

Python Cheat Sheet 1

Variables ampliadas por text (CONCATENATION)

Para encadenar texto

```
categoria1 = "verde"
color_detalle = categoria1 + ' ' + 'oscuro'

print(categoria1 + ' oscuro')
print(categoria1, 'oscuro')
```

type() and isinstance()

float/int/str(variable) cambia el tipo de data/type

type(variable) devuelve: class 'float/int/str'

isinstance(variable, float/int/str) comprobar el tipo de dato (devuelve True/False)

Operaciones Algebraicas

+ sumar	/ dividir
- restar	// divider y redondear (modulus)
* multiplicar	% resto de una division (floor division)
** elevar	round(x) redondear numero x

Operaciones Binarias

== comprobar si valores coinciden
is comprobar si valores son exacamente igual
!= comprobar si valores son diferentes
is not comprobar si valores no son exactamente iguales
> (>=) mayor que (mayor o igual que)
< (<=) menor que (menor o igual que)
and ambas verdaderas
or ambas o solo una verdadera
in/not in comprobar si hay un valor en una lista etc.

Metodos String

string.upper() MAYUSCULAS
string.lower() minusculas
string.capitalize() Primera letra de la frase en may.
string.title() Primera Letra De Cada Palabra En May.
string.swapcase() mINUSCULAS A mAYUSCULAS O vICEVERSA
string.strip() quita espacios del principio y final

string.split() divide string en lista - por espacios por defecto, o especifica otro divisor en ()
string.replace("frase", "frase") reemplaza la primera frase del string por el otro
" ".join(string) une los elementos de una lista en una string con el separador espificado en " "
list(string) convierte un variable string en una lista
string.find("substring") encuentra el indice en que empiece el substring/'-1' si no existe el substring

string[i] devuelve el elemento en la indice i
string[i:j] devuelve un rango de caracteres

metodos permanentes (cambia el variable, no devuelve nada)

Listas [] Metodos no permanentes

lista = [] crea una lista vacia
len(lista) devuelve el no. de elementos
min(lista)/max(lista) saca el valor minimo y maximo
lista.count() devuelve el no. de elementos que hay en la lista de un valor determinado en los()
sorted(lista) ordenar una lista de menor a mayor
lista.copy() hacer una copia de la lista

Metodos con indices

list.index(x) devuelve la indice de x en la lista
lista[i] devuelve el elemento en la indice i
[start:stop:step]
lista[i:j:x] devuelve los elementos por el rango de i a j (incluye i pero no j) saltando por x
lista[-i:-j] devuelve los elementos por los indices negativos (incluye -j pero no -i)

Listas – Acciones Permanentes

Ampliar una lista
[lista1, lista2] junta listas pero se mantienen como listas separadas
lista1 + lista2 hace una lista mas larga

.append()
lista.append(x)# añade un solo elemento (lista, string, integer o tuple) a la lista
.extend()
lista.extend(lista2)# añade los elementos de una lista al final de la lista
.insert()
.insert(i, x)# mete un elemento (x) en un índice(i)

Ordenar una lista
.sort()
lista.sort()# ordena de menor a mayor, usar con (reverse=True) para ordenar de mayor a menor
lista.reverse()# ordena los elementos al revés del orden guardado

Quitar elementos de una lista
.pop()
lista.pop(i)# quita el elemento en indice i y devuelve su valor
.remove()
lista.remove(x)# quita el primer elemento de la lista con valor x
lista.clear()# vacia la lista

del lista# borra la lista
del lista[i]# borra el elemento en indice i

Diccionarios { key : value , }

diccionario = {x:y} compuestos por un key(x) unica y un valor(y) (cualquier tipo de datos)
dict()
variable = dict(x=y, m=n) crear un diccionario
dicc.copy() crear una copia
len(dicc) devuelve el no. de elementos (x:y) hay en el diccionario
sorted(dicc) ordena los keys; usar con .items() para ordenar tuplas de los elementos o .values() para ordenar los valores solos

Diccionarios – Metodos

Obtener informacion de un diccionario
dicc.keys() devuelve todas las keys
dicc.values() devuelve todos los valores
dicc.items() devuelve tuplas de los key:value
in/not in comprobar si existe una clave
dicc.get(x, y) devuelve el valor asociado al key x, o si no existe devuelve el output y
dicc["key"] devuelve el valor del key (ver abajo que tiene mas usos)

