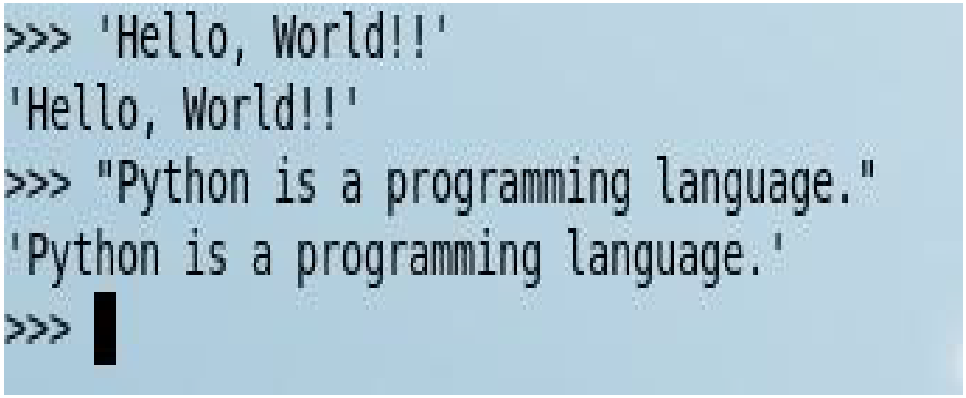
Γλωσσική Τεχνολογία

String Handling – Regular Expressions

Strings - Δήλωση

- Μπορείτε να γράψετε τα δικά σας string περικλείοντας απλά χαρακτήρες και αριθμούς μέσα σε μονά ('...') ή διπλά("...") αυτάκια.



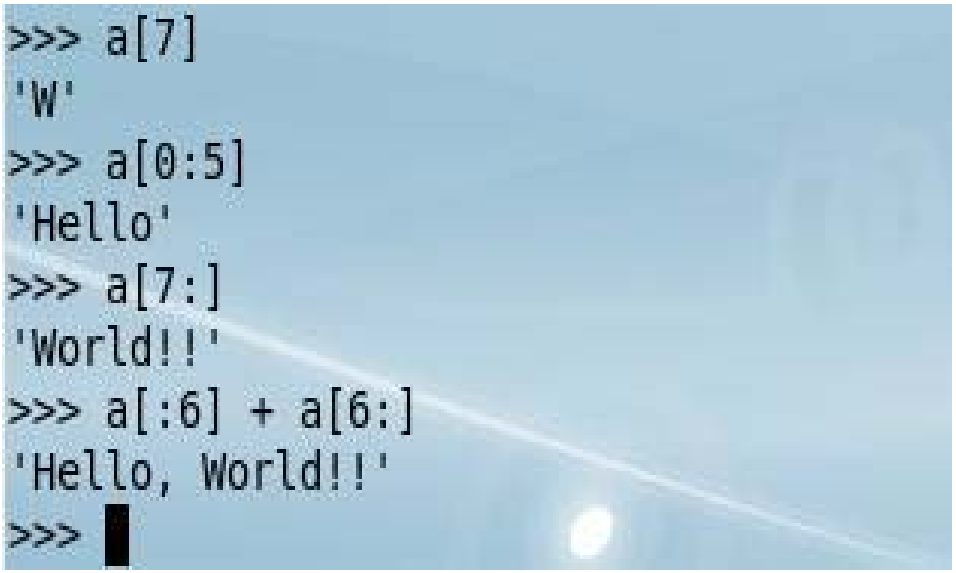
Strings - Τελεστές

- Μπορείτε να κάνετε χρήση των τελεστών “+” και “*” πάνω σε strings.
  - “+” (concatenation)
  - “*” (repetition)

```
>>> a = "Hello, World!!"
>>> b = "Python is a programming language."
>>> a + b
'Hello, World!!Python is a programming language.'
>>> a*3
'Hello, World!!Hello, World!!Hello, World!!'
>>> b + a*3 + b
'Python is a programming language.Hello, World!!Hello, World!!Hello, World!!Pyth
on is a programming language.'
>>>
```

Strings – Δεικτες I

- Μπορείτε να φανταστείτε τα string σαν πίνακες χαρακτήρων της C. Ο πρώτος χαρακτήρας έχει δείκτη 0.



