

Arquitecturas Cloud y Big Data



red.es

Centro de
Referencia Nacional
en Comercio Electrónico
y Marketing

CRN
Digital



"El FSE invierte en tu futuro"
Fondo Social Europeo

Crea un repositorio en GitHub llamado **curso Big Data** y sube todos los ejercicios resueltos separados por carpetas con un nombre descriptivo del ejercicio.

Además, crea un archivo **README.md** donde deberás escribir:

- Lenguaje utilizado.
- Lista de ejercicios con una descripción acotada.



Funciones lambda (o anónimas) de Python

- Son funciones anónimas. Por ejemplo, para sumar dos números:

```
lambda a, b: a + b
```

- Se pueden usar cuando haya que pasar una función como parámetro
- Tienen una única instrucción cuyo valor corresponde al valor devuelto

Introducción a PySpark: RDDs Master-Worker



RDD

Closure

SPARK: aportaciones

- Spark es una plataforma de computación para clústers
- Es de propósito general.
- Desarrollo simplificado
- Trabaja en memoria
- Rápido
- Permite trabajo interactivo, streaming...



SPARK: Java, Scala, Python, R

```
1 package org.myorg;
2
3 import java.io.IOException;
4 import java.util.*;
5
6 import org.apache.hadoop.fs.Path;
7 import org.apache.hadoop.conf.*;
8 import org.apache.hadoop.io.*;
9 import org.apache.hadoop.mapreduce.*;
10 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
11 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
12 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
13 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
14
15 public class WordCount {
16
17     public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
18         private final static IntWritable one = new IntWritable(1);
19         private Text word = new Text();
20
21         public void map(LongWritable key, Text value, Context context) throws IOException {
22             String line = value.toString();
23             StringTokenizer tokenizer = new StringTokenizer(line);
24             while (tokenizer.hasMoreTokens()) {
25                 word.set(tokenizer.nextToken());
26                 context.write(word, one);
27             }
28         }
29     }
30
31     public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
32
33         public void reduce(Text key, Iterable<IntWritable> values, Context context)
34             throws IOException, InterruptedException {
35             int sum = 0;
36             for (IntWritable val : values) {
37                 sum += val.get();
38             }
39             context.write(key, new IntWritable(sum));
40         }
41     }
42
43     public static void main(String[] args) throws Exception {
44         Configuration conf = new Configuration();
45
46         Job job = new Job(conf, "wordcount");
47
48         job.setOutputKeyClass(Text.class);
49         job.setOutputValueClass(IntWritable.class);
50
51         job.setMapperClass(Map.class);
52         job.setReducerClass(Reduce.class);
53
54         job.setInputFormatClass(TextInputFormat.class);
55         job.setOutputFormatClass(TextOutputFormat.class);
56
57         FileInputFormat.addInputPath(job, new Path(args[0]));
58         FileOutputFormat.setOutputPath(job, new Path(args[1]));
59
60         job.waitForCompletion(true);
61     }
62
63 }
```

Contar palabras en Hadoop (Java)

Contar palabras en Spark (Python API)

```
text_file = sc.textFile("hdfs://...")
counts = text_file.flatMap(lambda line: line.split(" ")) \
                    .map(lambda word: (word, 1)) \
                    .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://...")
```

Índice

1. RDDs: Transformaciones

2. RDDs: Acciones

3. Ejercicio práctico: archivo de texto



Crear un RDD a partir de una lista con sus elementos

- Crea un RDD a partir de una lista Python

```
numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10], 2)
```

Lista de python.
Puede ser de
números,
cadenas...

Número de
particiones en que
se dividirá la lista
(opcional)

SPARK: ejemplo de Evaluación Perezosa

```
numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10,11,24])
```

EVALUACIÓN PEREZOSA / DEMORADA

```
numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10,11])
```

EVALUACIÓN PEREZOSA / DEMORADA

```
print(numeros.count())
```

([1,2,3,4,
5,6,7,8,9,
10,11,])



- Spark “apunta” qué va a pasar
- No se calcula nada hasta que es necesario

1. RDDs: Transformaciones

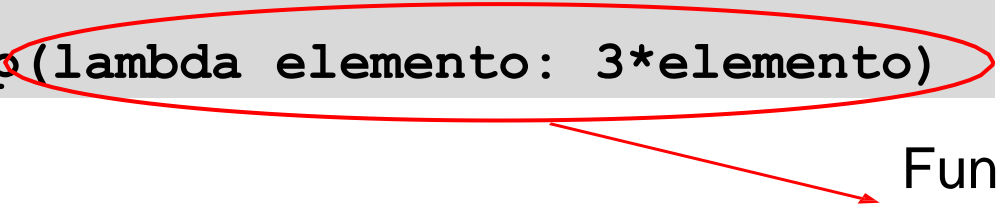


Transformación “map()”

- Aplica una transformación a cada elemento del RDD original

```
numeros = sc.parallelize([1,2,3,4,5])
```

```
num3 = numeros.map(lambda elemento: 3*elemento)
```



Función que se aplica a cada elemento del rdd números

- **Resultado:** [1,2,3,4,5] → [3,6,9,12,15]
- La función que se pasa a map debe:
 - Recibir un único parámetro, que serán elementos individuales del rdd de partida
 - Devolver el elemento transformado

RDDs: Transformaciones más comunes

Transformación	Descripción
map(func)	Crea un nuevo RDD a partir de otro aplicando una transformación a cada elemento original
filter(func)	Crea un nuevo RDD a partir de otro manteniendo solo los elementos de la lista original que cumplan una condición
flatMap(func)	Como map pero cada elemento original se puede mapear a 0 o varios elementos de salida
distinct()	Crea un nuevo RDD a partir de otro eliminando duplicados
union(otroRDD)	Une dos RDD en uno
sample()	Obtiene un RDD con una muestra obtenida con reemplazamiento (o sin) a partir de otro RDD.

