**EXPRESIONES REGULARES**

[lesson del profe](https://drive.google.com/drive/folders/16i3adHMdlHJQNJKTi2-i7E3jgKho-SOO) → python\_string\_operations

[internet doc python](https://docs.python.org/3/library/re.html), [internet doc google](https://developers.google.com/edu/python/regular-expressions)

Lo primero es importar su librería 

**Character Classes:**

**[a-z]**: Any lowercase letter between a and z.

**[A-Z]:** Any uppercase letter between A and Z.

**[0-9]:** Any numeric character between 0 and 9.

**\w:** Any alphanumeric character.

**\W:** Any non-alphanumeric character.

**\d:** Any numeric character.

**\D:** Any non-numeric character.

**\s:** Any whitespace characters.

**\S**: Any non-whitespace characters.

**.** Any character except newline (\n).

**Meta-characters:**

**[]**: Match set of characters

**.** Match any character except newline (\n)

**^** Match characters not listed if within set or match **beginning** of line

**[^ ]**:everything that doesn't match the sequence we have designated.

**$**: Match end of line

**|:** Functions as an "OR" operator

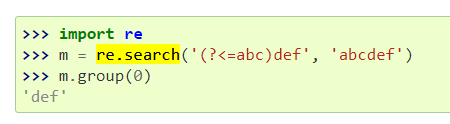
**Quantifiers:**

**\*:** Matches previous character 0 or more times

**+**: Matches previous character 1 or more times

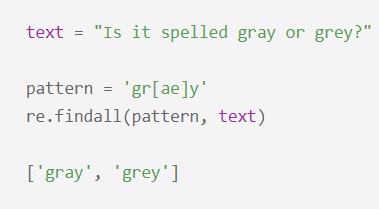
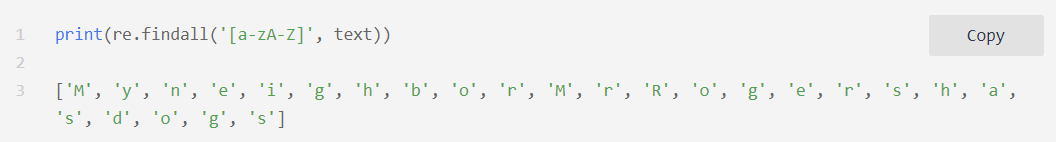
**?**: Matches previous character 0 or 1 times (optional)

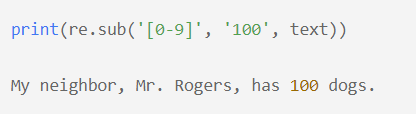
**{}**: Matches previous characters however many times specified within:

* **{n}** : Exactly n times
* **{n,}** : At least n times
* **{n,m}** : Between n and m times
* **re.search(r“patron regex”, str):** BUSCA un PATRÓN (**first instance**, osea que en cuanto lo cumpla una vez para) y **DEVUELVE** EN UN **STRING**
  + varx = **re.search(r“patron”, str)** → devolvería true o false
  + **if else**
  + varx**.group()** → devuelve el patrón (podríamos hacer un print o meterlo dentro de una nueva variable)
* **re.findall:** BUSCA **all instances** of an expression in a string and return them as a list. → BUSCA y **DEVUELVE** EN UNA **LISTA**
* **re.findall(“*palabras regex*”, variable)** → Busca todas las palabras que coincidan de una expresión y las devuelve como una lista



* **re.findall(“*[letras regex]*”, variable)** → Busca todas las letras que coincidan de una expresión y las devuelve como una lista



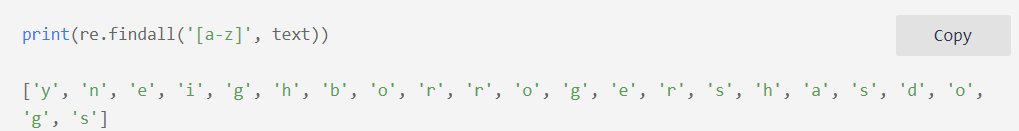
* **re.split(“caracter regex”, variable) →:** Splits a string based on a specified delimiter. → BUSCA y **DIVIDE** EN UNA **LISTA**
* **re.sub(“regex a buscar”,“sustituir”, variable) →:** Substitutes a string/substring with another → BUSCA y **SUSTITUYE** EN UN **STRING**
* **re.match(“regex a buscar”,variable) →:** Busca y devuelve un string

