Technical University of Lodz
Institute of Electronics

Algorithms and Data Structures

7. Strings and Text Manipulation II

Łódź 2012

Exercise

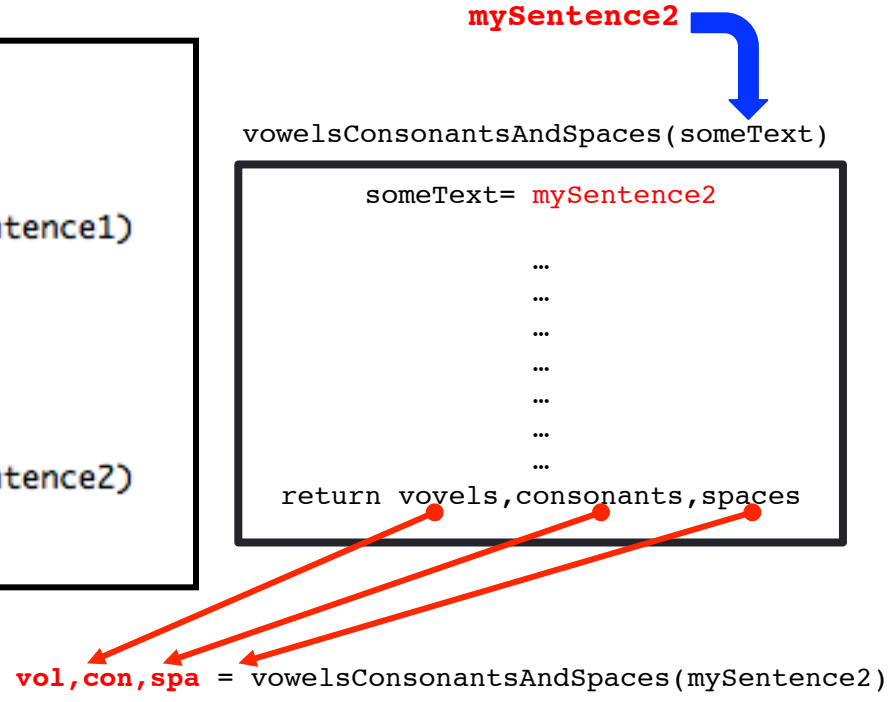
- Type in the program; Save it as **textManipulation_2.py;** Run the script

```
#Exercise 6.10
def vowelsConsonantsAndSpaces(someText):
    """Function returns tnumber of vowels, consonants and spaces in someText"""

    vowel = 0
    consonant = 0
    spaces = 0
    for c in someText:
        if c in 'aoeuyiAOEUYI':
            vowel += 1
        elif c != ' ':
            consonant += 1
        else:
            spaces += 1

    return vowel, consonant, spaces

```

Exercise

- We **have to call** the function! Type in in the same script

```
17
18  #=============== Option 1 ==============
19  mySentence1 = input("Enter some text")
20
21  vol,con,spa = vowelsConsonantsAndSpaces(mySentence1)
22  print vol, con, spa
23
24  #=============== Option 2 ==============
25  mySentence2 = "Ala ma kota"
26
27  vol,con,spa = vowelsConsonantsAndSpaces(mySentence2)
28  print vol, con, spa
29
```




```
>>> print someText
>>> print vowel
>>> print consonant
>>> print spaces
```

- What is the result of print()?
- Why?

