

Python para Análisis de datos: Introducción

Sesión 4

Alejandro Villar (avillar@ticnor.es)

8 Abril 2019

Expresiones regulares

Expresiones regulares en Python

Web scraping

Brevísima intro a HTML

Extraer datos con Pandas

BeautifulSoup

Scrapy

Expresiones regulares

Qué es una Expresión regular

- ▶ Expresión regular (regex, regexp): Patrón de búsqueda de texto
- ▶ Ejemplo:
 - ▶ `b{2,5}` -> "Entre 2 y 5 veces 'b' "
 - ▶ `\. +` -> "Un punto seguido de uno o más espacios"

Conceptos básicos (I)

- ▶ Alternación (una cosa u otra)
 - ▶ Ejemplo: `gris|verde`
- ▶ Agrupación (juntas cosas)
 - ▶ Ejemplo: `s(o|ó)lo`
- ▶ Comodín (el punto):
 - ▶ Ejemplo: `s.lo`
- ▶ Caracteres especiales: ¡escapar con `\`!

Conceptos básicos (II)

- ▶ Cuantificación (una cosa n veces)
 - ▶ `?` = “cero o una veces”
 - ▶ `*` = “cero o más veces”
 - ▶ `+` = “una o más veces”
 - ▶ `{n}` = “n veces”
 - ▶ `{n,}` = “n o más veces”
 - ▶ `{n,m}` = “entre n y m veces”

$abc? = ab / abc$
 $abc+ = abc / abcc / abccc / \dots$
 $abc* = ab / abc / abcc / \dots$
 $abc\{2,3\} = abcc / abccc$

Conceptos básicos (III)

- ▶ Clases de caracteres:
 - ▶ $[abc] = \text{"'a' o 'b' o 'c' "}$
 - ▶ $[a-z] = \text{"carácter de la 'a' a la 'z' "}$
 - ▶ $[a-zA-Z0-9] = \text{"'a' a la 'z' o '0' a '9' "}$
 - ▶ $[^a-z] = \text{"carácter que no es de la 'a' a la 'z' "}$

[0-2] [3-5]
[a-f]? [jkl]
[gq] [^u]
[ab]+[cd]?

Conceptos básicos (V)

- ▶ Clases de caracteres:
 - ▶ \s = [\t\n\r\f] = espacio
 - ▶ \d = [0-9] = dígito
 - ▶ \w = [A-Za-z0-9_] = carácter de palabra
 - ▶ En mayúsculas -> negados
- ▶ Otros caracteres especiales:
 - ▶ ^ = “principio de línea”
 - ▶ \$ = “final de línea”
 - ▶ \b = “límite de palabra”

Ejemplos

```
H[aäe]nse!  
;Go+1!  
(No m|M)e gusta Python  
Mi [mp]adre tiene \d+ años  
[a-z0-9]{4,8}  
^(python|java)$
```

Grupos y capturas

- ▶ Los paréntesis crean grupos que el motor “captura”
- ▶ Ejemplo:
 - ▶ Nueva (York|Inglaterra|Montaña)
 - ▶ ((Facultad|Escuela(Técnica Superior)?)) de (Medicina|Enfermería|Ingeniería)
- ▶ Grupos capturados -> Extracción, reemplazo, etc.

Problemas (I)

- ▶ Difíciles de entender para humanos (*write-only*)
 - ▶ <https://regex101.com/>
- ▶ No valen para todo
 - ▶ Sólo lenguajes regulares
 - ▶ Pueden llegar a ser muy complejas

Problemas (II)

Validación de email:

```
\A(?:[a-z0-9!#$%&'**+\/=^_`{|}~-]+(?:\.[a-z0-9!#$%&'**+\/=^_`{|}~-]+)*)|"(?:[\x0
```

Expresiones regulares en Python

Python soporta nativamente expresiones regulares mediante el módulo `re`.

```
>>> import re
>>> r = re.compile('a[bc]')
>>> r.match('ab')
<_sre.SRE_Match object; span=(0, 2), match='ab'>
```


Módulo re (I)

- ▶ `re.compile(patron, flags=0)` -> Compila expresiones
- ▶ `re.search(patron, string, flags=0)` -> Busca patron en cualquier sitio del string
- ▶ `re.match(patron, string, flags=0)` -> Busca patron **al principio** del string
- ▶ `re.fullmatch(patron, string, flags=0)` -> Comprueba que string concuerda completamente con patron

Módulo re (II)

- ▶ `re.split(patron, string, max=0, flags=0)` -> Divide string cuando se encuentra patron
- ▶ `re.findall(patron, string, flags=0)` -> Devuelve una lista con todas las coincidencias de patron en string
- ▶ `re.sub(patron, reempl, string, flags=0)` -> Reemplaza las apariciones de patron en string

