programa de BECAS
DGTIC
de FORMACIÓN en TIC

Seguridad Informática

Programación con Python

Jesús Enrique Pacheco Franco
Servando Miguel López Fernandez

Expresiones regulares

- Son una herramienta muy potente para la manipulación de cadenas.
- Las regex están presentes en la mayoría de los lenguajes de programación.
- Son útiles para:
  - Validar que una cadena cumple con un formato
  - Realizar sustituciones en una cadena
- Pueden ser usadas mediante el módulo ‘re’
- Algunas funciones útiles son:
  - Match
  - Find
  - Findall
  - Sub

Expresiones regulares - match

- Por comodidad, usaremos cadenas “puras/crudas” para representar expresiones regulares. Es decir, cadenas que no escapen su contenido:

a = r”esto es una regex”

- La función **match** permite validar si el principio de una cadena concuerda con una expresión regular.

```
>>> import re
>>> patron = r"spam"
>>> if re.match(patron,"spamhaus"):
...     print 'SI'
... else:
...     print 'NO'
...
SI
```

Expresiones regulares - search

- La función **search** valida si hay una coincidencia del patrón en cualquier parte de la cadena, y no únicamente al inicio.

```
>>> import re
>>> patron = r"becario"
>>> if re.search(patron,"yo soy un becario"):
...     print "SI"
... else:
...     print "NO"
...
SI
>>>
```

Expresiones regulares - findall

- La función **findall** regresa una lista de todas las subcadenas que coinciden con el patrón indicado.

```
>>> import re
>>> patron = r"python"
>>> re.findall(patron, 'python rocks! python rocks!')
['python', 'python']
>>>
```

Expresiones regulares

- La función **sub** busca un patrón en una cadena y lo substituye por otra cadena.

```
>>> import re
>>> patron = r"aprobar"
>>> cadena = "Voy a aprobar"
>>> nueva_cadena = re.sub(patron,"reprobar", cadena)
>>> nueva_cadena
'Voy a reprobar'
>>>
```

Clases de caracteres

- Las clases de caracteres ofrecen una forma sencilla de especificar un conjunto de caracteres.
- Una clase de caracteres se crea con corchetes.
- Para validar la coincidencia, se toma cualquier carácter de la clase.

```
>>> from re import findall
>>> patron = r"[aeiou]"
>>> findall(patron, "voy a reprobar")
['o', 'a', 'e', 'o', 'a']
>>>
```

Clases de caracteres

- Las clases permiten rangos de caracteres.
  - [a-z] coincide con cualquier letra minúscula
  - [0-9] coincide con cualquier dígito
  - [A-Za-z] coincide con cualquier letra
  - [^F-M] coincide con cualquier carácter menos las mayúsculas entre F y M

```
>>> from re import search
>>> patron = r"[a-z][a-z][a-z][0-9]"
>>> if search(patron,"aaf8"):
...     print "coincide"
...
coincide
>>> if search(patron,"aa8f"):
...     print "coincide"
```

Expresiones regulares - metacaracteres

Meta carácter	Significado
.	Cualquier carácter, excepto salto de línea
^	Inicio de una cadena
$	Final de una cadena
*	Cero o más repeticiones de “lo anterior”. Lo anterior puede ser un carácter, una clase de caracteres o un grupo
+	Una o más repeticiones de “lo anterior”
?	Cero o una repetición de “lo anterior”
{}	Las llaves se usan para indicar el número de repeticiones entre dos números de “lo anterior”. Por lo tanto {0,1} es equivalente a ?

Expresiones regulares - grupos

- Es posible agrupar ciertas parte de una expresión regular usando paréntesis para acceder a esa coincidencia en específico

```
In [14]: import re

In [15]: pattern = r':?([^:]+)'

In [16]: passwd = 'username:password:uid:gid:gecos:home:shell'

In [17]: re.findall(pattern, passwd)
Out[17]: ['username', 'password', 'uid', 'gid', 'gecos', 'home', 'shell']
```

Expresiones regulares - metacaracteres

- Cualquier carácter, excepto minúsculas, una o más veces
  - [^a-z]+
- 3 mayúsculas, seguidas de cualquier carácter
  - [A-Z]{3}.
  - [A-Z][A-Z][A-Z].
- Cualquier carácter al menos 4 veces, seguido de 2 minúsculas, seguido de uno o más dígitos
  - .{4,}[a-z]{2}[0-9]+

Expresiones regulares

- 4 mayúsculas o minúsculas, seguidas de 4 dígitos
- Cualquier carácter excepto “$” una o más veces, seguido de “$”
- Los comentarios multilínea en lenguaje C
- Un posible nombre de variable en Python
- Correos con el formato de correos del CERT

Caso I - Números Romanos

Expresión regular que valide en números romanos hasta el 3999.

