - Syntax: **`list.append[item]`**
  - *E.g.*

```
>>> l.append("end")
>>> l
[8.3, 1, ['2', 4], 3.0, "end"]
```

Lists: Methods

- Sorting
  - Syntax: **`list.sort()`**
  - *E.g.*

```
>>> l = [1, 3, 2.0, 4]
>>> l.sort()
>>> l
[1, 2.0, 3, 4]
>>> l=["c", "d", "a", "b"]
>>> l.sort()
>>> l
['a', 'b', 'c', 'd']
```

Lists: Methods

- Reversing
  - Syntax: **`list.reverse()`**
  - *E.g.*

```
>>> l = [1, 3, 2.0, 4]
>>> l.reverse()
>>> l
[4, 2.0, 3, 1]
```

Built-in Data Structures: Dictionaries

- A dictionary is an **unordered collection of objects indexed by keys**
- A*ny* object can be a key
- *Any* object can be a item indexed by a key
- Dictionaries are *mutable*
- Examples

`{}`	Empty dictionary
`{'item':'tire','price':20.99}`	Two-element dictionary

Dictionaries: Accessing items

- Syntax: **`list[key]`**
  - *E.g.*

```
>>> d = {'item':'tire','price':20.99}
>>> d[‘price']
20.99
>>> d[item]
Traceback (most recent call last):
  File "<pyshell#88>", line 1, in ?
    d[item]
NameError: name 'item' is not defined
>>> str = 'item'
>>> d[str]
'tire'
```

Dictionaries: Deleting items

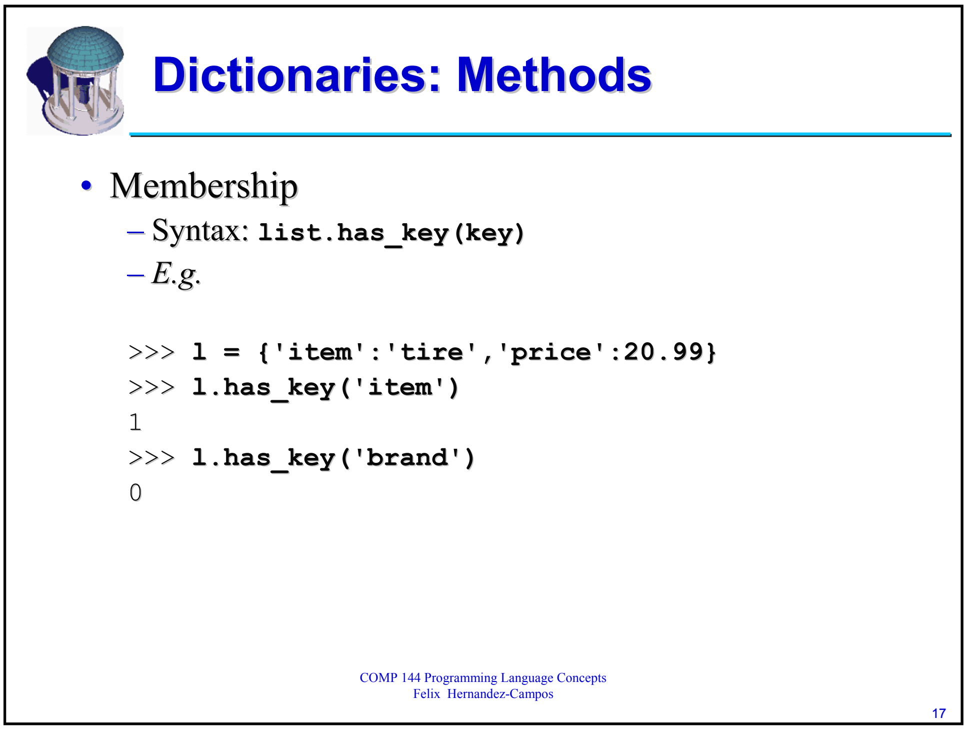
- Syntax: **`del list[key]`**
  - *E.g.*

```
>>> d = {'item':'tire','price':20.99}
>>> del d['item']
>>> d
{'price': 20.989999999999998}
>>> del d['brand']
Traceback (most recent call last):
  File "<pyshell#95>", line 1, in ?
    del d['brand']
KeyError: brand
```