Ampliar un diccionario
.update()
dicc.update({x:y})# para insertar nuevos elementos
dicc["key"] = valor# para insertar un nuevo key o valor, o cambiar el valor de un key
dicc.setdefault(x, y)# devuelve el value del key x, o si no existe la key x, la crea y asigna el valor y por defecto

Quitar elementos de un diccionario
dicc.pop(x)# elimina la key x (y lo devuelve)
dicc.popitem()# elimina el ultimo par de key:value
dicc.clear()# vacia el diccionario

Tuplas (,) inmutables, indexados

tupla = (x,y) tuplas se definen con () y , o solo ,
tupla1 + tupla2 juntar tuplas

tuple(lista) crear tuplas de una lista
tuple(dicc) crear tuplas de los keys de un diccionario
tuple(dicc.values()) crear tuplas de los valores
tuple(dicc.items()) crear tuplas de los key:valores

len(tupla) devuelve el no. de elementos
in/not in comprobar si hay un elemento
tupla.index(x) devuelve el indice de x
tupla.count(x) devuelve el no. de elementos con valor x en la tupla

para cambiar el contenido de una tupla hay que convertirla en una lista y luego a tupla

zip()

zip(iterable1, iterable2) crea una lista de tuplas de parejas de los elementos de las dos listas (mientras se puede)
listzip.sort() ordena las tuplas del zip por el primer elemento

Sets {} no permiten duplicados, no tienen orden

set = {x,y}
set(iterable) solo permite un argumento iterable; elimina duplicados

in/not in comprobar si hay un elemento
len(set) devuelve el no. de elementos

Ampliar un set
set.add(x)# añadir un elemento
set.update(set o lista)# añadir uno o mas elementos con [] o {} o un variable tipo lista o set

Quitar elementos de un set
set.pop()# elimina un elemento al azar
set.remove(x)# elimina el elemento x
set.discard(x)# elimina el elemento x (y no devuelve error si no existe)
set.clear()# vacia el set

Operaciones con dos Sets

set1.union(set2) devuelve la union de los dos sets: todos los elementos menos dupl.
set1.intersection(set2) devuelve los elementos comunes de los dos sets
set1.difference(set2) devuelve los sets que estan en set1 pero no en set2 (restar)
set1.symmetric_difference(set2) devuelve todos los elementos que no estan en ambos
set1.isdisjoint(set2) comprobar si todos los elementos de dos sets son diferentes
set1.issubset(set2) comprobar si todos los elementos de set1 estan en set2
set1.superset(set2) comprobar si todos los elementos de set2 estan en set1

input()

• permite obtener texto escrito por teclado del usuario
input("el texto que quieres mostrar al usuario")
• se puede guardar en un variable
• por defecto se guarda como un string
x = int(input("escribe un numero")) para usar el variable como integer o float se puede convertir en el variable

Sentencias de control

if ... elif ... else
if estableca una condición para que se ejecute el código que esta debajo del if. *tiene que estar indentado*
elif para chequear mas condiciones después de un if
else agrupa las condiciones que no se han cumplido; no puede llevar condiciones nuevas

if x > y:
 print("x es mayor que y")
elif x == y:
 print("x es igual que y")
else:
 print("x e y son iguales")

while
• repite el código mientras la condición sea True, o sea se parará cuando la condición sea False
• se pueden incluir condiciones con if... elif... else
• *pueden ser infinitos* (si la condición no llega a ser False)

while x < 5:
 print("x es mayor que 5")

For loops

• sirven para iterar por todos los elementos de un variable que tiene que ser un iterable (lista, diccionario, tupla, set, or string)
• se pueden combinar con if ... elif ... else, while, u otro for loop
• en diccionarios por defecto intera por las keys; podemos usar dicc.values() para acceder a los valores
for i in lista:
 print("hola mundo")

List comprehension

• su principal uso es para crear una lista nueva de un un for loop en una sola línea de código
[**lo que queremos obtener** **iterable** **condición** (opcional)]

try ... except

Se usan para evitar que nuestro código se pare debido a un error en el código. Se puede imprimir un mensaje que avisa del error.
try:
 print("2.split())
except:
 print("no funciona")

range()