$I = 1$

$V = 5$

$X = 10$

$L = 50$

$C = 100$

$D = 500$

$M = 1000$

CM

CD

De cero a tres caracteres C (Cero si el lugar de las centenas vale cero).

D, seguido de cero a tres caracteres C.

Los dos últimos patrones se pueden combinar:

Una D opcional, seguida de cero a tres caracteres C.

Expresiones Regulares Compactas vs Detalladas

Todas las expresiones regulares que hemos visto hasta el momento son expresiones regulares Compactas.

Existe la manera de documentar nuestras expresiones regulares para ofrecer ayuda respecto a que es lo que esta recuperando la expresión regular.

Dos diferencias:

- Se ignoran los espacios en blanco (espacios, tabuladores, retornos de carro y saltos de línea).
- Se ignoran los comentarios.

Expresión Regular Detallada

```
pattern = r'''
    ^                   # Comienzo de la Cadena
    M{0,3}              # Unidades de Millar - 0 a 3 M
    (CM|CD|D?C{0,3}) # Centenas - 900 (CM), 400 (CD), 0-300 (0 a 3 C) o 500 - 800 (D, seguido de 0 a 3 C)
    (XC|XL|L?X{0,3}) # Decenas -  90 (XC), 40 (XL), 0-30 (0 a 3 X) o 50 - 80 (L, seguido de 0 a 3 X)
    (IX|IV|V?I{0,3}) # Unidades - 9 (IX), 4 (IV), 0-3 (0 a 3 I) o 5 - 8 (V, seguido de 0 a 3 I)
    $                   # Fin de la Cadena
  '''
```

Caso II – Números Telefónicos

Código de Área Troncal Resto Extensión

800-555-1212

888 555 1212

800.555.1212

(800) 555-1212

800-555-1212-1234

800-555-1212x1234

800-555-1212 ext. 1234

Trabajo 1-(800) 555.1212 #1234

Ejercicio de clase ?

- Hacer una expresión regular que coincida con una dirección IP versión 4

- Hacer expresión regular que coincida con una dirección de correo electrónico

Algunos “trucos” pythonicos

- Desempaque de tuplas

```
>>> a,b,c = 1,2,3
>>> a,b,c
(1, 2, 3)
>>> a,b,c = [1,2,3]
>>> a,b,c
(1, 2, 3)
>>> a
1
```

Algunos “trucos” pythonicos

- Slices, la indexación puede ser con números positivos y con números negativos

```
>>> lista1 = [1,2,3,4,5]
>>> lista1[1:3]
[2, 3]
>>> lista1[-4:-2]
[2, 3]
>>>
```

Algunos “trucos” pythonicos

- Asignación usando slices

```
>>> lista1 = [1,2,3,4,5]
>>> lista1[1:3] = [20,30]
>>> lista1
[1, 20, 30, 4, 5]
>>> lista1[-2:] = [40,50]
>>> lista1
[1, 20, 30, 40, 50]
```

Algunos “trucos” pythonicos

- Compresión de listas

```
>>> a = [1,2,3,4]
>>> b = ['a','e','i','o','u']
>>> zip(a,b)
[(1, 'a'), (2, 'e'), (3, 'i'), (4, 'o')]
>>>
```

Entorno virtual

- Un entorno virtual permite usar versiones diferentes de python
- Se pueden instalar módulos de python sin necesidad de instalarlos en el PATH del sistema operativo
- Es ampliamente usado en aplicaciones como Django

Entorno virtual

- Instalar virtualenv con el manejador de paquetes del sistema operativo

```
CURSO-PYTHON> sudo apt install -y virtualenv
Reading package lists... Done
Building dependency tree
Reading state information... Done
virtualenv is already the newest version (15.1.0+ds-1).
0 upgraded, 0 newly installed, 0 to remove and 140 not upgraded.
CURSO-PYTHON>
```