Dictionaries: Length

- Syntax: **`len(list)`**
  - *E.g.*

```
>>> d = {'item':'tire','price':20.99}
>>> len(d)
2
```



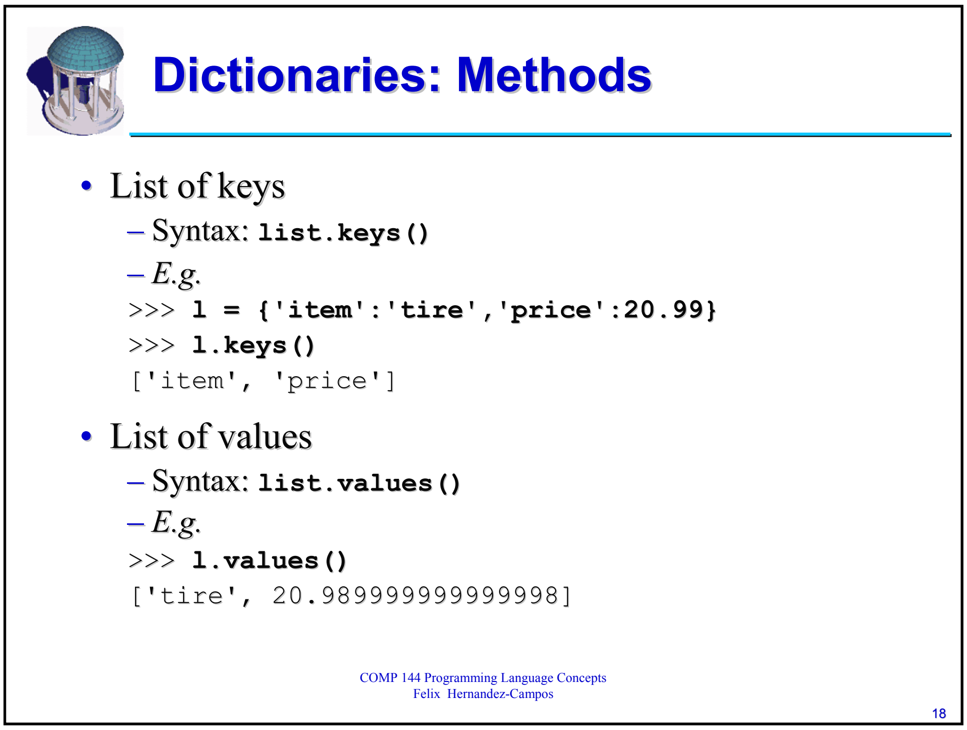
Dictionaries: Methods

- Membership
  - Syntax: **`list.has_key(key)`**
  - *E.g.*

```
>>> l = {'item':'tire','price':20.99}
>>> l.has_key('item')
1
>>> l.has_key('brand')
0
```

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

17



Dictionaries: Methods

- List of keys
  - Syntax: **`list.keys()`**
  - *E.g.*

```
>>> l = {'item':'tire','price':20.99}
>>> l.keys()
['item', 'price']
```

- List of values
  - Syntax: **`list.values()`**
  - *E.g.*

```
>>> l.values()
['tire', 20.989999999999998]
```

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

18

Built-in Data Structures: Tuples

- A tuple is an **ordered collection of objects**
- Tuples can contain *any* type of object
- Tuples are *immutable*
- Examples

`()`	Empty tuple
`1,`	One-element tuple (!)
`(1, ”2”, 3.0)`	Three-element tuple
`1, (”2”, 3.0)`	Nested tuple

Built-in Data Structures: Tuples

- **Commas** are used to define tuples
  - Parentheses around tuples are optional
  - *E.g.*

```
>>> 1,('2',2.0)
(1, ('2', 2.0))

>>> (1,('2',2.0))
(1, ('2', 2.0))
```

  - The one-element list <u>requires</u> a trailing comma

```
>>> 1,
(1,)
>>> (1)          ← This is not a tuple but a number
1
```

