Wyrażenia regularne

Funkcje obsługujące wyrażenia regularne w Pythonie znajdują się w module re.
>>> import re

1.1. Funkcja match. Funkcja re.match próbuje odszukać dany wzorzec na początku ciągu; jeśli go znajdzie, zwraca znaleziony obiekt match, jako powodzenie (Match) lub niepowodzenie (None).

Składania dla tej funkcji to:
re.match(pattern, string, flags=0)

pattern - wyrażenie regularne wykorzystywane do dopasowania
string - ciąg znaków, które b ędą przeszukiwane aby dopasować wzorzec do początku łańcucha
flags - modyfikatory; wykorzystywane w procesie wyszukiwania

Jak to wygląda w praktyce?

```
# Example 1
        r = re.match("P.t", "Python")
        print(r.group())
```

Uwaga: Funkcja **group**, zwraca nam ciąg znaków dopasowania.

1.2. Funkcja search. Funkcja re.search wyszukuje pierwsze dopasowanie wzoru do ciągu znaków. Funkcja re.search zwraca obiekt match, jako powodzenie (Match) lub niepowodzenie (None).

Składania dla tej funkcji to:
re.search(pattern, string, flags=0)

Jak to wygląda w praktyce?

```
# Example 1
        value="voorheesville"
        m=re.search("(vi.*)", value)
        if m: print("search:", m.group(1))
        m=re.match("(vi.*)", value)
        if m: print("match:", m.group(1))
# Example 2
        x = re.search("cat","A cat and a rat can't be
        friends.") print(x)
        x = re.search("cow","A cat and a rat can't be
        friends.") print(x)
# Example 3
        s = 'BEGIN hello world END'
        mo = re.search('BEGIN (.*) END', s)
        print(mo.group(1))
```

1.3. Funkcja compile. Funkcja re.compile kompiluje wyrażenie regularne i zwraca odpowiedni obiekt.

Składania dla tej funkcji to:
re.compile(pattern, flags=0)

Jak to wygląda w praktyce?

```
# Example 1
        patt = re.compile("\w+")
```

```
words = patt.findall("Hello World")
print(words)
# Example 2
s = 'Hello world, this is a test!'
print(re.findall(r'\S+', s))
print(re.sub( r'\S+' , 'WORD', s))
```

Uwaga: Jeśli łańcuch poprzedzimy znakiem **r** lub **R**, wtedy nie są uwzględniane znaki specjalne, a tekst jest traktowany dosłownie: R" … \n…" - tzw. łańcuch surowy.

1.4. Funkcja findall. Funkcja re.findall zwraca wszystkie (nie nachodzące na siebie) wystąpienia wzorca w danym łańcuchu.

Składania dla tej funkcji to:
re.findall(pattern, string, flags=0)

Jak to wygląda w praktyce?

```
# Example 1
sample="Weronika ma 15 lat, a Daniel ma 27 lat. Edward jest o 97 lat od niego    starszy  i jego dziadek Oskar ma 102 lata"
ages = re.findall(r'\d{1,3}',sample)
names = re.findall(r'[A-Z][a-z]*', sample)
print(ages)
print(names)
# Example 2
value="abc 123 def 456 dot map pat"
list=re.findall("[dp]\w+", value)
print(list)
```

1.5. Funkcja sub. Funkcja re.sub to funkcja podmiany, która zamienia wszystkie wystąpienia wzorca w napisie na podany ciąg.

Składania dla tej funkcji to:
re.sub(pattern, repl, string, max=0)

repl - ciąg funkcji/znaków

Jak to wygląda w praktyce?

```
# Example 1
str = "yes I said yes I will Yes."
res = re.sub("[yY]es", "no", str)
print (res)
# Example 2
v = "running eating reading"
v = re.sub(r"r.*?ing", "ring",v)
print(v)
```

1.6. Funkcja split. Funkcja re.split dzieli napis wg podanego wzorca.

Składania dla tej funkcji to:
re.split(pattern, string, max=0)

Jak to wygląda w praktyce?

```
# Example 1
value="one 1 two 2 three 3"
result=re.split("\D+", value)
for element in result:
        print (element)
# Example 2
a = "Potrzeba było dwóch uderze ń aby wydostać się z bunkra."
sample = a.split(" ",3)
print(sample)
```

1.7. Podstawowe wzorce wyrażeń regularnych

Identificators:	Modificators:	White Space Charts:
\d = any number \D = anything but a number \s = space \S = anything but a space \w = any letter \W = anything but a letter . = any character, except for a new line \b = space around whole words \. = period must use backslash, because normally means any character	{1,3} = for digits, u expect 1-3 counts of digits, or "places" + = match 1 or more ? = match 0 or 1 repetitions * = match 0 or MORE repetitions $ = matches at the end of string ^ = matches start of a string | = matches either/or (example: x|y = will match either x or y) [] = range, or "variance" {x} = expect to see this amount of the preceding code {x,y} = expect to see this x-y amounts of the precedng code	\n = new line \s = space \t = tab \e = escape \f = form feed \r = carriage return