Input and output

Inteligencia Artificial en los Sistemas de Control Autónomo
Máster en Ciencia y Tecnología desde el Espacio

Departamento de Automática

Universidad de Alcalá
ISG

Objectives

1. Being able to apply output formatting methods in Python.
2. Being able to manipulate files in Python.
3. Being able to understand the usefulness of Python serialization (*pickles*).

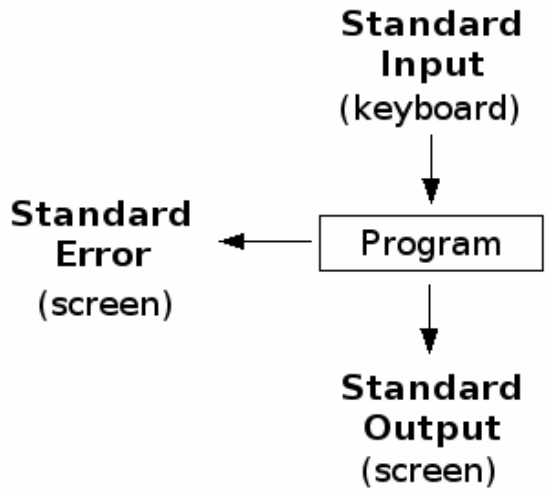
Table of Contents

Input and output

Introduction

Input/output: How the program can read and export data.

- So far, we have used two methods to output information:
  - Expressions statements and the `print()` function.
- A third method: Standard input and output.



Source: `http://labor-liber.org/en/gnu-linux/introduction/input_output`

I/O interactive

Data input

Enter data by keyboard (version 2.X)

```
>>> x = raw_input('Introduzca un numero:   ')
64.5
>>> y = float(x) ** 2
```

Enter data by keyboard (version 3.X)

```
>>> x = input('Introduzca un numero:   ')
64.5
>>> y = x ** 2
```

I/O interactive

Data output (I)

Print **not formatted** data (version 2.X):
Needs () for version 3.X

```
>>> print 'message', var1, var2, ..., vark
```

Prints on the screen: message var1 var2 vark

```
>>> name = 'John'
>>> age = 37
>>> print 'Name, age= ', name, age
Name, age= John 37
>>> print 'Name = ', name, ' age = ', age
Name = John age = 37
```

I/O interactive

Data output (II)

Print **formatted** data (version 2.X):

```
>>> print 'msg1 = %type1 msg2 = %type2' % (var1, var2)
```

where `type1` and `type2` indicate how to represent the variable:

%i and %d: integer number.

%f: real number with decimal point.

%e: real number in exponential format.

%g: remove not significant zeros.

%s: string.

```
>>> name = 'John'
>>> daybal = 55.5
>>> print '%s earns per month %6.2f euros' % (name,  daybal *30.)
John earns per month 1515.00 euros
```

Methods

Custom output

- Two methods to create custom output:
  - String manipulation.
  - The `str.format()` method.
- **Convert values to strings:**
  - `str()`: Human-readable format.
  - `repr()`: Interpreter-readable format.
  - Both, are quite similar. But, strings have two representations:

```
>>> str1 = ``Hellow\n"
>>> str(str1)
'Hellow\n'
>>> repr(str1)
``'Hellow\\n'"
>>> repr([234, ('hellow', 'bye')])
"[234, ('hellow', 'bye')]"
>>> str([234, ('hellow', 'bye')])
"[234, ('hellow', 'bye')]"
```

Examples of fancy output

Table of squares and cubes I

```
1 for x in range(1, 11):
2     print(repr(x).rjust(2), repr(x*x).rjust(3),end=' ')
3     print(repr(x*x*x).rjust(4))
```

Table of squares and cubes II

```
1 for x in range(1, 11):
2   print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
```

Useful methods

Method	Description
`str.rjust(n)`	Right justification `n` characters
`str.ljust(n)`	Left justification `n` characters
`str.center(n)`	Center `n` characters
`str.zfill(n)`	Fill left with `n` zero

The `format()` method

Use (I)

- Basic usage:

```
>>> print('{} and {}'.format('spam', 'eggs'))
spam and eggs
>>> print('{1} and {0}'.format('spam', 'eggs'))
eggs and spam
```

The `format()` method

Use (II)

- Additional formatting:

```
>>> import math
>>> math.pi
3.14159265358979 3
>>> print('PI values {0:.3f}'.format(math.pi))
PI values 3.142
```

- It's also possible to left or right justify data with the format method preceding the format with the options '<' (left justify) or '>' (right justify).

For more examples, *Click Here!*

The `format()` method

Use (III)

```
>>> 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
```

Path

- On Linux, the path is denoted by:

```
path = '/tmp/prueba.txt'
```

- On Windows, the path is denoted by:

```
path = 'C:\Windows\Temp'
```

And it is represented in Python by:

```
path = 'C:\\Windows\\Temp'
```

But by also using *raw string*:

```
path = r'C:\Windows\Temp'
```

Opening files

- All file operations are made through a *file object*.
- A file is a sequence of bytes. But ..., it's often useful to treat it as a sequence of lines.
- First of all: Call the `open()` function.