Transformación “filter()”

- Filtra un RDD manteniendo solo los elementos que cumplan una condición


```
numeros = sc.parallelize([1,2,3,4,5])  
  
pares_rdd = numeros.filter(lambda elemento: elemento%2==0)
```

Función que se aplica a cada elemento para filtrarlo

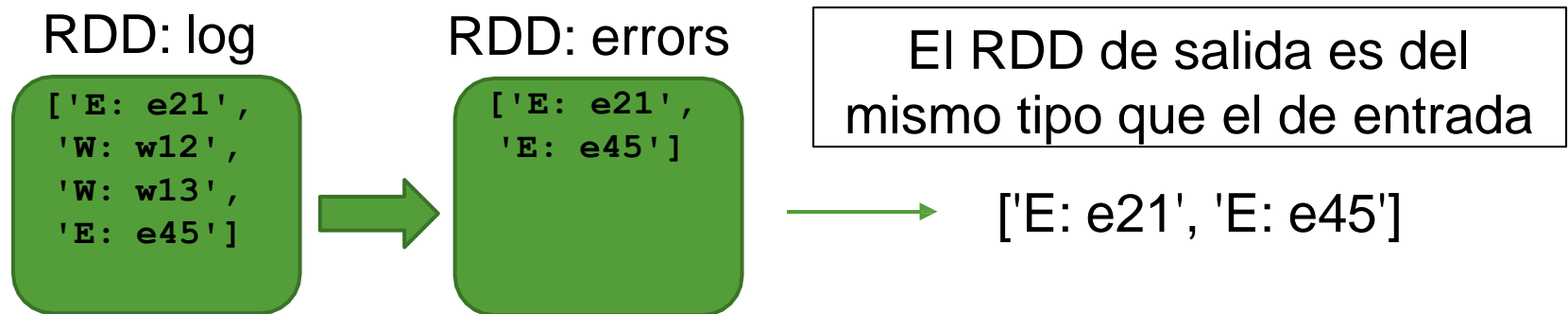
- Resultado: [1,2,3,4,5] → [2,4]
- **La función que se pasa a filter debe:**
 - Recibe un único parámetro, que serán los elementos individuales del rdd de partida
 - Devuelve True o False para indicar si el elemento pasa o no el filtro

Transformación: cuestiones sobre “filter()”

- ¿Cuál es el tamaño del rdd de salida?
 - Menor o igual que el original



```
log = sc.parallelize(['E: e21', 'W: w12', 'W: w13', 'E: e45'])  
  
errors = log.filter(lambda elemento: elemento[0]=='E')  
  
errors.collect()
```



Transformación “flatMap()”

- Como map pero por cada elemento puede crear más elementos

```
numeros = sc.parallelize([1,2,3,4,5])  
  
rdd = numeros.flatMap(lambda elemento :  
[elemento, 10*elemento])
```

- Resultado → [1, 10, 2, 20, 3, 30, 4, 40, 5, 50]
- **La función que se pasa a flatMap debe:**
 - Recibir un único parámetro, que serán elementos individuales del rdd de partida
 - Devolver una lista de elementos

Diferencias entre “flatMap()” y “map()”

```
lineas = sc.parallelize(['', 'a', 'a b', 'a b c'])

palabras_flat = lineas.flatMap(lambda elemento:
elemento.split())

palabras_map = lineas.map(lambda elemento:
elemento.split())
```

- Con flatMap → ['a', 'a', 'b', 'a', 'b', 'c']
- Con map → [[], ['a'], ['a', 'b'], ['a', 'b', 'c']]
- De aquí viene lo de *flat*, la lista de flatmap se ‘aplana’

Transformación “distinct()”

- Crea un nuevo RDD eliminando duplicados

```
numeros = sc.parallelize([1,1,2,2,5])  
  
unicos = numeros.distinct()  
unicos.collect()
```

- Resultado: [1,1,2,2,5] → [1, 2, 5]

Transformación “sample()”

- Remuestrea el RDD de entrada con o sin reemplazamiento.
- El primer parámetro indica si hay reemplazamiento
- El segundo parámetro indica la fracción de datos **aproximados** que se seleccionan.

```
numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10])  
  
muestra = numeros.sample(False, 0.5)
```

- Resultado -> **[1,4,6,9] (aleatorio)**
- Cada ejecución da un resultado distinto
- **Es útil cuando hay un número de datos demasiado elevado**
- Al menos en depuración

Transformación “union()”

- Une dos RDDs en uno

```
pares = sc.parallelize([2,4,6,8,10])  
impares = sc.parallelize([1,3,5,7,9])  
  
numeros = pares.union(impares)
```

