Python for Beginners – Cheat Sheet

Data types and Collections

integer	10
float	3.14
boolean	True/False
string	'abcde'
list	[1, 2, 3, 4, 5]
tuple	(1, 2, 'a', 'b')
set	{'a', 'b', 'c'}
dictionary	{'a':1, 'b':2}

Numerical Operators

+	addition
-	subtraction
*****	multiplication
/	division
******	exponent
%	modulus
//	floor division

Comparison Operators

<	less
<=	less or equal
>	greater
>=	greater or equal
==	equal
!=	not equal

Logical Operators

and	logical AND
or	logical OR
not	logical NOT

Operations

Index starts at 0

Strings:

s[i]	i:th item of s
s[-1]	last item of s

Lists:

l = []	define empty list
l[i:j]	slice in range i to j
l[i] = x	replace i with x
l[i:j:k]	slice range i to j, step k

Dictionaries:

d = {}	create empty dictionary
d[i]	retrieve item with key i
d[i] = x	store x to key l
i in d	is key i in dictionary

Membership Operators

in	value in object
not in	value not in object

Conditional Statements

```
if condition:
  <code>

elif condition:
  <code>

else:
  <code>
```

List Methods

l.append(x)	append x to end of list
l.insert(I, x)	insert x at position i
l.remove(x)	remove first occurrence of x
l.reverse()	reverse list in place

Dictionary Methods

d.keys()	returns a list of keys
d.values()	returns a list of values
d.items()	returns a list of (key, value)

String Methods

s.strip()	remove trailing whitespace
s.split(x)	return list, delimiter x
s.join(l)	return string, delimiter s
s.startswith(x)	return True if s starts with x
s.endswith(x)	return True if s ends with x
s.upper()	return copy, uppercase only
s.lower()	return copy, lowercase only

Import from Module

from module import func	import func
from module import func as f	import func as f

Python for Beginners – Cheat Sheet

Built-in Functions

float(x)	convert x to float
int(x)	convert x to integer
str(x)	convert x to string
set(x)	convert x to set
type(x)	returns type of x
len(x)	returns length of x
max(x)	returns maximum of x
min(x)	returns minimum of x
sum(x)	returns sum of values in x
sorted(x)	returns sorted list
round(x, d)	returns x rounded to d
print(x)	print object x

Loops

```
while condition:
  <code>

for var in list:
  <code>
```

Control statements:

break	terminate loop
continue	jump to next iteration
pass	does nothing

String Formatting

```
"Put {}into a {}".format("values", "string")
'Put values into a string'

"Put whitespace after: {:<10}, or before:{:>10}".format("a","b")
'Put whitespace after: a         , or before:         b'

"Put whitespace around:{:^10}.".format("c")
'Put whitespace around:     c     .
```

Regular Expressions

```
import re
```

p = re.compile(pattern)	compile search query
p.search(text)	search for all matches
p.sub(sub, text)	substitute match with sub

.	any one character
*****	repeat previous 0 or more times
+	repeat previous 1 or more times
?	repeat previous 0 or 1 times
\d	any digit
\s	any whitespace
[abc]	any character in this set {a, b, c}
[^abc]	any character *not* in this set
[a-z]	any letter between a and z
a|b	a or b

Reading and Writing Files

```
fh = open(<path>,'r')
for line in fh:
   <code>
fh.close()

out = open(<path>,'w')
out.write(<str>)
out.close()
```

Functions

```
def Name(param1, param2 = val):
   <code>
   #param2 optional, default: val
   return <data>
```

sys.argv

import sys	import module
sys.argv[0]	name of script
sys.argv[1]	first cmd line arg

Python 3 Beginner's Reference Cheat Sheet

Alvaro Sebastian
http://www.sixthresearcher.com

Main data types

boolean = *True* / *False*
integer = 10
float = 10.01
string = "123abc"
list = [value1, value2, …]
dictionary = { key1:value1, key2:value2, …}

Numeric operators

+	addition
-	subtraction
*	multiplication
/	division
**	exponent
%	modulus
//	floor division

