Python For Data Science *Cheat Sheet*

Python Basics

Learn More Python for Data Science Interactively at www.datacamp.com

Variables and Data Types

Variable Assignment

```
>>> x=5
>>> x
 5
```

Calculations With Variables

`>>> x+2`
`7`
	Sum of two variables

`>>> x-2`
`3`
	Subtraction of two variables

`>>> x*2`
`10`
	Multiplication of two variables

`>>> x**2`
`25`
	Exponentiation of a variable

`>>> x%2`
`1`
	Remainder of a variable

`>>> x/float(2)`
`2.5`
	Division of a variable

Types and Type Conversion

`str()`	`'5', '3.45', 'True'`	Variables to strings
`int()`	`5, 3, 1`	Variables to integers
`float()`	`5.0, 1.0`	Variables to floats
`bool()`	`True, True, True`	Variables to booleans

Asking For Help

```
>>> help(str)
```

Lists

Also see NumPy Arrays

```
>>> a = 'is'
>>> b = 'nice'
>>> my_list = ['my', 'list', a, b]
>>> my_list2 = [[4,5,6,7], [3,4,5,6]]
```

Selecting List Elements

Index starts at 0

Subset	
`>>> my_list[1]`	Select item at index 1
`>>> my_list[-3]`	Select 3rd last item
Slice	
`>>> my_list[1:3]`	Select items at index 1 and 2
`>>> my_list[1:]`	Select items after index 0
`>>> my_list[:3]`	Select items before index 3
`>>> my_list[:]`	Copy my_list
Subset Lists of Lists	

`>>> my_list2[1][0]`
`>>> my_list2[1][:2]`
	my_list[list][itemOfList]

List Operations

```
>>> my_list + my_list
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list * 2
['my', 'list', 'is', 'nice', 'my', 'list', 'is', 'nice']
>>> my_list2 > 4
True
```

List Methods

`>>> my_list.index(a)`	Get the index of an item
`>>> my_list.count(a)`	Count an item
`>>> my_list.append('!')`	Append an item at a time
`>>> my_list.remove('!')`	Remove an item
`>>> del(my_list[0:1])`	Remove an item
`>>> my_list.reverse()`	Reverse the list
`>>> my_list.extend('!')`	Append an item
`>>> my_list.pop(-1)`	Remove an item
`>>> my_list.insert(0,'!')`	Insert an item
`>>> my_list.sort()`	Sort the list

Libraries

Import libraries

```
>>> import numpy
>>> import numpy as np
```

Selective import

```
>>> from math import pi
```

pandas	Data analysis
learn	Machine learning
NumPy	Scientific computing
matplotlib	2D plotting

Install Python

ANACONDA	spyder	jupyter
Leading open data science platform powered by Python	Free IDE that is included with Anaconda	Create and share documents with live code, visualizations, text, ...

Numpy Arrays

Also see Lists

```
>>> my_list = [1, 2, 3, 4]
>>> my_array = np.array(my_list)
>>> my_2darray = np.array([[1,2,3],[4,5,6]])
```

Selecting Numpy Array Elements

Index starts at 0

Subset	

`>>> my_array[1]`
`2`
	Select item at index 1

Slice	

`>>> my_array[0:2]`
`array([1, 2])`
	Select items at index 0 and 1

Subset 2D Numpy arrays	

`>>> my_2darray[:,0]`
`array([1, 4])`
	my_2darray[rows, columns]

Numpy Array Operations

```
>>> my_array > 3
  array([False, False, False,  True], dtype=bool)
>>> my_array * 2
  array([2, 4, 6, 8])
>>> my_array + np.array([5, 6, 7, 8])
  array([6, 8, 10, 12])
```

Numpy Array Functions

`>>> my_array.shape`	Get the dimensions of the array
`>>> np.append(other_array)`	Append items to an array
`>>> np.insert(my_array, 1, 5)`	Insert items in an array
`>>> np.delete(my_array,[1])`	Delete items in an array
`>>> np.mean(my_array)`	Mean of the array
`>>> np.median(my_array)`	Median of the array
`>>> my_array.corrcoef()`	Correlation coefficient
`>>> np.std(my_array)`	Standard deviation

Strings

```
>>> my_string = 'thisStringIsAwesome'
>>> my_string
'thisStringIsAwesome'
```

String Operations

```
>>> my_string * 2
  'thisStringIsAwesomethisStringIsAwesome'
>>> my_string + 'Innit'
  'thisStringIsAwesomeInnit'
>>> 'm' in my_string
  True
```