Strings – Δεικτες II

- Μπορείτε ακόμη να χρησιμοποιήσετε και αρνητικούς δείκτες.

Strings – Δεικτες III

- Γενικά η δεικτοδότηση στα strings έχει ως εξής:

```
>>> a = "hello"
>>> for i in range(len(a)):
...     print i, a[i], "\t", -i, a[-i]
...
0 h     0 h
1 e     -1 o
2 l     -2 l
3 l     -3 l
4 o     -4 e
>>>
```

Συναρτήσεις Αναζήτησης I

- str.count(sub[,start[,end]])
  - Επιστρέφει τις μη επικαλυπτόμενες εμφανίσεις του substring sub στο διάστημα (προαιρετικά) [start, end].
- str.find(sub[,start[,end]])
  - Επιστρέφει τον μικρότερο δείκτη του string όπου το substring sub εμφανίζεται (προαιρετικά) στο διάστημα [start, end], διαφορετικά -1.
- str.index(sub[,start[,end]])
  - Όμοια με την find αλλά "πετάει" ValueError όταν δεν βρεθεί το substring sub.

Συναρτήσεις Αναζήτησης II

```
>>> a = "abababababa"
>>> a.count("bab")
2
>>> a.find("bab")
1
>>> a.find("bab",4)
5
>>> a.find("c")
-1
>>> a.index("bab",4)
5
>>> a.index("c")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: substring not found
>>>
```

Συναρτήσεις Ταυτοποίησης I

- str.startswith(prefix[,start[,end]])
  - Επιστρέφει boolean τιμή, ανάλογα με το αν το string ξεκινάει με το πρόθεμα prefix (προαιρετικά) στο διάστημα [start, end].
- str.endswith(suffix[,start[,end]])
  - Επιστρέφει boolean τιμή, ανάλογα με το αν το string τελειώνει με το επίθεμα suffix (προαιρετικά) στο διάστημα [start, end].
- Και στις δύο περιπτώσεις, τα prefix/suffix μπορεί να είναι tuple από προθέματα/επιθέματα, αντίστοιχα.

Συναρτήσεις Ταυτοποίησης II

```
>>> a = "abababababa"
>>> a.startswith("c")
False
>>> a.startswith("ab",4,6)
True
>>> a.endswith(("ab","c"))
False
>>> a.endswith(("ab","ba"),0,len(a))
True
>>>
```

Συναρτήσεις Ταυτοποίησης ΙΙΙ

- str.isalnum()
  - Επιστρέφει boolean ανάλογα με το αν το string περιέχει μόνο αλφαριθμητικούς χαρακτήρες και δεν είναι το κενό string.
- str.isalpha()
  - Επιστρέφει boolean ανάλογα με το αν το string περιέχει μόνο αλφαβητικούς χαρακτήρες και δεν είναι το κενό string.
- str.isdigit()
  - Επιστρέφει boolean ανάλογα με το αν το string περιέχει μόνο αριθμητικούς χαρακτήρες και δεν είναι το κενό string.

Συναρτήσεις Ταυτοποίησης IV

- str.isspace()
  - Επιστρέφει boolean ανάλογα με το αν το string περιέχει μόνο κενούς χαρακτήρες και δεν είναι το κενό string.
- str.islower()
  - Επιστρέφει boolean ανάλογα με το αν το string περιέχει μόνο lowercase χαρακτήρες και δεν είναι το κενό string.
- str.isupper()
  - Επιστρέφει boolean ανάλογα με το αν το string περιέχει μόνο uppercase χαρακτήρες και δεν είναι το κενό string.

