Lecture 13

Topics: *Chapter 5. Computing with strings*

5.8 Input/output as string manipulation

5.9 File processing

5.8 Input/output as string manipulation

What did we do so far with `print` method:

1. We can specify an end of line:

```
print("One")
print("Two")
print("Three")
```

```
OUTPUT:

One
Two
Three
```

```
print("One", end =' ')
print("Two", end =" ")
print("Three")
```

```
OUTPUT:

One Two Three
```

```
print("One",end ='ay')
print("Two",end ="ho")
print("Three")
```

```
OUTPUT:

OneayTwohoThree
```

5.8 Input/output as string manipulation

What did we do so far with `print` method:

1. We can specify an end of line:
   Note that you can put anything you like inside the quotes:

```
print("One",end ='ay')
print("Two",end ="ho")
print("Three")
```

```
OUTPUT:

OneayTwohoThree
```

5.8 Input/output as string manipulation

What did we do so far with `print` method:

2. We can specify an item separator:

```
print(“Hello”,”How”,”are”,”you”,”today?”,sep=“***”)
```

result: Hello***How***are***you***today?

5.8 Input/output as string manipulation

What did we do so far with `print` method:

3. We can use escape characters:

`\n`	New line
`\t`	Tabulation (skips few spaces)
`\'`	Single quote will be printed
`\”`	Double quote will be printed
`\\`	Backslash character will be printed

Example: `print(“One \t two \t \”three\””)`

Result: One	two	"three"

Exercise: print the following sentences in a python shell:
The symbol \ is called ‘backslash’.
“’” is a single quote, whereas “”” is a double quote

5.8 Input/output as string manipulation

String formatting

Basic string operations can be used to build nicely formatted output, but building up a complex output can be tedious.

Python provides a powerful *string formatting operation*.

type in the following in the interactive window:

```
>>> total=12.567
>>> print(“The total value is ${0:0.2f}. Good 
buy!”.format(total))
The total value is $12.57. Good buy!
>>>
```

5.8 Input/output as string manipulation

String formatting operator:

```
<template-string>.format(<values>),
```

formatting specifier has the following general form:

```
{<index>:<format-specifier>}
```

`index:` Tells which of the values is inserted into the slot.

`format-specifier: <width>.<precision><type>`

width: Number of spaces to use in displaying value.
(0 means «use as much space as needed»)

precision: How many decimal places (rounds off)

type: Format types:

d	decimal integer
f	float
s	string

we will see more

5.8 Input/output as string manipulation

String formatting

Type the following in the interactive window:

```
>>> “Good day {0} {1}, you have ${2} on your
account balance.”.format('Mr.','Smith',150000)
'Good day Mr. Smith, you have $150000 on your
account balance'

>>> “This integer number, {0:8}, was placed in a
field of width 8”.format(12)
'This integer number,       12, was placed in a
field of width 8'
```

5.8 Input/output as string manipulation

String formatting

Type the following in the interactive window:

```
>>> ”This decimal number, {0},  was rounded of to
three decimal places: {0:.3f}”.format(3.141592654)
```

```
OUTPUT: 'This decimal number, 3.141592654,  was
rounded of to three decimal places: 3.142'
```

Now try

```
>>> ”This decimal number, {0},  was rounded of to
three decimal places: {0:.30f}”.format(3.14)
```

```
>>> ”This decimal number, {0},  was rounded of to
three decimal places: {0:.2}”.format(33.14)
```

```
>>> ”This decimal number, {0},  was rounded of to
three decimal places: {0:.2}”.format(33.14)
```

5.8 Input/output as string manipulation

String formatting

Type the following in the interactive window:

```
>>> num,denom=3.123,4.234
>>> print("{0:.2f} / {1:.2f}= {2:.2f}".format(num,
denom, num/denom))
3.12 / 4.23= 0.74

>>>print(format(num,'.2f'),"/",format(denom,'.2f'),
"=",format(num/denom,'.2f'))
3.12 / 4.23 = 0.74
```

same results!

The built-in format function takes two arguments:

A numeric value, and

A format specifier

5.8 Input/output as string manipulation

String formatting

```
Type the following in the interactive window:
>>> n=23
>>> print("{0:4d}".format(n))
   23

>>> print(format(n,'4d'))
   23
```

5.8 Input/output as string manipulation

Conversion	Meaning
'd'	Signed decimal integer.
'i'	Signed decimal integer.
'o'	Signed octal integer.
'x'	Signed hexadecimal integer (lowercase).
'X'	Signed hexadecimal integer (uppercase).
'e'	Floating point exponential format (lowercase).
'E'	Floating point exponential format (uppercase).
'f' or 'F'	Floating point decimal format.
'c'	Single character (accepts integer or single character string).
's'	String (converts any Python object using str()).

5.9 File processing

Programs must be able to read data from file and to write data to files. It is especially needed when we have a large volume of data.

Python supports a built-in class file to manipulate files on the computer.

Constructor of Python's file class accepts two parameters:

- *file name* (as string), and
- *access mode* (as string, optional)
    - r – for reading (default mode)
    - w – for (over)writing
    - a – for appending to the end of the file

5.9 File processing

Constructor of Python's file class accepts two parameters:

- *file name* (as string), and
- *access mode* (as string, optional)

r – for reading (default mode)
w – for (over)writing
a – for appending to the end of the file

Example:

```
file1 = open('inputData.txt')
```

→ File inputData will be open in read-only mode

```
file2 = open('outputData.txt','w')
```

→ File outputData will be open for writing (re-writing)

Syntax	Semantics
open()	Returns a file object (two arguments)
close()	Disconnects the file object from the associated file (saving it, if necessary)
flush()	Flushes the buffer of written characters, saving the underlying file
read()	Returns a string representing the (remaining) contents of the file
read(size)	Returns a string representing the specified number of bytes next in the file
readline()	Returns a string representing the next line of the file
readlines()	Returns a list of strings representing the remaining lines of the file
write(s)	Writes the given string to the file. No newlines are added.
writelines(seq)	Writes each of the strings to the file. No newlines are added.
for line in f	Iterates through the file~f, one line at a time

5.9 File processing

Example 1: Let's open a file and display everything it has.

data.txt:

```
12 23
14 45
-4 -20
-15 -60
15 40
17 60
-10 -48
```

numbers.txt

```
1 3 2 4 6 5 2 3
```

names.txt

```
Maria 6
Anna 7
Alex 10
Frank 11
Uma 6
Nicholas 13
```

See programs readAllFromFile.py and readAllFromFile_mod.py

5.9 File processing

<u>Example 2</u>: Let's generate data this time: write a program than generates n pairs of values (x,y), where $x \in [-100,100]$ and $y \in [0,1000]$ randomly. n is provided by the user.
These pairs of values are stored in a file “outData.txt”.

Design / algorithm:

```
open a file
prompt for n
for i in range(n)
    generate x-value, record into a file adding space at the end
    generate y-value, record into a file adding “end of line”
close file
```

see program createDataFile.py

5.9 File processing

Example 3: Let's process the data from file “outData.txt”: find the average of *x*-values and *y*-values separately

Design / algorithm:

```
open a file
sumX =0 for sum of x-values, sumY = 0 for sum of y-values
counter = 0 for counting pairs
for line in file
    split line into two parts,
    convert each part to integer value (x and y)
    sumX += x
    sumY += y
    counter += 1
output sumX / counter and sumY / counter
close file
```

see program processDataFile.py