ΠΡΟΑΠΑΙΤΟΥΜΕΝΑ:

1. Python Advanced: Μάθημα 2 - Iterators

ΠΕΡΙΕΧΟΜΕΝΑ:

1. Pattern και matcher
2. Συντακτικό re
    1. Στοιχειώδη Ταιριάσματα
    2. Επανάληψη
    3. Χαρακτήρες Αποφυγής
    4. Ομάδες Χαρακτήρων
    5. OR και group χαρακτήρων
    6. Ταίριασμα στα άκρα

Κατερίνα Τ.

Σμαραγδένιος Χορηγός Μαθήματος

Πάνος Γ.

Ασημένιος Χορηγός Μαθήματος

ΜΑΘΗΜΑ 8.1: Το module re (Α') — 1. Pattern και matcher

Τα **regular expressions** (regexp, re, μτφ: κανονικές εκφράσεις) είναι μία γλώσσα περιγραφής ομάδων συμβολοσειρών.

- π.χ. η κανονική έκφραση **ab.*cd** περιγράφει τις συμβολοσειρές που ξεκινούν με ab και τελειώνουν με cd
- και λέμε ότι π.χ. η συμβολοσειρά «abacd» **ταιριάζει (matches)** στην κανονική έκφραση (μπορεί να προκύψει από αυτήν)
- ενώ π.χ. η συμβολοσειρά «aa» δεν ταιριάζει στην καν.έκφραση.

Η κλάση **Pattern** (πρότυπο) περιτυλίσσει μία κανονική έκφραση

- Αρχικοποιείται μέσω της μεθόδου:

Στατ. Μέθοδος	Επεξήγηση
compile(string re)	Αρχικοποιεί και επιστρέφει ένα αντικείμενο τύπου Pattern, το οποίο εκφράζει την κανονική έκφραση re

- Προσοχή! Οι κανονικές εκφράσεις χρησιμοποιούν στο συντακτικό τους, κάποιους χαρακτήρες (όπως το '\') που είναι χαρακτήρες διαφυγής σε τυπικές συμβολοσειρές.
- Γι' αυτό συνίσταται να χρησιμοποιούμε στο πρότυπο **raw strings** που συντάσσεται με r μπροστά από τη συμβολοσειρά **(r"…")**

- Μεταξύ των μεθόδων της Pattern, βρίσκεται και η μέθοδος:

Μέθοδος	Επεξήγηση
fullmatch(str)	Τσεκάρει αν η συμβολοσειρά str ταιριάζει στην κανονική έκφραση.

  - Επιστρ. None, αν δεν υπάρχει ταίριασμα
  - Επιστρ. αντικείμενο **matcher** αν υπάρχει ταίριασμα.

- Η κλάση matcher προσφέρει πλούσια πρόσθετη λειτουργικότητα (όπως π.χ. να βρούμε σε ποιο σημείο της συμβολοσειράς βρέθηκε το pattern κ.λπ), την οποία θα μελετήσουμε στο επόμενο μάθημα.
- Απλοποιούμε (καθαρά για λόγους μελέτης του συντακτικού re) με μία δική μας συνάρτηση ως:

```
import re
def matches(text, regexp):
    pattern = re.compile(regexp)
    matcher = pattern.fullmatch(text)
    if matcher is None:
        return False
    else:
        return True
```

Παράδειγμα 1: PatternMatcher

```
print(matches("abba", r"abba"))
print(matches("abba", r"baba"))
```

- Παραπάνω βλέπουμε το απλούστερο ταίριασμα: Μία συμβολοσειρά ταιριάζει στον εαυτό της (και σε καμία άλλη συμβολοσειρά)

Ταίριασμα με έναν χαρακτήρα:

re	Ταίριασμα με:
.	Ακριβώς ένας χαρακτήρας

Παράδειγμα 2: dot

```
r = r"a.b."

print(matches("abba", r))
print(matches("baba", r))
print(matches("abbaa", r))
print(matches("acbz", r))
print(matches("aaabba", r))

print("=" * 20)
print(matches("abc", r"..."))
print(matches("abcde", r"..c.."))
print(matches("aa\nbb", r".."))
print(matches("\n\n", r".."))
```