Συναρτήσεις Ταυτοποίησης V

```
>>> a = "abababababa"
>>> a.isalnum()
True
>>> a.isalpha()
True
>>> a.isdigit()
False
>>> a.isspace()
False
>>> a.islower()
True
>>> a.isupper()
False
>>> a.istitle()
False
>>>
```

Συναρτήσεις Μορφοποίησης I

- str.lower()
  - Επιστρέφει ένα αντίγραφο του string με όλα του τα στοιχεία μικρά.
- str.upper()
  - Επιστρέφει ένα αντίγραφο του string με όλα του τα στοιχεία κεφαλαία.
- str.capitalize()
  - Επιστρέφει ένα αντίγραφο του string με το πρώτο του στοιχείο κεφαλαίο.
- str.title()
  - Επιστρέφει ένα αντίγραφο του string με όλα τα πρώτα γράμματα κάθε λέξης κεφαλαία.

Συναρτήσεις Μορφοποίησης II

```
>>> a = "abababababa"
>>> a.upper()
'ABABABABABA'
>>> a.title()
'Abababababa'
>>> a.lower()
'abababababa'
>>> a
'abababababa'
>>> b = a.capitalize()
>>> a
'abababababa'
>>> b
'Abababababa'
>>>
```

Συναρτήσεις Αντικατάστασης

- str.replace(old,new[,count])
  - Επιστρέφει ένα αντίγραφο του string όπου το substring old έχει αντικατασταθεί από το substring new. Αν δοθεί το προαιρετικό όρισμα count, πραγματοποιούνται μόνο οι πρώτες count αλλαγές.

```
>>> a = "Hello, World!!"
>>> b = a.replace("World","girl")
>>> c = b.replace("Hello,","OK bye..")
>>> a
'Hello, World!!'
>>> b
'Hello, girl!!'
>>> c
'OK bye.. girl!!'
>>>
```

Συναρτήσεις Διαχωρισμού I

- str.split([sep[,maxsplit]])
  - Επιστρέφει μία λίστα από λέξεις του string χρησιμοποιώντας το sep ως delimeter. Αν δοθεί το προαιρετικό όρισμα maxsplit, τότε μόνο maxsplit "κοψίματα" πραγματοποιούνται.
  - Έχει σημασία αν δοθεί ή όχι το όρισμα sep καθώς σε κάθε περίπτωση εκτελείται διαφορετικός αλγόριθμος.
  - Αν δοθεί (μπορεί να είναι και παραπάνω από ένας χαρακτήρας), χρησιμοποιείται ως έχει, με την ιδιότητα ότι οι delimeters δεν ομαδοποιούνται.
  - Αν όχι, χρησιμοποιείται ως separator το κενό, και με την ιδιότητα συνεχόμενα κενά μέσα στο string να λαμβάνονται ως ένα.

Συναρτήσεις Διαχωρισμού II

```
>>> a = '  1$$2$3 '
>>> a
'  1$$2$3 '
>>> a.split("$")
['  1', '', '2', '3 ']
>>> a.split(" ")
['', '', '1$$2$3', '']
>>> a.split()
['1$$2$3']
>>> b = ""
>>> b
''
>>> b.split(" ")
['']
>>> b.split()
[]
>>>
```

Συναρτήσεις Συνένωσης

- str.join(seq)
  - Επιστρέφει ένα string το οποίο είναι η συνένωση των strings της ακολουθίας seq. Όπου seq, μπορεί να είναι string, tuple, list, κτλ

```
>>> ls = ["Two", "beer", "or", "not", "two", "beer", "?"]
>>> " ".join(ls)
'Two beer or not two beer ?'
>>>
```

Module StringIO

- import StringIO
  - Με το παραπάνω module μπορεί κανείς να μεταχειριστεί τα strings σαν αρχεία, καθώς δύνεται δυνατότητα για ανάγνωση/εγγραφή ενός string buffer.

```
>>> output = StringIO.StringIO()
>>> output.write("First line.\n")
>>> print >>output,"Second line."
>>> contents = output.getvalue()
>>> output.close()
>>> contents
'First line.\nSecond line.\n'
>>>
```

Regular Expressions - γενικά

- Βασική ιδέα: έχουμε ένα pattern και ένα κείμενο εισόδου. Εφαρμόζουμε το pattern στο κείμενο και μπορούμε:
  - Να ελέγξουμε αν μέρος του κειμένου συμφωνεί με το pattern.
  - Να εξάγουμε κομμάτια του κειμένου που επιθυμούμε
  - Να αντικαταστήσουμε κομμάτια του κειμένου κλπ.
- Πολύ δυνατό εργαλείο διαχείρισης strings.
- Επιτρέπει τις λειτουργίες find και replace σε string, με πολύ περισσότερες δυνατότητες, όχι απλό string matching.
- Πολύ γρήγορα!!!

