

Introduction to Python

Edward Loper



1

CIS-530

Outline

- **Data**
 - strings, variables, lists, dictionaries
- **Control Flow**
- **Working with files**
- **Modules**
- **Functions**
- **Classes**
- **Etc.**



2

CIS-530

Python

- **Python is an open source scripting language.**
- **Developed by Guido van Rossum in the early 1990s**
- **Named after Monty Python**
- **Available on unagi, linc, gradient**
/pkg/p/Python2.1.1/bin/python
- **Available for download from**
<http://www.python.org>



3

CIS-530

The Python Interpreter

- **Interactive interface to Python**
% /pkg/p/Python-2.1.1/bin/python
Python 2.1.1 (#6, Aug 28 2001, 15:43:27)
[GCC 2.95.2 19991024 (release)] on sunos5
Type "copyright", "credits" or "license" for more information.
>>>
- **Enter an expression, and Python evaluates it:**
>>> 3*(7+2)
27



4

CIS-530

Strings

- **A string is a single piece of text.**
 - **Strings are written '...' or "..."**

```
>>> "the king of spain"
the king of spain
>>> 'the king said "hello."'
the king said "hello."
```
 - **Spaces are significant**

```
>>> ' the knights of ni '
the knights of ni
```
 - **Backslashes mark special characters**

```
>>>'hello\nworld' # '\n' is a newline
hello
world
```



5

CIS-530

Operations on Strings

```
>>> 'the' + 'king'
'theking'
>>> len('the df')
6
>>> 'the df'.count('the')
1
>>> 'the king'.replace('the', 'a')
'a king'
>>> 'the king'.upper()
'THE KING'
>>> ' hello there '.strip()
'hello there'
```



6

CIS-530

Variables

- **A variable is a name for a value.**
 - **Use “=” to assign values to variables.**

```
>>> first_name = 'John'
>>> last_name = 'Smith'
>>> first_name + ' ' + last_name
'John Smith'
```
 - **Variable names are case sensitive**
 - **Variable names include only letters, numbers, and “_”**
 - **Variable names start with a letter or “_”**
 - **Any variable can hold any value (no typing)**



7

CIS-530

Lists

- **A list is an ordered set of values**
 - **Lists are written [elt₀, elt₁, ..., elt_{n-1}]**

```
>>> [1, 3, 8]
>>> ['the', 'king', 'of', ['spain', 'france']]
>>> []
>>> [1, 2, 'one', 'two']
```
 - **lst[i] is the ith element of lst.**
 - **Elements are indexed from zero**

```
>>> words = ['the', 'king', 'of', 'spain']
>>> words[0]
'the'
>>> words[2]
'of'
```



8

CIS-530

Indexing Lists

```
>>> lst = ['a', 'b', 'c', ['d', 'e']]
>>> determiners[0]           # 0th element
'a'
>>> determiners[-2]         # N-2th element
'c'
>>> determiners[-1][0]      # sublist access
'd'
>>> determiners[0:2]        # elements in [0, 2)
['a', 'b']
>>> determiners[2:]         # elements in [2, N)
['c', ['d', 'e']]
```

	'a'	'b'	'c'	['d', 'e']	
0	1	2	3	4	
-4	-3	-2	-1		



9

CIS-530

Operations on Lists

```
>>> determiners = ['the', 'an', 'a']
>>> len(determiners)
3
>>> determiners + ['some', 'one']
['the', 'an', 'a', 'some', 'one']
>>> determiners
['the', 'an', 'a']
>>> determiners.index('a')
2
>>> [1, 1, 2, 1, 3, 4, 3, 6].count(1)
3
```



10

CIS-530

Operations on Lists 2: List Modification

```
>>> determiners
['the', 'an', 'a']
>>> del determiners[2]           # remove the element at 2
>>> determiners.append('every')  # insert at the end of the list
>>> determiners.insert(1, 'one') # insert at the given index
>>> determiners
['the', 'one', 'an', 'every']
>>> determiners.sort()          # sort alphabetically
>>> determiners
['an', 'every', 'one', 'the']
>>> determiners.reverse()       # reverse the order
['the', 'one', 'every', 'an']
```



11

CIS-530

Lists and Strings

- **Strings act like lists of characters in many ways.**

```
>>> 'I saw a man with a telescope'[-9:]
'telescope'
```
- **Converting strings to lists:**

```
>>> list('a man')           # get a list of characters
['a', ' ', 'm', 'a', 'n']
>>> 'a man'.split()         # get a list of words
['a', 'man']
```
- **Converting lists to strings:**

```
>>> str(['a', 'man'])       # a representation of the list
"['a', 'man']"
>>> '-'.join(['a', 'man'])  # combine the list into one string
'a-man'
```



12

CIS-530

Dictionaries

- A dictionary maps keys to values
 - Like a list, but indexes (keys) can be anything, not just integers.
 - Dictionaries are written `{key:val, ...}`

```
>>> numbers = {'one':1, 'two':2, 'three':3}
```
 - Dictionaries are indexed with `dict[key]`

```
>>> numbers['three']
3
>>> numbers['four'] = 4
```
 - Dictionaries are *unordered*.