Tuples: Accessing Items

- Syntax: **`tuple[index]`**
  - *E.g.*

```
>>> t = (1, 2, (3, 4, 5))
>>> t[1]
2
>>> t[-1]
(3, 4, 5)
>>> t[-1][1]
4
>>> t[3]
Traceback (most recent call last):
  File "<pyshell#110>", line 1, in ?
    t[3]
IndexError: tuple index out of range
```

Tuples: No Deletion and Length

- No deletion!
  - Tuples are immutable
- Length:
  - Syntax: **`len(tuple)`**
  - *E.g.*

```
>>> t = (1,2,(3,4,5))
>>> len(t)
3
>>> len(t[1])
Traceback (most recent call last):
  File "<pyshell#117>", line 1, in ?
    len(t[1])
TypeError: len() of unsized object
>>> len(t[2])
3
```

Tuples: Constructing Tuples

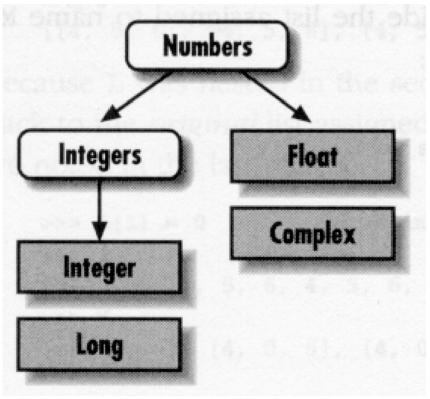
- Concatenation
  - Syntax: `tuple1 + tuple2`
  - *E.g.*

```
>>> t = (1,2) + (3,)
>>> t
(1, 2, 3)
```

- Repetition
  - Syntax: `tuple * integer`
  - E.g.

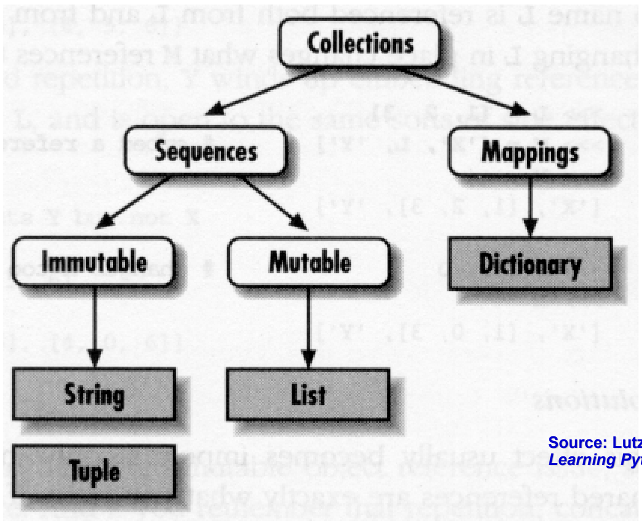
```
>>> t * 5
(1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3)
```

Hierarchy of Numbers



Source: Lutz & Ascher, *Learning Python*, Figure 2-3

Hierarchy of Built-in Collections



Source: Lutz & Ascher, *Learning Python*, Figure 2-3

Statements: Assignment

- Syntax: `reference = object` or `reference`
  - *E.g.*

```
>>> a = 3
>>> a
3
>>> s1, n, m = "hello", 4.0, a
>>> s1
'hello'
>>> n
4.0
>>> m
3
```