Παρατηρήσεις:

- Τα ταιριάσματα πραγματοποιούνται στην 1η γραμμή του κειμένου
- Θα δούμε τρόπους ορισμού του ταιριάσματος σε πολλαπλές γραμμές στη συνέχεια του μαθήματος

Επανάληψη 0 ή περισσότερες φορές:

re	Ταίριασμα με:
c*	0 ή περισσότερες φορές το c (,c,cc,ccc,…)

Το c είναι είτε:

- ένας χαρακτήρας
- ακολουθία χαρακτήρων (πρέπει να εντίθενται σε παρενθέσεις), π.χ. (ab)* ή (.a)*

Παράδειγμα 3: star

```
print(matches("", r"a*"))
print(matches("a", r"a*"))
print(matches("aa", r"a*"))
print(matches("aab", r"a*"))

print(matches("Niger", r".*er"))
print(matches("Niger", r".*em"))
print(matches("Mexico", r"Me.*"))
print(matches("Nexico", r"Me.*"))
print(matches("Nicaragua", r".*ar.*"))
print(matches("Nicaaagua", r".*ar.*"))
print(matches("Morocco", r"M.*ro.*o"))
print(matches("Mexico", r"(..)*"))
print(matches("USA", r"(..)*"))
print(matches("Burundi", r"B(u.)*di"))
```

Ειδικοί χαρακτήρες επανάληψης (quantifiers):

re	Ταίριασμα με:
c*	0 ή περισσότερες φορές το c (,c,cc,ccc,…)
c+	1 ή περισσότερες φορές το c (c,cc,ccc,…)
c?	0 ή 1 φορές
c{n,m}	n έως m φορές
c{n,}	τουλάχιστον n φορές
c{n}	ακριβώς n φορες

Το c είναι είτε:

- ένας χαρακτήρας
- ακολουθία χαρακτήρων (πρέπει να εντίθενται σε παρενθέσεις), π.χ. (ab)* ή (.a)*

Παράδειγμα 4: quantifiers

```
print(matches("abb", r"ab+"))
print(matches("abb", r"ab{3,}"))
print(matches("abab", r"(ab){2}"))
print(matches("abab", r"(ab){2}"))

print("=" * 10)

print(matches("aaaa", r"aa(aa)?"))
print(matches("aaaa", r"(..)+"))
print(matches("aaaa", r".(..)*"))
```

Οι ειδικοί χαρακτήρες επανάληψης συγκεκριμενοποιούνται με δύο αλγοριθμικές παραλλαγές:

re	Ταίριασμα με:
c*	0 ή περισσότερες φορές το c (,c,cc,ccc,…) [**greedy**] [άπληστος, θα δοκιμάσει το ταίριασμα μεγαλύτερου μήκους, αν το επόμενο ταίριασμα της κ.ε. αποτύχει, θα δοκιμάσει το αμέσως μικρότερο ταίριασμα]
c*?	0 ή περισσότερες φορές το c (c,cc,ccc,…) [**minimal, lazy**] [διστακτικός, θα δοκιμάσει το ταίριασμα μικρότερου μήκους, αν το επόμενο ταίριασμα της κ.ε. αποτύχει, θα δοκιμάσει το αμέσως μεγαλύτερο ταίριασμα]

- ισχύουν και για τους υπόλοιπους quantifiers (+,?, {})
- και το c πάλι είναι χαρακτήρας ή ακολουθία χαρ/ρων

- Βλέπουμε ακόμη μία πολύ χρήσιμη συνάρτηση του re:

Μέθοδος	Επεξήγηση
findall(text, re)	Επιστρέφει λίστα με διαδοχικές εμφανίσεις υποσυμβολοσειρών της text, που ταιριάζουν στην re

Παράδειγμα 5: quantifiers2

```
text = "Men occasionally stumble over the truth, but most of them "
    "pick themselves up and hurry off as if nothing had happened."

print(re.findall(r" o.* ", text))
print(re.findall(r" o.*? ", text))
```

Οι χαρακτήρες *, +, ? κ.λπ. που είδαμε στην προηγούμενη διαφάνεια, λέγονται και **μεταχαρακτήρες (meta-characters)** γιατί αποδίδουν ένα ειδικό νόημα στην κανονική έκφραση.