13

CIS-530

Operations on Dictionaries

```
>>> determiners = {'the':'def', 'an':'indef',
...               'a':'indef'}
>>> determiners.keys()
['an', 'a', 'the']
>>> determiners.has_key('an')
1                               # 1 is true
>>> del determiners['an']
>>> determiners.has_key('an')
0                               # 0 is false
>>> determiners.items()
['the':'def', 'a':'indef']
```



14

CIS-530

Truth Values

- Every expression has a truth value
 - 0 is false, all other numbers are true.
 - "" is false, all other strings are true
 - [] is false, all other lists are true

```
>>> 5 == 3+2           # == tests for equality
1
>>> 5 != 3*2          # != tests for inequality
1
>>> 5 > 3*2           # >, <, >=, <= test for ordering
0
>>> 5 > 3*2 or 5<3*2  # or, and combine truth values
0
```



15

CIS-530

Control Flow

- **if** statement tests if a condition is true
 - If so, it executes a body
 - Otherwise, it does nothing

```
>>> if len(determiners) > 3:
...     del determiners[3]
...     print 'deleted the 3rd determiner'
```

body {

 - Indentation is used to mark the body.
 - Note the ":" at the end of the if line.



16

CIS-530

Control Flow 2

- **if-else statement**

```
>>> if len(sentence) > 3:
...     print sentence
>>> else:
...     print 'too short'
```
- **if-elif statement**

```
>>> if x<3:
...     print x*3
>>> elif x<6:
...     print x*2
>>> else:
...     print x
```
- **while statement**

```
>>> while x<1000:
...     x = x*x+3
```
- **for statement**

```
>>> for n in [1, 8, 12]:
...     print n*n
```
- **range ()**

```
>>> for n in range(0, 10):
...     print n*n
```



17

CIS-530

Working with Files

- **To read a file:**

```
>>> for line in open('corpus.txt', 'r').readlines():
...     print line
```
- **To write to a file:**

```
>>> outfile = open('output.txt', 'w')
>>> outfile.write(my_string)
>>> outfile.close()
```
- **Example:**

```
>>> outfile = open('output.txt', 'w')
>>> for line in open('corpus.txt', 'r').readlines():
...     outfile.write(line.replace('a', 'some'))
>>> outfile.close()
```



18

CIS-530

Modules

- **A module is a collection of useful operations and objects.**
 - e.g., networking, graphics, threads, NLP
- **Access modules with import**

```
>>> import re # regular expressions
>>> re.search('[ab]c+', mystring)
```
- **Or use from...import**

```
>>> from re import search
>>> search('[ab]c+', mystring)
```



19

CIS-530

Getting Help

- **The pydoc module can be used to get information about objects, functions, etc.**

```
>>> from pydoc import help
>>> help(re)
```
- **pydoc can also be used from the command line, to provide manpage-like documentation for anything in Python:**

```
% pydoc re
```
- **dir() lists all operations and variables contained in an object (list, string, etc):**

```
>>> dir(re)
['DOTALL', 'I', ..., 'split', 'sub', 'subn', 'template']
```



20

CIS-530

Functions

- A **function** is a reusable piece of a program.

- **Functions are defined with `def`**

```
>>> def square(x):
...     return x*x
>>> print square(8)
64
```

- **Optional arguments:**

```
>>> def power(x, exp=2):           # exp defaults to 2
...     if x <= 0: return 1
...     else: return x*power(x, exp-1)
```



21

CIS-530

Classes

- A **class** is a kind of object (like lists or strings) that contains variables and operations (or **methods**)

- **The simplest class:**

```
>>> class Simple: pass
```

- **Class objects are created with the constructor, which has the same name as the class:**

```
>>> obj = Simple()
```

- **Variables are accessed as `obj.var`**

```
>>> obj.x = 3
```



22

CIS-530

An Example Class

```
>>> class Account:
...     def __init__(self, initial):
...         self.balance = initial
...     def deposit(self, amt):
...         self.balance = self.balance + amt
...     def withdraw(self, amt):
...         self.balance = self.balance - amt
...     def getbalance(self):
...         return self.balance
```

- **`__init__` defines the constructor**
- **`self` is the object that is being manipulated.**
 - It is the first argument to every method.



23

CIS-530

Using the example class

```
>>> a = Account(1000.00)
>>> a.deposit(550.23)
>>> print a.getbalance()
1550.23
>>> a.deposit(100)
>>> a.withdraw(50)
>>> print a.getbalance()
1600.23
```



24

CIS-530