Entorno virtual

- Crear directorio que contendrá el ambiente virtual y posteriormente crear el ambiente virtual

```
CURSO-PYTHON> mkdir venv-curso
CURSO-PYTHON> virtualenv venv-curso/
Running virtualenv with interpreter /usr/bin/python2
New python executable in /home/prueba/curso_python/dia4/venv-curso/bin/
python2
Also creating executable in /home/prueba/curso_python/dia4/venv-curso/b
in/python
Installing setuptools, pkg_resources, pip, wheel...done.
```

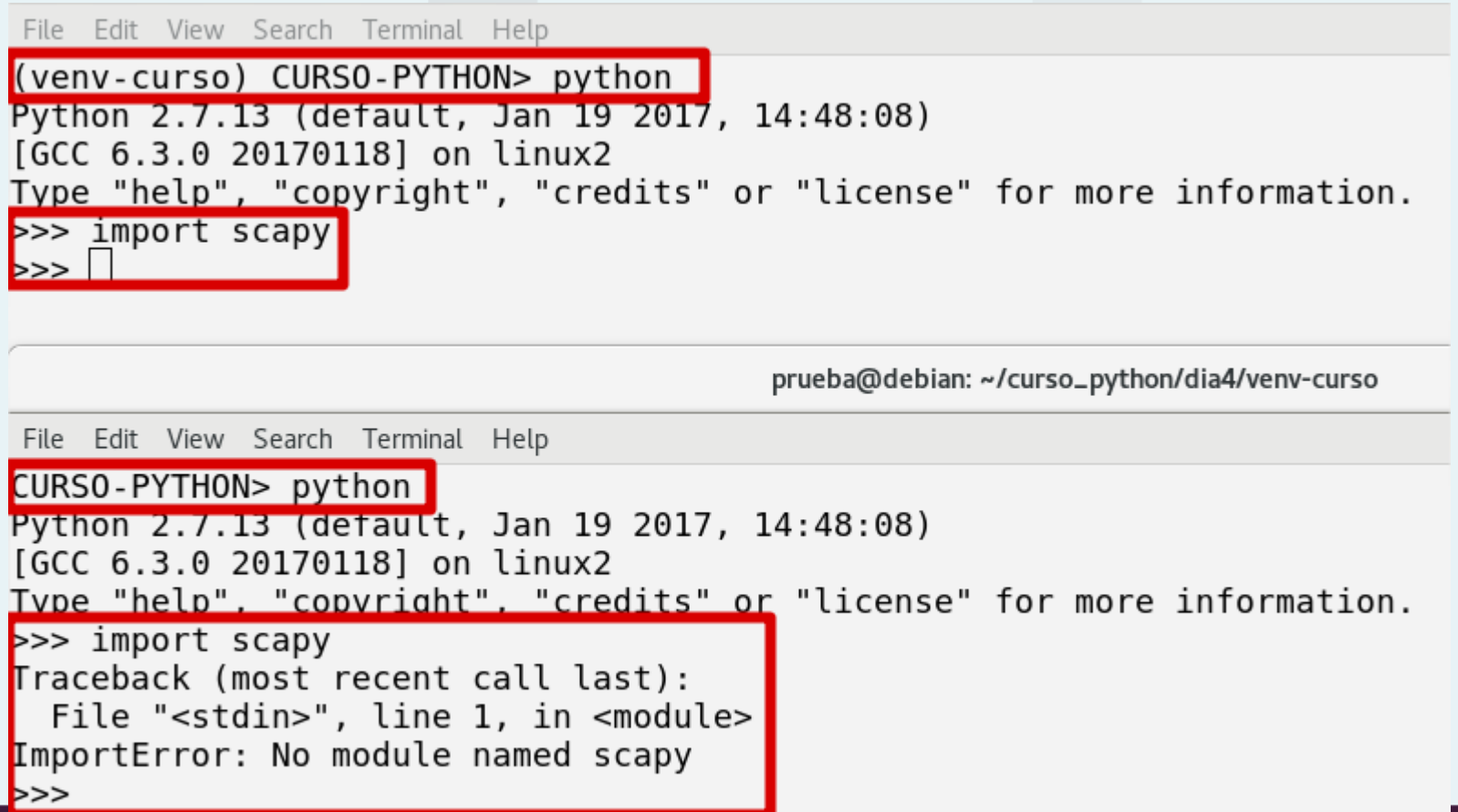
Entorno virtual

- Acceder al ambiente usando “source”. El ejecutable “actívate” es el encargado de lograr esto.

```
CURSO-PYTHON> source venv-curso/bin/activate
(venv-curso) CURSO-PYTHON>
```