Statements: Print

- Syntax: **`print object`** or **`reference`**
  - *E.g.*

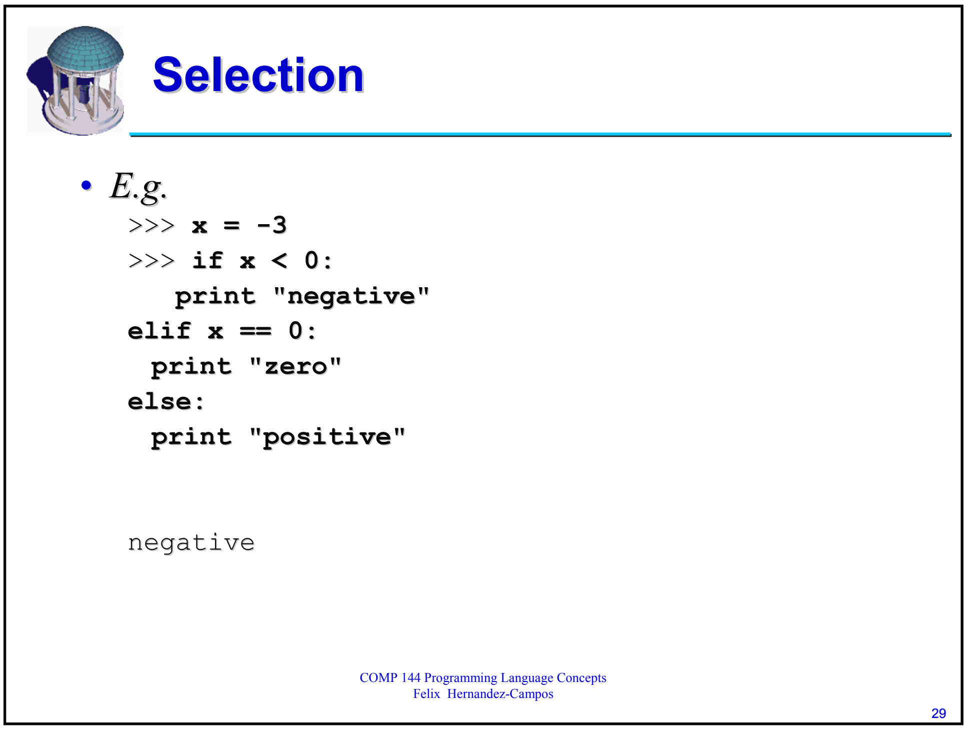
```
>>> print "hello", 'again'
hello again
>>> print 3.0e5
300000.0
>>> name = "python"
>>> ver = 2.2
>>> print "This is %(name)s %(ver).3f" % vars()
This is python 2.200
```

Selection

- Syntax:

```
if test:
    statements
elif test:
    statements
else:
    statements
```

- Conditional expressions:
  - `>, <, >=, <=, ==, and, or, not`

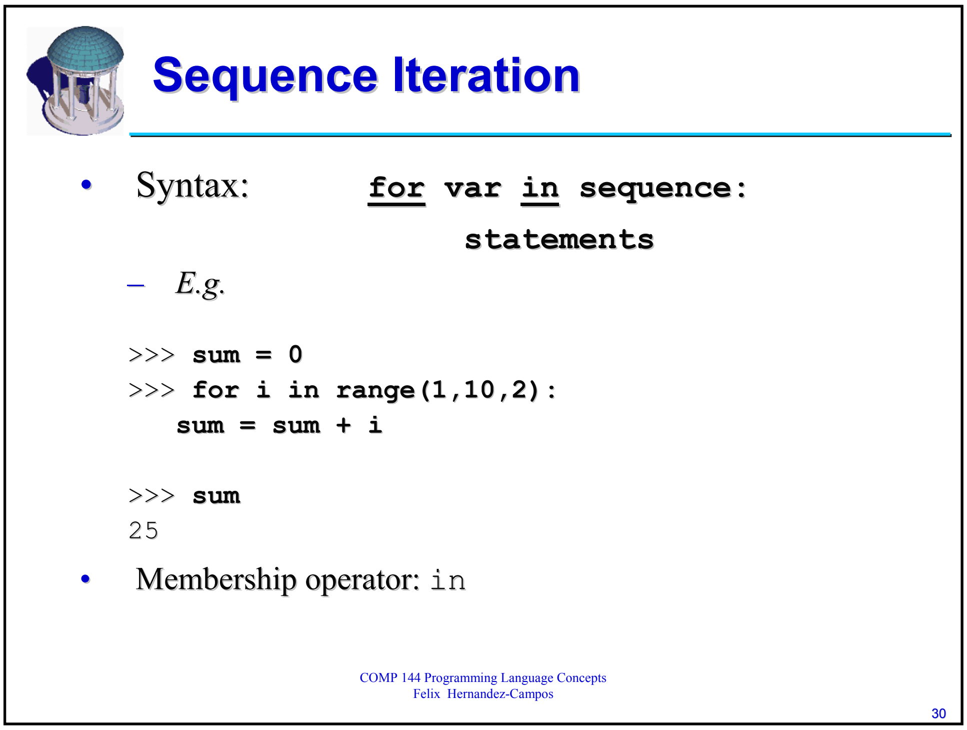


Selection

- *E.g.*

```
>>> x = -3
>>> if x < 0:
   print "negative"
elif x == 0:
  print "zero"
else:
  print "positive"


negative
```



