
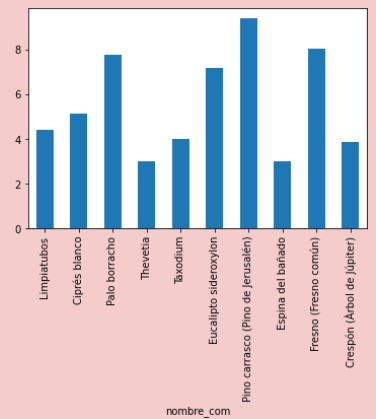


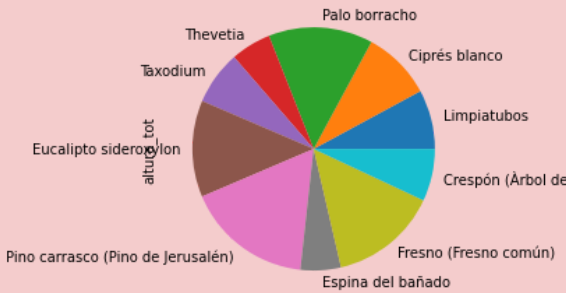
General

Operación	Operador	Ejemplo
Suma	+	4+9
Resta	-	12-2
Multiplicación	*	2*3
División	/	16/2
Resto - saber el resto de una cuenta	%	if (numero) % 2 == 0: return "par"
Redondear	round	round (4.3)
Valor absoluto	abs	abs (-123)
Máximo	max	max(7,8) (va a dar de rtado 8) max (funcion1, funcion2) (va a dar de rtado la función que tenga el nro más alto)
Mínimo	min	min (7,8) (va a dar de rtado 7)
Menor / mayor	< / >	2>7 (va a dar de rtado "false")
Menor igual/ mayor igual	≤ / ≥	
Número en un string	str (nro)	str (5)
Como saber si un string está incluido en otro	in	
Empezar con un string	str.startswith	 str.startswith("Fundación e imperio", "Fundación") True
Terminar con un string	str.endswith	
Contar caracteres	len	len("¿como te va?") (va a dar de rtado 12)
Eliminar espacios del costado.	str.strip	(" papafrita ") == "papafrita"
Pasar las mayúsculas a minúsculas	str.lower	
Pasar las minúsculas a mayúsculas	str.upper	

Operación	Operador	Ejemplo															
	and																
	or																
<p>Saber si los caracteres son lo mismo.</p> <p>Si son diferentes</p> <p>_____</p> <p>Saber si uno es más largo que otro</p>	<p>==</p> <p>!=</p> <p>_____</p> <p>es_mas_largo_que</p>	<p>len("dog") == len("cats") va a dar de resultado false</p> <p>_____</p> <p>es_mas_largo_que("dog", "cats") va a dar de resultado true</p>															
<p>Dar una respuesta booleana pero al revés que la tabla normal</p>	xor re	<table border="1"> <thead> <tr> <th>A</th><th>B</th><th>A x B</th></tr> </thead> <tbody> <tr> <td>V</td><td>V</td><td>F</td></tr> <tr> <td>V</td><td>F</td><td>V</td></tr> <tr> <td>F</td><td>V</td><td>V</td></tr> <tr> <td>F</td><td>F</td><td>F</td></tr> </tbody> </table> <p>¿Como se define? para que de verdadero: def xor (a,b): return a and not b or b and not a</p> <p>para que de falso def xor (a,b): return a and b or b and a</p>	A	B	A x B	V	V	F	V	F	V	F	V	V	F	F	F
A	B	A x B															
V	V	F															
V	F	V															
F	V	V															
F	F	F															
<p>Pasar un numero con “,” y te lo devuelve como entero</p> <p>Pasar de numero string a numero</p>	int	<p>int funcion/ 3000 devuelve un número entero</p> <p>int ("10") devuelve 10</p>															