**ALGUNOS EJEMPLOS:**

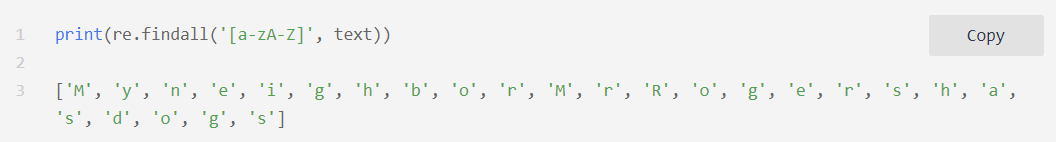
* **re.findall(“*palabras*”, variable)** → Busca todas las palabras que coincidan de una expresión y las devuelve como una lista

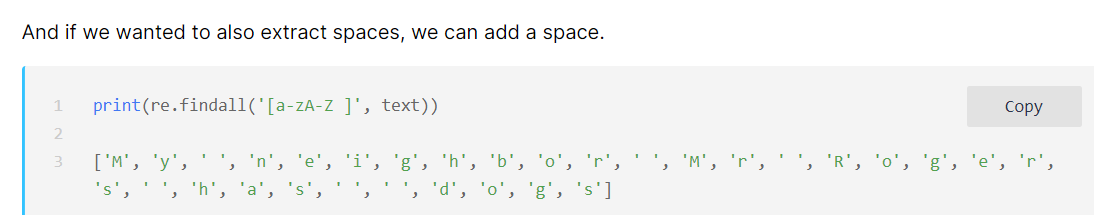


* **re.findall(“*[letras]*”, variable)** → Busca todas las letras que coincidan de una expresión y las devuelve como una lista
* **[a-z]:** Any lowercase letter between a and z.

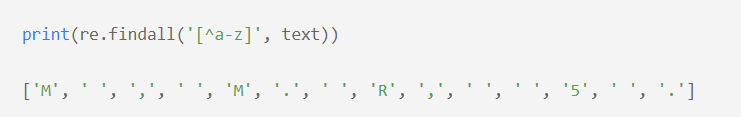


* **[A-Z]:** Any uppercase letter between A and Z.

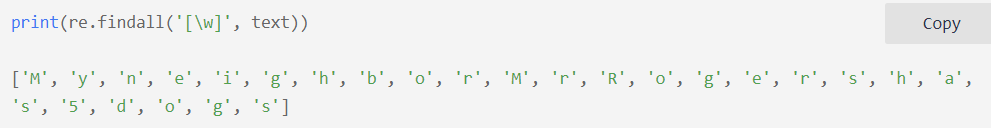




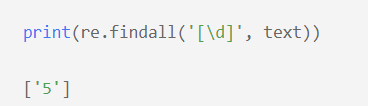
* **[0-9]:** Any numeric character between 0 and 9.
* **[^ ]:**everything that **doesn't match** the sequence we have designated.



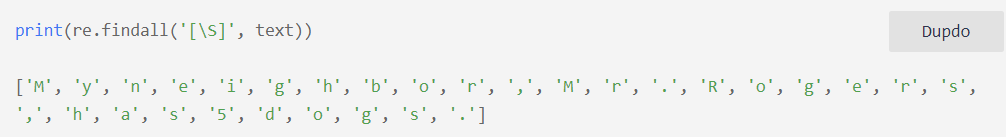
* **\w:** “w” de word → Any alphanumeric character.
* **\W:** Any non-alphanumeric character.



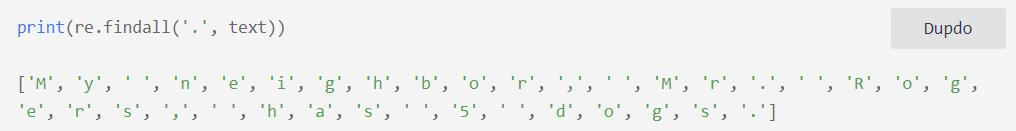
* **\d:** “d” de word → Any numeric character.
* **\D:** Any non-numeric character.



* **\s:** (espacios) → Any whitespace characters.
* **\S:** Any non-whitespace characters.

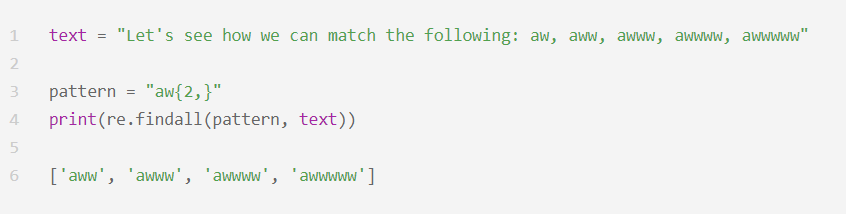


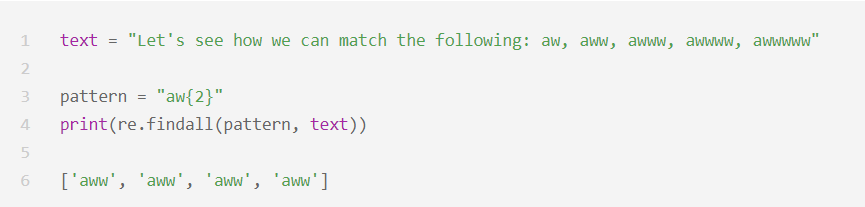
* .: Any character (incluidos los espacios) except newline (\n).