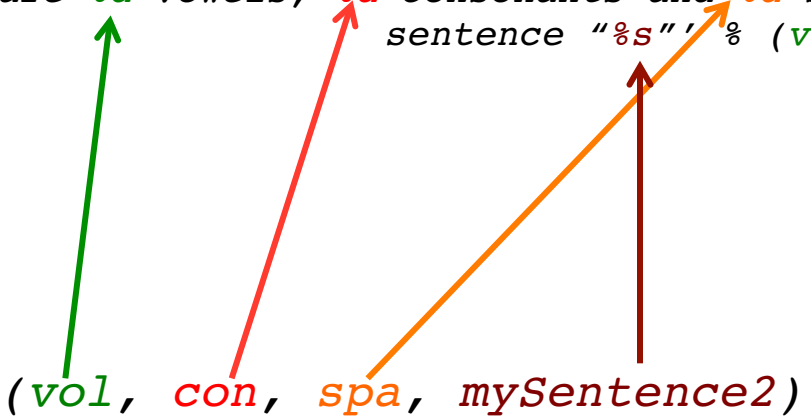
Exercise

How to get a nice information about vowels, consonants and spaces?

There are 5 vowels, 4 consonants and 2 spaces in the sentence "Ala ma kota"

These numbers will be different in each sentence!

```
>>> print 'There are %d vowels, %d consonants and %d spaces in the\
                                  sentence "%s"' % (vol, con, spa, mySentence2)
```



Formatting with %

The format operator % formats values to strings using C conventions

```
>>> s1 = "some numbers:"
>>> x = 1.34
>>> y = 2
>>> t = "%s %f, %d" % (s1,x,y)
>>> print t


>>> y = -2.1
>>> print "%f\n%f" % (x,y)


>>> print "% f\n% f" % (x,y)


>>> print "%4.2f" % x
>>> print "%5.2f" % x
>>> print "%6.2f" % x
```

	CONVERTION CODES
d or i	Signed integer decimal
o	Unsigned octal
u	Unsigned decimal
x	Unsigned hexadecimal (lowercase)
X	Unsigned hexadecimal (uppercase)
e	Floating point exponential format (lowercase)
E	Floating point exponential format (uppercase)
F or f	Floating point decimal format
G or g	Floating point decimal format or exponential
c	Single character
r	Converts objects using repr()
s	Converts object using str()

String Formatting operations (chapter 5.6.2):

http://docs.python.org/2/library/stdtypes.html

Based on Enthought , Inc. www.enthought.com

Escape characters

How to get even a nicer information about vowels, consonants and spaces?

```
There are:
5 vowels,
        4 consonants
                and 2 spaces
                        in the sentence "Ala ma kota"
```

Escape sequence:
- are located inside a string
- begin with **backslash** "\"
- are used to insert non-alphabetic characters into a string

\n	Newline
\t	Tab.
\"	To get " inside a double-quoted string
\'	To get ' inside a single-quoted string
\\	If you need a backslesh itself

```
>>> print 'There are\n%d vowels,\n\t%d consonants\n\t\tand %d spaces\n\t\t\tin
the sentence "%s"' % (vol, con, spa, sentence2)
```

Strings Methods

```
>>> s = 'algorithms and DATA structures'
```

The `s` variable is an **instance** of the **class** String. So it has many **methods** predefined.

```
>>> type(s)
```

```
>>> s.
```

 hit "Tab" key

```
In [48]: s.
s.capitalize  s.format      s.isupper     s.rindex      s.strip
s.center      s.index       s.join        s.rjust       s.swapcase
s.count       s.isalnum     s.ljust       s.rpartition  s.title
s.decode      s.isalpha     s.lower       s.rsplit      s.translate
s.encode      s.isdigit     s.lstrip      s.rstrip      s.upper
s.endswith    s.islower     s.partition   s.split       s.zfill
s.expandtabs  s.isspace     s.replace     s.splitlines
s.find        s.istitle     s.rfind       s.startswith
```

To get more information use "?": `>>> s.capitalize?`

Object Oriented Programming (OOP)

Object oriented programming (OOP) languages (like Python) let you create an entirely new kind of objects using **a *class***. A class is like a template that you use to create new objects.

