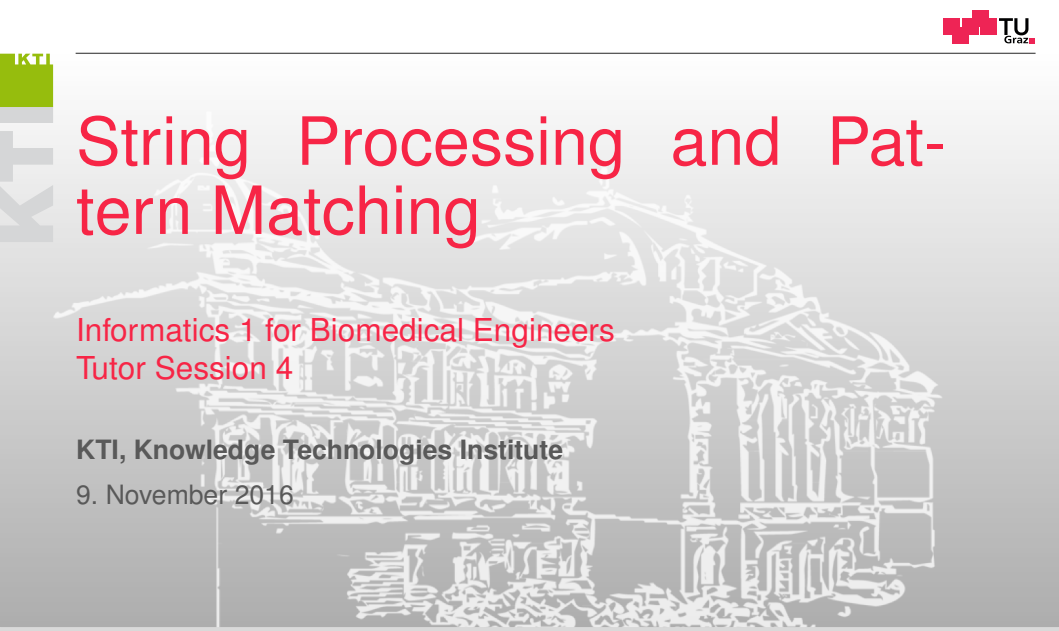
String Processing and Pattern Matching



Informatics 1 for Biomedical Engineers
Tutor Session 4

KTI, Knowledge Technologies Institute

9. November 2016

Today's Topics

- Strings in python
- Pattern Matching
- Regular Expressions

Student Goals

- Learn about the characteristics of strings in python
- Learn how to process strings: find, join, replace,. . .
- Understand the basics of regular expressions and how to use them

Data types revisited

- Logic: `boolean`
- Numeric types: `int`, `float`
- Sequences: `list`, `tuple`
- Text Sequence: `str`
- ...

Strings in python

- String variable

```
1  a = "Good morning"
2  b = 'Good evening'
```

- Multi-line string variable

```
1  a = '''Good morning,
2  How are you feeling today?'''
```

Special characters

- How to define a string with line breaks/characters that break the syntax?
- → Escape characters

```
1  print('Hi,\n\'how\' are you?')
```

Output:

```
1  Hi,
2  'how' are you?
```

Important Escape Characters[1]

Special Character	Python Encoding
line break	\ n
tab	\ t
\	\\
‘ , “	\‘ , \ “

Easy solution for string containing ‘ or “:

```
1 string = 'Hi, "how" are you?'
```

[1] https://docs.python.org/3/reference/lexical_analysis.html#string-and-bytes-literals

Working with Strings

- Concatenate, Format
- Substrings
- Contains
- Find
- Replace, Strip
- Split
- Join

Strings are immutable, always remember to store results in a (new) variable.

Working with Strings – Concat and Format

- Concatenate from separate parts
- Add variable values

```
1  # Concatenation
2  a = 'Part␣1␣' + 'part␣2.'
3
4  # Format with variable
5  field_of_study = 'Field␣of␣study:␣{}'.format('Biomedical␣Engineering
6  bpm = 'Blood␣pressure:␣systolic␣{0}␣mmHg,␣diastolic␣{1}␣mmHg'
7  bpm_formatted = bpm.format(118, 70)
8
9  # convert to lower case
10 lower_case = a.lower()
```

Working with Strings – Substrings

- strings are sequences of characters
- define a start and end offset using square brackets: `[0:2]`
- important: start at index `0`, end at `len(string) - 1`
- leave out indices for entire start/end: `[:]`
- you can also count from the back: `[:-2]`

```
1  a = 'Hello World.'
2  substring1 = a[2:4] # ll
3  substring2 = a[:] # Hello World.
4  substring3 = a[2:-2] # llo Worl
```

Working with Strings – Contains

- check if one string contains another
- returns boolean value `True` or `False`

```
1  'Hel' in 'Hello World' # True
2  'hel' in 'Hello World' # False
```