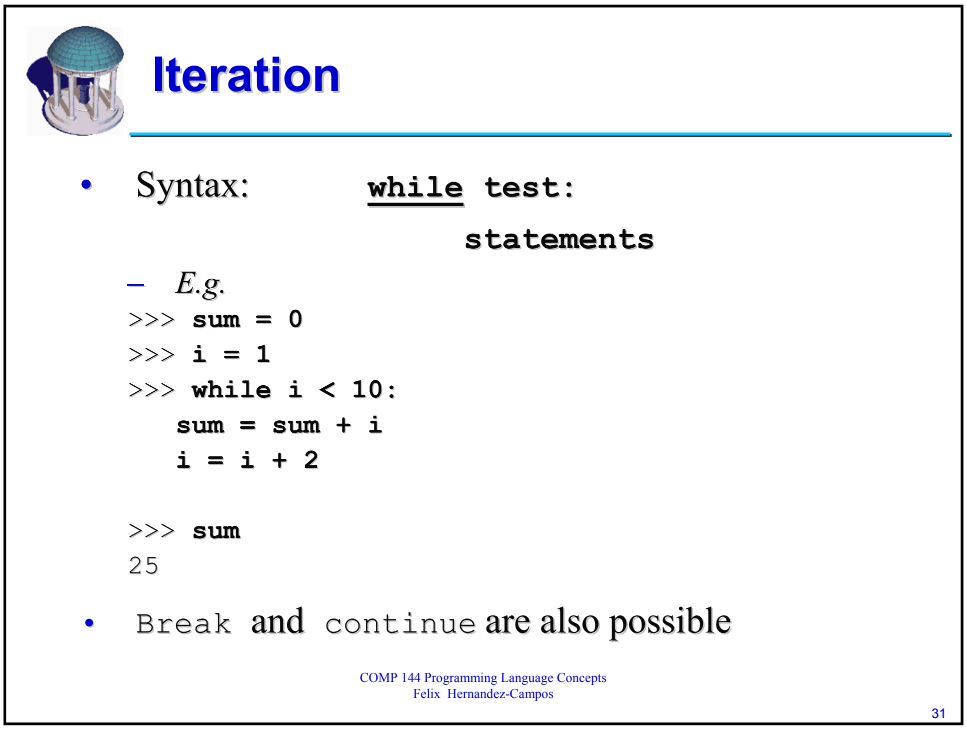
Sequence Iteration

- Syntax: `for var in sequence:`
  `statements`
  - *E.g.*

```
>>> sum = 0
>>> for i in range(1,10,2):
   sum = sum + i

>>> sum
25
```