Regular Expressions #1

Pattern	
.	Ταιριάζει σε οποιονδήποτε χαρακτήρα. πχ το pattern a.c ταιριάζει στα strings abc, a_c, afc, a0c κλπ
^	Ταιριάζει στην αρχή του string. πχ το pattern ^abc ταιριάζει στα strings που ξεκινούν από abc
$	Ταιριάζει στο τέλος του string. πχ το pattern abc$ ταιριάζει στα strings που τελειώνουν με abc
*	Ταιριάζει σε 0 ή περισσότερες εμφανίσεις του pattern που προηγείται. πχ το pattern a* ταιριάζει στο άδειο string και στα a, aa, aaa, aaaa κλπ.
+	Ταιριάζει σε 1 ή περισσότερες εμφανίσεις του pattern που προηγείται. πχ το pattern a+ ταιριάζει στα a, aaa, aaaa, κλπ
?	Ταιριάζει σε 0 ή 1 εμφανίσεις του pattern που προηγείται. πχ το pattern a?bc ταιριάζει στα strings abc και bc
{m}	Ταιριάζει σε m εμφανίσεις του pattern που προηγείται. πχ το pattern a{3} ταιριάζει στο string aaa
{m,n}	Ταιριάζει σε m έως n εμφανίσεις του pattern που προηγείται. πχ το pattern a{1,3} ταιριάζει στα a, aa, aaa

Regular Expression #2

Pattern	
[]	Ταιριάζει σε σύνολο χαρακτήρων. πχ [0-9] είναι ένα οποιοδήποτε αριθμητικό ψηφίο. [a-zA-Z] ένα οποιοδήποτε γράμμα. [134] είναι ένα οποιοδήποτε ψηφίο από τα 1 3 ή 4. [0-9]+ ταιριάζει σε οποιονδήποτε ακέραιο [a-zA-Z]+ ταιριάζει σε οποιαδήποτε λέξη έχει μόνο γράμματα.
[^]	Ταιριάζει σε οτιδήποτε δεν περιέχεται στο σύνολο χαρακτήρων. πχ το [^0-9] ταιριάζει σε οτιδήποτε δεν είναι ψηφίο.
|	x|y ταιριάζει είτε στο x είτε στο y
( )	Ομαδοποίηση ενός υποσυνόλου του pattern. πχ (01)+ ταιριάζει στα strings 01,0101,010101 κλπ ενώ το 01+ ταιριάζει στα 01,011, 0111 κλπ
\	Escape character. Αν θέλω να συμπεριλάβω κάποιον ειδικό χαρακτήρα για την τιμή του, χρησιμοποιώ τον escape. πχ το \+ ταιριάζει στον χαρακτήρα + και όχι σε επαναλήψεις του προηγούμενου.

Regular Expressions #3

Pattern	
\t	Tab
\n	New line
\s	Ταιριάζει στα κενά. (whitespace, tab, new line κλπ)
\d	Ταιριάζει σε ψηφία. Ισοδύναμο με [0-9]
\w	Ταιριάζει σε ψηφία, γράμματα ή underscore. Ισοδύναμο με το [0-9a-zA-Z_]

Regular Expressions – look(ahead|behind)

Pattern	
(?=pattern)	Positive lookahead – ταιριάζει με το string που ακολουθείται από το pattern που περιγράφεται. πχ το readme(?=\.txt) ταιριάζει με το readme μόνο αν έχει κατάληξη .txt Προσοχή: αυτό που ταιριάζει είναι το readme! Το .txt δεν καταναλώνεται!
(?!pattern)	Negative lookahead – ταιριάζει με το string που δεν ακολουθείται από το pattern που περιγράφεται.
(?<=pattern)	Positive lookbehind – ταιριάζει με το string από το οποίο προηγείται το pattern που περιγράφεται. πχ. το (?<=readme) \.txt ταιριάζει με το .txt μόνο αν προηγείται το readme.
(?<!pattern)	Negative lookbehind – ταιριάζει με το string από το οποίο δεν προηγείται το pattern που περιγράφεται.