Sección	Nombre	Función	Ejemplo
LISTAS		Conjunto de elementos Pueden ser strings o booleanos	[]
	lista vacía	Que nos devuelva una lista	<code>un_lista_vacia = []</code>
	len	Cuenta el contenido de una lista	<code>numeros_de_loteria = [2, 11, 17, 32, 36, 39]</code> <code>len([[1,2],[8],[14, 87]])</code> → = 3 <hr/> <code>len(numeros_de_loteria)</code> → = 6
	list.index	Saber la posición de un elemento. Dato: se empieza a contar desde 0	<code>dias_laborales = ["lunes", "martes", "miercoles", "jueves", "viernes"]</code> <code>list.index(dias_laborales, "lunes")</code>
	in	Saber si un elemento está en la lista	<code>7 in [1, 6, 7, 6]</code> True
		Saber que elemento está en X posición	<code>["ese", "perro", "tiene", "la", "cola", "peluda"] [1]</code> "perro"
	.pop	Elimina el elemento que esta en la posición que le digo	<pre>def elemento (lista2): lista2.pop(0) return lista2 lista2= ["atun", "lapiz", "cuaderno", "papel"] print (elemento(lista2))</pre>
	for		
	list.append (lista, "comida")	procedimiento que devuelve una lista con un elemento extra	<code>grupos_lh=[Sima, Guibo, Gama]</code> <code>list.append(grupos, "Goza")</code> <u>rtado:</u> <code>grupos_lh</code> ["Sima", "Guibo", "Gama", "Goza"]
	+= []	agregar más de un elemento a la lista	<code>grupos_lh += ["Jiu", "Amitz"]</code> <u>rtado:</u> <code>grupos_lh</code> ["Sima", "Guibo", "Gama", "Goza", "Jiu", "Amitz"]

	list.remove (lista, "arroz")	<u>procedimiento</u> que devuelve una lista sin un elemento	
	sorted	<u>procedimiento</u> que devuelve una lista ordenada	
	sort_values	<u>función</u> que ordena una lista ascendente	
	sort_values ("calle", ascending=False)	<u>función</u> que ordena una lista descendentemente	florerias.sort_values("calle", ascending = False)
	sort_values (by..) data frames	<u>lista de criterios</u>	florerias.sort_values(by = ["barrio", "calle", "altura"], ascending = [True, True, False])
		Lo que hay que hacer antes para pedir las alturas (ej) para después poder graficar	alturas_promedio = arbolado.groupby("nombre_com").altura_tot.mean().sample(10) alturas_promedio
	reset_index ()		
	as_index		
GRÁFICO			
	florerias_por_calle [florerias].plot.bar	gráficas (de barras)	 <p>empresas_por_anio.plot.bar('empresa', 'anio', figsize=(25, 8))</p>
	alturas_promedio.plot.hist()		

	alturas_promedio.plot.pie()	gráfico de torta	
PANDAS		<u>import pandas as pd</u>	
	nunique()	pedir valores únicos en una serie Retorna la cantidad de valores únicos, excluyendo a los valores NaN	Calle.nunique()
	len(Calle) pd.unique	cuantas calles hay	len(pd.unique(florerias.calle))
		el número de filas de una serie	florerias["Calle"]
		ver cuantas calles hay	pd.unique(florerias.calle)
	unique	Cuales son los valores que no se repiten en el dataframe	florerias.Calle.unique()
	groupby	Encontrar cuántas “florerias” hay por “calle”	florerias_por_calle=florerias.groupby("Calle").count() florerias_por_calle
	to_dict	Devolver una lista con menos cosas??????????	Bicicleteros.to_dict("records")
	[iloc]	devolver todos los datos de las filas seleccionadas entre corchetes.	florerias.iloc[15]
	[[]]	quedarse/mostrar solo las columnas que quiero	florerias[['nombre', 'direccion', 'web', 'horario_de']] florerias.iloc[29][['nombre', 'direccion', 'web', 'horario_de']]
	Head	primeras filas	arbolado.head(10)
	Tail	Últimas filas	arbolado.tail(10)

	sample (número de la fila)	mostrar n filas al azar	
	CSV	valores separados por coma.	
	Len	cantidad de filas en un dataframe. cantidad de caracteres de un string	<code>len(arbolado)</code>
	Promedio	hacer un promedio	def: return: sum(lista)/len(lista)
	Data frame	tipo de variable de la lista → ej: floreria	
	pd.value_counts	agrupa, cuenta y ordena las calles. cuántas veces se repite un valor en una columna	<code>pd.value_counts(bibliotecas["barrio"])</code>
	id	id: es una abreviatura de identifier Los ids son identificadores únicos de una fila o "cosa"	
	idxmax idxmin	índice de la primera fila que hace máxima ó mínima a la columna dada → quien contiene a ese maximo	<code>arbolado.diametro.idxmax()/arbolado.diametro.idxmin()</code> <code>arbolado.diametro.max() == arbolado.diametro.iloc[arbolado.diametro.idxmax()]</code>
	max	máximo	<code>max</code>
	min	mínimo	<code>min</code>
			<code>idxmax</code>
	mean	calcular el promedio de una columna Nos devuelve la mediana de una columna (o lo que es lo mismo, el cuantil 50%, también llamado Q2).	
	median	calcular el número que está justo en el medio de []	<code>median</code>
	sum	suma de todas las medidas	
	drop	eliminar columnas	<code>arbolado_simplificado.drop(columns = ["nombre_cie", "esta_actualizado"], inplace = True)/ arbolado_simplificado = arbolado_simplificado.drop(columns = ["nombre_cie", "esta_actualizado"])</code>