Comparison operators

==	equal
!=	different
>	higher
<	lower
>=	higher or equal
<=	lower or equal

Boolean operators

and	logical AND
or	logical OR
not	logical NOT

Special characters

#	coment
\n	new line
\<char>	scape char

String operations

string[i]	retrieves character at position i
string[-1]	retrieves last character
string[i:j]	retrieves characters in range i to j

List operations

list = []	defines an empty list
list[i] = x	stores x with index i
list[i]	retrieves the item with index I
list[-1]	retrieves last item
list[i:j]	retrieves items in the range i to j
del list[i]	removes the item with index i

Dictionary operations

dict = {}	defines an empty dictionary
dict[k] = x	stores x associated to key k
dict[k]	retrieves the item with key k
del dict[k]	removes the item with key k

String methods

string.upper()	converts to uppercase
string.lower()	converts to lowercase
string.count(x)	counts how many times x appears
string.find(x)	position of the x first occurrence
string.replace(x,y)	replaces x for y
string.strip(x)	returns a list of values delimited by x
string.join(L)	returns a string with L values joined by string
string.format(x)	returns a string that includes formatted x

List methods

list.append(x)	adds x to the end of the list
list.extend(L)	appends L to the end of the list
list.insert(i,x)	inserts x at i position
list.remove(x)	removes the first list item whose value is x
list.pop(i)	removes the item at position i and returns its value
list.clear()	removes all items from the list
list.index(x)	returns a list of values delimited by x
list.count(x)	returns a string with list values joined by S
list.sort()	sorts list items
list.reverse()	reverses list elements
list.copy()	returns a copy of the list

Dictionary methods

dict.keys()	returns a list of keys
dict.values()	returns a list of values
dict.items()	returns a list of pairs (key,value)
dict.get(k)	returns the value associtated to the key k
dict.pop()	removes the item associated to the key and returns its value
dict.update(D)	adds keys-values (D) to dictionary
dict.clear()	removes all keys-values from the dictionary
dict.copy()	returns a copy of the dictionary

Legend: x,y stand for any kind of data values, **s** for a string, **n** for a number, **L** for a list where **i,j** are list indexes, **D** stands for a dictionary and **k** is a dictionary key.

Python 3 Beginner's Reference Cheat Sheet

Alvaro Sebastian
http://www.sixthresearcher.com

Built-in functions

print(x, sep='y')	prints x objects separated by y
input(s)	prints s and waits for an input that will be returned
len(x)	returns the length of x (s, L or D)
min(L)	returns the minimum value in L
max(L)	returns the maximum value in L
sum(L)	returns the sum of the values in L
range(n1,n2,n)	returns a sequence of numbers from n1 to n2 in steps of n
abs(n)	returns the absolute value of n
round(n1,n)	returns the n1 number rounded to n digits
type(x)	returns the type of x (string, float, list, dict …)
str(x)	converts x to string
list(x)	converts x to a list
int(x)	converts x to a integer number
float(x)	converts x to a float number
help(s)	prints help about x
map(function, L)	Applies function to values in L

Conditional statements

```
if <condition> :
    <code>
else if <condition> :
    <code>
...
else:
    <code>

if <value> in <list>:
```

Data validation

```
try:
    <code>
except <error>:
    <code>
else:
    <code>
```

Working with files and folders

```
import os
os.getcwd()
os.makedirs(<path>)
os.chdir(<path>)
os.listdir(<path>)
```

Loops

```
while <condition>:
    <code>

for <variable> in <list>:
    <code>

for <variable> in
range(start,stop,step):
    <code>

for key, value in
dict.items():
    <code>
```

Loop control statements

break	finishes loop execution
continue	jumps to next iteration
pass	does nothing

Running external programs

```
import os
os.system(<command>)
```

Functions

```
def function(<params>):
    <code>
    return <data>
```

Modules

```
import module
module.function()

from module import *
function()
```

Reading and writing files

```
f = open(<path>,'r')
f.read(<size>)
f.readline(<size>)
f.close()

f = open(<path>,'r')
for line in f:
    <code>
f.close()

f = open(<path>,'w')
f.write(<str>)
f.close()
```

Legend: x,y stand for any kind of data values, **s** for a string, **n** for a number, **L** for a list where **i,j** are list indexes, **D** stands for a dictionary and **k** is a dictionary key.