• nos devuelve una lista de numeros que por defecto se aumentan de uno en uno empezando por 0
range(start:stop:step)
• se puede especificar por donde empieza y el limite (que debe ser +1 por que se para uno antes del limite que ponemos como stop)
• tambien se puede especificar saltos

Python Cheat Sheet 2
Funciones
Definir una funcion: <code>def nombre_funcion(parametro1, parametro2, ...): return valor_del_return</code>
Lllamar una funcion: <code>nombre_funcion(argumento1, argumento2, ...)</code>
return: es opcional, pero sin return devuelve None parametros por defecto: – siempre deben ser lo ultimo
*args: una tupla de argumentos sin limite **kwargs: diccionarios cuyas keys se convierten en parámetros y sus valores en los argumentos de los parámetros
<code>def nombre_funcion(parametros, *args, **kwargs, parametro_por_defecto = valor) arg/kwarg: sin */** dentro de la funcion arg[0]</code>
Lllamar una funcion con *args: <code>nombre_funcion(argumento, argumento, argumento, ...) o nombre_funcion(*[lista_o_tupla_de_args])</code>
Lllamar una funcion con **kwargs: <code>nombre_funcion(**diccionario)</code>
Clases
Definir una clase: <code>class NombreClase:</code>
<code> def __init__(self, atributo1, atributo2): self.atributo1 = atributo1 self.atributo2 = atributo2 self.atributo_por_defecto = ‘valor’</code>
<code> def nombre_funcion1(self, parametros) self.atributo += 1 return f“el nuevo valor es {self.atributo}”</code>
Definir una clase hija: <code>class NombreClaseHija(NombreClaseMadre): def __init__(self, atributo1, atributo2): super().__init__(atributo_hereditado1, ...)</code>
<code> def nombre_funcion_hija (self, parametros):</code>
Crear un objeto de la clase: <code>variable_objeto = NombreClase(valor_atributo1, valor_atributo2)</code> instanciar (crear) un objeto <code>variable_objeto.atributo</code> devuelve el valor del atributo guardado para ese objeto <code>variable_objeto.atributo = nuevo_valor</code> para cambiar el valor del atributo <code>variable_objeto.nombre_funcion()</code> llamar una funcion
<code>print(help(NombreClase))</code> imprime informacion sobre la clase

Regex
- una abreviatura de `expresión regular`, `regex` es una cadena de texto que permite crear patrones que ayudan a emparejar, localizar y gestionar strings <code>import re</code> para poder trabajar con regex
Operadores communes de regex + coincide con el carácter precedente una o más veces * coincide con el carácter precedente cero o más veces u opcional ? indica cero o una ocurrencia del elemento precedente . coincide con cualquier carácter individual ^ coincide con la posición inicial de cualquier string \$ coincide con la posición final de cualquier string
Sintaxis básica de regex \\w cualquier caracter de tipo alfabético \\d cualquier caracter de tipo numérico \\s espacios \\n saltos de línea \\w cualquier caracter que no sea una letra \\D cualquier caracter que no sea un dígitos \\S cualquier elemento que no sea un espacio () aísla sólo una parte de nuestro patrón de búsqueda que queremos devolver [] incluye todos los caracteres que queremos que coincidan e incluso incluye rangos como este: a-z y 0-9 es como el operador ‘or’ \ señala una secuencia especial (escapar caracteres especiales) { } Exactamente el número especificado de ocurrencias {n} Exactamente n veces {n,} Al menos n veces {n,m} Entre n y m veces
Métodos Regex <code>re.findall(“patron”, string)</code> busca en todo el string y devuelve una lista con todas las coincidencias en nuestro string <code>re.search(“patron”, string_original)</code> busca en todo el string y devuelve un objeto con la primera coincidencia en nuestro string <code>re.match(“patron”, “string_original)</code> busca en la primera linea del string y devuelve un objeto con la primera coincidencia en nuestro string <code>resultado_match.span()</code> devuelve la referencia de las posiciones donde hizo el “match” <code>resultado_match.group()</code> devuelve el element resultando de la coincidencia del “match” <code>re.split(“patron”, “string_original”) </code> busca en todo el string y devuelve una lista con los elementos separados por el patron <code>re.sub(“patron”, “string_nuevo”, “string_original”) </code> busca en todo el string y devuelve un string con el element que coincide