	rename	para renombrar columnas (para guardar la modificación deberíamos agregar inplace = True o guardarlo en una variable)	arbolado_simplificado.rename(columns = {"nombre_com": "nombre_comercial", "altura_tot": "altura_total"})
	dropna (NaN's)	limpieza (sacarlo cuando quieras/sea necesario) → EJ: Te permite ver que árboles hay sin nombre (para guardar la modificación deberíamos agregar inplace = True o guardarlo en una variable)	arbolado_simplificado[arbolado_simplificado.nombre_com.isna()]
	drop.duplicate	eliminar filas completas duplicadas	arbolado_simplificado = arbolado_simplificado.drop_duplicates()
	subset	eliminar varios criterios en simultáneo	nombres_comerciales_por_barrio = arbolado_simplificado.drop_duplicates(subset = ["nombre_com", "barrio"])
	isna	filtrar el dataframe, quedándonos con las filas que cumplieran cierto criterio. → Nos permite saber si los valores de una columna son nulos o no nulos, respectivamente. Nos retorna una columna booleana, útil para realizar filtrados.	arbolado_simplificado[arbolado_simplificado.nombre_com.isna()]
	notna	→ Nos permite saber si los valores de una columna son nulos o no nulos, respectivamente. Nos retorna una columna booleana, útil para realizar filtrados.	
	& (and)		fresnos_americanos_de_floresta = arbolado_simplificado[(arbolado_simplificado['nombre_com'] == "Fresno americano") & (arbolado_simplificado.barrio == "FLORESTA")]
	(or)	quedar con los árboles que cumplan uno u otro criterio (o ambos) → disyunción	arboles_flores_y_floresta = arbolado_simplificado[(arbolado_simplificado['barrio'] == "FLORESTA") (arbolado_simplificado.barrio == "FLORES")]
	~ (not)	quedarnos con todos los árboles menos los de Floresta	todos_menos_floresta = arbolado_simplificado[~(arbolado_simplificado['barrio'] ==

			"FLORESTA")]
	isin	permite saber si un elemento está en una lista → se piden lo que quieras EJ: ver si los barrios que vos pensas están y te salen solo esos	arbolado_simplificado[arbolado_simplificado["barrio"].isin(["FLORES", "FLORESTA", "LINIERS", "CONSTITUCION"])]
	str.	obtener los árboles cuyo nombre contenga la palabra "Eucalipto" → te salen todos los que tienen la palabra "eucalipto" en su nombre	arbolado_validado[arbolado_validado.nombre_com.str.contains("Eucalipto")]
	str.replace	para reemplazar un valor por otro → ejemplo el barrio se llamaba PALERMO y yo lo reemplazo y lo pasó a llamar PALERMO SOHO	arbolado_validado.barrio.str.replace("VILLA GRAL. MITRE", "VILLA GENERAL MITRE")
	.str.contains	llamas a esa columna específico	recorridos.nombre_estacion_origen.str.contains("Parque") recorridos["nombre_estacion_origen"].str.contains("Parque") → es por dataframe
	range	se puede usar con 3 parámetros distintos	
	title	para ponerle mayúscula a las primeras letras de cada palabra → en un string	.title()
	list	para pedir que se convierta en una lista	
	itertuples	nos permite recorrer directamente cada fila	.itertuples()
ERRORES			
	Index error list index out of range	pedís un elemento = o > al tamaño de la lista	
	SyntaxError	error de sintaxis	invalid syntax
	Identificacion Error	error en el tab	expected an indented block

número → es un int

[:5] → significa que va desde 0 a 4 → porque son 5 "numeros/espacios" porque se cuenta el 0