Python Cheat Sheet

JUST THE BASICS

Created By: Arianne Colton and Sean Chen

General

- Python is case sensitive
- Python index starts from 0
- Python uses whitespace (tabs or spaces) to indent code instead of using braces.

HELP

Help Home Page	`help()`
Function Help	`help(str.replace)`
Module Help	`help(re)`

MODULE (AKA LIBRARY)

Python module is simply a '.py' file

List Module Contents	`dir(module1)`
Load Module	`import module1` *
Call Function from Module	`module1.func1()`

* import statement creates a new namespace and executes all the statements in the associated *.py* file within that namespace. If you want to load the module's content into current namespace, use `'from module1 import *'`

Scalar Types

Check data type : `type(variable)`

SIX COMMONLY USED DATA TYPES

1. **int/long*** - Large int automatically converts to long
2. **float*** - 64 bits, there is no 'double' type
3. **bool*** - True or False
4. **str*** - ASCII valued in Python 2.x and Unicode in Python 3
   - String can be in single/double/triple quotes
   - String is a sequence of characters, thus can be treated like other sequences
   - Special character can be done via `\` or preface with `r`

```
str1 = r'this\f?ff'
```

   - String formatting can be done in a number of ways

```
template = '%.2f %s haha $%d';
str1 = template % (4.88, 'hola', 2)
```

Scalar Types

* str(), bool(), int() and float() are also explicit type cast functions.

5. **NoneType(None)** - Python 'null' value (ONLY one instance of None object exists)
   - None is not a reserved keyword but rather a unique instance of 'NoneType'
   - None is common default value for optional function arguments :

```
def func1(a, b, c = None)
```

   - Common usage of None :

```
if variable is None :
```

6. **datetime** - built-in python 'datetime' module provides 'datetime', 'date', 'time' types.
   - 'datetime' combines information stored in 'date' and 'time'

Create datetime from String	`dt1 = datetime.strptime('20091031', '%Y%m%d')`
Get 'date' object	`dt1.date()`
Get 'time' object	`dt1.time()`
Format datetime to String	`dt1.strftime('%m/%d/%Y %H:%M')`
Change Field Value	`dt2 = dt1.replace(minute = 0, second = 30)`
Get Difference	`diff = dt1 - dt2` # diff is a 'datetime.timedelta' object

Note : Most objects in Python are mutable except for 'strings' and 'tuples'

Data Structures

Note : All non-Get function call i.e. `list1.sort()` examples below are in-place (without creating a new object) operations unless noted otherwise.

TUPLE

One dimensional, fixed-length, **immutable** sequence of Python objects of ANY type.

Create Tuple	`tup1 = 4, 5, 6` or `tup1 = (6,7,8)`
Create Nested Tuple	`tup1 = (4,5,6), (7,8)`
Convert Sequence or Iterator to Tuple	`tuple([1, 0, 2])`
Concatenate Tuples	`tup1 + tup2`
Unpack Tuple	`a, b, c = tup1`

Application of Tuple

Swap variables	`b, a = a, b`

Data Structures

LIST

One dimensional, variable length, **mutable** (i.e. contents can be modified) sequence of Python objects of ANY type.

Create List	`list1 = [1, 'a', 3]` or `list1 = list(tup1)`
Concatenate Lists*	`list1 + list2` or `list1.extend(list2)`
Append to End of List	`list1.append('b')`
Insert to Specific Position	`list1.insert(posIdx, 'b')` **
Inverse of Insert	`valueAtIdx = list1.pop(posIdx)`
Remove First Value from List	`list1.remove('a')`
Check Membership	`3 in list1 => True` ***
Sort List	`list1.sort()`
Sort with User-Supplied Function	`list1.sort(key = len)` # sort by length

* List concatenation using '+' is expensive since a new list must be created and objects copied over. Thus, `extend()` is preferable.

** Insert is computationally expensive compared with append.

*** Checking that a list contains a value is lot slower than dicts and sets as Python makes a linear scan where others (based on hash tables) in constant time.

Built-in 'bisect module‡

- Implements binary search and insertion into a sorted list
- 'bisect.bisect' finds the location, where 'bisect.insort' actually inserts into that location.

‡ WARNING : bisect module functions do not check whether the list is sorted, doing so would be computationally expensive. Thus, using them in an unsorted list will succeed without error but may lead to incorrect results.