**{}**: Matches previous characters however many times specified within:

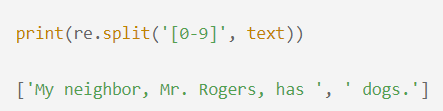
* **{n}** : Exactly n times
* **{n,}** : At least n times
* **{n,m}** : Between n and m times



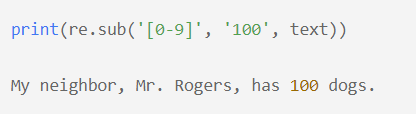


* **re.split(“*caracter*”, variable)** → Busca y divide lo que le indiques en una lista





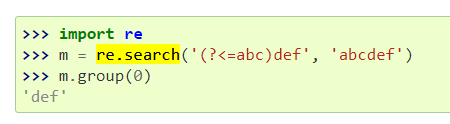
* **re.subs(“*a buscar*”,“*sustituir*”, variable)** → Busca y divide lo que le indiques en una lista



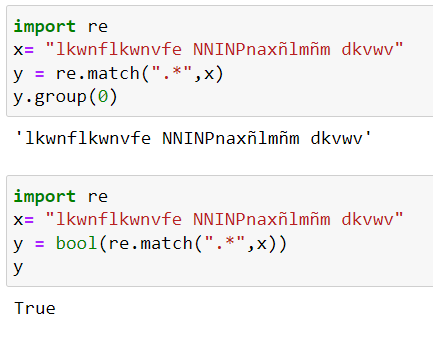
* **re.search(r“**word:\w\w\w**”, str)** → toma un patrón de expresión regular y una cadena y busca ese patrón dentro de la cadena. Si la búsqueda tiene éxito, search () devuelve un objeto de coincidencia o None de lo contrario. Por lo tanto, la búsqueda generalmente es seguida inmediatamente por una instrucción if para probar si la búsqueda tuvo éxito

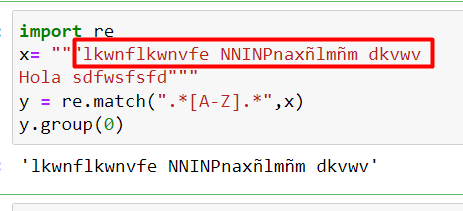


1. varx = **re.search(r“patron”, str)** → devolvería true o false
2. **if else**
3. varx**.group()** → devuelve el patrón (podríamos hacer un print o meterlo dentro de una nueva variable)

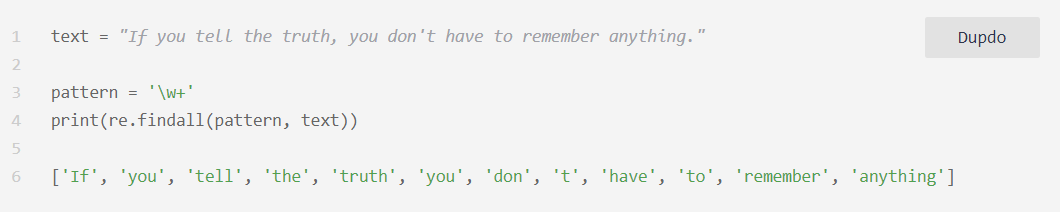


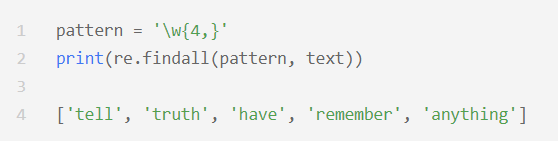
* **re.match(“regex a buscar”,variable) →:** Busca y devuelve un string de la primera linea del string (sin parar en la primera instancia)