The `open()` function

`open(filename[, mode])`

Description: The function returns an object file.

- *filename*: String with the file name.
- *mode*: Characters describing how the file will be used:
  - r: Reading mode, *w*: Writing mode, +: Reading/Writing mode.
  - *b*: Binary mode, *a*: Appending mode.

Remember: Always, always, always close the file: `f.close()`

Methods of file objects

Reading files (I)

The `read()` function

`f.read([size])`

- *size*: The number of bytes to be read from the file.
- Return value: The bytes read in string.

Option r: Read the entire file (`f.read()`)

```
>>> f = open("/tmp/file", 'r+')
>>> f.read()
'This is the entire file.\\n'
>>> f.read()
''
>>> f.close()
```

Methods of file objects

Reading files (II)

Option 2: Read a single line (`f.readline()`)

```
>>> f = open("/tmp/file2", 'r+')
>>> f.readline()
'This is the first line of the file.\n'
>>> f.readline()
'This is the second line of the file\n'
>>> f.readline()
''
>>> f.close()
```

Methods of file objects

Reading files (III)

Option 3: Read lines as list (`f.readlines()`)

```
>>> f = open("/tmp/file2", 'r+')
>>> f.readlines()
['This is the first line of the file.\n',
'This is the second line of the file\n']
>>> f.close()
```

Option 4: Read in a loop

```
f = open("/tmp/file2", 'r+')
for line in f:
    print(line, end='')
f.close()
```

Methods of file objects

Writing files (I)

The `write()` function

`f.write(string)`

- *string*: String to write in file.
- Return value: The number of written bytes.

Example 1: Write a line

```
>>> f = open("/tmp/file", 'w+')
>>> f.write('This is a test\n')
15
>>> f.read()
''
>>> f.close()
```

Methods of file objects

Writing files (II)

Example 2: Write a number

```
>>> f = open("/tmp/file", 'w+')
>>> f.write(str(42))
2
>>> f.close()
```

Others file management methods

Useful methods

Method	Description
`f.tell()`	Returns the pointer's position
`f.seek(n)`	Moves the pointer `n` bytes
`f.close()`	Closes a file. Use it always!

```
>>> f = open("/tmp/file", 'rb+')
>>> f.write(b'0123456789abcdef')
16
>>> f.seek(5)
5
>>> f.read(1)
b'5'
```

Example 1

Calculating the average of characters per line of file `example.txt`

```
1 file_ex = open('example.txt', 'r')
2 num_total_char = 0
3 count_line= 0
4
5 for line in file_ex:
6   count_line += 1
7   num_total_char += len(line)
8 file_ex.close()
9 print('average', float(num_total_char) / float(count_line))
```

Example 2

Reading a line each time

```
1 count_line = 0
2 with open('/Users/julia/code/names.txt') as arch_names:
3     for line in arch_names:
4         count_line += 1
5         print('{:<10}{}'.format(count_line, line.rstrip()))
```

names.txt

```
1 Juan
2 Laura
3 Pablo
4 Enrique
5 Javier
```

Output

```
1         Juan
2         Laura
3         Pablo
4         Enrique
5         Javier
```

The `pickle` module

Introduction

- What happens if we need to store complex data structures?
  - Think about lists, dictionaries or even objects ...
  - The `pickle` module comes to help.
- *Pickling*: Transform an object to string representation.
- *Unpickling*: Reconstruct an object from its string representation.
- Given an object `x` and a file object `f`.

```
>>> pickle.dump(x, f)
>>> x = pickle.load(f)
```

The `pickle` module

Example: Save/load data structure to/from a file

Save a list to a file

```
1 import pickle
2
3 list_number = [2, 5, 7, 8]
4
5 pickle.dump(list_number, open('file1_list.txt', 'wb'))
```

Load a list from a file

```
1 import pickle
2
3 list_number = pickle.load(open('file1_list.txt', 'rb'))
4 print(list_number)
```

Bibliographic references I

[van Rosum, 2012] G. van Rossum, Jr. Fred L. Drake.
Python Tutorial Release 3.2.3, chapter 7.
Python Software Foundation, 2012.

[Pilgrim, 2004] M. Pilgrim.
Dive into Python, chapter 6.
Ed. Prentice Hall, 2004.

[Marzal, 2014] A. Marzal Varó, I. Gracia Luengo, P. García Sevilla.
Introducción a la programación con Python 3, capítulo 8.
Universidad Jaime I.

Bibliographic on scientific programming I


[Langtangen, 2008] H. Petter Langtangen.
Python Scripting for Computational Science.
Ed. Springer, 2008.


[McKinney, 2013] W. McKinney
Python for Data Analysis.
Ed. O'Reilly, 2013.