Think of the class like a *cookie-cutter*, and think of the object as the cookie that is created based on the class. As all the cookies are created from the same cookie-cutter, they all have the **same characteristic**, even though they are all **individual** cookies. When an individual objects is created from a class, it's referred to as **an instance** of that class.*

* Paul Barry & David Griffiths, Head First Programming, O'REILLY Media Inc, 2009

A String Class

```
>>> s = 'algorithms and DATA structures'
>>> s1 = "Biomedical Engineering"
>>> s2 = 'Today is Monday"
>>> s3 = 'Ala ma kota'
```

*s, s1, s2, s3 are instances of class String. Each of them have the same set of **attributes** and **methods**.*

```
>>> s.   ⟵ hit "Tab" key
```

```
s.capitalize  s.format      s.isupper     s.rindex      s.strip
s.center      s.index       s.join        s.rjust       s.swapcase
s.count       s.isalnum     s.ljust       s.rpartition  s.title
s.decode      s.isalpha     s.lower       s.rsplit      s.translate
s.encode      s.isdigit     s.lstrip      s.rstrip      s.upper
s.endswith    s.islower     s.partition   s.split       s.zfill
s.expandtabs  s.isspace     s.replace     s.splitlines
s.find        s.istitle     s.rfind       s.startswith
```

```
>>> s1.   ⟵ hit "Tab" key
```

```
s.capitalize  s.format      s.isupper     s.rindex      s.strip
s.center      s.index       s.join        s.rjust       s.swapcase
s.count       s.isalnum     s.ljust       s.rpartition  s.title
s.decode      s.isalpha     s.lower       s.rsplit      s.translate
s.encode      s.isdigit     s.lstrip      s.rstrip      s.upper
s.endswith    s.islower     s.partition   s.split       s.zfill
s.expandtabs  s.isspace     s.replace     s.splitlines
s.find        s.istitle     s.rfind       s.startswith
```

```
>>> s2.   ⟵ hit "Tab" key
```

```
s.capitalize  s.format      s.isupper     s.rindex      s.strip
s.center      s.index       s.join        s.rjust       s.swapcase
s.count       s.isalnum     s.ljust       s.rpartition  s.title
s.decode      s.isalpha     s.lower       s.rsplit      s.translate
s.encode      s.isdigit     s.lstrip      s.rstrip      s.upper
s.endswith    s.islower     s.partition   s.split       s.zfill
s.expandtabs  s.isspace
s.find        s.istitle
```

```
>>> s3.   ⟵ hit "Tab" key
```

```
s.capitalize  s.format      s.isupper     s.rindex      s.strip
s.center      s.index       s.join        s.rjust       s.swapcase
s.count       s.isalnum     s.ljust       s.rpartition  s.title
s.decode      s.isalpha     s.lower       s.rsplit      s.translate
s.encode      s.isdigit     s.lstrip      s.rstrip      s.upper
s.endswith    s.islower     s.partition   s.split       s.zfill
s.expandtabs  s.isspace     s.replace     s.splitlines
s.find        s.istitle     s.rfind       s.startswith
```

Strings Methods

```
>>> s = 'algorithms and DATA structures'
```

Check and describe each of the methods:

s.title()
s.capitalize()
s.upper()
s.lower()
s.center()
s.replace()

s.count()
s.index()
s.find()
s.rfind()
s.lfind()
s.rjust()
s.ljust()

s.startswith()
s.endswith()
s.isalpha()
s.isupper()
s.islower()
s.isdigit()
s.zfill()

s.replace()
s.strip()
s.lstrip()
s.rstrip()
s.swapcase()
s.split()
s.join()

String Formatting

The `format()` method replaces the replacement fields in the string with the values given as arguments. Any other text in the string remains unchanged.

```
>>> 'We have {} students during {} lecture today'.format(24, 'AandDS')
```

Replacement field: `{<field_name>` Optional `:<format_spec>` `}`

- If `field_name` is an integer, it refers to a position in the positional arguments:

```
>>> "We have {1} students during '{0}' lecture today".format('AandDS',24)
```