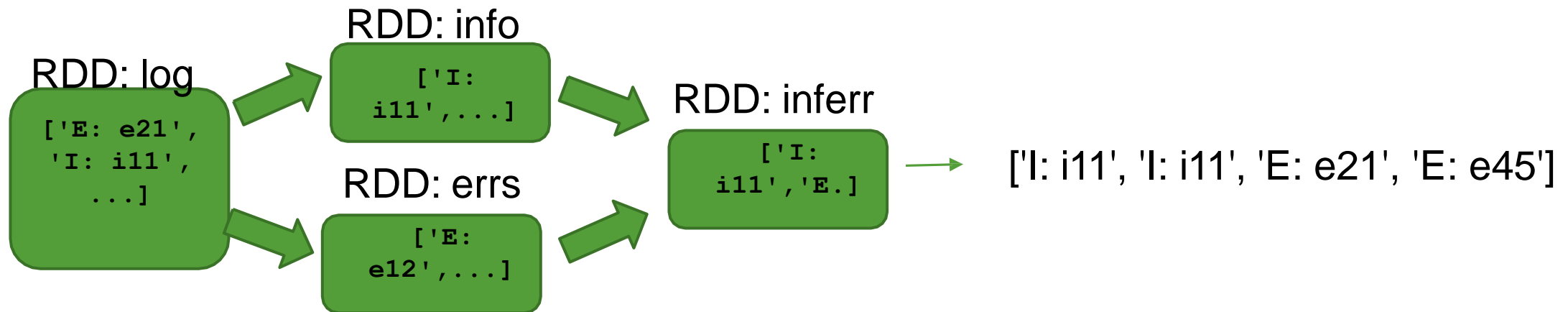
- Resultado: → [2, 4, 6, 8, 10, 1, 3, 5, 7, 9]

Transformación “union()”: ejemplo de uso sencillo

```
log = sc.parallelize(['E: e21', 'I: i11', 'W: w12', 'I: i11', 'W: w13', 'E: e45'])

info = log.filter(lambda elemento: elemento[0]=='I')
errs = log.filter(lambda elemento: elemento[0]=='E')
inferr = info.union(errs)

inferr.collect()
```



2. RDDs: Acciones



RDDs: Acciones

- Devuelven un resultado
- Desencadena la ejecución de toda la secuencia de RDD necesarios para calcular lo requerido.
- Ejecuta la "receta"

```
rdd2 = rdd1.flatMap(...).filter(...)  
  
print(rdd.count())
```

RDDs: Acciones más comunes

Acción	Descripción
<code>count()</code>	Devuelve el número de elementos del RDD
<code>reduce(func)</code>	Agrega los elementos del RDD usando <i>func</i>
<code>take(n)</code>	Devuelve una lista con los primeros n elementos del RDD
<code>collect()</code>	Devuelve una lista con todos los elementos del RDD
<code>takeOrdered(n[,key=func])</code>	Devuelve n elementos en orden ascendente. Opcionalmente se puede especificar la clave de ordenación

Acción “count”

- Devuelve el número de elementos del RDD

```
numeros = sc.parallelize([1,2,3,4,5,6,7,8,9,10])  
  
pares = numeros.filter(lambda elemento: elemento%2==0)  
  
➔ pares.count()
```

RDD: numeros

```
[1, 2, 3,  
4, 5, 6,  
7, 8, 9,  
10]
```



RDD: pares

```
[2, 4,  
6, 8,  
10]
```


Acción “reduce”

- Agrega todos los elementos del RDD **por pares** hasta obtener un único valor (expresión **lambda**)

```
numeros = sc.parallelize([1,2,3,4,5])  
  
print(numeros.reduce(lambda elem1,elem2: elem1+elem2))
```

- Resultado: 15
- La función que se pasa a reduce debe:
 - Recibir dos argumentos y devolver uno **de tipo compatible**
 - Operación debe ser conmutativa y asociativa, de forma que se pueda calcular bien en paralelo (workers)

Acción “take”

- Devuelve una lista con los primeros n elementos del RDD

```
numeros = sc.parallelize([5,3,2,1,4])  
  
print(numeros.take(3))
```

- Resultado: [5,3,2]

Acción “collect”

- Devuelve una lista con todos los elementos del RDD

```
numeros = sc.parallelize([5,3,2,1,4])  
  
print(numeros.collect())
```

- Resultado: [5, 3, 2, 1, 4]

Acción “takeOrdered”

- Devuelve una lista con los primeros n elementos del RDD en orden ascendente

```
numeros = sc.parallelize([3,2,1,4,5])  
  
print(numeros.takeOrdered(3))
```

- Resultado: [1,2,3]

Acción “takeOrdered”: cambiar criterio ordenación

- También podemos pasar una función para ordenar como queramos

```
numeros = sc.parallelize([3,2,1,4,5])  
  
print(numeros.takeOrdered(3, lambda elem: -elem))
```

