Calendario

- Expresiones regulares
- Meta caracteres
  - Anclas
  - Clases
  - Rangos
- Clases Predefinidas
- Cuantificadores
- Ejemplos
- Módulo re
  - Pattern, search, sub
- Ejercicios

Expresiones regulares

Las expresiones regulares, también llamadas regex o regexp, consisten en patrones que describen conjuntos de cadenas de caracteres.

Por ejemplo:

- Correos electrónicos
- Números de teléfonos
- Validar cualquier campo de entrada.
- URL de un sitio

El objetivo es validar que en una cadena se cumplan ciertos patrones

Meta caracteres

- Los meta caracteres son caracteres que tienen un significado especial en el contexto de expresiones regulares.
- **Anclas**: Delimitan el principio o final de la cadena:

  Los meta caracteres para usar anclas son ^ (circunflejo) y $ (pesos)

  - ^palabra: coincide con cualquier cadena que comience con “palabra”
  - palabra$: coincide con cualquier cadena que termine con “palabra”.
  - ^palabra$: coincide con la cadena exacta “palabra”.

Meta caracteres (2)

- **Clases de caracteres:** Se utilizan cuando se quiere buscar un carácter en varias opciones.
  - [abc]: coincide con a, b, o c
  - [38ab]: coincide con 3, 8, a ó b
- Rangos: Si queremos encontrar un número, podemos usar una clase como [0123456789], o podemos utilizar un rango. También sirve para letras.
  - [a-c]: equivale a [abc]
  - [0-9]: equivale a [0123456789]
  - [a-d5-8]: equivale a [abcd578]

Meta caracteres (3)

- Rango negado:
  - También podemos listar caracteres que NO deben aparecer.
  - [^abc]: coincide con cualquier carácter distinto de a, b y c
  - El rango sin negar se puede interpretar como un OR lógico, mientas que el rango negado como un AND

Clases Predefinidas

Hay algunas clases que se usan frecuentemente y por eso existen formas abreviadas para ellas.

"\d" : un dígito. Equivale a [0-9]

"\D" : cualquier carácter que NO sea un dígito. Equivale a [^0-9]

"\w" : cualquier carácter alfanumérico. Equivale a [a-zA-Z0-9_]

"\W" : cualquier carácter NO alfanumérico. Equivale a [^a-zA-Z0-9_]

"\s" :cualquier carácter en blanco. Equivale a [\t\n\r\f\v]

"\S" : cualquier carácter que NO sea un espacio en blanco. Equivale a [^ \t\n\r\f\v]

"." : cualquier carácter

Cuantificadores

- Son caracteres que se utilizan para repetir la aplicación del patrón que les precede. Mientras que con las clases de caracteres podemos buscar un dígito, o una letra, con los cuantificadores podemos multiplicar esa búsqueda
  - ?: coincide con cero ó una ocurrencia del patrón (dicho de otra forma: hace que el patrón sea opcional)
  - +: coincide con una o más ocurrencias del patrón
  - *: coincide con cero o más ocurrencias del patrón.

Cuantificadores (2)

- {x}: coincide con exactamente x ocurrencias del patrón
- {x, y}: coincide con al menos x ocurrencias y no más de y
  - Si se omite x, el mínimo es cero, si se omite y, no hay máximo.
  - Esto permite especificar a los otros cuantificadores como casos particulares: ? es {0,1}, + es {1,} y * es {,} ó {0,}.

Otros metacaracteres

- (): Los paréntesis agrupan patrones.
- Ejemplo:
  - ab+ coincide con ab, abb, abbbbb, …, mientras que (ab)+ coincide con ab, abab, abab…
- |: permite definir opciones para el patrón:
  - perro|gato coincide con perro o con gato.

Ejemplos

- .* : cualquier cadena, de cualquier largo (incluyendo una cadena vacía)
- [a-z]{3,6}: entre 3 y 6 letras minúsculas
- \d{4,}: al menos 4 dígitos
- .*hola!?: una cadena cualquiera, seguida de hola, opcionalmente termina con un !
- ((oo)+(aa)+): una cadena de texto formada por la letra “o” intercalada por la letra “a”, pero tiene que haber un número par de “o” y de “a”

Ejemplos (2)

- **(o.o):** todas las palabras de tres letras que empiecen y acaben por “o”, como oso, ojo u oro.
- **[0-9]{2}[/-][0-9]{2}[/-]([0-9]{2}|[0-9]{4}):** fechas con dos dígitos para día y mes, y dos o cuatro dígitos para el año. Además es posible utilizar tanto barras “/” como guiones “-” como carácter separador.
- **(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)*\/?$:** Valida una url

Módulo Re - Patrones

- Vamos a aplicar regex en Python. Para ellos PY provee el módulo re.
- Podemos crear objetos de tipo patrón y generar objetos tipo matcher. Los objetos patron expresan la regex mientras que los matcher contienen información acerca de la coincidencia (o no) con la cadena evaluada.
- Para crear un objeto patrón, importamos el módulo re y utilizamos la función compile:
  - import re
  - patron = re.compile('a[3-5]+')
  - # letra a, seguida de al menos 1 dígito entre 3 y 5