Working with Strings – Find

- get the index at which a string occurs in another one
- returns -1 if the searched string was not found
- only returns the first occurrence!

```
1  a = 'Hello World'
2  find1 = a.find('l') # 2
3  find2 = a.find('x') # -1
4  find3 = a.find(' Wo') # 5
```

Working with Strings – Replace

- replace all occurrences of one string in another one

```
1  a = 'Hello World.'
2  replace1 = a.replace(' ', ';') # Hello;World.
3  replace2 = a.replace('l', ';') # He;;o Wor;d.
4  replace3 = a.replace('llo', 'x') # Hex World.
```

Working with Strings – Strip

- remove whitespace characters at the beginning/end of a string

```
1 a = ' Hello World. '
2 strip = a.strip() # 'Hello World.'
```

Working with Strings – Split

- break a string at the occurrence of a specified string
- returns a sequence of the parts, without the split characters
- always returns a string, even when the input is a number

```
1  a = 'Hello World.'
2  split1 = a.split(' ') # ['Hello', 'World.']
3  split2 = a.split('lo') # ['Hel', ' World.']
4  split3 = a.split('.') # ['Hello World', '']
```

Working with Strings – Join

- combine a sequence to a single string
- using a specified separator string

```
1  sequence = ['a', 'b', 'c']
2  joined1 = '␣'.join(sequence) # a b c
3
4  # join a sequence of characters, i.e. a string
5  separator = ','
6  joined2 = separator.join('Hello␣World.') # H,e,l,l,o, ,W,o,r,l,d,.
```

Regular Expressions – Introduction

- What are regular expressions and what are they used for?
  - search occurrences of a given pattern
  - for search/replace implementation
- How do they work?
  - define the pattern you are looking for
  - e.g. “The letter A followed by two lower-case characters and one digit”
  - match a string/text based on this pattern

Regular Expressions – Sample Expressions[2]

- digits: `[0-9]`, `\d`
- whitespace: `\s`
- wildcard (arbitrary character): `.`
- Encoding for “The letter A followed by two lower-case characters and one digit”:
  `A[a-z][a-z]\d`

[2] `https://docs.python.org/3/library/re.html`

Regular Expressions – Email Addresses

The patterns can be arbitrarily complex.
Example: Email validation according to RFC standard

```
1  (?:[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*
2  |"(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09
3  \x0b\x0c\x0e-\x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)
4  +[a-z0-9](?:[a-z0-9-]*[a-z0-9])?|\[(?:(?:25[0-5]|2[0-4][0-9]|[01]?
5  [0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?|[a-z0-9-]*
6  [a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21-\x5a\x53-\x7f]|\\
7  [\x01-\x09\x0b\x0c\x0e-\x7f])+)\])
```

Regular Expressions – Groups

- Which pattern was matched by which part of the text? How to retrieve and address that part?
- $\rightarrow$ use groups which are filled with the matching result
- defined with `(<expression>)`
- E.g. `([A-Z][a-z])ll` for “Hello World”:
  group 0 is “He”
- also used to describe frequency of partial patterns, e.g. `(ab)+|(cde)`

Regular Expressions in python

- using the regular expressions module[3]
- general usage:
  - `re.search(<pattern>, <string>)`
    returns a match object which contains the whole match and groups
  - `re.findall(<pattern>, <string>)`
    returns a list of the matched strings

[3] `https://docs.python.org/3/library/re.html`

Regular Expressions in python

```
1  # import the regular expressions module
2  import re
3
4  # re.search(pattern, string)
5  match1 = re.search('l', 'Hello World.')
6  # the entire match is saved as group 0
7  print(match1.group(0)) # l
8
9  # all matches as a list
10 match2 = re.findall('l[a-z]', 'Hello World.')
11 print(match2) # ['ll', 'ld']
```

Regular Expressions in python – Groups

- address groups in the match object using the group index
- the 0th group is the entire matched string

```
1  # find parts of a phone number
2  match3 = re.search('(\+\d+)-(\d+)-(\d+)', '+43-680-1234567')
3  print('entire number ' + match3.group(0)) # entire number +43-680-12
4  print('country code ' + match3.group(1)) # country code +43
5  print('operator code ' + match3.group(2)) # operator code 680
6  print('phone number ' + match3.group(3)) # phone number 1234567
```

Complex example – Hangman Part 1

1. Search for a word list file[4]
2. Optionally clean that list or shorten it
3. Save that word list as a file and load it into python
4. Store all unique words. Hint: use a set[5]

[4] e.g. `https://github.com/dwyl/english-words`

[5] `https://docs.python.org/3/tutorial/datastructures.html#sets`

Complex example – Hangman Part 1

1. Randomly select one of the words. This will be the solution for a round of hangman

```
1          import random
2
3          # get a random sample from the set.
4          # Returns a list, so we choose the first (and only) element
5          selected_word = random.sample(<set>, 1)[0]
```