Modulos/Librerias (paquetes de funciones)
Importar y usar modulos y sus funciones <code>import modulo</code> para importar un modulo <code>from modulo import funcion</code> importar solo una funcion <code>modulo.funcion()</code> usar una funcion de un modulo <code>modulo.clase.funcion()</code> para usar una funcion de una clase <code>import modulo as md</code> asignar un alias a un modulo
Libreria os <code>os.getcwd()</code> devuelve la ruta de donde estamos trabajando; se puede guardar en un variable e.g. ruta = os.getcwd() <code>os.listdir()</code> devuelve una lista de los archivos y carpetas donde estamos trabajando <code>os.listdir(‘carpeta’)</code> devuelve los contenidos de otra carpeta <code>os.chdir(‘ruta’)</code> cambia la carpeta en la que estes <code>os.mkdir(‘nueva_carpeta’)</code> crear una nueva carpeta <code>os.rename(‘nombre_carpeta’, ‘nueva_nombre’)</code> cambia el nombre de una carpeta <code>os.rmdir(‘carpeta’)</code> borra la carpeta
Libreria shutil from shutil import rmtree <code>rmtree(‘carpeta’)</code> borra la carpeta y subcarpetas
Abrir y cerrar ficheros Primero hay que guardar la ruta del archivo: ubicacion_carpeta = os.getcwd() nombre_archivo = “text.txt” ubicacion_archivo = ubicacion_carpeta + “/” + nombre_archivo f = open(ubicacion_archivo) abrir un archivo en variable f f.close() cerrar un archivo * IMPORTANTE * with open(ubicacion_archivo) as f: codigo e.g. variable = f.read() abre el archivo solo para ejecutar el codigo indicado (y despues lo deja)
Encoding from locale import getpreferredencoding getpreferredencoding() para saber que sistema de encoding estamos usando f = open(ubicacion_archivo, encoding="utf-8") abrir un archivo y leerlo con el encoding usado; guardar con .read()
mode: argumento opcional al abrir un archivo r – read w – write - sobreescribe x – exclusive creation, sólo crearlo si no existe todavía a – appending, añadir texto al archivo sin manipular el texto que ya habia hay que anadir otra letra: t – texto – leer en texto b – bytes – leer en bytes (no se puede usar con encoding)
<code>f = open(ubicacion_archivo, mode = “rt”)</code>
Leer ficheros f.read() leer el contenido de un archivo f.read(n) leer los primeros n caracteres de un archivo variable = f.read() guardar el contenido del archivo (o n caracteres de un archivo) en un variable f.readline(n) por defecto devuelve la primera linea o n lineas f.readlines() devuelve una lista de todas las lineas del archivo (cada linea es un elemento); se usa vacio sin n y list_name[x:] para seleccionar lineas especificas
Escribir en ficheros with open(ubicacion_archivo, “w”) as f: f.write(“Texto que va en el fichero.”) para escribir with open(ubicacion_archivo, “a”) as f: f.write(“Texto que va en el fichero.”) para anadir texto f.writelines(‘lista’) para anadir lineas de texto de una lista