- Αν θέλουμε όμως η κ.ε. μας να περιέχει κυριολεκτικά κάποιον τέτοιο χαρακτήρα, τότε θα πρέπει να προσδιορίσουμε τη χρήση του ως κανονικός χαρακτήρας.
- Αυτό καλείται **«αποφυγή χαρακτήρα» (escape character)** και το κάνουμε θέτοντας ένα backslash (\) πριν από το μεταχαρακτήρα.

Παρατήρηση:

- Το παραπάνω ισχύει εφόσον χρησιμοποιούμε raw strings για την απεικόνιση της κανονικής έκφρασης.
- Αν χρησιμοποιούμε κανονικές συμβολοσειρές, πρέπει να αποφύγουμε το \ και έπειτα το μεταχαρακτήρα (άρα πρακτικά χρησιμοποιούμε \\ για την αποφυγή)

Παρατήρηση:

- Χρησιμοποιείται στο παράδειγμα η ακόλουθη μέθοδος του re:

split(re, test)	Χρησιμοποιεί την re ως διαχωριστή, για να χωρίσει την text σε υποσυμβολοσειρές

Παράδειγμα 6: escaping

```
print(matches("a+aa", r"a+aa"))
print(matches("a+aa", r"a\+aa"))
print(matches("a+aa", "a\\+aa"))
print(re.split(r"\. *", text))
```

Ενώ υπάρχουν και οι ακόλουθοι ειδικοί χαρακτήρες, για ειδική χρήση:

re	Ταίριασμα με:
\n	Αλλαγή γραμμής
\t	tab
\	Το backslash

- και υπάρχουν και άλλοι πιο προχωρημένοι (κωδικοποίηση χαρακτήρων ως 8δικοί, 16δικοί, Unicode και ειδικοί χαρακτήρες όπως το \b (beep), \f (form-feed), \r (carriage return κ.α)

Παράδειγμα 7: special_characters

```
print(matches("\\\\", r"\\{2,}"))

print("=" * 10)

text = "Everything about me is a contradiction, " + \
    "and so is everything about everybody else.\n " + \
    ...
    "You just recognize them."

print(re.split(r"\n *", text))
```

Τα ακόλουθα τμήματα κανονικής έκφρασης ορίζουν **ομάδες χαρακτήρων (character classes)**:

- θα ταιριάξει με ένα χαρακτήρα που ανήκει στην συγκεκριμένη ομάδα

re	Ταίριασμα με:
[abc]	a ή b ή c
[^abc]	Οποιοσδήποτε χαρακτήρας εκτός των a,b,c
[a-zA-Z]	Εύρος χαρακτήρων. Ισοδύναμα [a-z] ή [A-Z]

και υπάρχουν και ομάδες που έχουν οριστεί ώστε να «γκρουπάρουν» ομοειδείς χαρακτήρες:

re	Ταίριασμα με:
\d	ψηφίο [0-9]
\D	όχι ψηφίο [^0-9]
\s	whitespace[\t\n\r\f\v]
\S	όχι whitespace [^\s]
\w	word character [a-zA-Z]
\W	όχι word character

Παράδειγμα 8: character_classes

```
print(matches("abb", r"[abc]*"))
print(matches("abZ", r"[a-z]*"))
print(matches("abdm", r"[a-dm-p]*"))
print(matches("ffxm", r"[^a-d]*"))

print("=" * 10)

print(matches("psounis21@gmail.com",
        r"[a-zA-Z1-9]{8,12}@gmail.com"))
print(matches("1.psounis@gmail.com",
        r"[a-zA-Z1-9]{8,12}@gmail.com"))
print(matches("psounis-21@gmail.com",
        r"[a-zA-Z0-9_-]+@[a-zA-Z]+\.com"))
```

Παράδειγμα 9: predefined_character_classes

```
regexp = r"[0-2][0-3]:[0-5]\d"
print(matches("01:49", regexp))
print(matches("11:01", regexp))
print(matches("25:11", regexp))

print("=" * 10)
regexp = r"[0-2][0-3]:[0-5]\d:[0-5]\d(.\d{1,4})?"
print(matches("01:49:12", regexp))
print(matches("11:01:11.11", regexp))
print(matches("22:11:23.11213", regexp))
```