Complex example – Hangman Part 3

1. Using console input, the player can guess *one* letter at a time
2. Check if the random word contains that letter
3. Track progress:
   - how many guesses does the user have left?
     7 guesses are allowed
   - which characters were guessed correctly?
4. Notify the user when she has won/lost

Complex example – Hangman Step 1

```
1  # Load words into a set
2  word_list = open('words.txt', 'r')
3
4  words = []
5
6  for line in word_list:
7      words.append(line.strip())
8
9  word_list.close()
```

Complex example – Hangman Step 2

See instruction slide

```
1   # Select a random word
2
3   import random
4   selected_word = random.choice(words)
5   # make it lower case so we don't have to worry about cases
6   selected_word = selected_word.lower()
```

Complex example – Hangman Step 3

```
1  # Set up game tracking variables
2
3  correct_letters = [False] * len(selected_word)
4  guessed_letters = []
5  current_word = ['_'] * len(selected_word)
6  wrong_letters_count = 0
```

Complex example – Hangman Step 4

```
1   # Ask player for character input
2   while wrong_letters_count < 7 and False in correct_letters:
3       # ask the user to enter a letter
4       letter = input('Guess a letter: ')
5       letter = letter.lower()
6       # only one letter at a time!
7       while (len(letter) is not 1):
8           letter = input('Guess one letter only: ')
9           letter = letter.lower()
10      # insert [Step 5] here
11
12   # the word has not been guessed but all guesses are used up - the player lo
13  if False in correct_letters:
14      print('Sorry, you lost.')
15  # everything was guessed correctly
16  else:
17      print('Yay, you won!')
```

Complex example – Hangman Step 5

```
1   # has this letter already been guessed before? Only continue if it hasn't
2       if letter not in guessed_letters:
3           # the word contain the player's letter!
4           if letter in selected_word:
5               # insert [Step 6] here
6
7           else:
8               # that means one attempt less available...
9               wrong_letters_count += 1
10              print('Sorry, my word does not contain ' + letter + '. You have '
11                  + str(7 - wrong_letters_count + 1) + ' guesses remaining.')
12
13      # this letter cannot be used anymore
14      guessed_letters.append(letter)
```

Complex example – Hangman Step 6

```
1       # now where do we find the letter?
2       # We're not using find() because it only finds the first occurrence
3       for i in range(len(selected_word)):
4           if selected_word[i] == letter:
5               # this letter was guessed correctly,
6               # so let's set that position to True
7               correct_letters[i] = True
8
9               # show the player the current status by showing all
10              # correct letters in the word
11              current_word[i] = selected_word[i]
12      # the known letters are in a list, so to print them correctly,
13      # they need to be joined to a string
14      print('This␣is␣what␣you␣know␣so␣far:␣' + '␣'.join(current_word))
```

Student task – find a DNA pattern in a DNA string

- Load the DNA of baker's yeast[6] and the genome AXL2[7] from a file. Write a function for this!
- normalise the data: remove all whitespace characters ' '
- Find some shorter base sequences, e.g. "AGT", "GTCC",... Not only the first occurrence, but all of them!
- Does baker's yeast have the AXL2 gene?

[6] `http://www.ncbi.nlm.nih.gov/genbank/samplerecord`

[7] `http://www.yeastgenome.org/locus/S000001402/overview`

(both preprocessed in the folder for this unit)

Student task – Sample Solution Step 1

```
1  # Define a function to read a DNA file and return it as a string
2  def get_dna_sequence(file_name):
3      dna_sequence = ""
4      dna_file = open(file_name, "r")
5
6      for line in dna_file:
7          dna_sequence += line
8
9      dna_sequence = dna_sequence.replace("␣", "")
10     dna_sequence = dna_sequence.replace("\n", "")
11
12     dna_file.close()
13
14     return dna_sequence
```

Student task – Sample Solution Step 2

```
1  # Load the baker's yeast file
2  # check for some simple base sequences
3  yeast_dna = get_dna_sequence("bakers_yeast.txt")
4
5  find_agt = yeast_dna.find("AGT")
6  print(find_agt) # 72
```

Student task – Sample Solution Step 3

```
1   # That only gives us the first occurrence
2   # - let's make it recursive and look at substrings!
3   matches = []
4   index = 0
5   sequence_to_find = "AGT"
6   while True:
7       new_index = yeast_dna[index:].find(sequence_to_find)
8       if new_index is -1:
9           break
10      else:
11          index += new_index + 1
12          matches.append(index)
13  print(matches) # [73, 90, 110, ..., 4926]
14  # analogously for other sequences
```

Student task – Sample Solution Step 4

```
1   # Load the AXL2 genome and check if baker's yeast has it
2
3   genome_dna = get_dna_sequence("AXL2_genomic.txt")
4
5   print(yeast_dna.find(genome_dna)) # True
```