Módulo re (III)

- ▶ Flags

- ▶ `re.I` / `re.IGNORECASE`
- ▶ `re.M` / `re.MULTILINE`
- ▶ `re.S` / `re.DOTALL`

- ▶ Ejemplo de uso:

```
re.compile(r'[a-z]+', re.I | re.S)
```

Módulo re (IV)

- ▶ `re.search`, `re.match` y `re.fullmatch` devuelven `None` o un objeto `Match`
 - ▶ `Match.group(num)` -> grupo capturado (0 = todo el *match*, resto numerados)
 - ▶ `Match.groups()` -> todos los grupos
 - ▶ `Match.start()` y `Match.end()` -> Inicio y final de coincidencia (pueden aceptar el núm. del grupo).

Ejemplos

```
re.sub(r'\. {2,}', ' ' texto)
re.sub(r'(F(ernando|rancisco))', r'Julián (antes \1)', texto, re.I)
re.findall(r'((\d+),)*(\d+)(\.(\d+))?', texto)
re.split(r'\.s*', texto)
```

Otras características avanzadas

- ▶ *Backreferences*
- ▶ Grupos con nombre
- ▶ *Lookahead* y *Lookbehind*
- ▶ Condicionales
- ▶ ...

<https://www.regular-expressions.info/>

Ejercicios

1. Escribir una función que compruebe que una cadena de caracteres sólo tiene caracteres de la a a la f, con un mínimo de 8.
2. Escribir una función que compruebe que todas las iniciales de una frase son mayúscula.
3. Escribir una función que compruebe que una cadena empieza por a, termina por b, y por el medio tiene al menos una o.

Ejercicios

4. Escribir una función que reemplace los espacios múltiples en una cadena de caracteres por un espacio sencillo.
5. Escribir una función que divida una cadena en varias, haciendo corte cuando aparezcan @, | o ; al menos 2 veces seguidas.
6. Escribir una función que encuentre minúsculas seguidas de punto y uno o varios espacios.

Ejercicios

7. Escribir una función que encuentre todas las palabras que empiezan por a y terminan por o en un texto, sin importar mayúsculas/minúsculas.
8. Escribir una función que reemplace números con el formato 1,234,567.89 por 1.234.567,89 en un texto.

Web scraping

Qué es



Extracción de datos desde páginas web.


Utilidades

- ▶ Extracción de datos no estructurados (p.e., sin APIs de datos) o con formatos incorrectos (páginas con errores de sintaxis)
- ▶ Indexación (motores de búsqueda)
- ▶ Minería de datos
- ▶ Monitorización (precios, cambios, etc.)

Ejemplo

1. Definir punto de entrada (p.e., portada).
2. Rastrear elementos con contenido a extraer (p.e., listado de productos)
3. Localizar enlaces a siguientes páginas -> Añadir a cola de rastreo
4. Pasar a siguiente página en la cola y volver a 2

 Articles About 1,810,000 results (0.07 sec)

Any time

Since 2018

Since 2017

Since 2014


Custom range...

Sort by relevance


Sort by date


☒ include patents

☒ include citations

 Create alert

User profiles for **einstein**



 Albert **Einstein** - Cited by 121742

 Andrew J. **Einstein** - Verified email at columbia.edu - Cited by 9536

Can quantum-mechanical description of physical reality be considered complete?



[A. Einstein](#), B Podolsky, N Rosen - Physical review, 1935 - APS

In a complete theory there is an element corresponding to each element of reality. A sufficient condition for the reality of a physical quantity is the possibility of predicting it with certainty, without disturbing the system. In quantum mechanics in the case of two physical ...

  Cited by 17189 [Related articles](#) [All 112 versions](#)

[CITATION] A. **Einstein**, B. Podolsky, and N. Rosen, Phys. Rev. 47, 777 (1935).




[A Einstein](#) - Phys. Rev., 1935

  Cited by 1409 [Related articles](#)

[BOOK] Ideas and opinions

[A Einstein](#), C Seelig, S Bargmann, I Unna, B Wolff - 1954 - ias.ac.in




" Bear in mind that the wonderful things you learn in your schools are the work of many generations, produced by enthusiastic effort and infinite labor in every country of the world. All this is put into your hands as your inheritance in order that you may receive it, honor it ...


  Cited by 1913 [Related articles](#) [All 7 versions](#) 

[BOOK] Investigations on the Theory of the Brownian Movement

[A Einstein](#) - 1956 - books.google.com

The " Brownian movement" was first described in 1828 by the botanist Robert Brown. While investigating the pollen of several different plants, he observed that pollen dispersed in water in a great number of small particles which he perceived to be in uninterrupted and ...