Ficheros xml
<code>import xml.etree.ElementTree as ET</code> importa la librería xml <code>variable_tree = ET.parse(‘ruta/archivo.xml’)</code> abre el archivo <code>variable_root = variable_tree.getroot()</code> saca el elemento que envuelve todo (el elemento raíz) en una lista <root> <child_tag atributo1=“valor” atributo2=valor> <subchild_tag> elemento </subchild_tag> </child_tag> </root> <code>variable_root.tag</code> devuelve el nombre del tag del raiz <code>variable_root.attrib</code> devuelve los atributos del fichero
<code>variable_root.find(“tag”).find(“childtag”).text</code> devuelve la primera ocasión en que el tag de un elemento coincida con el string <code>variable_root.findall(“tag”).findall(“childtag”).text</code> devuelve todos los elementos cuyos tag coincide
MySQL Connector/Python
Conectar a una base de datos <code>import mysql.connector</code> para importar MySQL Connector pip install mysql-connector pip install mysql-connector-Python <code>connect()</code> para conectar a una base de datos: <code>variable_cnx = mysql.connector.connect(user='root', password='AlumnaAdalab', host='127.0.0.1', database='nombre_BBDD')</code> <code>from mysql.connector import errorcode</code> importar errores <code>mysql.connector.Error</code> se puede usar en un try/except <code>cnx.close()</code> desconectar de la base de datos
Realizar queries <code>variable_cursor = cnx.cursor()</code> crear el objeto cursor que nos permite comunicar con la base de datos <code>variable_cursor.close()</code> desconectar el cursor <code>variable_query = (“SQL Query”)</code> guardar un query en un variable <code>variable_cursor.execute(variable_query)</code> ejecutar el query; devuelve una lista de tuplas <code>import datetime</code> sacar fechas en el formato AAAA-MM-DD <code>datetime.date(AAAA, M, D)</code> devuelve el formato de fecha <code>variable_query = “SQL Query... %s AND %s”</code> query dinamica <code>variable_cursor.execute(query, (variable1, variable2))</code> valores que van en lugar de los %s <code>variable_cursor.execute(“SHOW DATABASES”)</code> mostrar las BBDD <code>variable_cursor.execute(“SHOW TABLES”)</code> mostrar las tablas de la BBDD indicado en la conexión <code>variable_cursor.execute(“SHOW TABLES”)</code> <code>variable_cursor.execute(“SHOW COLUMNS FROM bbdd.table”)</code> mostrar las columnas de la tabla especificada; hay que conectarse a la bbdd information_schema
Argumentos cursor: <code>variable_cursor = cnx.cursor([arg=value[, arg=value]...])</code> <code>buffered=True</code> devuelve todas las filas de la bbdd <code>raw=True</code> el cursor no realizará las conversiones automáticas entre tipos de datos <code>dictionary=True</code> devuelve las filas como diccionarios <code>named_tuple=True</code> devuelve las filas como named tuples <code>cursor_class</code> un argumento que se puede usar para indicar que subclase queremos usar para instanciar el nuevo cursor

Obtener resultados de una query <code>variable_cursor.fetchone()</code> devuelve el primer resultado <code>variable_cursor.fetchall()</code> devuelve todos los resultados como iterable – cada fila es una tupla
Pandas dataframe with SQL <code>import pandas as pd</code> <code>variable_df = pd.DataFrame(variable_resultado_fetchall, columns = [‘columna1’, ‘columna2’, ...])</code> crear un dataframe con los resultados de una query en una variable <code>variable_df.head(n)</code> devuelve las n primeras filas del df, o 5 por defecto <code>variable_df = pd.read_sql_query(variable_query, variable_cnx)</code> convertir los resultados de la query en df <code>pd.read_sql(variable_query, variable_cnx)</code> <code>variable_df.to_csv(“nombre_archivo.csv”)</code> guardar en csv <code>variable_df.to_string()</code> formatear el dato en string <code>variable_df.to_latex()</code> formatear el dato en un string que facilite la inserción en un documento latex
Crear y alterar una base de datos <code>variable_cursor.execute(“CREATE DATABASE nombre_BBDD”)</code> <code>variable_cursor.execute(“CREATE TABLE nombre_tabla (nombre_columna TIPO, nombre_columna2 TIPO2)”)</code> <code>variable_cursor.execute(“ALTER TABLE nombre_tabla ALTERACIONES”)</code>
Insertar datos <code>variable_query = “INSERT INTO nombre_tabla (columna1, columna2) VALUES (%s, %s)”</code> <code>variable_valores = (valor1, valor2)</code> <code>variable_cursor.execute(variable_query, variable_valores)</code> otro método: <code>variable_query = “UPDATE nombre_tabla SET nombre_columna = “nuevo_valor” WHERE nombre_columna = “valor”</code>
Insertar múltiples filas a una tabla <code>variable_valores_en_tuplas = ((valor1columna1, valor1columna2), (valor2columna1, valor2columna2))</code> <code>variable_cursor.executemany(variable_query, variable_valores_en_tuplas)</code>
<code>variable_conexion.commit()</code> después de ejecutar la inserción, para que los cambios efectúen en la BBDD <code>variable_conexion.rollback()</code> se puede usar después de execute y antes de commit para deshacer los cambios <code>print(variable_cursor.rowcount, “mensaje”)</code> imprimir el numero de filas en las cuales se han tomado la accion
Eliminar registros <code>variable_query = “DROP TABLE nombre_tabla”</code>
Añadir errores importar errorcode y usar try/except: <code>try: accion except mysql.connector.Error as err: print(err) print("Error Code:", err.errno) print("SQLSTATE", err.sqlstate) print("Message", err.msg)</code>