Διάζευξη κανονικών εκφράσεων:

- Μπορούμε να επιβάλλουμε διαφορετικές περιπτώσεις στις κανονικές εκφράσεις με χρήση του συντακτικού:

re	Ταίριασμα με:
re1 | re2	re1 ή re2

Παράδειγμα 10: or

```
text = "Computer Science is no more about computers " + \
    "than astronomy is about telescope"

print(re.findall(r"Sci.*? |ast.*? ", text))
```

- **Δίνεται η δυνατότητα να πάρουμε τμήματα του ταιριάσματος μέσω των group**
  - Ορίζουμε ένα group ενθέτοντας το τμήμα της κανονικής έκφρασης σε παρενθέσεις
  - Η αρίθμηση των groups γίνεται από αριστερά προς τα δεξιά στην κανονική έκφραση.
  - Όλο το ταίριασμα νοείται ότι είναι στο group 0.
  - Ανασύρουμε το περιεχόμενο κάθε group με τη μέθοδο **group(n)**, ή indexed, π.χ. **matched_object[0]**, του αντικειμένου match.

Παράδειγμα 11: groups

```
text = "Computer Science is no more about computers " + \
    "than astronomy is about telescope"

m = re.match(r".* (\w+?) is (\w+?) .*", text)
print(m.group(0) + "|" + m.group(1) + "|" + m.group(2))
print(m[0] + "|" + m[1] + "|" + m[2])
```

- Μπορούμε να δώσουμε και όνομα στο group γράφοντας στις παρενθέσεις (?P<group_name> re) όπου group_name το όνομα που αποδίδουμε στο group και re η κανονική έκφραση.

Παράδειγμα 12: groups2

```
text = "Computer Science is no more about computers " + \
    "than astronomy is about telescope"

m = re.match(r".* (?P<previous>\w+?) is (?P<next>\w+?) .*", text)
print(m["previous"], m["next"])
```

Παρατήρηση:

- Η συνάρτηση match εντοπίζει μόνο το τελευταίο ταίριασμα επί της συμβολοσειράς.
- Θα δούμε τρόπους για να πάρουμε όλα τα ταιριάσματα στο επόμενο μάθημα.

Μέσω ειδικών χαρακτήρων μπορούμε να κάνουμε ταίριασμα σε άκρα της συμβολοσειράς.

- Οι παρακάτω επιβάλλουν το άκρο της συμβολοσειράς να έχει συγκεκριμένη μορφή:

re	Ταίριασμα με:
^	ταίριασμα στην αρχή (^re…)
$	ταίριασμα στο τέλος (…re$)

Παρατήρηση:

- Κάποιες μέθοδοι (όπως η findall) δέχονται προαιρετικό όρισμα με σημαίες. Π.χ. η findall δέχεται προαιρετικό 3ο όρισμα με τιμή re.MULTILINE.
- Τότε επιστρέφονται όλα τα ταιριάσματα (στην αρχή ή στο τέλος) της κάθε γραμμής.

Παράδειγμα 13: boundary

```
with open("pies.html", encoding="utf-8") as f:
    text = f.read()
    print(text)
res = re.findall(r"^\s*?<(.+?)>", text, re.MULTILINE)
print(res)
res2 = re.findall(r"^\s*?<(.+?)>.*<(/.+?)>$", text, re.MULTILINE)
print(res2)
```

- Ενδιαφέρον έχει και οι ακόλουθοι δύο ειδικοί χαρακτήρες

re	Ταίριασμα με:
\b	άκρο λέξης. Επιβάλλεται στην αρχή, στο τέλος ή και στα δύο.
\B	επέκταση άκρου λέξης. Επιβάλλεται στην αρχή, στο τέλος ή και στα δύο.

Παράδειγμα 14: boundaryWord

```
with open("pies.html", encoding="utf-8") as f:
    text = f.read()
    print(text)


res = re.findall(r"\bπίτες\b", text)
print(res)

res2 = re.findall(r"(\b\w*?)\Bπιτες\b", text)
print(res2)

res3 = re.findall(r"\ba.*?\b", text)
print(res3)
```