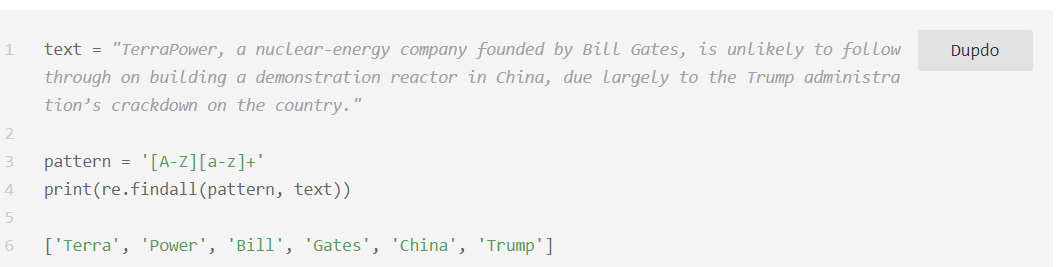




## **ESCRIBIR EXPRESIONES REGULARES MÁS COMPLEJAS**



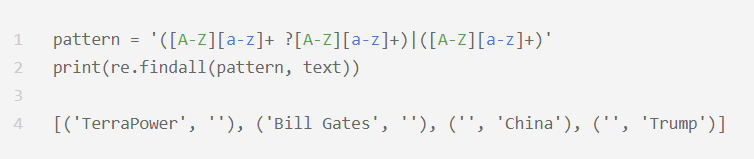




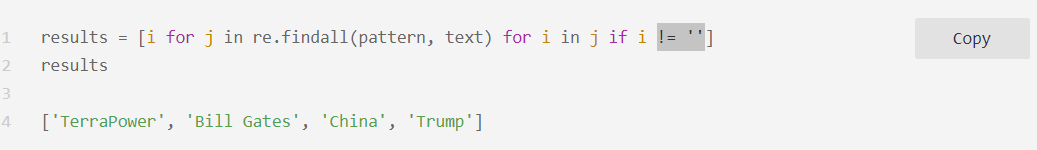
#### **Grouping**

Cuando utilizamos un “|” coincidirá con TODA la regex de atrás o TODA la siguiente regex.

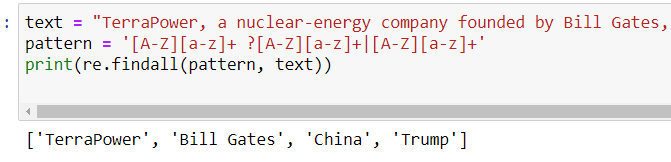
→ Si le ponemos () aislamos, pero nos devuelve un TUPLE con el contenido de la regex y su posición en el or



→ tendriamos que limpiarlo, por ejemplo iterando y guardando lo que no coincida con “” (!= '')



Si no ponemos los () nos daría el siguiente resultado:



EN INGLÉS POR SI HE DICHO ALGO MAL:

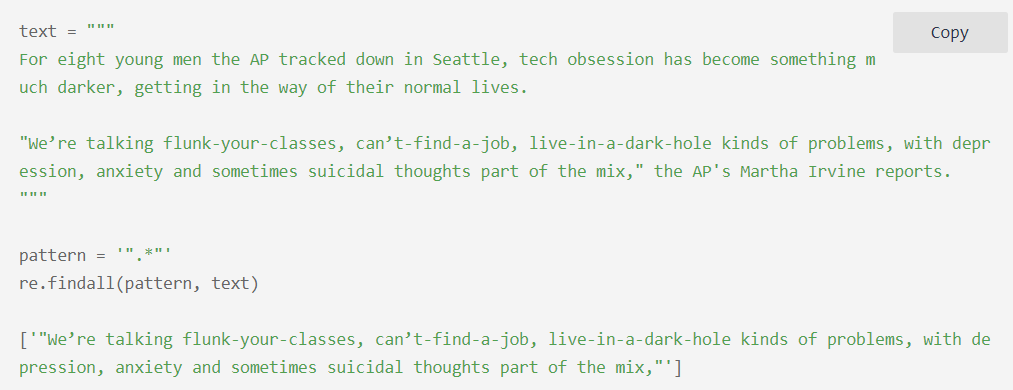
In the example above, the pattern we are defining is looking for an uppercase letter followed by a series of one or more lowercase letters. This gives us the individual capitalized words, which is good, but ideally, TerraPower and Bill Gates should be grouped together. It turns out that we can improve upon this by using groups. Grouping is done by enclosing regular expression components that belong together within parentheses, and it allows us to create more complex regular expressions like the kind we would need to group consecutively capitalized words together.

In this example, we have two groups in our regular expression. The first matches a capitalized first letter followed by a series of lowercase characters, an optional space, and then another capitalized first letter and series of lowercase characters (essentially two capitalized words optionally separated by a space). The second expression simply matches a single capitalized word, and the OR operator in between them specifies that one or the other should be returned if matched. You can see that the results are formatted as a list of tuples where the result is positioned as the first element if it matches the first grouped expression and the second element if it matches the second grouped expression.

If you didn't care for the nested structure and just wanted a single neat list with the results, you could use a list comprehension to obtain that as follows.

### **Extracting Quotes from Text**

* SI TENEMOS UN SALTO DE LINEA (\n)



### **Extracting Formatted Numbers**