String Operations

Index starts at 0

```
>>> my_string[3]
>>> my_string[4:9]
```

String Methods

`>>> my_string.upper()`	String to uppercase
`>>> my_string.lower()`	String to lowercase
`>>> my_string.count('w')`	Count String elements
`>>> my_string.replace('e', 'i')`	Replace String elements
`>>> my_string.strip()`	Strip whitespaces

Python For Data Science *Cheat Sheet*

Importing Data

Learn Python for data science **Interactively** at www.DataCamp.com

Importing Data in Python

Most of the time, you'll use either **NumPy** or **pandas** to import your data:

```
>>> import numpy as np
>>> import pandas as pd
```

Help

```
>>> np.info(np.ndarray.dtype)
>>> help(pd.read_csv)
```

Text Files

Plain Text Files

```
>>> filename = 'huck_finn.txt'
>>> file = open(filename, mode='r')
>>> text = file.read()
>>> print(file.closed)
>>> file.close()
>>> print(text)
```

Open the file for reading
Read a file's contents
Check whether file is closed
Close file

Using the context manager `with`

```
>>> with open('huck_finn.txt', 'r') as file:
        print(file.readline())
        print(file.readline())
        print(file.readline())
```

Read a single line

Table Data: Flat Files

Importing Flat Files with numpy

Files with one data type

```
>>> filename = 'mnist.txt'
>>> data = np.loadtxt(filename,
                      delimiter=',',
                      skiprows=2,
                      usecols=[0,2],
                      dtype=str)
```

String used to separate values
Skip the first 2 lines
Read the 1st and 3rd column
The type of the resulting array

Files with mixed data types

```
>>> filename = 'titanic.csv'
>>> data = np.genfromtxt(filename,
                         delimiter=',',
                         names=True,
                         dtype=None)
```

Look for column header

```
>>> data_array = np.recfromcsv(filename)
```

The default `dtype` of the `np.recfromcsv()` function is `None`.

Importing Flat Files with pandas

```
>>> filename = 'winequality-red.csv'
>>> data = pd.read_csv(filename,
                       nrows=5,
                       header=None,
                       sep='\t',
                       comment='#',
                       na_values=[""])
```

Number of rows of file to read
Row number to use as col names
Delimiter to use
Character to split comments
String to recognize as NA/NaN

Excel Spreadsheets

```
>>> file = 'urbanpop.xlsx'
>>> data = pd.ExcelFile(file)
>>> df_sheet2 = data.parse('1960-1966',
                           skiprows=[0],
                           names=['Country',
                                  'AAM: War(2002)'])
>>> df_sheet1 = data.parse(0,
                           parse_cols=[0],
                           skiprows=[0],
                           names=['Country'])
```

To access the sheet names, use the `sheet_names` attribute:

```
>>> data.sheet_names
```

SAS Files

```
>>> from sas7bdat import SAS7BDAT
>>> with SAS7BDAT('urbanpop.sas7bdat') as file:
        df_sas = file.to_data_frame()
```

Stata Files

```
>>> data = pd.read_stata('urbanpop.dta')
```

Relational Databases

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite://Northwind.sqlite')
```

Use the `table_names()` method to fetch a list of table names:

```
>>> table_names = engine.table_names()
```

Querying Relational Databases

```
>>> con = engine.connect()
>>> rs = con.execute("SELECT * FROM Orders")
>>> df = pd.DataFrame(rs.fetchall())
>>> df.columns = rs.keys()
>>> con.close()
```

Using the context manager `with`

```
>>> with engine.connect() as con:
        rs = con.execute("SELECT OrderID FROM Orders")
        df = pd.DataFrame(rs.fetchmany(size=5))
        df.columns = rs.keys()
```

Querying relational databases with pandas

```
>>> df = pd.read_sql_query("SELECT * FROM Orders", engine)
```

Exploring Your Data

NumPy Arrays

```
>>> data_array.dtype
>>> data_array.shape
>>> len(data_array)
```

Data type of array elements
Array dimensions
Length of array

pandas DataFrames

```
>>> df.head()
>>> df.tail()
>>> df.index
>>> df.columns
>>> df.info()
>>> data_array = data.values
```

Return first DataFrame rows
Return last DataFrame rows
Describe index
Describe DataFrame columns
Info on DataFrame
Convert a DataFrame to an a NumPy array

Pickled Files

```
>>> import pickle
>>> with open('pickled_fruit.pkl', 'rb') as file:
        pickled_data = pickle.load(file)
```

HDF5 Files