Entorno virtual

- Es posible instalar módulos en el ambiente virtual que no se encontrarán en el resto del sistema.
- Los ejecutables instalados se crearán en la carpeta del ambiente, en este caso es “venv-curso”



Entorno virtual

- Es posible crear un ambiente virtual para versiones diferentes de python.

```
File  Edit  View  Search  Terminal  Help
CURSO-PYTHON> mkdir venv-python3
CURSO-PYTHON> virtualenv -p /usr/bin/python3 venv-python3/
Already using interpreter /usr/bin/python3
Using base prefix '/usr'
New python executable in /home/prueba/curso_python/dia4/venv-python3/bin/python3
Also creating executable in /home/prueba/curso_python/dia4/venv-python3/bin/python
Installing setuptools, pkg_resources, pip, wheel...done.
CURSO-PYTHON>
```

Entorno virtual

- Ahora, la versión de python por defecto en el ambiente virtual es la 3.
- No es necesario usar el comando “python3” para ejecutar scripts de esta versión.

```
CURSO-PYTHON> source venv-python3/bin/activate
(venv-python3) CURSO-PYTHON> python
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170118] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Python 2 vs Python 3

- La filosofía del lenguaje es exactamente la misma sin importar la versión, sin embargo si hay múltiples diferencias a considerar.

- Python 2 sigue siendo muy utilizado pues es la versión por defecto en la mayoría de las distribuciones de GNU/Linux y MacOS, sin embargo es buena idea usar las versiones más nuevas.

- La principal desventaja de Python 3 es que hay módulos de python 3 que aún no tienen soporte en python 3.

Python 2 vs Python 3 - print

- En python 2, “print” es una instrucción.
- En python 3, “print” es una función, por lo que los paréntesis son obligatorios.

```
CURSO-PYTHON> python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170118] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print 'hello world'
  File "<stdin>", line 1
    print 'hello world'
                       ^
SyntaxError: Missing parentheses in call to 'print'
>>> print('hello world')
hello world
>>>
```

Python 2 vs Python 3 – división entera

- En python 2, la división entre enteros resulta en un entero
- En python 3, la división entre enteros resulta en un flotante

```
CURSO-PYTHON> python
Python 2.7.13 (default, Jan 19 2017, 14:48:08)
[GCC 6.3.0 20170118] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 15/2
7
>>>
```

prueba@debian: ~/curso_python/dia4

File Edit View Search Terminal Help

```
CURSO-PYTHON> python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170118] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 15/2
7.5
>>>
```

Python 2 vs Python 3 – range

- En python 2, la función range genera una lista con los elementos indicados
- En python 3, la función range genera un objeto de la clase “Range”. Se debe convertir explícitamente a una lista

```
CURSO-PYTHON> python
Python 2.7.13 (default, Jan 19 2017, 14:48:08)
[GCC 6.3.0 20170118] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> range(4,10)
[4, 5, 6, 7, 8, 9]
>>>
```

prueba@debian: ~/curso_python/dia4

File Edit View Search Terminal Help

```
CURSO-PYTHON> python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170118] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> range(4,10)
range(4, 10)
>>> list(range(4,10))
[4, 5, 6, 7, 8, 9]
>>>
```

Python 2 vs Python 3 – input

- En python 2, la función input podía regresar valores de diferentes tipos. Para asegurar una cadena, se usa raw_input()
- En python 3, la función input regresa siempre una cadena

```
CURSO-PYTHON> python
Python 2.7.13 (default, Jan 19 2017, 14:48:08)
[GCC 6.3.0 20170118] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> a = input()
4
>>> type(a)
<type 'int'>
>>> a = raw_input()
4
>>> type(a)
<type 'str'>
```

prueba@debian: ~/curso_python/dia4

File Edit View Search Terminal Help

```
CURSO-PYTHON> python3
Python 3.5.3 (default, Jan 19 2017, 14:11:04)
[GCC 6.3.0 20170118] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = input()
4
>>> type(a)
<class 'str'>
>>>
```

PYTHON ROCKS