  Cited by 4282 [Related articles](#) [All 4 versions](#) 

 [Next](#)

1 2 3 4 5 6 7 8 9 10

Brevísima intro a HTML

- ▶ HTML: lenguaje de marcado (*markup language*)
- ▶ Especificación del W3C
- ▶ Base de la web
- ▶ Árbol de <elementos> anidados
- ▶ Los elementos pueden tener atributos, contenido textual y comentarios
- ▶ El espaciado extra se ignora


```
<html>
  <head>
    <title>Mi HTML</title>
  </head>
  <body>
    <h1 id="titulo-pagina">Esto es un título</h1>
    <p class="texto-normal">Esto es un párrafo</p>
  </body>
</html>
```

Elementos importantes

- ▶ html
 - ▶ head
 - ▶ title
 - ▶ body
 - ▶ div
 - ▶ span
 - ▶ p
 - ▶ a
 - ▶ img
 - ▶ h1, h2, h3..., h6

Tablas

- ▶ `table` (tabla)
 - ▶ `thead` (grupo de cabecera, opcional)
 - ▶ `tbody` (grupo de cuerpo, opcional)
 - ▶ `tr` (fila)
 - ▶ `th` (celda de cabecera)
 - ▶ `td` (celda de contenido)

Atributos importantes

- ▶ `id` (único en el documento)
- ▶ `class`
- ▶ `href`
- ▶ `src`

Selectores CSS (I)

- ▶ Sintaxis:
 - ▶ elemento
 - ▶ #id
 - ▶ .clase
 - ▶ [atributo]
 - ▶ [atributo="valor"]

Selectores CSS (II)

- ▶ Se pueden anidar y combinar:
 - ▶ body h1
 - ▶ p.texto-normal
 - ▶ body > p
 - ▶ p a[href]

```
<html>
  <head>
    <title>Mi HTML</title>
  </head>
  <body>
    <h1 id="titulo-pagina">Esto es un título</h1>
    <h2>Subtítulo</h2>
    <p class="texto-normal">
      Esto es un párrafo
      
      <a href="/" class="back-link"><span class="enfasis">Volver</span></a>
    </p>
    <div class="pie">
      Copyright Ticnor 2018.
    </div>
  </body>
</html>
```

Extraer datos con Pandas

Pandas

- ▶ Pandas es un módulo para trabajar con *tablas* (DataFrame) de datos.
- ▶ Ofrece métodos para crear DataFrames desde diversas fuentes (CSV, Excel, Web...).

Ejemplo

```
>>> import pandas as pd
>>> dfs = pd.read_html('https://www.sepe.es/contenidos/personas/prestaciones/du
>>> type(dfs)
<class 'list'>
>>> type(dfs[0])
<class 'pandas.core.frame.DataFrame'>
```

```
>>> df = dfs[0]
>>> df
```

	Días de cotización	Días de prestación
0	de 360 a 539	120
1	de 540 a 719	180
2	de 720 a 899	240
3	de 900 a 1079	300
4	de 1080 a 1259	360
5	de 1260 a 1439	420
6	de 1440 a 1619	480
7	de 1620 a 1799	540
8	de 1800 a 1979	600
9	de 1980 a 2159	660
10	desde 2160	720

```
>>> df.loc[3]
Días de cotización    de 900 a 1079
Días de prestación          300
Name: 3, dtype: object
>>> df[df['Días de cotización'] == 'de 900 a 1079']
Días de cotización  Días de prestación
3      de 900 a 1079          300
>>> df[df['Días de prestación'] > 300]
Días de cotización  Días de prestación
4      de 1080 a 1259          360
5      de 1260 a 1439          420
6      de 1440 a 1619          480
7      de 1620 a 1799          540
8      de 1800 a 1979          600
9      de 1980 a 2159          660
10     desde 2160          720
```

BeautifulSoup

BeautifulSoup es un módulo para extraer información (de cualquier tipo) de páginas HTML.

Ejemplo

```
from bs4 import BeautifulSoup
import requests

r = requests.get("https://www.ticnor.es")
data = r.text
soup = BeautifulSoup(data)
for link in soup.find_all('a'):
    print(link.get('href'))
```

findy find_all

- ▶ find -> primer resultado; find_all -> todos
- ▶ Aceptan diversos argumentos para encontrar **elementos**: string, expresión regular (re.compile(...)), listas...


```
# Por nombre
soup.find_all(['th', 'td'])

# Por atributo
soup.find_all(href=re.compile('^http'))

# Combinando
soup.find_all('a', href=True)

# Usando dict de atributos
atributos = {'src': True}
soup.find_all(attrs=atributos)

# Por clase (cualquier clase)
soup.find_all(class_='menu__link')
```

Moviéndonos por el documento

- ▶ Cuando tenemos un elemento localizado con `find_all`:
 - ▶ `find_parents()` / `find_parent()` (ancestros o ancestro)
 - ▶ `find_next_siblings()` / `find_next_sibling()` (hermanos a continuación)
 - ▶ `find_previous_siblings()` / `find_previous_sibling()` (hermanos antes)

Selectores CSS

```
select() y select_one()  
soup.select("title")  
soup.select("a[href]")  
soup.select("h2")  
soup.select(".servicios a")
```

Scrapy

Qué es Scrapy

Scrapy es una librería de Python para hacer *web crawlers* (recolectores de información en HTML).