```
>>> import h5py
>>> filename = 'H-H1_LOSC_4_v1-815411200-4096.hdf5'
>>> data = h5py.File(filename, 'r')
```

Matlab Files

```
>>> import scipy.io
>>> filename = 'workspace.mat'
>>> mat = scipy.io.loadmat(filename)
```

Exploring Dictionaries

Accessing Elements with Functions

```
>>> print(mat.keys())
>>> for key in data.keys():
        print(key)
meta
quality
strain
>>> pickled_data.values()
>>> print(mat.items())
```

Print dictionary keys
Print dictionary keys
Return dictionary values
Returns items in list format of (key, value) tuple pairs

Accessing Data Items with Keys

```
>>> for key in data ['meta'].keys()
        print(key)
Description
DescriptionURL
Detector
Duration
GPSstart
Observatory
Type
UTCstart
>>> print(data['meta']['Description'].value)
```

Explore the HDF5 structure
Retrieve the value for a key

Navigating Your FileSystem

Magic Commands

```
!ls
%cd ..
%pwd
```

List directory contents of files and directories
Change current working directory
Return the current working directory path

`os` Library

```
>>> import os
>>> path = "/usr/tmp"
>>> wd = os.getcwd()
>>> os.listdir(wd)
>>> os.chdir(path)
>>> os.rename("test1.txt",
              "test2.txt")
>>> os.remove("test1.txt")
>>> os.mkdir("newdir")
```

Store the name of current directory in a string
Output contents of the directory in a list
Change current working directory
Rename a file
Delete an existing file
Create a new directory

Python 3 Cheat Sheet

Official Python documentation on http://docs.python.org/py3k

Base Types

integer, float, boolean, string

```
  int  783    0    -192
float  9.23   0.0   -1.7e-6
 bool  True   False
  str  "One\nTwo"   'I\'m'
```

$-1.7\underbrace{\text{e-6}}_{10^{-6}}$; `"One\nTwo"`: `\n` new line ; `'I\'m'`: `'` escaped

`str` ↑ immutable, ordered sequence of chars

multiline
```
"""X\tY\tZ
1\t2\t3"""
```
`\t` tab char

Container Types

▪ ordered sequence, fast index access, repeatable values

```
 list  [1,5,9]   ["x",11,8.9]   ["word"]    []
tuple  (1,5,9)    11,"y",7.4    ("word",)   ()
```

`11,"y",7.4`: expression with just comas

immutable ↗ `tuple` ↘ `str` as an ordered sequence of chars

▪ no *a priori* order, unique key, fast key access ; keys = base types or tuples

```
dict  {"key":"value"}                              {}
      {1:"one",3:"three",2:"two",3.14:"π"}
```

dictionary — *key/value associations*

```
set  {"key1","key2"}    {1,9,3,0}    set()
```

Identifiers

for variables, functions, modules, classes… names

`a..zA..Z_` followed by `a..zA..Z_0..9`

- ▫ diacritics allowed but should be avoided
- ▫ language keywords forbidden
- ▫ lower/UPPER case discrimination

☺ `a toto x7 y_max BigOne`

☹ ~~`8y`~~ ~~`and`~~

Variables assignment

$\texttt{x} = \underbrace{\texttt{1.2+8+sin(0)}}_{\text{value or computed expression}}$

↑ variable name (identifier)

$\underbrace{\texttt{y,z,r}}_{\text{variables names}} = \underbrace{\texttt{9.2,-7.6,"bad"}}_{\text{container with several values (here a tuple)}}$

`x+=3` ← increment
decrement → `x-=2`

`x=None` « undefined » constant value

Conversions

`type(`*expression*`)`

`int("15")`	can specify integer number base in 2nd parameter
`int(15.56)`	truncate decimal part (`round(15.56)` for rounded integer)
`float("-11.24e8")`	
`str(78.3)`	and for litteral representation → `repr("Text")`

see other side for string formating allowing finer control

`bool` → use comparators (with `==`, `!=`, `<`, `>`, …), logical boolean result

`list("abc")` —*use each element from sequence*→ `['a','b','c']`

`dict([(3,"three"),(1,"one")])` → `{1:'one',3:'three'}`

`set(["one","two"])` —*use each element from sequence*→ `{'one','two'}`

$\underbrace{\texttt{":"}}_{\text{joining string}}\texttt{.join(}\underbrace{\texttt{['toto','12','pswd']}}_{\text{sequence of strings}}\texttt{)} \rightarrow$ `'toto:12:pswd'`

`"words with  spaces".split()` → `['words','with','spaces']`

$\texttt{"1,4,8,2".split(}\underbrace{\texttt{","}}_{\text{splitting string}}\texttt{)} \rightarrow$ `['1','4','8','2']`

Sequences indexing

for lists, tuples, strings, …

