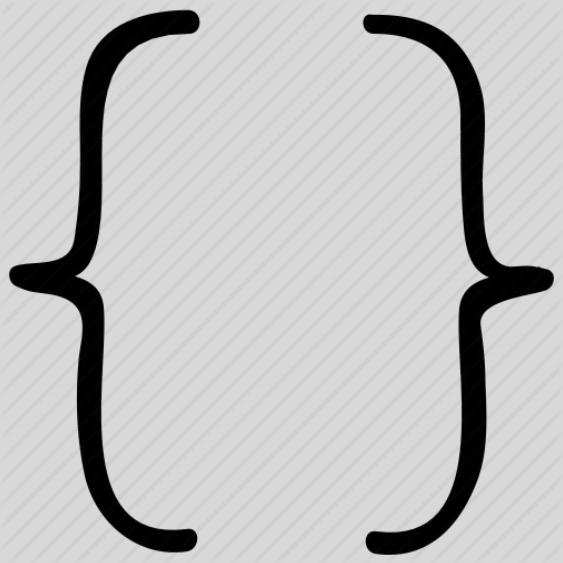

DATA SCIENCE

MÓDULO 1

Python Avanzado.

Definición de Funciones
Listas por Comprensión
Módulos en Python


Repaso / Tips
Ejercicio
Resolución independiente
Puesta en común

REPASO: INDENTACIÓN EN PYTHON

Python tiene reglas estrictas de indentación. En Python la indentación se usa para definir donde termina la ejecución de las estructuras de control (if … else, for, while) y también la definición de funciones.

Se puede indentar con un TAB o con exactamente 4 espacios.

```
a = 2
if a == 1:
    print("Dentro del if")
    print("También dentro del if")
print ("Fuera del if")
```

```
Fuera del if
```

```
a = 0
while a < 3:
    a = a + 1
    print("Dentro del while")
  print("Indentación incorrecta")
print ("Fuera del while")
```

```
  File "<tokenize>", line 5
    print("Indentación incorrecta")
    ^
IndentationError: unindent does not match any outer
```

BREVE REPASO:

Listas

- Una lista es una secuencia de datos mutable.
- Está contenida por corchetes, por ejemplo, [1, 2, 3].
- Puede contener objetos de distinta clase

Tuplas

- Es una secuencia fija e inmutable de valores. (1,2,3)

Diccionarios

- Consisten en pares de elementos que contienen una clave (key) y un valor. Las { } encierran diccionarios. El primer elemento del par de un diccionario es el key; el segundo es el valor. {"clave1":1 , "clave2":2}

Las funciones son fragmentos de código que devuelven un valor. Opcionalmente pueden recibir o no parámetros.

- **Reutilización**
- **Claridad**
- **Menos propenso a errores**
- **Abordaje a problemas en partes / división de problemas**

Recap de sintaxis de funciones:

Sintaxis

```
def functionName(parameters):
    "Function docstring"
    function_suite
    return [expression]
```

Ejemplo

```
def printme(str):
    "Imprime el parámetro que recibe esta función"
    print str
    return
```

La función del ejemplo no devuelve ningún valor en la sentencia *return*. El ejemplo de más abajo es la función identidad:

```
In [1]: def identidad(variable):
            return variable

In [2]: identidad(1)
Out[2]: 1
```

Un conjunto descripto por **extensión** es aquel que enumera uno a uno a todos sus elementos.

Por ejemplo,

C = {2, 4, 6, 8, 10}

S = {1, 4, 9, 16, 25, 36, 49, 64, 81, 100}

Un conjunto descripto por **comprensión** es aquel que determina las propiedades que caracterizan a sus elementos

Por ejemplo,

C = {n | n es un número par y 1 ≤ n ≤ 10}

S = {n | n = k^2, k es natural y n ≤ 100}

La sintaxis de las listas por comprensión es: [*expresión* for *nombre* in *lista* [if *condicion*]]

Ejemplo: [*n* for *n* in *lista* if *n%2==0*]

- **Facilidad y velocidad para el data scientist**: Las Listas por comprensión en Python son una sintaxis simple y poderosa que, una vez dominada, permite una manipulación rápida, eficiente e intuitiva de tipos de datos como arreglos o listas.

- **Código conciso y fácil de leer**: Puede ser que las listas por comprensión resulten confusas en un principio. Sin embargo, una vez adquirido el hábito y comprendidas tornan un código complejo en uno conciso y fácil de leer.

- **Reemplaza las estructuras iterativas (while / for)**: Las listas por comprensión son esencialmente un reemplazo para sentencias de control iterativas. Compararemos las alternativas de utilizar y no utilizar comprensión para ilustrar su funcionamiento y sus ventajas.

Son conjuntos de recursos que se incorporan en un sistema para aprovechar su funcionalidad.

- **Reutilización entre distintos sistemas**
- **Estandarización**
- **Aprovechar desarrollos complejos que ya están hechos**
- **Trabajar en línea con la comunidad**

Algunas de las librerías que vamos a usar en este curso

- **NumPy**
  - Permite trabajar de manera eficiente con operaciones matemáticas sobre arrays
- **Pandas**
  - Se usa para análisis y manipulación de datos como tablas (dataframes)
- **Matplotlib**
  - Es una librería de visualización, para gráficos y tableros
- **Seaborn**
  - Otra librería de visualización, basada en matplotlib

Además de utilizar librerías externas, podemos desarrollar nuestros propios módulos y llamarlos de la misma manera desde las notebooks.

Ejemplo de un módulo sencillo que contiene dos funciones. Este código se guarda en el archivo fibo.py.

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

Si importamos el contenido del módulo, podemos usar las funciones desde el código.

```
In [1]: # Desde la notebook importamos el módulo fibo
        import fibo

In [2]: # Y usamos las funciones desde el código
        fibo.fib(5)

        1
        1
        2
        3
```