{ } → diccionario

[] → listas

```
import pandas as pd
```

```
bicicleterias = pd.read_csv("LINK") --> SI ES NECESARIO AGREGAR TRIPLE COMILLA
```

Gráficos

- FALTAN ALGUNOS GRAFICOS Y OTROS QUE NO ENTIENDO

(<https://colab.research.google.com/drive/1BVqM4fvOvQDKL2E0TjoVvm4H5X4aeWxn?usp=sharing#scrollTo=O1XUOwek5cms>)

`plot.line`

`plot.scatter`

`boxplot`

RECURSOS/RESUMEN → ESTÁN TREMENDOS

https://flbulgarelli.github.io/recursos-python/1_introduccion_a_la_programacion/15_integraci%C3%B3n/referencia_r%C3%A1pida_python/#referencia-rapida-del-lenguaje-python → FOR / IF (y más)

https://flbulgarelli.github.io/recursos-python/1_introduccion_a_la_programacion/15_integraci%C3%B3n/referencia_r%C3%A1pida_pandas/ → pandas

- Intro listas:

https://colab.research.google.com/drive/1UOJon5wTyc5Zsfum5LVzsV7_hcattCCN#scrollTo=7ICtIFdN2zMM

Combinación

`merge`

`concat`

- Herramientas de programación
 - Operadores matemáticos, lógicos y de strings
 - Funciones, parámetros y procedimientos
 - Alternativa condicional (if)
 - Variables
 - Repetición indexada (for ... in)
- Estructuras de datos
 - Listas: colecciones homogéneas con repetidos que preservan el orden
 - Diccionarios: colecciones heterogéneas de pares clave-valor (llamados entradas) que no tienen un orden particular
 - Tablas (DataFrames): estructuras tabulares con filas y columnas
- Procesamiento de datos
 - Se puede hacer de diferentes formas:
 - con operaciones de alto nivel:
 - len, all, any, max, sum, sorted *listas por comprensión*, etc (**listas**)
 - groupby, dropna, value_counts, head, filtros, etc (**tablas/DataFrames**)
 - con operaciones de bajo nivel (**recorridos**), combinando las herramientas antes vistas: **for**, **if**, variables acumuladoras, variables contadoras, segmentos, acceso indexado, operaciones de alto nivel, etc
 - Los distintos procesamientos que vimos caen en general en las siguientes categorías:
 - **Agregación:**
 - sumatorias (sum)
 - promedios (mean)
 - medianas (median)
 - cuantiles (quantile)
 - máximos y mínimos (max, min, idxmax, idxmin)
 - **Ordenamiento** (sort_values, sorted)
 - **Recorte:**
 - Primeros N elementos (head: cabeza)
 - Últimos N elementos (cola: tail)
 - Elementos N al M (secciones: slices)
 - **Agrupación (groupby)**
 - **Transformación:**
 - Renombrado de columnas / entradas (rename)
 - Creación de columnas / entradas
 - Eliminación de columnas / entradas (del, drop)
 - **Filtrado**
 - **Acceso indexado (iloc, una_lista[indice])**

Ejercicio 1: Ejercicio 1



Cuando tomamos decisiones muchas veces necesitamos opciones. Pero, ¿qué pasa si las opciones son demasiadas? ¿Y si son demasiado pocas? Por eso queremos programar algo que nos pueda asegurar de que no sean tantas ni tan pocas:

- si tenemos opciones de más, las queremos descartar 😊;
- y si tenemos opciones de menos, vamos a repetir las veces que sean necesarias la última opción que nos dieron 😊.

Definí la función `ni_tanto_ni_tan_poco` que tome una lista de opciones y la cantidad mínima y máxima de opciones que necesitamos, y devuelve una nueva lista de opciones acorde. Por ejemplo:

```
➤ ni_tanto_ni_tan_poco(["jugar fútbol", "jugar basquet", "jugar ajedrez", "jugar al truco", "ir a bailar"], 3, 5)
["jugar fútbol", "jugar basquet", "jugar ajedrez", "jugar al truco", "jugar al truco"] # todo en orden, porque queríamos entre 3 y 5 opciones, así que completamos lo que faltaba con la última opción que nos dieron 😊

➤ ni_tanto_ni_tan_poco(["jugar fútbol", "jugar basquet", "jugar ajedrez", "jugar al truco", "ir a bailar"], 2, 3)
["jugar fútbol", "jugar basquet", "jugar ajedrez"] # recortamos opciones, porque queríamos 3 como máximo

➤ ni_tanto_ni_tan_poco(["jugar fútbol", "jugar basquet"], 4, 5)
["jugar fútbol", "jugar basquet", "jugar basquet", "jugar basquet"] # queríamos entre 4 y 5 opciones, así que completamos lo que faltaba con la última opción que nos dieron 😊
```