Python Cheat Sheet 3

Pandas

Series: estructuras en una dimension

Crear series

serie = pd.Series() crear serie vacía
serie = pd.Series(array) crear serie a partir de un array con el indice por defecto
serie = pd.Series(array, index = ['a', 'b', 'c'...]) crear una serie con indice definida; debe ser lista de la misma longitude del array
serie = pd.Series(lista) crear una seria a partir de una lista
serie = pd.Series(numero, indice) crear una serie a partir de un escalar con la longitude igual al numero de indices
serie = pd.Series(diccionario) crear una serie a partir de un diccionario

Acceder a informacion de una serie

serie.index devuelve los indices
serie.values devuelve los valores
serie.shape devuelve la forma (no. filas)
serie.size devuelve el tamaño
serie.dtypes devuelve el tipo de dato

serie[i] devuelve el valor del elemento en indice i
serie[[i,j]] devuelve el valor de los dos elementos
serie[i:m] devuelve el valor de un rango

serie["etiqueta"] devuelve el valor de los elementos en indices i y j

Operaciones con series

serie1 +/- serie2 suma/resta/multiplica/divide las filas con indices comunes entre las dos series
serie1.add(serie2, fill_value = numero) suma las filas con indices comunes, y suma el fill value a los valores sin indice comun
serie1.sub(serie2, fill_value = numero) restan las filas de la seria2 de la serie1 cuando tienen indices comunes, y resta el fill value de las otras indices de serie1
serie1.mul(serie2, fill_value = numero) multiplica las filas con indices comunes y multiplica el fill value con las otras *usar 1 para conservar el valor*
serie1.mul(serie2, fill_value = numero) divida las filas de la serie1 entre las de la serie2 cuando tienen indices comunes, y divide las otras por el fill value
serie1.mod(serie2, fill_value = numero) devuelve el modulo (division sin resta)
serie1.pow(serie2, fill_value = numero) calcula el exponencial
serie1.ge(serie2) compara si serie1 es mayor que serie2 y devuelve True o False
serie1.le(serie2) compara si serie1 es menor que serie2 y devuelve True o False

Filtrado booleanos

serie < > >= <= == valor devuelve True o False segun si cada condicion cumple la condicion
serie1[serie1 < > >= <= == valor] devuelve solo los valores que cumplen la condicion
np.nan crear valor nulo (NaN)
serie.isnull() devuelve True o False segun si los valores existen o son nulos ("" no cuenta como nulo)
serie.notnull() devuelve True o False segun si los valores existen o son nulos ("" no cuenta como nulo)

DataFrames: estructuras en dos dimensiones

Crear DataFrames

df = pd.DataFrame(data, index, column)
data: NumPy Array, diccionario, lista de diccionarios
index: indice que por defecto se asigna como 0-(n-1), n siendo el numero de filas;
index = [lista] para asignar “etiquetas” (nombres de filas)
column: nombre de las columnas; por defecto 0-(n-1);
columns =[lista] para poner mas nombres

df = pd.DataFrame(array) crear un dataframe a partir de un array con indices y columnas por defecto
df = pd.DataFrame(diccionario) crear un dataframe a partir de un diccionario – los keys son los nombres de las columnas