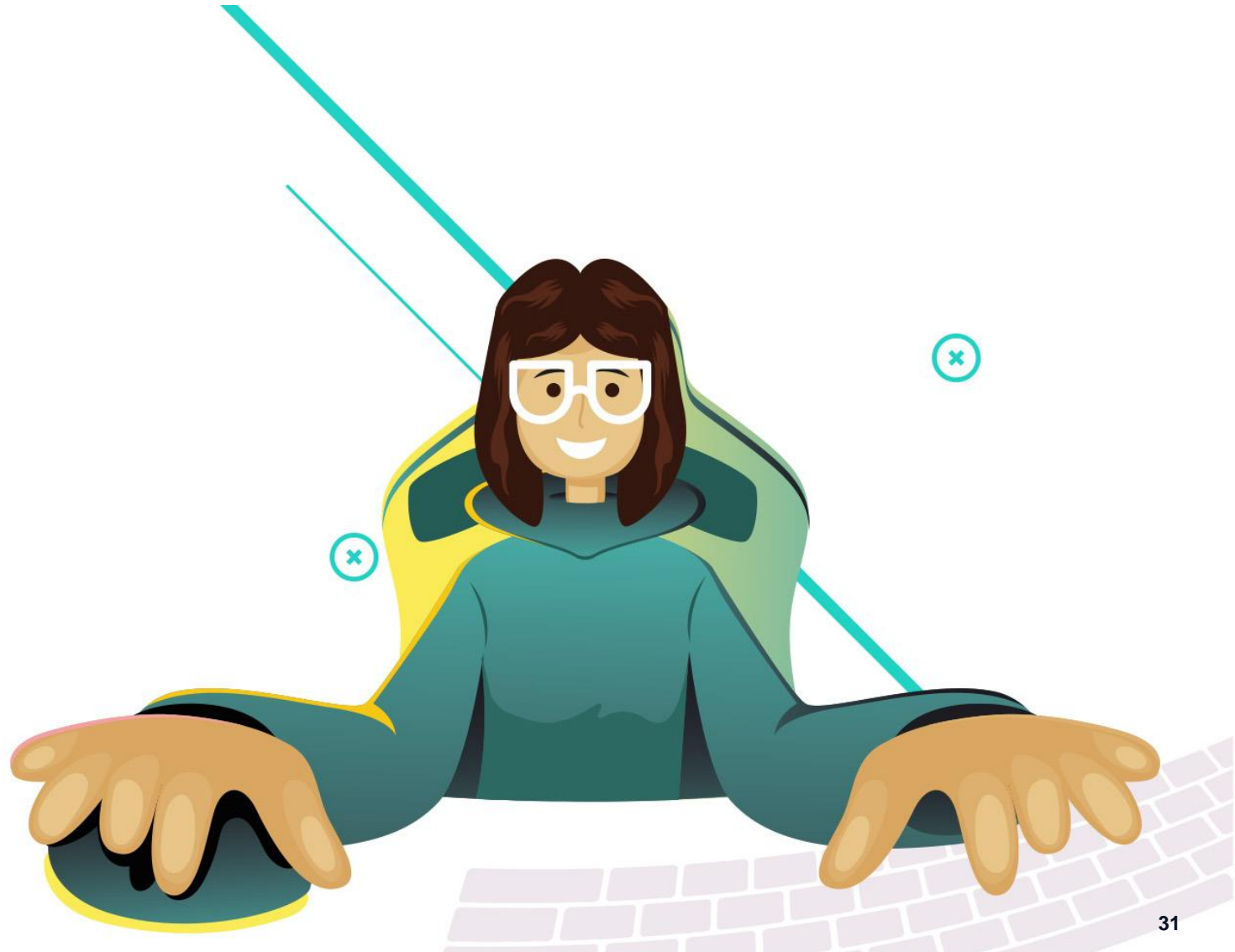
- Resultado: [5,4,3]
- ¿Cómo ordenarías para que primero aparezcan los pares ordenados y luego los impares?