```
negative index  -6    -5     -4      -3    -2     -1
positive index   0     1      2       3     4      5
          lst=[11,   67,   "abc",  3.14,  42,  1968]
positive slice 0    1     2       3      4     5      6
negative slice -6   -5    -4      -3     -2    -1
```

`len(lst)` → 6

individual access to items via `[`*index*`]`

`lst[1]`→`67`	`lst[0]`→`11` *first one*
`lst[-2]`→`42`	`lst[-1]`→`1968` *last one*

access to sub-sequences via `[`*start slice*`:`*end slice*`:`*step*`]`

```
lst[:-1]→[11,67,"abc",3.14,42]      lst[1:3]→[67,"abc"]
lst[1:-1]→[67,"abc",3.14,42]        lst[-3:-1]→[3.14,42]
lst[::2]→[11,"abc",42]              lst[:3]→[11,67,"abc"]
lst[:]→[11,67,"abc",3.14,42,1968]   lst[4:]→[42,1968]
```

Missing slice indication → from start / up to end.

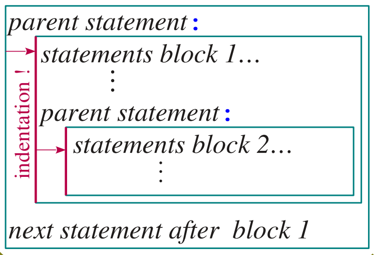
On mutable sequences, usable to remove `del lst[3:5]` *and to modify with assignment* `lst[1:4]=['hop',9]`

Boolean Logic

Comparators: `<  >  <=  >=  ==  !=`
≤ ≥ = ≠

`a and b`	logical and — *both simultaneously*
`a or b`	logical or — *one or other or both*
`not a`	logical not
`True`	true constant value
`False`	false constant value

Statements Blocks



Conditional Statement

statements block executed only if a condition is true

```
if logical expression:
    statements block
```

can go with several elif, elif... and only one final else, example :

```
if x==42:
    # block if logical expression x==42 is true
    print("real truth")
elif x>0:
    # else block if logical expression x>0 is true
    print("be positive")
elif bFinished:
    # else block if boolean variable bFinished is true
    print("how, finished")
else:
    # else block for other cases
    print("when it's not")
```

Maths

☝ *floating point numbers… approximated values!* *angles in radians*

Operators: `+ - * / // % **`
× ÷ ; `//` integer ÷ ; `%` ÷ remainder ; `**` a^b

```
(1+5.3)*2→12.6
abs(-3.2)→3.2
round(3.57,1)→3.6
```

```
from math import sin,pi…
sin(pi/4)→0.707…
cos(2*pi/3)→-0.4999…
acos(0.5)→1.0471…
sqrt(81)→9.0      √
log(e**2)→2.0     etc. (cf doc)
```

Conditional loop statement

statements block executed as long as condition is true

```
while logical expression:
    statements block
```

```
s = 0 } initializations before the loop
i = 1 }
```

condition with at least one variable value (here **i***)*

```
while i <= 100:
    # statement executed as long as i ≤ 100
    s = s + i**2
    i = i + 1 } ☝ make condition variable change
print("sum:",s) } computed result after the loop
```

$$s=\sum_{i=1}^{i=100} i^2$$

☝ *be careful of inifinite loops !*

Loop control

break *immediate exit*

continue *next iteration*

Iterative loop statement

statements block executed for each item of a container or iterator

```
for variable in sequence:
    statements block
```

Go over sequence's **values**

```
s = "Some text" } initializations before the loop
cnt = 0
```

loop variable, value managed by **for** *statement*

```
for c in s:
    if c == "e":
        cnt = cnt + 1
print("found",cnt,"'e'")
```

Count number of **e** *in the string*

loop on dict/set = loop on sequence of keys

use slices to go over a subset of the sequence

Go over sequence's **index**

- ▫ modify item at index
- ▫ access items around index (before/after)