Ejemplo

```
import scrapy

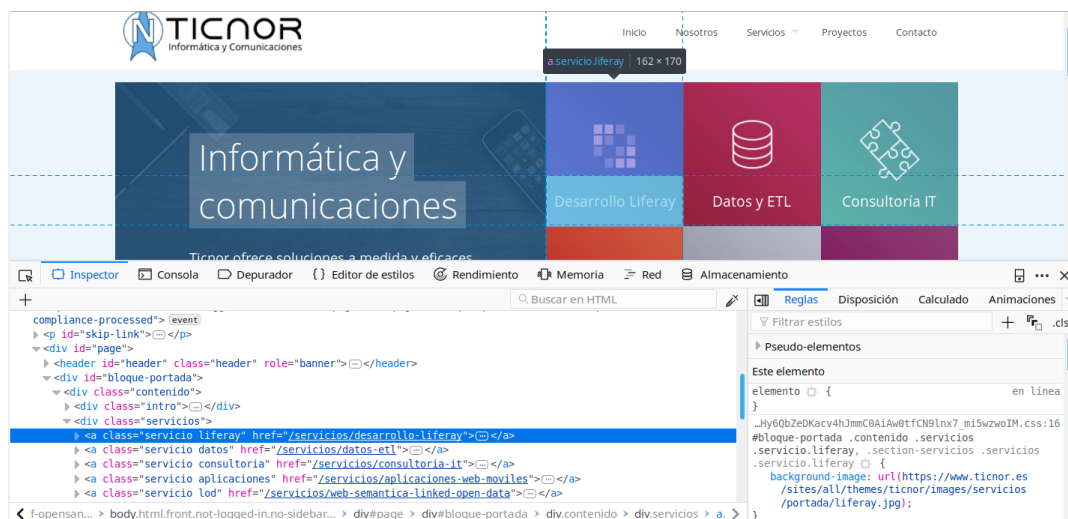
class BlogSpider(scrapy.Spider):
    name = 'blogspider'
    start_urls = ['https://blog.scrapinghub.com']

    def parse(self, response):
        for title in response.css('.post-header>h2'):
            yield {'title': title.css('a ::text').extract_first()}

        for next_page in response.css('div.prev-post > a'):
            yield response.follow(next_page, self.parse)

$ scrapy runspider blogspider.py
```

- ▶ Es necesario tener algo de conocimiento sobre la estructura del sitio que se quiere rastrear:
 - ▶ ¿Qué elementos / atributos / textos queremos extraer?
 - ▶ ¿Qué enlaces (<a>) debemos seguir (paginación, categorías...)?



Mejor método: Inspeccionar Elemento

Procedimiento

1. Definir las URL iniciales
 - ▶ Ej: `https://scholar.google.es/scholar?q=einstein`
2. Localizar los elementos que queremos extraer
 - ▶ Ej: `.gs_ri h3 a ::text`
3. Localizar los enlaces a las siguientes páginas a navegar
 - ▶ Ej: `#gs_n a`

```
import scrapy

class EinstenSpider(scrapy.Spider):
    name = 'einsten_spider'
    start_urls = ['https://scholar.google.es/scholar?q=einstein']

    def parse(self, response):
        for texto in response.css('.gs_ri h3 a ::text'):
            yield {'publicacion': texto.extract()}

        for next_page in response.css('#gs_n a'):
            yield response.follow(next_page, self.parse)

scrapy runspider einstein.py -o resultados.json
```

¿Qué más ofrece scrapy?

- ▶ Extraer con métodos independientes distintas páginas (p.e., navegar listados, pero extraer datos de vistas individuales).
- ▶ Pasar argumentos desde la línea de comandos.
- ▶ Su consola (`scrapy shell https://www.ticnor.es`).

Y más, y más...

- ▶ Exportar a distintos formatos (JSON, JSONL, CSV, XML).
- ▶ Seleccionar con XPath.
- ▶ Integrarlo en módulos (no sólo desde línea de comandos).
- ▶ ...

Ejercicio

Extraer un listado de todas las calderas con su precio de <https://euroclimaonline.es/>

```
[  
  { "modelo": "XYZ", "precio": "335,30 €"},  
  { "modelo": "TRS", "precio": "299,99 €"},  
  ...  
]
```