Technical University of Lodz
Institute of Electronics

Algorithms and Data Structures

6. Strings and Text Manipulation I

Łódź 2012

Exercise

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

```
1  # string and text manipulation 1
2
3  def countChar(text, myChar):
4      """Functions counts appearence of myChar in the text"""
5      count = 0
6      for c in text:
7          if c == myChar:
8              count +=1
9      print 'Character %s appeares %d time(s) in the string "%s"'%(myChar,count,text)
10
11 def insertTextNTimes(text, pos, insertText, n):
12     """Function insert insertText n times into Text at a position pos"""
13     return text[:pos] + insertText*n + text[pos:]
14
15 def main():
16     sentence = 'Ala ma kota'
17     charToFind = 'a'
18     textToInsert = ' i Ola'
19
20     countChar(sentence, charToFind)
21     print insertTextNTimes(sentence, 3, textToInsert, 4)
22
23 main()
```

Strings

- **String** is a basic type in Python
- **String** is an example of data structure
- **String** is a collection of data organized to efficiently support a particular set of access methods and operations
- **String** is a sequence of characters inside quotation marks
  1. Single quotes (‘)
  2. Double quotes (“)
  3. Triple quotes – for multi-line strings (“””)

```
>>> subject = ‘Algorithms and Data Structures’
>>> name =  “Grzegorz”
>>> surname = “Brzeczyszczykiewicz”
>>> multi_line_string = “””This is a very long string, that is spread
    across many, many lines, for example a description of some newly
    created function “””
```

Basic operations

Concatenation

```
>>> letters = "ab" + "cd"
>>> letters += 'ab'
>>> "Ala" + ' ma ' + "kota!"
```

`s1 + s2` The string `s1` followed by `s2`
`s1 += s2` Shorthand for `s1 = s1 + s2`

Repetition

```
>>> "Hello"*4
>>> "bye! "*7
>>> 10*"-*-?-*"
```

`s*n` The string `s+s+…+s+s,n` times
`n*s` The same as `s*n`

String Accumulators

```
>>> s = 'a'
>>> for i in range(4):
>>>       s += 'b'
```

```
<accumulator> = <string value>
loop:
        <accumulator> += <string to add>
```

Comparition

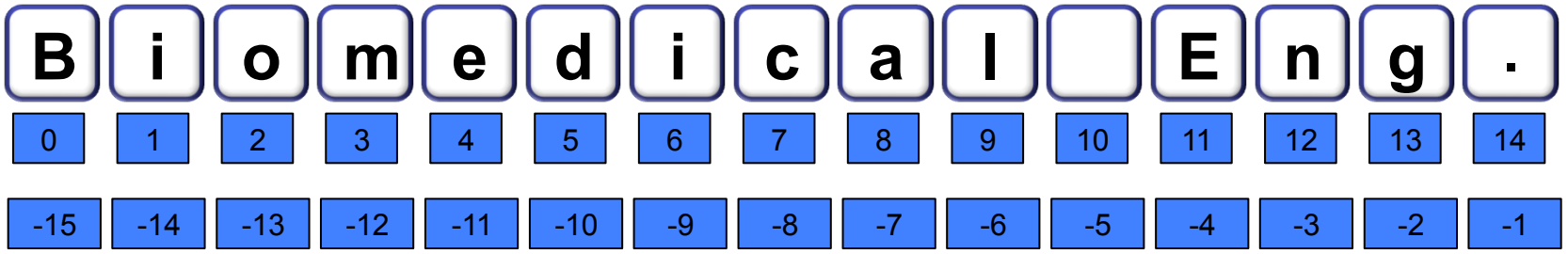
```
>>> s1 = 'ala'; s2 = 'ala'; s3 = 'ola'
>>> s1 == s2
>>> s1 == s3
```

