Теоретический материал к занятию Введение в регулярные выражения.

Регулярные выражения(regex) - механизм поиска и замены необходимого текста в строке, либо же во всем файле. Регулярные выражения, достаточно мощный инструмент позволяющий выполнить любой тип обработки текста. Регулярные выражения часто применяются для поиска и проверки данных. Например пользователь вводит email и мы проверяем формат введенного адреса.

Для работы с регулярными выражениями в Python, существует модуль стандартной библиотеки re. Соответственно устанавливать для работы нам ничего не надо. Требуется только импортировать библиотеку

```
import re
```

Возьмем строку ‘hello world!’, в которой нам требуется найти слово ‘hello’

```
import re

sentence = ‘hello world!’

r = re.match(‘hello’, sentence)
print(r)
```

В данном примере мы с вами использовали функцию match модуля re. Данная функция ищет совпадение в начале строки по заданному шаблону и возвращает объект match.
Синтаксис функции match
re.match(pattern, string, flag)
pattern - регулярное выражение
string - строка по которой производится поиск
flag - модификатор. Например с помощью модификаторов мы можем игнорировать регистр текста.

Функция match ищет только в начале строки. Если мы например попробуем изменить искомое слово с ‘hello’ на ‘llo’, то функция вернет None.

Для того, чтобы вывести начальный и конечный индексы в виде кортежа, мы можем воспользоваться методом span, объекта match

```
print(r.span())
```

Вывести начальный и конечный индексы по отдельности можно с помощью start() и end() соответственно

```
print(r.start())
print(r.end())
```

Вывести искомое слово, а вернее полное совпадение

```
print(r.group(0))
```

Очень часто мы можем встретить запись шаблона с указанием сырой строки(необработанной строки). Для этого перед строкой указывается r(raw).

r = re.match(r'\d+', sentence)

Необработанные строки используются для того, чтобы знак \ не экранировал символы(\n ,\t и т.д) Если в строке есть \, который нужен сам по себе, то его нужно дополнительно экранировать самим \.
Пример:
"C:\\Users\\Admin"
Для raw-строк текст станет более человеко-читаемый:
r"C:\Users\Admin"

Модернизируем первый пример, для того чтобы при поиске игнорировался регистр, применив модификатор.

```
import re

sentence = 'Hello world!'

r = re.match(r'hello', sentence, re.I)
if r:
  print(r.group(0))
else:
  print('Не найдено совпадений')
```

В примере выше мы указали модификатор re.I(IGNORECASE), который позволяет игнорировать регистр текста. Также добавили условие, которое при проверке, если она удачна, выводит нам совпадение, в противном же случае(если match вернул None) мы выводим сообщение, о том, что совпадений не найдено.

Теперь давайте рассмотрим функцию search. Данная функция, также возвращает объект match, синтаксис точно такой же, но в отличие от match, функция search ищет по всей строке.

```
import re

sentence = 'Hello World!'

r = re.search(r'world', sentence, re.I)
if r:
  print(r.group(0))
```

```
else:
  print('Не найдено совпадений')
```

Как можно заметить основное отличие в этих функциях, места поиска в строке необходимого шаблона.

Теперь давайте рассмотрим что же такое group() и для чего он может принимать. При составлении регулярного выражения, мы можем группировать отдельные элементы с помощью круглых скобок. После этого используя метод group(), в него мы передаем номер группы.

```
import re

sentence = 'Hello World!'

r = re.search(r'(\w+)\s(\w+)', sentence, re.I)
if r:
  print(r.group(0))
  print(r.group(1))
  print(r.group(2))
else:
  print('Не найдено совпадений')
```

Результатом данного примера в group(0) выведется вся найденная подстрока, в group(1) - первое значение(hello), а в group(2) соответственно ‘world’). Символ \w означает любое буквенное значение.

Помимо функций match и search, еще существует функция findall(). В отличие двух предыдущих она ищет по всей строке все возможное вхождения. Match и search ищут только до первого встреченного совпадения.

```
import re

sentence = 'Python! Регулярные выражения в python'

r = re.findall(r'python', sentence, re.I)
print(r)
```

Findall() из примера мы видим возвращает уже список совпадений, в данном случае список из двух элементов.