```
lst = [11,18,9,12,23,4,17]
lost = []
for idx in range(len(lst)):
    val = lst[idx]
    if val > 15:
        lost.append(val)
        lst[idx] = 15
print("modif:",lst,"-lost:",lost)
```

Limit values greater than 15, memorization of lost values.

Go simultaneously over sequence's **index** and **values**:

```
for idx,val in enumerate(lst):
```

Display / Input

```
print("v=",3,"cm :",x,",",y+4)
```

items to display: litteral values, variables, expressions

print options:

- ▫ `sep=" "` (items separator, default space)
- ▫ `end="\n"` (end of print, default new line)
- ▫ `file=f` (print to file, default standard output)

```
s = input("Instructions:")
```

☝ **input** always returns a **string**, convert it to required type (cf boxed *Conversions* on on ther side).

Operations on containers

Note: For dictionaries and set, these operations use ***keys****.*

`len(c)` → items count

`min(c)` `max(c)` `sum(c)`

`sorted(c)` → sorted copy

`val in c` → boolean, membersihp operator `in` (absence `not in`)

`enumerate(c)` → *iterator* on (index,value)

Special for ***sequence containeurs*** *(lists, tuples, strings) :*

`reversed(c)` → reverse *iterator* `c*5` → duplicate `c+c2` → concatenate

`c.index(val)` → position `c.count(val)` → events count

Generator of int sequences

frequently used in **for** *iterative loops*

default 0 not included

`range([start,]stop [,step])`

```
range(5)          → 0 1 2 3 4
range(3,8)        → 3 4 5 6 7
range(2,12,3)     →  2 5 8 11
```

range returns a « generator », converts it to list to see the values, example:

```
print(list(range(4)))
```

Operations on lists

☝ modify original list

`lst.append(item)`	add item at end
`lst.extend(seq)`	add sequence of items at end
`lst.insert(idx,val)`	insert item at index
`lst.remove(val)`	remove first item with value
`lst.pop(idx)`	remove item at index and return its value
`lst.sort()` `lst.reverse()`	sort / reverse list *in place*

Function definition

function name (identifier)

named parameters

```
def fctname(p_x,p_y,p_z):
    """documentation"""
    # statements block, res computation, etc.
    return res
```

result value of the call. if no computed result to return: `return None`

☝ parameters and all of this bloc only exist *in* the block and *during* the function call *("black box")*

Function call

```
r = fctname(3,i+2,2*i)
```

one argument per parameter

retrieve returned result (if necessary)

Operations on dictionaries

`d[key]=value` `d.clear()`

`d[key]`→*value* `del d[clé]`

`d.update(d2)` } *update/add associations*

`d.keys()`
`d.values()` } *views on keys, values associations*
`d.items()`

`d.pop(clé)`

Operations on sets

Operators:

- `|` → union (vertical bar char)
- `&` → intersection
- `-  ^` → difference/symetric diff
- `<  <=  >  >=` → inclusion relations

`s.update(s2)`

`s.add(key)` `s.remove(key)`

`s.discard(key)`

Files

storing data on disk, and reading it back

```
f = open("fil.txt","w",encoding="utf8")
```

file variable for operations

name of file on disk (+path…)

opening **mode**
- ▫ `'r'` read
- ▫ `'w'` write
- ▫ `'a'` append…

encoding of chars for text files: `utf8` `ascii` `latin1` …

cf functions in modules `os` and `os.path`

writing

```
f.write("hello")
```

☝ *text file → read /write only* ***strings****, convert from/to required type.*

reading

empty string if end of file

```
s = f.read(4)
```

if char count not specified, read whole file

read next line

```
s = f.readline()
```

```
f.close()
```

☝ don't forget to close file after use

Pythonic automatic close : `with open(…) as f:`

very common: iterative loop reading lines of a text file

```
for line in f :
    # line processing block
```

Strings formating

formating directives values to format

```
"model {} {} {}".format(x,y,r) → str
"{selection:formating!conversion}"
```

▫ **Selection** :
```
2
x
0.nom
4[key]
0[2]
```

Examples:

```
"{:+2.3f}".format(45.7273)
→'+45.727'
"{1:>10s}".format(8,"toto")
→'      toto'
"{!r}".format("I'm")
→'"I\'m"'
```

▫ **Formating** :

fillchar alignment sign minwidth.precision~maxwidth type

`< > ^ =`   `+ - space`   `0` at start for filling with 0

integer: `b` binary, `c` char, `d` decimal (default), `o` octal, `x` or `X` hexa…

float: `e` or `E` exponential, `f` or `F` fixed point, `g` or `G` appropriate (default), `%` percent

string : `s` …

▫ **Conversion** : `s` (readable text) or `r` (litteral representation)

sys Variables

argv	Command line args
builtin_module_names	Linked C modules
byteorder	Native byte order
check_interval	Signal check frequency
exec_prefix	Root directory
executable	Name of executable
exitfunc	Exit function name
modules	Loaded modules
path	Search path
platform	Current platform
stdin,stdout,stderr	File objects for I/O
version_info	Python version info
winver	Version number

sys.argv for python foo.py bar -c qux --h

sys.argv[0]	foo.py
sys.argv[1]	bar
sys.argv[2]	-c
sys.argv[3]	qux
sys.argv[4]	--h

os Variables

altsep	Alternative separator
curdir	Current dir string
defpath	Default search path
devnull	Path of null device
extsep	Extension separator
linesep	Line separator
name	Name of OS
pardir	Parent dir string
pathsep	Path separator
sep	Path separator

NOTE OS name can be `posix`, `nt`, `mac`, `os2`, `ce`, `java` or `riscos`

Class Special Methods

__new__(cls)	__lt__(self,other)
__init__(self,args)	__le__(self,other)
__del__(self)	__gt__(self,other)
__repr__(self)	__ge__(self,other)
__str__(self)	__eq__(self,other)
__cmp__(self,other)	__ne__(self,other)
__index__(self)	__nonzero__(self)
__hash__(self)	__call__(self,args,kwargs)
__getattr__(self,name)	__setattr__(self,name,attr)
__getattribute__(self,name)	__delattr__(self,name)

String Methods

capitalize() *	lstrip()
center(width)	partition(sep)
count(sub,start,end)	replace(old,new)
decode()	rfind(sub,start,end)
encode()	rindex(sub,start,end)
endswith(sub)	rjust(width)
expandtabs()	rpartition(sep)
find(sub,start,end)	rsplit(sep)
index(sub,start,end)	rstrip()
isalnum() *	split(sep)
isalpha() *	splitlines()
isdigit() *	startswith(sub)
islower() *	strip()
isspace() *	swapcase() *
istitle() *	title() *
isupper() *	translate(table)
join()	upper() *
ljust(width)	zfill(width)
lower() *	

NOTE Methods marked * are locale dependant for 8-bit strings

List Methods

append(item)	pop(position)
count(item)	remove(item)
extend(list)	reverse()
index(item)	sort()
insert(position,item)	

Indexes and Slices (of a=[0,1,2,3,4,5])

len(a)	6
a[0]	0
a[5]	5
a[-1]	5
a[-2]	4
a[1:]	[1,2,3,4,5]
a[:5]	[0,1,2,3,4]
a[:-2]	[0,1,2,3]
a[1:3]	[1,2]
a[1:-1]	[1,2,3,4]

Datetime Methods

today()	fromordinal(ordinal)
now(timezoneinfo)	combine(date,time)
utcnow()	strptime(date, format)
fromtimestamp(timestamp)	utcfromtimestamp(timestamp)

Time Methods

replace()	utcoffset()
isoformat()	dst()
__str__()	tzname()
strftime(formato)	

Date Formatting (strfime and strptime)

%a	Abbreviated weekday (Sun)
%A	Weekday (Sunday)
%b	Abbreviated month name (Jan)
%B	Month name (January)
%c	Date and Time
%d	Day (leading zeros) (01 to 31)
%H	24 hour (leading zeros) (00 a 23)
%I	12 hour (leading zeros) (01 a 12)
%j	Day of the year (001 a 366)
%m	Month (01 a 12)
%M	Minute (00 a 59)
%p	AM or PM
%S	Second (00 a 61)[1]
%U	Week number[2] (00 a 53)
%w	Weekday[3] (0 a 6)
%W	Week number[4] (00 a 53)
%x	Date
%X	Time
%y	Year without century (00 a 99)
%Y	Year (2009)
%Z	Time zone (GMT)
%%	A literal "%" character (%)

1 -- Not a mistake. Range takes account of leap seconds

2 -- Sunday as start of week.

3 -- 0 is Sunday, 6 is Saturday.

4 -- Monday as start of week.

File Methods

close()	readlines(size)
flush()	seek(offset)
fileno()	tell()
isatty()	truncate(size)
next()	write(string)
read(size)	writelines(list)
readline(size)	

Created with rst2pdf:	http://rst2pdf.googlecode.com
Droid Typeface:	http://www.droidfonts.com
Homepage:	http://netmanagers.com.ar/machete