[Solución](#) [>_ Consola](#)

```
1 def ni_tanto_ni_tan_poco(opciones, minimo, maximo):
2     numero = len(opciones)
3     if numero < minimo:
4         for opcion in range(0, minimo-numero):
5             opciones.append(opciones[-1])
6     return opciones
7     elif numero > maximo:
8         return opciones[:maximo]
9     else:
10        return opciones
```

▶ Enviar

¿Tenés dudas? ¡Levantá la mano!

Ejercicio 2: Ejercicio 2



El *tabú* es un juego donde no podés decir ciertas palabras.

Por eso vamos a necesitar una función `pierde_tabu` que nos diga si la frase contiene alguna de las palabras prohibidas:

```
➤ pierde_tabu("esta frase gana", ["blanco", "negro"])
False
➤ pierde_tabu("esta frase no pierde", ["si", "no"])
True
➤ pierde_tabu("bajo el nogal", ["arriba", "árbol", "sombra", "nuez"])
False
```

[Solución](#) [>_ Consola](#)

```
1 def pierde_tabu(frase, palabrasProhibidas):
2     perdio = False
3     for palabra in palabrasProhibidas:
4         if palabra in str.split(frase):
5             perdio = True
6     return perdio
```

▶ Enviar

Ejercicio 3: Ejercicio 3



Necesitamos una función `usuarios_segun dominio` que dado una lista de emails, nos diga qué usuarios hay en cada dominio. Por ejemplo:

```
➤ usuarios_segun_dominio(["nadia@gmail.com", "franco@hotmail.com", "ana@yahoo.com", "guille@yahoo.com"])
{'gmail.com': ['nadia'],
 'hotmail.com': ['franco'],
 'yahoo.com': ['ana', 'guille']}
```

[Solución](#) [>_ Consola](#)

```
1 def usuarios_segun_dominio(lista):
2     respuesta = {}
3     for mail in lista:
4         if mail.split("@")[1] in respuesta:
5             respuesta[mail.split("@")[1]].append(mail.split("@")[0])
6         else:
7             respuesta[mail.split("@")[1]] = [mail.split("@")[0]]
8     return respuesta
```

Ejercicio 4: Ejercicio 4



Cargá en un cuaderno de colab el dataset <https://raw.githubusercontent.com/Gustruccio/IPC-UCEMA/master/terrenos/ds-47.csv> y respondé las siguientes preguntas:

1. ¿Cuántos terrenos hay?
2. ¿Cuál es la valuación total en dólares de los terrenos durante el tercer trimestre?
3. ¿Cuál es la comuna con más terrenos?

Además, ingresá el link a tu cuaderno.

Pregunta 1

1937

Pregunta 2

79035000

Pregunta 3

15

Enlace a tu cuaderno colab

Cliente	Alta de la cuenta	Monto	contacto
La rosa	15 Marzo 1910	15000	NaN
El diario	8 Julio 2001	30000	el_diego_10@plumitamail.com
Nené	15 Septiembre 1998	15000	nene_569@hotmail.com
Lucho	8 Agosto 2013	60000	lucho@media.com
Coco	NaN	50	coqito@unmail.com

- ☐ Usando `tabla['La rosa']` puede obtener los datos de la cliente Rosa
- ☐ Usando `tabla['Cliente'] == "La rosa"` puede obtener los datos de la cliente Rosa
- ☒ Usando `tabla.iloc[0]` puede obtener los datos de la cliente Rosa
- ☐ Usando `tabla.iloc[1]` puede obtener los datos de la cliente Rosa
- ☒ Si ejecuta el código `tabla.dropna()` el len del DataFrame será 3
- ☒ Si ejecuta el código `tabla.dropna()` se pierden los datos de Coco y La Rosa
- ☐ Si ejecuta el código `tabla.dropna()` el len del DataFrame será 5
- ☒ La cantidad de clientes únicos es 5
- ☐ La cantidad de montos únicos es 5
- ☐ El máximo monto puede obtenerse haciendo `tabla.[Monto].max()`