- Membership operator: `in`



Iteration

- Syntax: **while test:**

```
                    statements
```

  – *E.g.*

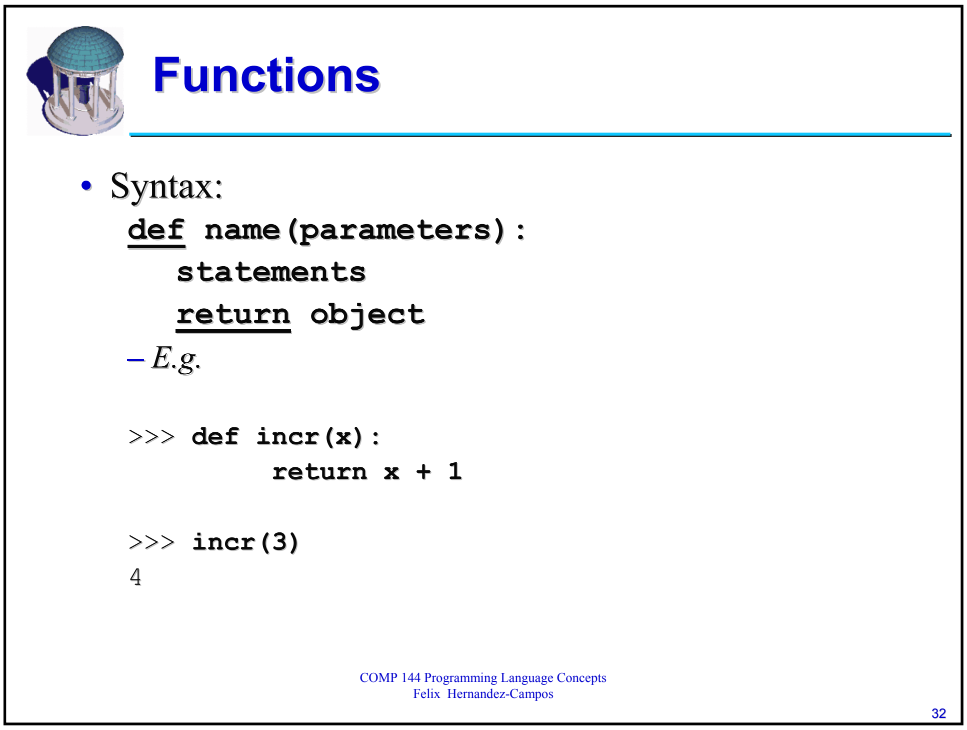
```
>>> sum = 0
>>> i = 1
>>> while i < 10:
   sum = sum + i
   i = i + 2

>>> sum
25
```

- `Break` and `continue` are also possible

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

31



Functions

- Syntax:

```
def name(parameters):
   statements
   return object
```

  – *E.g.*

```
>>> def incr(x):
         return x + 1

>>> incr(3)
4
```

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

32

Functions

- Default values
  - *E.g.*

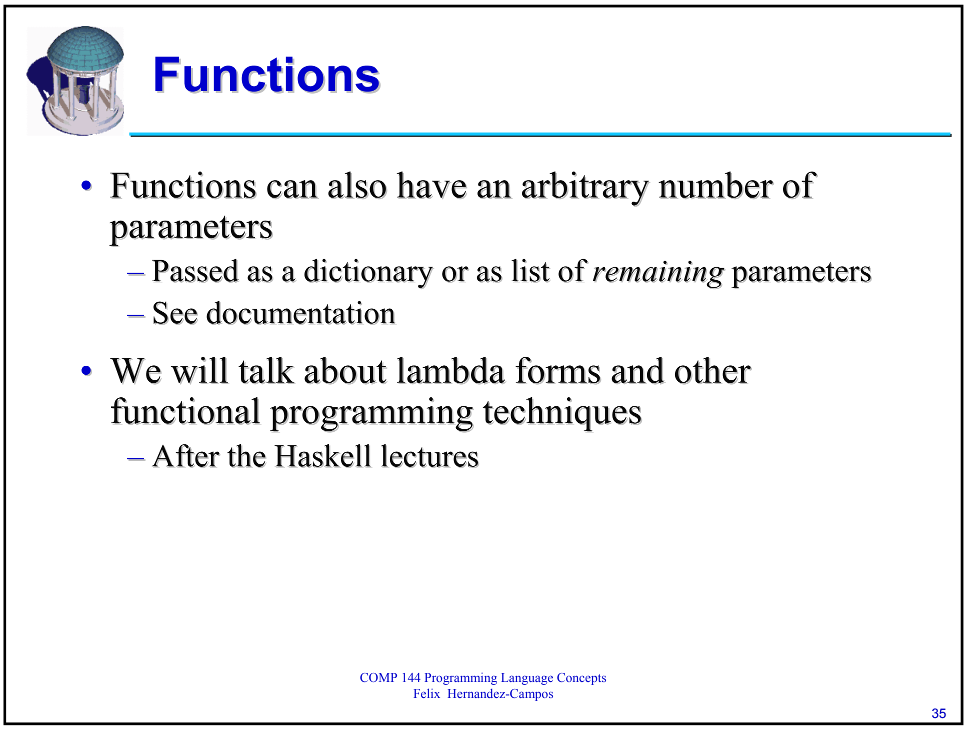
```
def ask_ok(prompt, retries=4, complaint='Yes or no!'):
         while 1:
             ok = raw_input(prompt)
             if ok in ('y', 'ye', 'yes'): return 1
             if ok in ('n', 'no', 'nop', 'nope'):
                  return 0
             retries = retries - 1
             if retries < 0:
                 raise IOError, 'refusenik user'
             print complaint
```