SLICING FOR SEQUENCE TYPES†

† Sequence types include 'str', 'array', 'tuple', 'list', etc.

Notation	`list1[start:stop]`
	`list1[start:stop:step]` (If step is used) §

Note :
- 'start' index is included, but 'stop' index is NOT.
- start/stop can be omitted in which they default to the start/end.

§ Application of 'step' :

Take every other element	`list1[::2]`
Reverse a string	`str1[::-1]`

DICT (HASH MAP)

Create Dict	`dict1 ={'key1' :'value1', 2 :[3, 2]}`
Create Dict from Sequence	`dict(zip(keyList, valueList))`
Get/Set/Insert Element	`dict1['key1']`* `dict1['key1'] = 'newValue'`
Get with Default Value	`dict1.get('key1', defaultValue)` **
Check if Key Exists	`'key1' in dict1`
Delete Element	`del dict1['key1']`
Get Key List	`dict1.keys()` ***
Get Value List	`dict1.values()` ***
Update Values	`dict1.update(dict2)` # dict1 values are replaced by dict2

* 'KeyError' exception if the key does not exist.

** 'get()' by default (aka no 'defaultValue') will return 'None' if the key does not exist.

*** Returns the lists of keys and values in the same order. However, the order is not any particular order, aka it is most likely not sorted.

Valid dict key types

- Keys have to be immutable like scalar types (int, float, string) or tuples (all the objects in the tuple need to be immutable too)
- The technical term here is 'hashability', check whether an object is hashable with the `hash('this is string'), hash([1, 2])` - this would fail.

SET

- A set is an **unordered** collection of UNIQUE elements.
- You can think of them like dicts but keys only.

Create Set	`set([3, 6, 3])` or `{3, 6, 3}`
Test Subset	`set1.issubset(set2)`
Test Superset	`set1.issuperset(set2)`
Test sets have same content	`set1 == set2`

- **Set operations :**

Union(aka 'or')	`set1 | set2`
Intersection (aka 'and')	`set1 & set2`
Difference	`set1 - set2`
Symmetric Difference (aka 'xor')	`set1 ^ set2`

Functions

Python is **pass by reference**, function arguments are passed by reference.

- Basic Form :

```
def func1(posArg1, keywordArg1 =
1, ..):
```

Note :
- Keyword arguments MUST follow positional arguments.
- Python by default is NOT "lazy evaluation", expressions are evaluated immediately.

- Function Call Mechanism :
  1. All functions are local to the module level scope. See 'Module' section.
  2. Internally, arguments are packed into a tuple and dict, function receives a tuple 'args' and dict 'kwargs' and internally unpack.
- Common usage of 'Functions are objects' :

```
def func1(ops = [str.strip, user_
define_func, ..], ..):
    for function in ops:
        value = function(value)
```

RETURN VALUES

- **None** is returned if end of function is reached without encountering a return statement.
- Multiple values return via ONE tuple object

```
return (value1, value2)
value1, value2 = func1(..)
```

ANONYMOUS (AKA LAMBDA) FUNCTIONS

- What is Anonymous function?
  A simple function consisting of a single statement.

```
lambda x : x * 2
# def func1(x) : return x * 2
```

- Application of lambda functions : 'curring' aka deriving new functions from existing ones by partial argument application.

```
ma60 = lambda x : pd.rolling_mean(x,
60)
```

USEFUL FUNCTIONS (FOR DATA STRUCTURES)

1. **Enumerate** returns a sequence (i, value) tuples where i is the index of current item.

```
for i, value in enumerate(collection):
```

   - Application : Create a dict mapping of value of a sequence (assumed to be unique) to their locations in the sequence.
2. **Sorted** returns a new sorted list from any sequence

```
sorted([2, 1, 3]) => [1, 2, 3]
```

   - Application :

```
sorted(set('abc bcd')) => [' ',
'a', 'b', 'c', 'd']
# returns sorted unique characters
```

3. **Zip** pairs up elements of a number of lists, tuples or other sequences to create a list of tuples :

```
zip(seq1, seq2) =>
[('seq1_1', 'seq2_1'), (..), ..]
```

   - Zip can take arbitrary number of sequences. However, the number of elements it produces is determined by the 'shortest' sequence.
   - Application : Simultaneously iterating over multiple sequences :

```
for i, (a, b) in
enumerate(zip(seq1, seq2)):
```

   - Unzip - another way to think about this is converting a list of rows to a list of columns.

```
seq1, seq2 = zip(*zipOutput)
```

4. **Reversed** iterates over the elements of a sequence in reverse order.

```
list(reversed(range(10))) *
```

* reversed() returns the iterator, `list()` makes it a list.