- If `field_name` is a name, it refers to a keyword arguments:

```
>>> "Student: {name} {surname}!".format(name='Jan', surname='Nowak')
>>> "Student: {surname} {name}!".format(name='Jan', surname='Nowak')
```

```
some_string.format(*args, **kwargs)
```

Based on Enthought , Inc. www.enthought.com

String Formatting

Fixed point format and a named keyword argument

```
>>> print "[{z:5.0f}]  [{z:5.1f}]  [{z:5.2f}]".format(z=12.3456)
```

Sign options

```
>>> print "{:-f}  {:-f}".format(3.14, -3.14)      #Default
>>> print "{:+f}  {:+f}".format(3.14, -3.14)      #Use +
>>> print "{: f}  {: f}".format(3.14, -3.14)      #Use ' '
```

Alignment and using a numbered positional argument

```
>>> print "[{0:<10s}]  [{0:>10s}]  [{0:*^10s}]".format('PYTHON')
```

Alignment with fill character

```
>>> print "[{0:*<10s}]  [{0:*>10s}]  [{0:*^10s}]".format('PYTHON')
```

Different bases for an intefer (hex, decima, octal, binary)

```
>>> print "{0:X} {0:x} {0:d}  {0:o}  {0:b}".format(255)
```

Based on Enthought , Inc. www.enthought.com

Some String Methods and Functions

Alternative numbers notations

```
>>> 0xFF  # hexadecimal
>>> 255      # decimal
>>> 023   # octal
>>> 19       #decimal
>>> 0b00111  #binary
>>> 7        #decimal
```

Numbers to string

```
>>> str(1.1 + 2.2)
>>> '3.3'
>>> repr(1.1+ 2.2)
>>> '3.3000000000000003'
>>> str(1)
>>> '1'
>>> hex(255)
>>> '0xff'
>>> oct(19)
>>> '023'
>>> bin(7)
>>> '0b111'
```

String to numbers

```
>>> float('23')
>>> 23.0
>>> int('23')
>>> 23
>>> int('FF'16)
>>> 255
>>> int('23',8)
>>> 19
>>> int('0111',2)
>>> 7
```

```
int('value_as_a_string', base)
```

Char ⇔ ASCII convertion

```
>>> ord('A')
>>> 65
>>> chr(65)
>>> 'A'
```

ASCII

ASCII

From Wikipedia, the free encyclopedia
(Redirected from Ascii)

Not to be confused with Windows-1252, also known as "ANSI", or other types of Extended ASCII, often just called "ASCII".

This article is about the character encoding. For other uses, see ASCII (disambiguation).

The **American Standard Code for Information Interchange** (**ASCII**, pronunciation: */ˈæski/ ass-kee*;[3]) is a character-encoding scheme originally based on the English alphabet. ASCII codes represent text in computers, communications equipment, and other devices that use text. Most modern character-encoding schemes are based on ASCII, though they support many additional characters.

ASCII developed from telegraphic codes. Its first commercial use was as a seven-bit teleprinter code promoted by Bell data services. Work on the ASCII standard began on October 6, 1960, with the first meeting of the American Standards Association's (ASA) X3.2 subcommittee. The first edition of the standard was published during 1963,[4][5] a major revision during 1967,[6] and the most recent update during 1986.[7] Compared to earlier telegraph codes, the proposed Bell code and ASCII were both ordered for more convenient sorting (i.e., alphabetization) of lists and added features for devices other than teleprinters.

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

All 128 ASCII characters, including non-printable characters (represented by their abbreviations).
The 95 ASCII graphic characters are numbered from 20_{hex} to $7E_{hex}$ (decimal 32 to 126). The space character is considered a non-printing graphic.[1][2]

ASCII includes definitions for 128 characters: 33 are non-printing control characters (many now obsolete)[8] that affect how text and space is processed[9] and 95 printable characters, including the space (which is considered an invisible graphic[1][2]).