`s1 == s2` returns `True`
`s1 == s3` returns `False`

Indexing

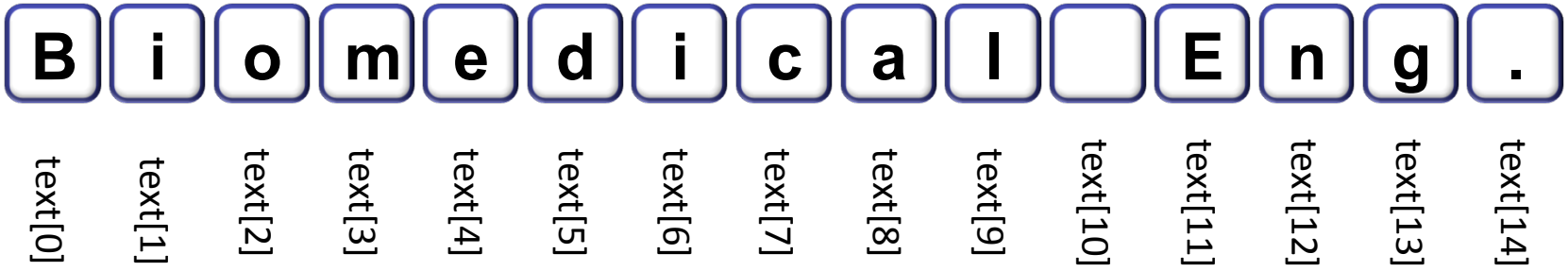
```
>>> text = 'Biomedical Eng.'
```

The `text` variable is stored in a memory like below. Each, individual character is stored in consecutive memory location



Each number along the bottom row is called **index** of the character above it.

Indexing



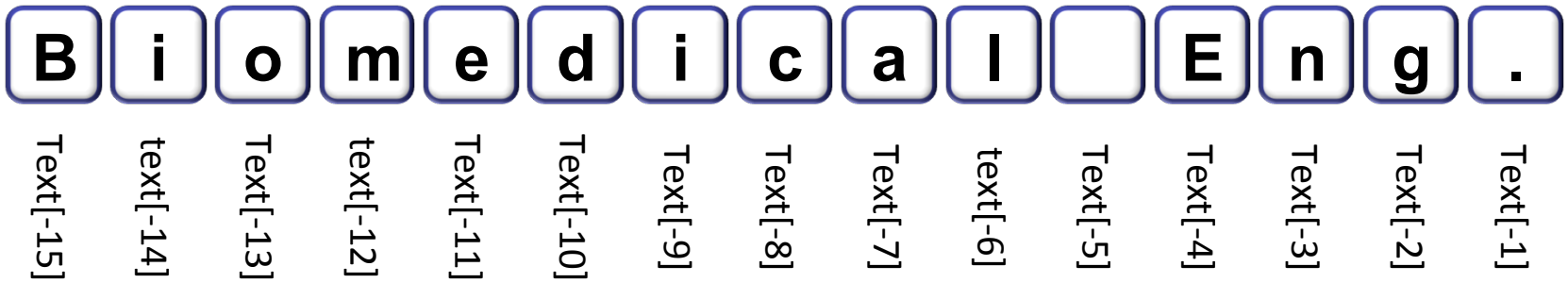
Type in following commands and check the result:

```
>>> text[0] = ?
>>> text[1] = ?
>>> text[5] = ?
>>> text[8] = ?
>>> text[11] = ?
```

`string[i]`	Character at index `i`.

Indexing a string always returns a single character

Indexing



Type in following commands and check the result:

```
>>> text[-1] = ?
>>> text[-5] = ?
>>> text[-7] = ?
>>> text[-11] = ?
>>> text[-15] = ?
```

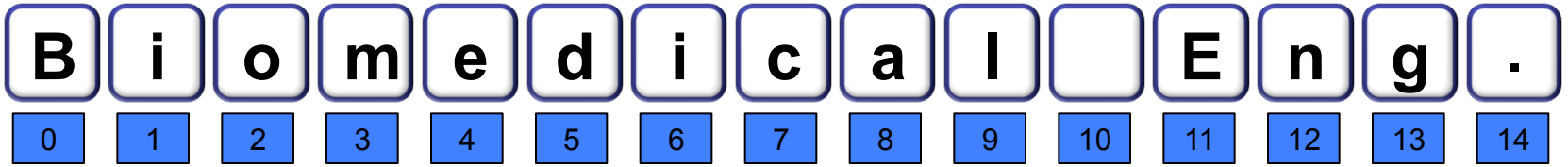
`string[i]` Character at index `i`.