3. EJERCICIO

PRÁCTICO: archivo de texto

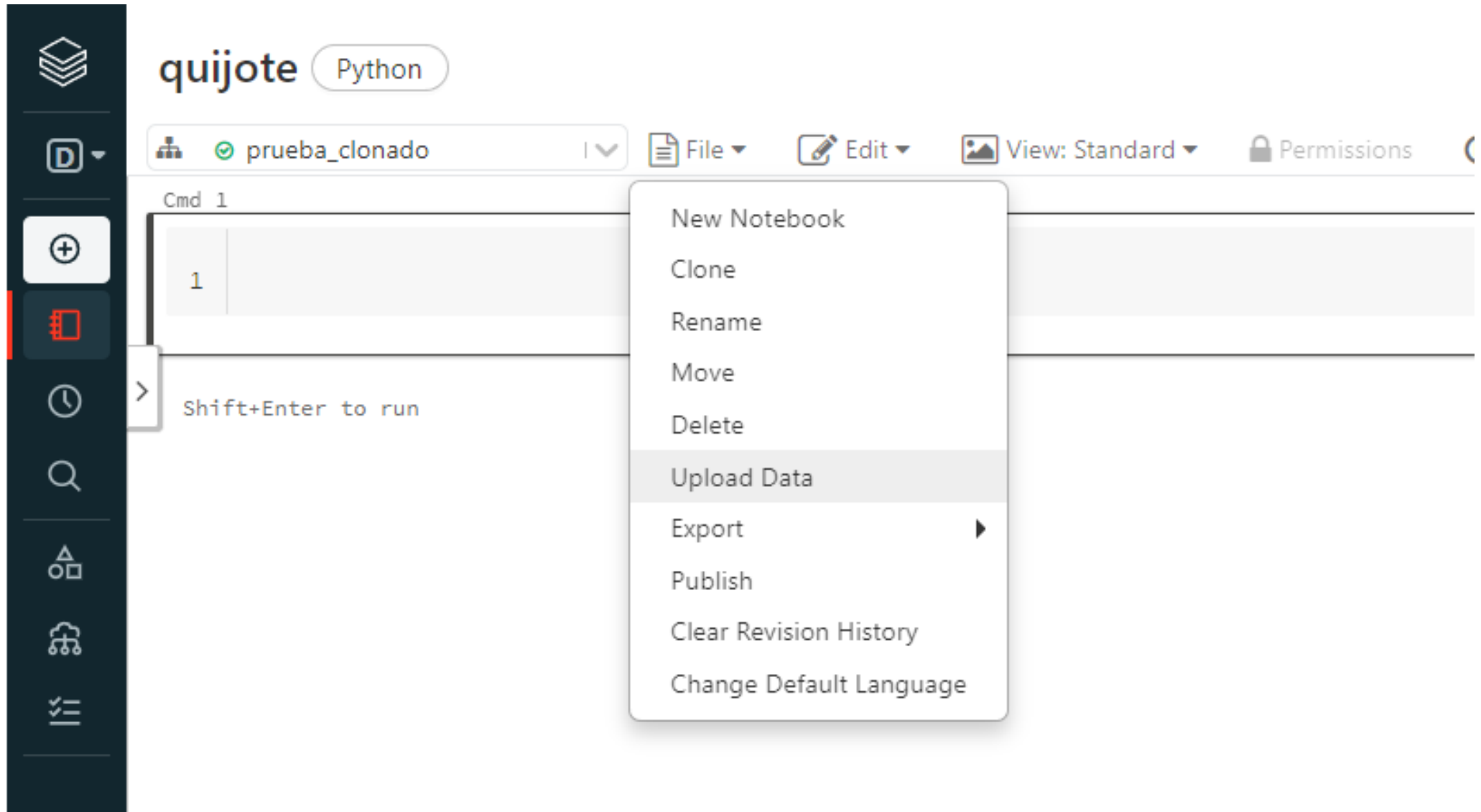


EJERCICIO 1: contar caracteres de un fichero



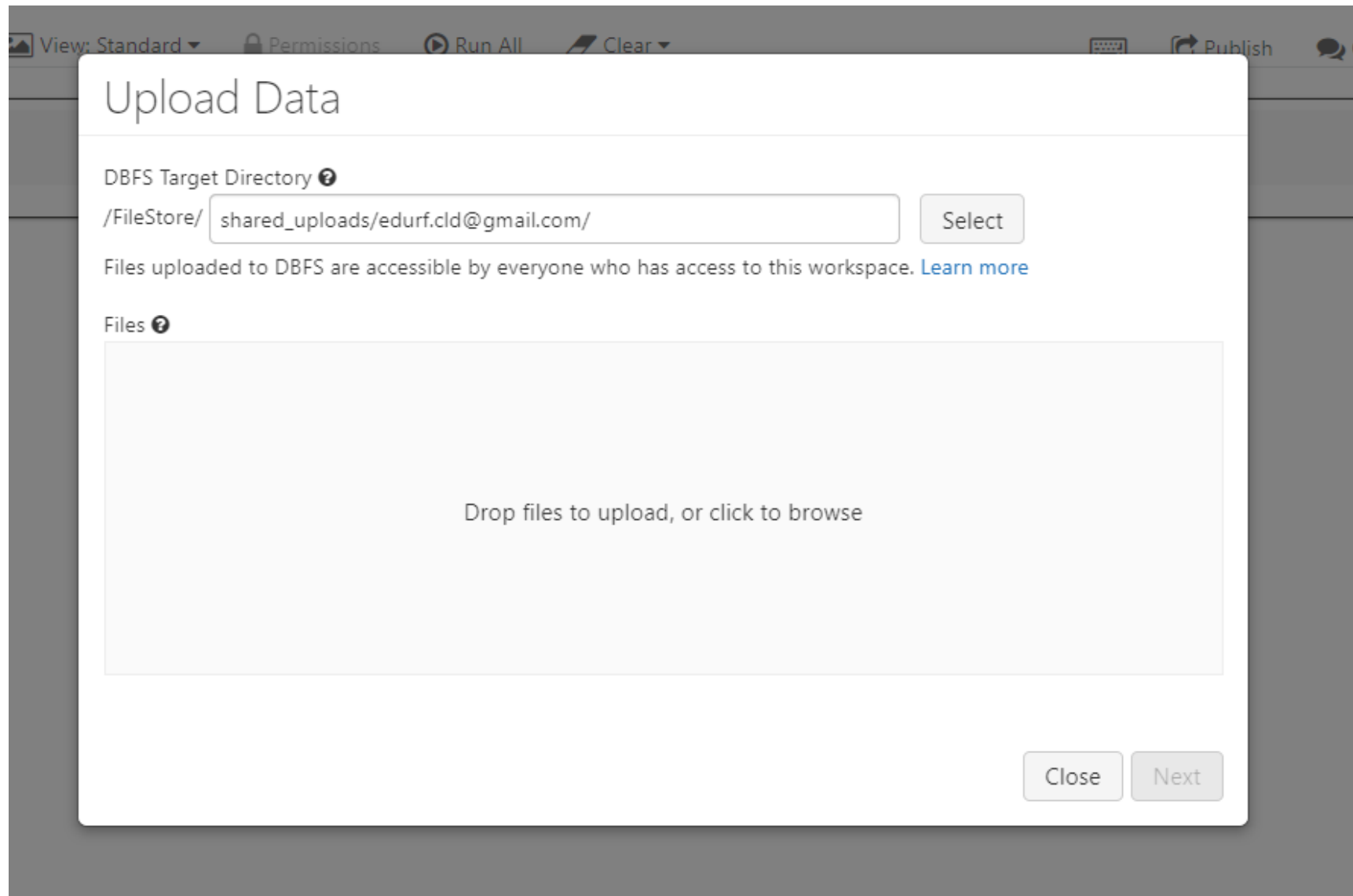
Cómo “subimos” un archivo de texto

- En el notebook nos vamos en el menú de arriba a “File”, desplegamos y picamos en “Upload Data”