Control and Flow

1. Operators for conditions in 'if else' :

Check if two variables are same object	`var1 is var2`
. . . are different object	`var1 is not var2`
Check if two variables have same value	`var1 == var2`

WARNING : Use 'and', 'or', 'not' operators for compound conditions, not &&, ||, !.

2. Common usage of 'for' operator :

Iterating over a collection (i.e. list or tuple) or an iterator	`for element in iterator :`
. . . If elements are sequences, can be 'unpack'	`for a, b, c in iterator :`

3. 'pass' - no-op statement. Used in blocks where no action is to be taken.
4. Ternary Expression - aka less verbose 'if else'
   - Basic Form :

```
value = true-expr if condition
else false-expr
```

5. No switch/case statement, use if/elif instead.

Object-Oriented Programming

1. **'object'** is the root of all Python types
2. Everything (number, string, function, class, module, etc.) is an object, each object has a 'type'. Object variable is a pointer to its location in memory.
3. All objects are reference-counted.

```
sys.getrefcount(5)  =>  x
a = 5, b = a
# This creates a 'reference' to the object on the
right side of =, thus both a and b point to 5
sys.getrefcount(5)  => x + 2
del(a); sys.getrefcount(5) => x + 1
```

4. **Class** Basic Form :

```
class MyObject(object):
    # 'self' is equivalent of 'this' in Java/C++
    def __init__(self, name):
        self.name = name
    def memberFunc1(self, arg1):
        ..
    @staticmethod
    def classFunc2(arg1):
        ..
obj1 = MyObject('name1')
obj1.memberFunc1('a')
MyObject.classFunc2('b')
```

5. Useful interactive tool :

```
dir(variable1) # list all methods available on
the object
```

Common String Operations

Concatenate List/Tuple with Separator	`', '.join([ 'v1', 'v2', 'v3']) =>  'v1, v2, v3'`
Format String	`string1 = 'My name is {0} {name}'` `newString1 = string1.format('Sean', name = 'Chen')`
Split String	`sep = '-';` `stringList1 = string1.split(sep)`
Get Substring	`start = 1;  string1[start:8]`

String Padding with Zeros	`month = '5';`
	`month.zfill(2) => '05'`
	`month = '12';`
	`month.zfill(2) => '12'`

Exception Handling

1. Basic Form :

```
try:
    ..
except ValueError as e:
    print e
except (TypeError, AnotherError):
    ..
except:
    ..
finally:
    .. # clean up, e.g. close db
```

2. Raise Exception Manually

```
raise AssertionError # assertion failed
raise SystemExit     # request program exit
raise RuntimeError('Error message :
..')
```

List, Set and Dict Comprehansions

Syntactic sugar that makes code easier to read and write

1. **List comprehensions**
   - Concisely form a new list by filtering the elements of a collection and transforming the elements passing the filter in one concise expression.
   - Basic form :

```
[expr for val in collection if condition]
```

   A shortcut for :

```
result = []
for val in collection:
    if condition:
        result.append(expr)
```

   The filter condition can be omitted, leaving only the expression.
2. **Dict Comprehension**
   - Basic form :

```
{key-expr : value-expr for value in
collection if condition}
```

3. **Set Comprehension**
   - Basic form : same as List Comprehension except with curly braces instead of `[]`
4. **Nested list Comprehensions**
   - Basic form :

```
[expr for val in collection for
innerVal in val if condition]
```

Created by Arianne Colton and Sean Chen
data.scientist.info@gmail.com

Based on content from
'Python for Data Analysis' by Wes McKinney

Updated: May 3, 2016

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_list[list][itemOfList]
`>>> my_list2[1][:2]`	

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 — Leading open data science platform powered by Python

spyder — Free IDE that is included with Anaconda

jupyter — 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