Indexing a string always returns a single character

Slicing

Accessing a group of consecutive characters

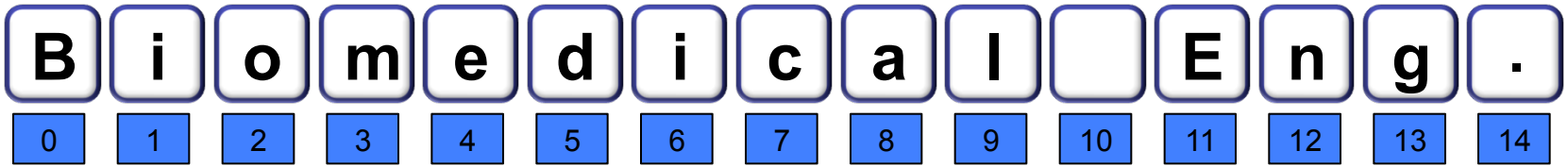
`text[i:j]` Slice from `i` to `j-1`




```
>>> text[0:3] = ?
>>> text[3:6] = ?
>>> text[11:14] = ?
>>> text[0:15] = ?
```

Does not include `text[j]`!

Slicing




```
>>> text[:3] = ?
>>> text[3:] = ?
>>> text[:] = ?
```

`text[:j]`	Slice from beginning to `j-1`
`text[i:]`	Slice from `i` to the end
`text[:]`	Full slice, which creates a **copy** of `s`

Optional third parameter `k` specifies a stepsize other 1 (see the `range()` function)

```
>>> text[2:12:2] = ?
>>> text[3::3] = ?
>>> text[::2] = ?
>>> text[::-1] = ?
>>> text[9:2:-1] = ?
>>> text[:5:-1] = ?
>>> text[5::-1] = ?
```

If `k` is negative, the index `i` will still is the starting point. If `i` is omitted with `k<0`, the beginning is considered the right end (`index-1`), and if `j` is omitted with a `k<0`, the end is at the left (index `0`)

In and Not In

To test whether or not a characters in `x` are a substring of `s`. Substring means that the characters of `x` occur as a consecutive slice of `s`

`x in s`	`True` if `x` is a substring of `s`; otherwise `False`
`x not in s`	`True` if `x` is not a substring of `s`; otherwise `True`

```
>>> s = 'ala ma kota'
```

```
>>> 'a' in s = ?
>>> 'ala' in s = ?
>>> 'a ma k' in s = ?
>>> 'alama' in s = ?
>>> 'kot' in s = ?
```

```
>>> 'a' not in s = ?
>>> 'ala' not in s = ?
>>> ' ma ' not in s = ?
>>> 'alama' not in s = ?
>>> 'ola' not in s = ?
```

Length

Length and type of the string

```
>>> len(subject)
>>> len(text)
>>> type(name)
>>> type(student1)
```

`len(string)`

`type(object)`

What character is at the index 15?

```
>>> text[15]
```

```
----> 1 text[15]
IndexError: string index out of range
```

The "type" of the error that occurred

Short message about why it occurred

The first character of text is `text[0]`, not `text[1]`. Remember that in programming, counting usually starts at `0`, not 1.

Changing characters

Suppose we have a string

```
>>> day = 'Todai is Monday'
```

We want to repair spelling mistake

```
>>> day[4] = 'y'
```

Python strings are immutable

```
----> 1 day[4]='y'
TypeError:        'str' object does not support item assignment
```

The "type" of the error that occurred

Short message about why it occurred

We have to create a string and assign it to `day`

```
>>> day = day[:4] + 'y' + day[5:]
>>> print day
```

Looping

If we want to scan through a string one character at a time we can use a for loop

```
>>> day = 'Today is Monday'
```

```
for <variable> in <string>: # loop over each character
        <body>
```

The easiest way is as follows

(without holding the current location in the string)

```
>>> for c in day:
        print c
```

Use comma at the end of `print()` command. What was changed?

```
>>> for c in day:
        print c,
```

Task: create string like below:

```
>>> T * o * d * a * y *   * i * s *   * M * o * n * d * a * y *
```

Looping

If we want to scan through a string with information about current location

```
>>> day = 'Today is Monday'
```

Let's recall the for loop syntax

```
for <variable> in <sequence>:
        <body>
```

```
>>> for i in range(5):
        print i,

>>> 0 1 2 3 4
```

