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 ***break long statements into multiple lines***:

```
print(“Hello, today's temperature is”, degrees, \
          “degrees, but it feels like”, feels, \
          “degrees.”)
```

same will be for almost any statement, for example:

```
result = 1*2 + 1*2**2 + 1 * 3**2 + 4 * 2**5 + \
             6 * 2**6 + 7 * 2**7
```

5.8 Input/output as string manipulation

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

2. The ***newline*** character

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

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

- one space between the single quotes

5.8 Input/output as string manipulation

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

3. we can specify an item separator: ***break long statements into multiple lines***:

```
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:

4. 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"

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!
>>>
```

```
print("The total value is ${0:0.2f}. Good buy!".format(total))
```

string formatting operator

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

formatting specifier has the following general form:
`{<index>:<format-specifier>}`

`index` tells which of the parameters is inserted into the slot.
`format-specifier` is `<width>.<precision><type>`

width: how many spaces to use in displaying value
0 - «use as much space as needed» (or simply allot it)
precision: how many decimal places (rounds off)
type: format types
d decimal
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)
'This decimal number, 3.141592654,  was rounded of to three decimal places: 3.142'
```

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 in Python 2

In **Python 2** print was not formally a function, but instead a *keyword* of the language. Therefore its arguments were not specified within parentheses.

In **Python 2** to output formatted strings we can use the % operator (modulo).
% sign starts the conversion 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

String formatting

Type the following in the interactive window:

```
>>> data = [1024,4,16,32]
>>> print("{0:4d} {1:4d} {2:4d} {3:4d}".
format(data[0], data[1], data[2], data[3]))
1024    4   16   32
>>> print('---- ' * len(data))
---- ---- ---- ----

>>> print(format(data[0],'4d'),
format(data[1],'4d'),format(data[2],'4d'),
format(data[3],'4d'))
1024    4   16   32
>>> print('---- ' * len(data))
---- ---- ---- ----
```

5.8 Input/output as string manipulation

Conversion	Meaning
'd'	Signed integer decimal.
'i'	Signed integer decimal.
'o'	Signed octal value.
'x'	Signed hexadecimal (lowercase).
'X'	Signed hexadecimal (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()).

All the examples are gathered in program experiments.py

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

Answer the question:
Why is the code in readAllFromFile_mod.py is better than the code in readAllfromFile.py?

5.9 File processing

Example 2: 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