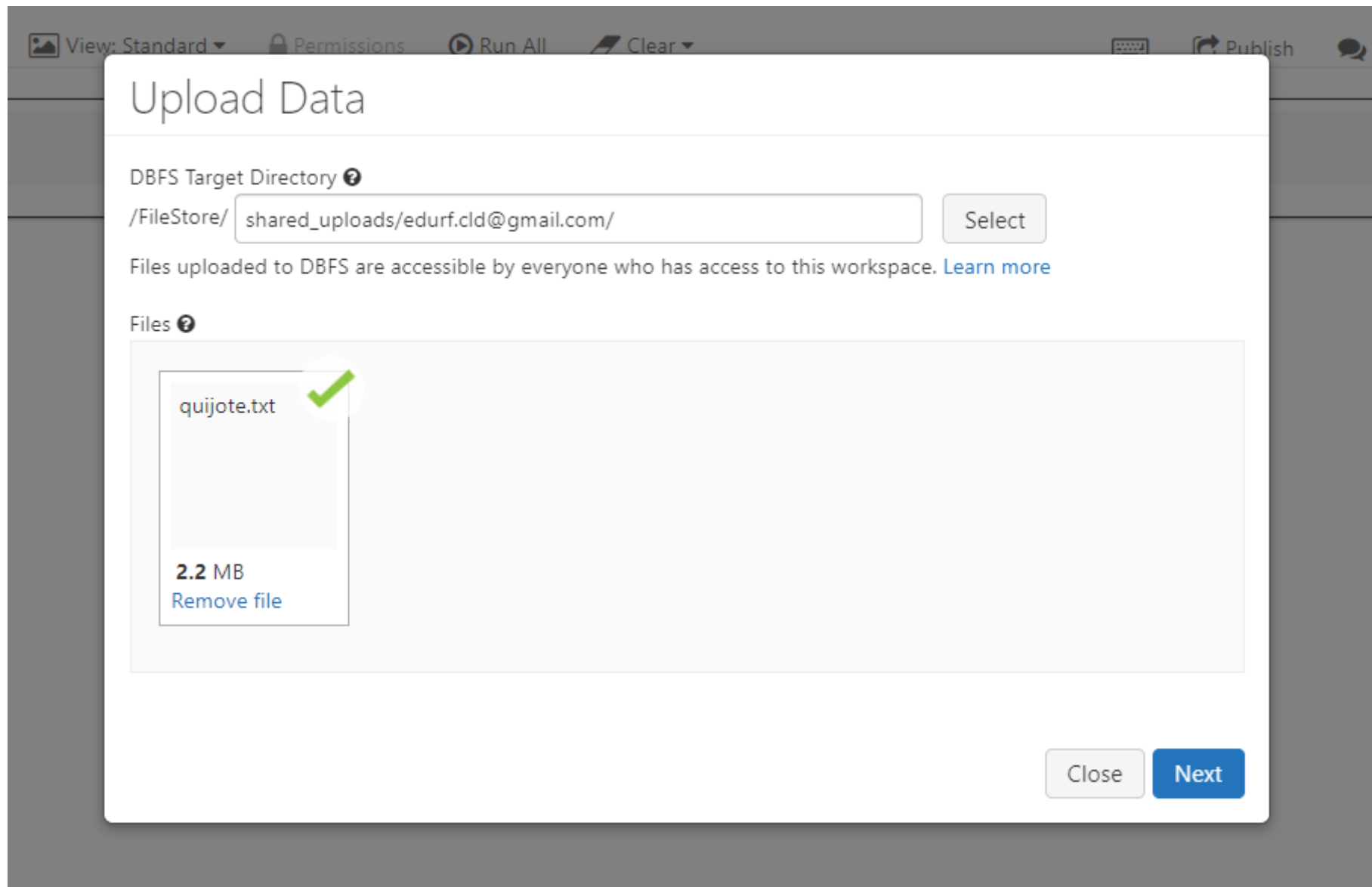
Cómo “subimos” un archivo de texto

- En la nueva ventana podemos clicar para buscar el archivo, o directamente arrastrar al recuadro de fondo gris el archivo.