Functions

- Parameter passing by position and by name
  - E.g.

```
def parrot(voltage, state='a stiff', action='voom',
  type='Norwegian Blue'):
         print "-- This parrot wouldn't", action,
         print "if you put", voltage, "Volts through
  it."
         print "-- Lovely plumage, the", type
         print "-- It's", state, "!"

>>> parrot(1000)
>>> parrot(action = 'VOOOOOM', voltage = 1000000)
>>> parrot('a thousand', state = 'pushing up the
  daisies')
>>> parrot('a million', 'bereft of life', 'jump')
```

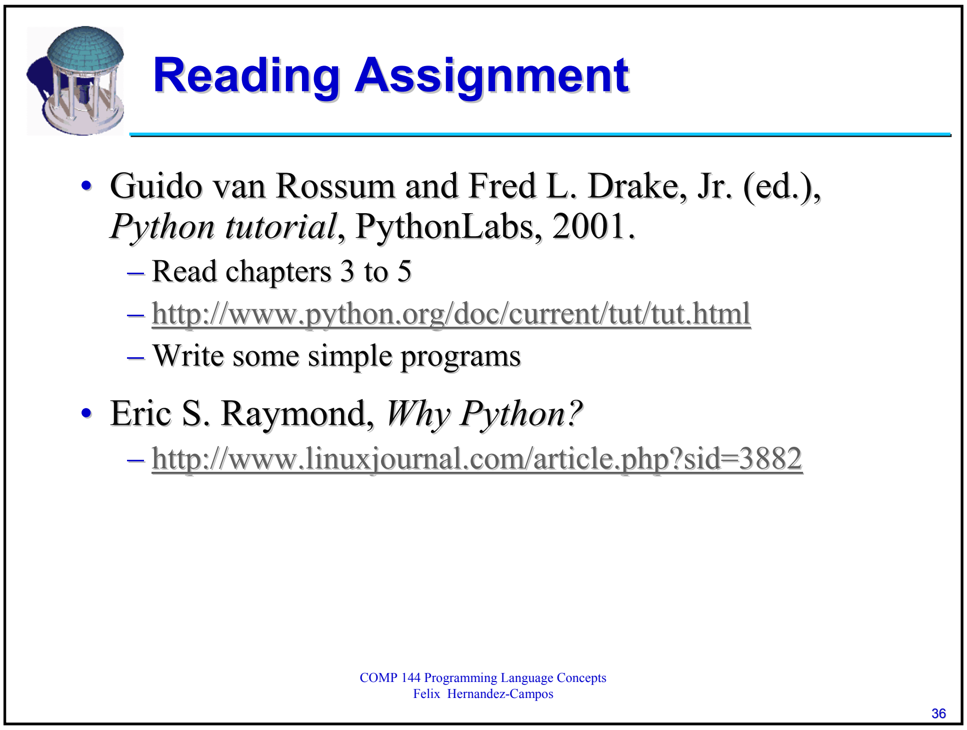


Functions

- Functions can also have an arbitrary number of parameters
  - Passed as a dictionary or as list of *remaining* parameters
  - See documentation
- We will talk about lambda forms and other functional programming techniques
  - After the Haskell lectures

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

35



Reading Assignment

- Guido van Rossum and Fred L. Drake, Jr. (ed.), *Python tutorial*, PythonLabs, 2001.
  - Read chapters 3 to 5
  - http://www.python.org/doc/current/tut/tut.html
  - Write some simple programs
- Eric S. Raymond, *Why Python?*
  - http://www.linuxjournal.com/article.php?sid=3882

COMP 144 Programming Language Concepts
Felix Hernandez-Campos

36