Python – re module

- Module: re
- Υποστηρίζει:
  - search
  - match
  - groups
  - named groups

Python - examples

- re.search(pattern, input): Εφαρμόζει το pattern στο input
  - Επιλογή και ανάκτηση του group χρησιμοποιώντας τις παρενθέσεις
- re.findall(pattern, input): Βρίσκει πολλαπλές εμφανίσεις στο input

>>> import re

```
>>> m = re.search('([a-zA-Z]+) ([a-zA-Z]+)', 'Stand back! I know regular expressions!')
>>> m.group(0)
'Stand back'
>>> m.group(1)
'Stand'
>>> m.group(2)
'back'
```

```
>>> lst = re.findall('[a-zA-Z]+','Stand back! I know regular expressions!')
>>> lst
['Stand', 'back', 'I', 'know', 'regular', 'expressions']
```

Python - examples

```
>>> test = '000 001 010 011 100 101 110 111'
>>> lst = re.findall('[01]{3}',test)
>>> lst
['000', '001', '010', '011', '100', '101', '110', '111']
>>> lst = re.findall('[^ ]+',test)
>>> lst
['000', '001', '010', '011', '100', '101', '110', '111']
```

```
>>> m = re.search('[0-9]+(\\.[0-9]+)?','6.23')
>>> m.group(0)
'6.23'
>>> m = re.search('[0-9]+(\\.[0-9]+)?','166')
>>> m.group(0)
'166'
```

```
>>> path = "C:\\Users\\Vivi\\test\\regex\\readme.txt"
>>> lst = re.findall('[^\\\\]+',path)
>>> lst
['C:', 'Users', 'Vivi', 'test', 'regex', 'readme.txt']
```

Escape Hell

- Τα regular expressions χρησιμοποιούν ως escape character το \.
- Το pattern είναι ένα string που γίνεται compile ανεξάρτητα από τη γλώσσα προγραμματισμού.
- Για να γραφτεί το \ σε string, η γλώσσα προγραμματισμού χρειάζεται το δικό της escape character που είναι το \!
- Παράδειγμα:
  - Το pattern "\\\\" ταιριάζει με το χαρακτήρα \. Έχει ένα \ για το escape του regular expression και ένα επιπλέον για το escape του string.
- Λύση:
  - Προσοχή στο μέτρημα των escapes ☺ ή
  - Χρήση raw strings για το pattern:

```
>>> lst = re.findall(r'[^\\]+',path)
>>> lst
['C:', 'Users', 'Vivi', 'test', 'regex', 'readme.txt']
```

Python match vs search

- Υποστηρίζονται δύο τρόποι ψαξίματος:
  - re.search : ψάχνει σε οποιοδήποτε μέρος του string
  - re.match : ψάχνει μόνο στην αρχή του string
- **>>>** re.match("c", "abcdef") *# No match*
- **>>>** re.search("c", "abcdef") *# Match* <_sre.SRE_Match object at ...>

Python – named groups

- Με τη χρήση του pattern (?P<a_name>pattern) η python υποστηρίζει named groups.
- **>>>** m = re.match(r"(?P<first_name>\w+) (?P<last_name>\w+)", "Malcom Reynolds")
- **>>>** m.group('first_name')
  'Malcom'
- **>>>** m.group('last_name')
  'Reynolds'

Regular Expressions - more

- Επιπλέον δυνατότητες των regular expressions:
  - split σε input strings
  - replace
  - Επιλογές για το αν η είσοδος θα είναι multiline, το encoding κλπ.
  - ...