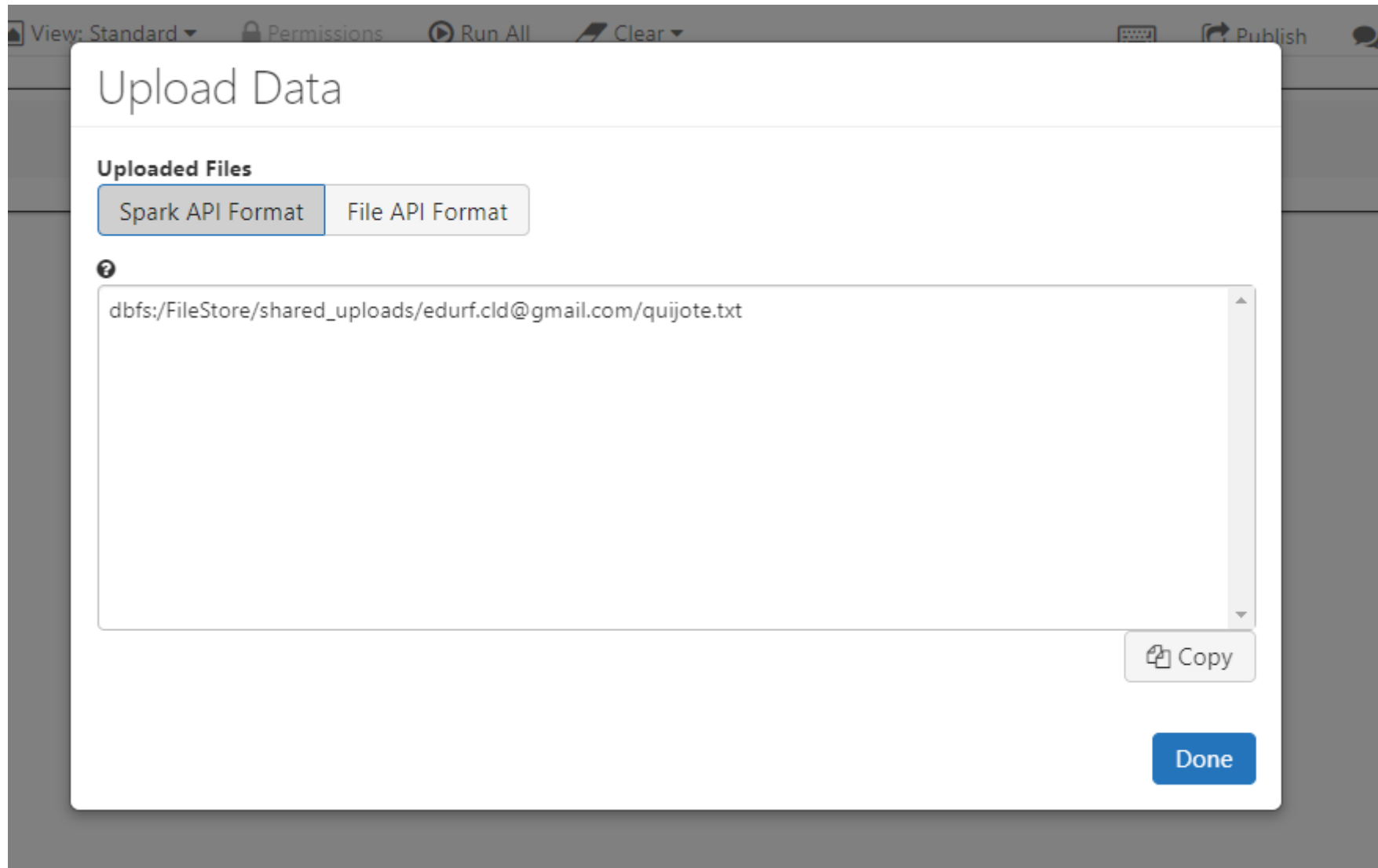
Cómo “subimos” un archivo de texto

- Una vez subido (tarda según el tamaño del archivo, conexión) le damos al botón “**Next**”:



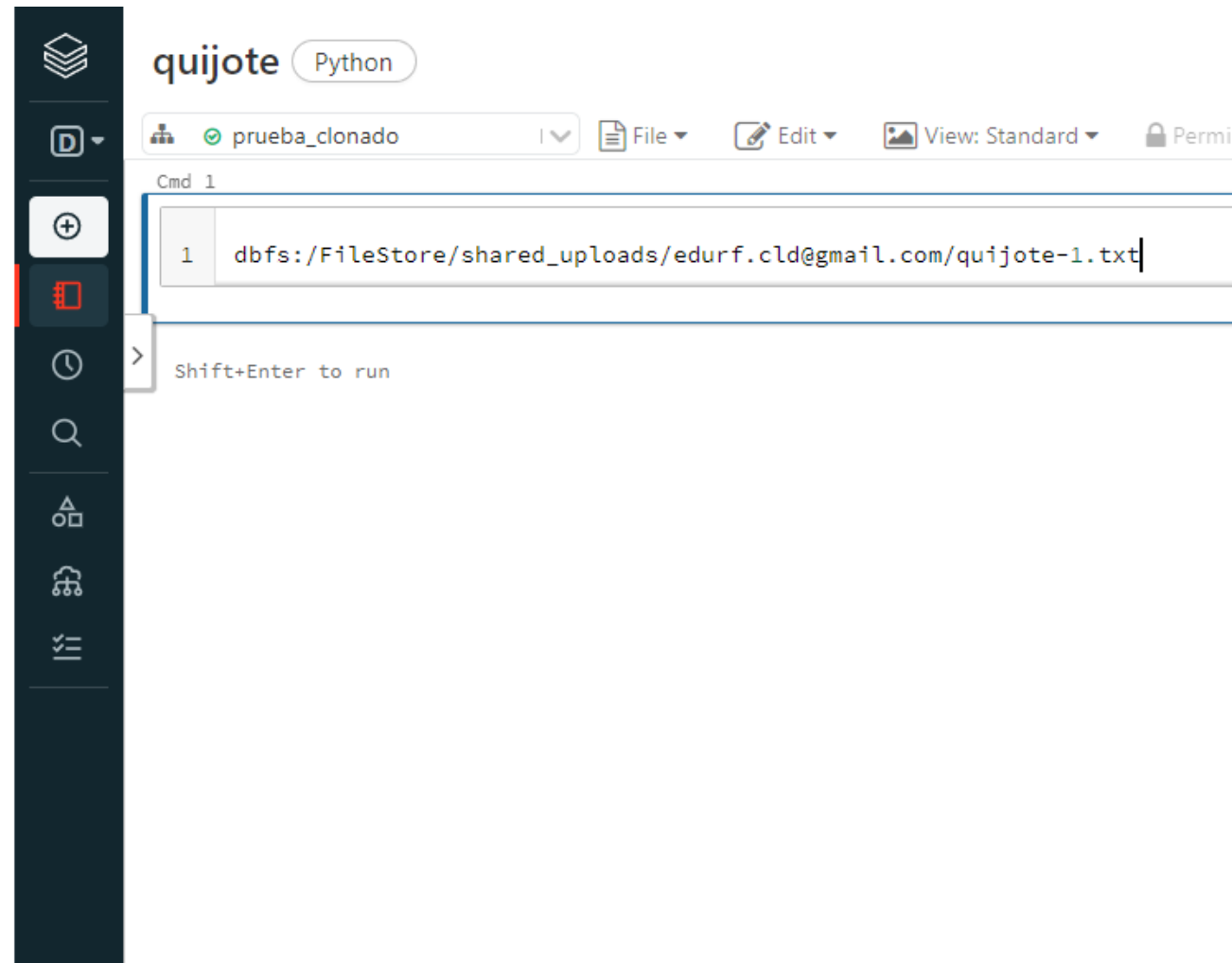
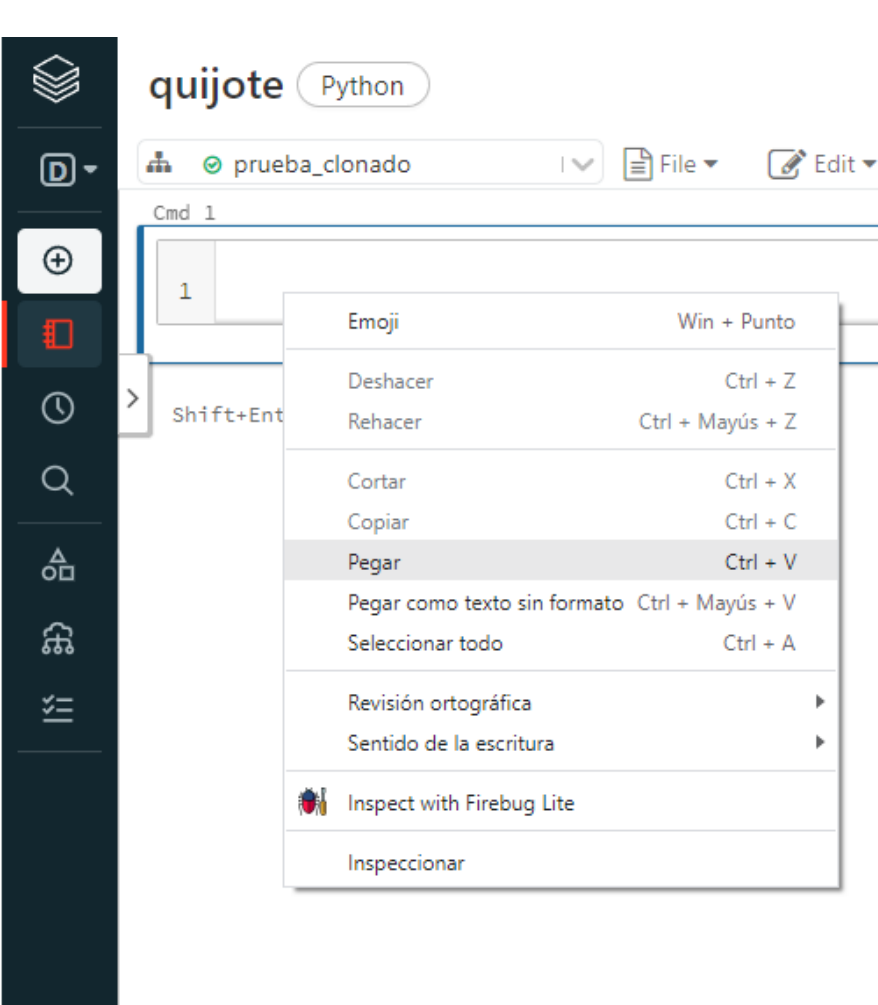
Cómo “subimos” un archivo de texto