Acceder a informacion de un DataFrame

df.loc[“etiqueta_fila”, “etiqueta_columna”] devuelve el contenido de un campo en una columna de una fila
df.loc[“etiqueta_fila”,:] devuelve los valores de todas las columnas de una fila
df.loc[:,“etiqueta_columna”] devuelve los valores de todas las filas de una columna
df.iloc[indice_fila, indice_columna] devuelve el contenido de un campo en una columna de una fila
df.iloc[indice_fila, :] devuelve los valores de todas las columnas de una fila
df.iloc[:,indice_columna] devuelve el contenido de un campo en una columna de una fila
df.loc[[lista_etiquetas_filas], [lista_etiquetas_columnas]] devuelve el contenido de varias filas / varias columnas
df.loc[[lista_indices_filas], [lista_indices_columnas]] devuelve el contenido de varias filas / varias columnas - se puede usar los indices/rangos de las listas [start:stop:step] dentro de los loc/iloc
df.loc[df.etiqueta > x] seleccionar datos basado en una condicion usando operadores comparativos
df.loc[(df.etiqueta > x) & (df.etiqueta == y)] seleccionar datos que tienen que cumplir las dos condiciones (and)
df.loc[(df.etiqueta > x) | (df.etiqueta == y)] seleccionar datos que tienen que deben cumplir una de las dos condiciones (or)
df.iloc[list(df.etiqueta > x), :] iloc no acepta una Serie booleana; hay que convertirla en lista
variable_df.head(n) devuelve las n primeras filas del df, o 5 por defecto

Crear columnas

df[“nueva_columna”] = (df[“etiqueta_columna”] + x) crea una nueva columna basada en otra
df.assign(nueva_columna= df[“etiqueta_columna”] + x) crea una nueva basada en otra
df.assign(nueva_columna= [lista_valores]) crea una nueva columna de una lista de valores *tiene que ser de la misma longitud como el numero de filas del dataframe*
df.insert[indice_nueva_columna, “nombre_columna”, valores] crea una nueva columna en la indice indicada
allow_duplicates = True parametro cuando queremos permitir columnas duplicadas (por defecto es False)

NumPy (Numerical Python)

Informacion sobre arrays

array = np.array([valorfila1columna1, valorfila1columna2], [valorfila2columna1, valorfila2columna2], etc)

array.shape devuelve la forma de nuestro array (filas, columnas)
array.size devuelve el numero de elementos de nuestro array (filas * columnas)
array.ndim devuelve el numero de dimensiones de nuestro array (que pueden ser de 1D, 2D, o 3D)
array.dtype devuelve el tipo de datos contenidos en nuestro array (tiene que ser solo un tipo)

Crear arrays con valores aleatorios

array = np.random.randint(inicio, final, forma_matriz) crea un array de numeros aleatorios entre dos valores;
forma_matriz: (z,x,y)
z: numero de arrays
x: numero de filas
y: numero de columnas
array = np.random.randint(inicio, final) devuelve un numero aleatorio en el rango
array = np.random.rand(z,x,y) crea un array de floats aleatorias con la forma que le especificemos; por defecto genera numeros aleatorios entre 0-1
array = np.random.random_sample((z,x,y)) crea un array de floats aleatorias con la forma que le especificemos; por defecto genera numeros aleatorios entre 0-0.9999999...
array = np.random.random_sample(size=None) devuelve un numero aleatorio en 0 y 0.9999999999999...
np.round(np.random.rand(z,x,y), n) crear array con floats de n decimales

Crear arrays de listas

array = np.array(lista, dtype= tipo) crea un array unidimensional de una lista
array = np.array([lista1, lista2]) crea un array bidimensional de dos listas
array = np.array([listadelistas1, listadelistas2]) crea un array bidimensional de dos listas

Crear otros tipos de arrays

array = np.arange(valor_inicio, valor_final, saltos) crea un array usando el formato [start:stop:step]
array = np.ones(z,x,y) crea un array de todo unos de la forma especificada
array2 = np.ones_like(array1) crea un array de todo unos de la forma basada en otra array
array = np.zeros(z,x,y) crea un array de todo zeros de la forma especificada
array2 = np.zeros_like(array1) crea un array de todo zeros de la forma basada en otra array
array = np.empty((z,x,y), tipo) crea un array vacio con datos por defecto tipo float
array2 = np.empty_like(array1) crea un array vacia con la forma basada en otra array
array = np.eye(z,x,y, k = n) crea un array con unos en diagonal empezando en la posicion k
array = np.identity(x) crea una matriz de identidad con ceros en filas y unos en la diagonal, de forma cuadrada

Operaciones con arrays

np.add(array1, array2) suma dos arrays
np.subtract(array1, array2) resta el array2 del array1
np.multiply(array1, array2) multiplica dos arrays
np.divide(array1, array2) divide el array1 por el array2

Operaciones con escalares (un número)

array + x
x * array etc. – con cualquier operador algebraico