LING 408/508: Programming for Linguists

Lecture 21
November 18th
Administrivia

• Email from me about HW group 7, 8, and 9.

• Reminder:
  – Quickie exercises HW 10 due Saturday by midnight
# futval.py
# A program to compute the value of an investment
# carried 10 years into the future

def main():
    print "This program calculates the future value"
    print "of a 10 year investment."

    principal = input("Enter the initial principal: ")
    apr = input("Enter the annualized interest rate: ")

    for i in range(10):
        principal = principal * (1 + apr)

    print "The value in 10 years is:", format(principal,'10.2f')

main()
Formated Output

• Lots of ways: (Old way)

```python
>>> print "x is %.2f" % 5
x is 5.00
```

```python
>>> print "x is %.2f and %2d" % (5,5)
x is 5.00 and  5
```

Notation comes originally from the C Programming Language:
`sprintf` function
http://www.tutorialspoint.com/c_standard_library/c_function_sprintf.htm
Formatted Output

<template-string> % (<values>)

- \%<width>.<precision><type>
- <type> = d, f, s
- <width> = minimum number of characters; right-justified by default, -width => left-justified 0 = as wide as needed
- <precision> = number of places after decimal point

- e.g. 02d two digits wide, pad with 0 if needed
Formatted Output

• Newer way:
  – [https://docs.python.org/2/tutorial/inputoutput.html#fancier-output-formatting](https://docs.python.org/2/tutorial/inputoutput.html#fancier-output-formatting)
  – Use {} for each argument
    • (can be numbered, e.g. {0}, {1},... or referenced by keyword {x})

```python
>>> import math
>>> print 'The value of PI is approximately {0:.3f}'.format(math.pi)
The value of PI is approximately 3.142.
```

```python
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}
>>> for name, phone in table.items():
...     print '{0:10} ==> {1:10d}'.format(name, phone)
...
Jack ==> 4098
Dcab ==> 7678
Sjoerd ==> 4127
```
Last Time

- Chapter 4

```python
SBS2893:Desktop sandiway$ python
Python 2.7.5 (default, Mar 9 2014, 22:15:05)
[GCC 4.2.1 Compatible Apple LLVM 5.0 (clang-500.0.68)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> x = input("Enter: ")
Enter: abc
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1, in <module>
NameError: name 'abc' is not defined
>>> x = input("Enter: ")
Enter: 'ab'
>>> print x
ab
>>> >>>
```
String slicing

- String is like an array of characters (strings):
  - `str[i]` index i (from 0 to `len(str)-1`)
  - `str[-i]` index i from the end (1 = last)
  - `str[i:j]` slice from index i until index j-1
  - `str[:j]` slice from index 0 until index j-1
  - `str[i:]` slice from index i until end of the string

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Concatenation</td>
</tr>
<tr>
<td>*</td>
<td>Repetition</td>
</tr>
<tr>
<td>&lt;string&gt;[]</td>
<td>Indexing</td>
</tr>
<tr>
<td>&lt;string&gt;[: ]</td>
<td>Slicing</td>
</tr>
<tr>
<td>len(&lt;string&gt;)</td>
<td>Length</td>
</tr>
<tr>
<td>for &lt;var&gt; in &lt;string&gt;</td>
<td>Iteration through characters</td>
</tr>
</tbody>
</table>

Table 4.1: Python string operations.
**List vs. Strings**

- Lists are mutable, Strings are not

```python
>>> myList = [34, 26, 15, 10]
>>> myList[2]
15
>>> myList[2] = 0
>>> myList
[34, 26, 0, 10]
>>> myString = "Hello World"
>>> myString[2]
'l'
>>> myString[2] = 'z'
Traceback (innermost last):
  File "<stdin>", line 1, in ?
TypeError: object doesn’t support item assignment
```
Strings and Unicode

- ASCII ↔ string:

  >>> ord("a")
  97
  >>> ord("A")
  65

  >>> chr(97)
  'a'
  >>> chr(90)
  'Z'

  >>> x = "é"
  >>> x[0]
  '\xe9'

  >>> ord(x)
  Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
  TypeError: ord() expected a character, but string of length 2 found

  >>> ord(x[0])
  195
  >>> ord(x[1])
  169
Strings and Unicode

- Example: u (unicode) prefix

```python
>>> x = "á".encode('utf-8')
>>> type(x)
<type 'str'>
>>> x[0]
'\xc3'
>>> x[1]
'\xa1'

>>> x = "á".encode('utf-8')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeDecodeError: 'ascii' codec can't decode byte 0xc3 not in range(128)

>>> x = u"á".encode('utf-8')
>>> y = x.decode('utf-8')
>>> type(y)
<type 'unicode'>

>>> y[0]
'\xe1'
>>> y[1]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```
Like all other programming languages, uses a file handle, called file variable: `open()`

```python
f = codecs.open('file.txt', encoding='utf-8')
infile = open("file.txt","r")
outfile = open("results.txt","w")
```

- `<filevar>.read()` Returns the entire remaining contents of the file as a single (potentially large, multi-line) string.
- `<filevar>.readline()` Returns the next line of the file. That is all text up to and including the next newline character.
- `<filevar>.readlines()` Returns a list of the remaining lines in the file. Each list item is a single line including the newline character at the end.

```python
infile = open(someFile, 'r')
for line in infile.readlines():
    # process the line here
infile.close()

infile = open(someFile, 'r')
for line in infile:
    # process the line here
infile.close()
```