ASCII

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Char ⇔ ASCII convertion

```
>>> ord('A')
>>> 65
>>> chr(65)
>>> 'A'
>>> hex(65)
>>> '0x41'
```

```
>>> ord('a')
>>> 97
>>> chr(97)
>>> 'a'
>>> hex(97)
>>> '0x61'
```

```
>>> ord('G')
>>> 71
>>> chr(71)
>>> 'G'
>>> hex(65)
>>> '0x47'
```

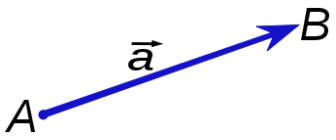
```
>>> ord('g')
>>> 103
>>> chr(103)
>>> 'g'
>>> hex(103)
>>> '0x67'
```

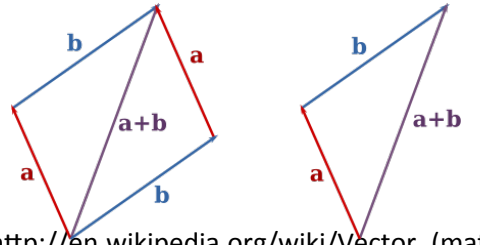
```
>>> ord('a')-ord('A')
>>> 32
>>> ord('g')-ord('G')
>>> 32
>>> ord('z')-ord('Z')
>>> 32
```

Exercises

8.1 Think over how to design following classes:

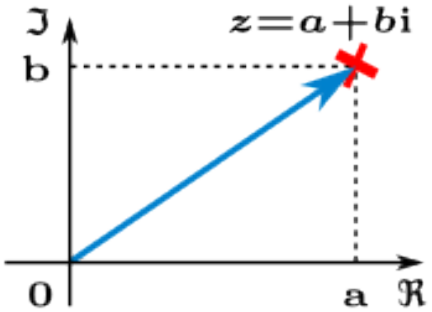
- Student (contains records of each student BE, I sem.)
- Vector





http://en.wikipedia.org/wiki/Vector_(mathematics_and_physics)

- Complex



http://en.wikipedia.org/wiki/Complex_number

What functionality should have each class?

What attributes and methods should contain?

Exercises

8.2 Write a function that prints numbers between 0 and 32 in 4 different notations (decimal, binary, hexadecimal and octal).

8.3 Write a function that has the same functionality as methods below:

- swapcase()
- capitalize()
- upper()
- lower()
- strip()
- lstrip()
- rstrip()
- title()

Summary

1. To change numbers to string use following functions `str(),hex(),bin(),oct()`
2. ASCII is a character-encoding standard to represent text in computers.
3. Use `int('string', base)` function to change sting to integer with given base.
4. Escape characters allows to insert into string non-alphabetic characters.
5. A class is a template of objects that have the same methods and attributes.
6. Class String has many predefined functions to deal with text
7. To format numbers inside text use `format()` method or %

Summary - Python data types

We have already known data types:

```
>>> a = 5 # int
>>> b = 3.14 # float
>>> name = "Ala" # string
```

The data types that we will learn soon :

```
>>> c = [1,2.25,3,'Tom'] # list
>>> d,e,f = (34, 67, 100) # tuple
>>> g = {1:100, 2:200, 3:300} # dictionary
>>> h = np.array([[1,2,3],[4,5,6],[7,8,9]]) # n-d array
>>> i = 6 +7j # complex
>>> k = Student() # own data types – design a new Class
```

Literature

Brian Heinold, Introduction to Programming Using Python, Mount St. Mary's University, 2012 (*http://faculty.msmary.edu/heinold/python.html*).

Brad Dayley, Python Phrasebook: Essential Code and Commands, SAMS Publishing, 2007 (dostępne też tłumaczenie: B. Dayley, Python. Rozmówki, Helion, 2007).

Mark J. Johnson, A Concise Introduction to Programming in Python, CRC Press, 2012.

Paul Barry & David Griffiths, Head First Programming, O'REILLY Media Inc, 2009