How define a scope of a `range()` function?

```
>>> for i in range(???):
        print i,
```

The scope of a `range()` is the length of a string

```
>>> for i in range(len(day)):
        print i,

>>> 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
```

How to get a single character form the string?

```
>>> for i in range(len(day)):
        print day[i],

>>> T o d a y   i s   M o n d a y
```

Task: create strings (below, and on the right):

```
>>> i s  M o n d a y
```

```
0 -> T
1 -> o
2 -> t
3 -> a
4 -> y
5 ->
6 -> i
7 -> s
8 ->
9 -> M
10 -> o
11 -> n
12 -> d
13 -> a
14 -> y
```

Exercise

6.0 Fill in the red gaps in the body of the function `deleteChar(text, myChar)`

```
1   # string and text manipulation 1 - exercises
2
3   def deleteChar(text, myChar):
4       """Function deletes first appearance of a given character"""
5       for i in range(len(text)):
6           if [          ] == myChar:
7               text1 = text[ ] + text[     ]
8               break #comment this line, what is the difference?
9           else:
10              text1 = text
11      return text1
```

Exercises

6.1 Write a Python script to print a patterns like the below
(Hint: use for loop and string repetition, use 10 stars).

A)

B)

Exercises

6.2 Determine the values of these expressions:

```
>>> subject = "Algorithms and Data Structures"
```

```
subject[0]
subject[5:]
subject[:5]
subject[::2]
```

```
subject[1:5]
subject[:-4]
subject[-15:]
subject[-20:20]
```

6.3 Crate Python expressions using variable `subject` that have these values:

```
>>> subject = "Algorithms and Data Structures"
```

```
Algorithms
Data
Structures
Alg
```

```
serutcurtS ataD dna smhtiroglA
AotsnDatcr #every third
suust amig #every third negative dir.
smhtiroglA #algorithms neg.
```

Exercises

6.4 Write a script **email.py** that builds e-mail address for a student. Ask the user for booth first and second name and surname. Generate e-mail address with the use of pattern: initials@p.lodz.pl, e.g. mmk@p.lodz.pl

6.5 Write a script `stringAccumlator1.py` that accumulates characters until 'q' appear

6.6 Write a script `stringAccumlator2.py` that accumulates characters until string 'quit' appear

6.7 Write a script *myStringFunctions.py*. In the script write a function `stringAccumlator(someText)` that accumulates every 2nd (or 3rd) character from `someText`

Exercises

6.8 In the script *myStringFunctions.py* write a function `stringReverse(someText)` that returns reversed `someText`

6.9 In the script *myStringFunctions.py* write a function `getSpaces(someText)` that returns number of spaces and index of the last space

6.10 In the script *myStringFunctions.py* write a function `vowelsConsonantsAndSpaces(someText)` that returns number of vowels, consonants and spaces in the `someText`

6.11 In the script *myStringFunctions.py* create a function `replaceSpaces (someText, newChar)` that replaces the spaces with the `newChar` given as an argument

Exercises

6.12 In the script *myStringFunctions.py* create a function `replaceCharacter(someText, charToReplace, newChar)` that replaces the specified char (`charToReplace`) in `someText` with the `newChar` given as an argument

6.13 In the script *myStringFunctions.py* create a function `deleteReplaceTripleCharacter(someText, charToChange, flag)` that depends on `flag` value replaces/delete/triples given `charToChange`. The help of the function is as follows:

```
""" Function deletes, replace with space or triple given letter (character)
depends on flag value:
          d -> delete
          s -> replace with space
          t -> triple
Usage:
1)deleteReplaceTripleCharacter('Biomedical','m','d') -> 'Bioedical'
2)deleteReplaceTripleCharacter('Biomedical','m','s') -> 'Bio edical'
3)deleteReplaceTripleCharacter('Biomedical','m','t') -> 'Biommmedical'
"""
```

Summary

1. String is a sequence of characters inside quotation marks
2. Strings can be added, repeated, accumulated and compared
3. A indexing method is used to get individual character from a strong
4. Python strings can have positive and negative indexes
5. To get several characters from a string we use slicing
6. Boolean values are returned by In and Not In operators
7. To change a character inside string we have to create a new
8. To scan through a string we use for loop

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.

12-11-10 00:14