- En esta nueva ventana solo clickamos en “**Copy**” en la esquina inferior derecha (resto opciones por defecto las dejamos) y finalmente a “**Done**”:



Cómo “subimos” un archivo de texto

- Volvemos al notebook y en una de las celdas copiamos el contenido. Ya tenemos la ruta del archivo.



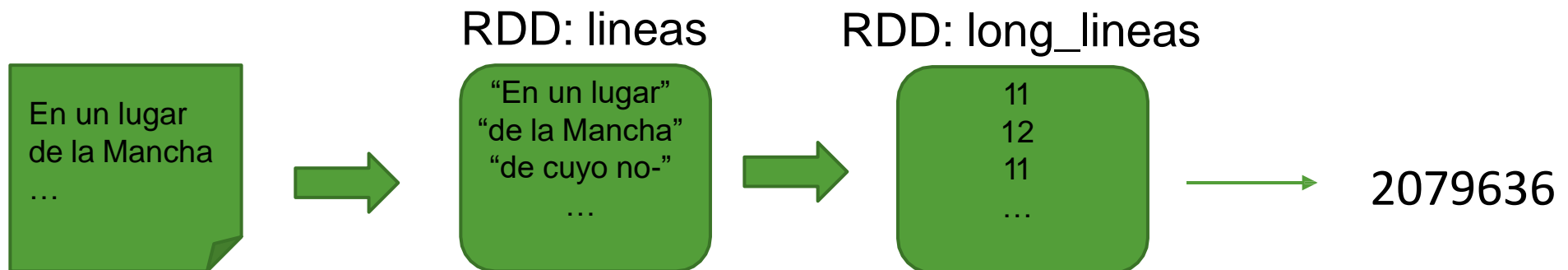
EJERCICIO 1: contar caracteres de un fichero

```
file = 'dbfs:/FileStore/shared_uploads/edurf.cld@gmail.com/quijote-1.txt'

lineas = sc.textFile(file)

long_lineas = lineas.map(lambda elemento: len(elemento))

print(long_lineas.reduce(lambda elem1,elem2: elem1 + elem2))
```





red.es



UNIÓN EUROPEA

"El FSE invierte en tu futuro"

Fondo Social Europeo