Módulo Re - Matchers

- Para buscar un patrón en una cadena, **Python** provee los métodos **match** y **search**. Ambos se aplican sobre un objeto **Pattern**

  match(string)

  - Trata de aplicar el patrón desde el comienzo de la palabra. Retorna un objeto de tipo match si encuentra una ocurrencia o None en otro caso.

```
patron = re.compile('python')

if patron.match("python"):

   print ("cierto")
```

  - El ejemplo es engañoso, funciona sólo porque None (cuando no machea con el patrón) es evaluado como False en el contexto de un if.
  - Cuando machea, no se devuelve True (no es una función booleana, retorna, como se mencionó arriba, un objeto, que tiene cargada información acerca de la coincidencia (o macheo)

Módulo Re - Matchers (2)

- Tipos devueltos por las funciones compile y match

```
patron = re.compile('a[3-5]+')
print (type(patron)) → <class '_sre.SRE_Pattern'>
print (type(patron.match("a3"))) → <class '_sre.SRE_Match'>
```

- ¿Coincide?

```
if patron.match("a38"):
    print ("cierto")
```

- Si tuviera esta regex ¿qué pasa con el match anterior? ¿Coincide?
  patron = re.compile('a[3-5]+$')

Módulo Re - Buscar patrón en cadena

search(string)

- Busca dentro del string que se cumpla con el patrón definido. Retorna un objeto del tipo match si encuentra ocurrencia, en otro caso None
- Funciona de forma similar a match, la única diferencia es que al utilizar **match** la cadena debe ajustarse al patrón desde el primer carácter. Con **search** buscamos cualquier parte de la cadena que se ajuste al patrón.
- **search** al igual que **match**, se detienen en la **1era ocurrencia**

Ejemplos

- Si quisiéramos comprobar si la cadena es “python” , “jython” ,“cython”.
- Utilizamos el comodín “.”
  - patron.match(“jython”)
  - patron.match(“hola jython”)
  - patron = re.compile('.ython')
  - patron.match(“python”)
  - patron.search(“hola jython”)

Ejemplos (2)

- Que comience con python y termine con un número
  - patron = re.compile(“python[0-9]”)
  - patron.match(“python0”)
- Que comience con python y que no termine con un numero o una letra minúscula.
  - patron = re.compile(“python[^0-9a-z]”)
  - patron.match(“pythonUY”)

Capturar la(s) coincidencia(s)

Ejemplo para entender qué datos tiene cargados el objeto devuelto por función search

```
regex = re.compile("([a-zA-Z]+) (\d+)")
match = regex.search("bla June 24 bla")
if match:
    print ("Index {},{}".format((match.start(),match.end()))
    #salida: Index 4, 11
    print ("Tuple match: {}".format(match.groups()))
    #salida: Tuple match: ('June', '24')
    print ("Full match: {}".format(match.group(0)))
    #salida: June 24
    print ("Month: {}".format(match.group(1)))
    #salida: "June"
    print ("Day: {}".format( match.group(2)))
    #salida: "24"
```

Capturar la(s) coincidencia(s) (2)

- Para obtener todas las ocurrencias debemos utilizar la función **findAll**

```
regex = re.compile("([a-zA-Z]+) (\d+)+")
matches = regex.findall("June 24, August 9, Dec 12")
for match in matches:
    print (match)
#salida ('June', '24') ('August', '9')('Dec', '12')
```

- Para obtener las posiciones: **finditer**

```
matches = regex.finditer("June 24, August 9, Dec 12")
for match in matches:
  print ("init {},fin {}".format(match.start(),match.end()))
# salida: init 0,fin 7 init 14,fin 22 init 24,fin 30
```

Reemplazo de cadenas: sub

- El método **sub** tiene como función encontrar todas las coincidencias de un patrón y sustituirlas por una cadena.
- Recibe dos parámetros:
  - la cadena con la que se sustituirá el patrón y
  - la cadena sobre la que queremos aplicar la sustitución.

```
regex = re.compile("(\w+) World")
s = regex.sub("Earth", "Hello World que tal”)
print (s)  #salida: Earth que tal
```

Reemplazo de cadenas: sub (2)

- Utilizando el parámetro *count*

```
oracion = 'la norma es la norma'

patron = re.compile('norma')

print(patron.sub('ley', oracion))  # la
ley es la ley


patron = re.compile('la')

print(patron.sub('LA', oracion, count=1))

# LA norma es la norma
```

Separar strings con regex: split

```
value = "one 1 two 2 three 3"
# Separate on one or more non-digit characters.
regex = re.compile("\d+")
result = regex.split(value)
for element in result:
    print(element)

#salida:
one
two
three
```

Ejercicios

- **Ejercicio 1:**
  - Escribir una función llamada **validar_correo** que dado un string retorne verdadero si el correo es del formato palabra@dominio.com.xx
- **Ejercicio 2:**
  - Escribir una función **validar_dirección** que dado un string retorne verdadero si la dirección web es del formato http://www.dominio.com.xx o http://dominio.com.xx