PYTHON

Contenido

[PRINT 3](#_Toc83204791)

[VARIABLES 3](#_Toc83204792)

[TIPOS DE DATOS 3](#_Toc83204793)

[STRINGS 3](#_Toc83204794)

[CONVERSIÓN DE DATOS 4](#_Toc83204795)

[DATOS ESTRUCTURADOS 4](#_Toc83204796)

[LIST 4](#_Toc83204797)

[DICTIONARY 5](#_Toc83204798)

[TUPLE 5](#_Toc83204799)

[FUNCIONES GENERALES 6](#_Toc83204800)

[INPUT 6](#_Toc83204801)

[RANGE 6](#_Toc83204802)

[ERROR HANDLING 6](#_Toc83204803)

[NAMEERROR 6](#_Toc83204804)

[SYNTAXERROR 6](#_Toc83204805)

[TYPEERROR 6](#_Toc83204806)

[INDEXERROR 6](#_Toc83204807)

[KEYERROR 6](#_Toc83204808)

[ATTRIBUTEERROR 6](#_Toc83204809)

[VALUEERROR 7](#_Toc83204810)

[FUNCIONES DEFINIDAS 7](#_Toc83204811)

[EJEMPLO SIMPLE 7](#_Toc83204812)

[EJEMPLO CON PARAMETRO 7](#_Toc83204813)

[EJEMPLO CON RETURN 7](#_Toc83204814)

[EJEMPLO CON DOBLE LLAMADO 7](#_Toc83204815)

[EJEMPLO CON PARAMETROS OPCIONALES 7](#_Toc83204816)

[EJEMPLO COMPLEJO 8](#_Toc83204817)

[SLICING 8](#_Toc83204818)

[OPERATORS 8](#_Toc83204819)

[Assignment Operators 8](#_Toc83204820)

[Arithmetic Operators 9](#_Toc83204821)

[LOGICAL OPERATORS 9](#_Toc83204822)

[RELATIONAL OR COMPARISON 9](#_Toc83204823)

[MEMBERSHIP OPERATORS 9](#_Toc83204824)

[IDENTITY OPERATORS 9](#_Toc83204825)

[BITWISE OPERATORS 9](#_Toc83204826)

[.format() 10](#_Toc83204827)

[.join() 10](#_Toc83204828)

[.split() 10](#_Toc83204829)

[.strip() 10](#_Toc83204830)

[dir() 10](#_Toc83204831)

[DATOS ANIDADOS 10](#_Toc83204832)

[IF-ELIF-ELSE 11](#_Toc83204833)

[FOR LOOP 11](#_Toc83204834)

[WHILE LOOP 11](#_Toc83204835)

[BREAK & CONTINUE 12](#_Toc83204836)

[LAMBDA 12](#_Toc83204837)

[ZIP() 12](#_Toc83204838)

[MAP() 13](#_Toc83204839)

[FILTER() 13](#_Toc83204840)

[SORTED() 13](#_Toc83204841)

[List Comprehension 14](#_Toc83204842)

[Dict Comprehension 14](#_Toc83204843)

[HELP() 15](#_Toc83204844)

[DEBUGGING 15](#_Toc83204845)

[PASS STATEMENT 15](#_Toc83204846)

[JSON 15](#_Toc83204847)

[WEB REQUESTS 16](#_Toc83204848)

[TRY/EXCEPT 17](#_Toc83204849)

# PRINT

EJEMPLO 1

print("Hello" , "My name is" , "John")

EJEMPLO 2

print("Model S" , "Model 3" , sep="--")

EJEMPLO 3

print("USA" , "France" , end="|")

EJEMPLO 4

my\_text="probar resultado"

print(my\_text)

# VARIABLES

EJEMPLO 1

glass\_of\_water= 3

print("I drank", glass\_of\_water, "glasses of water today.")

EJEMPLO 2

glass\_of\_water=3

glass\_of\_water=glass\_of\_water + 1

print(glass\_of\_water)

# TIPOS DE DATOS

1. **int** : this type of data consists of numbers and particularly integers.

men\_stepped\_on\_the\_moon= 12

print(men\_stepped\_on\_the\_moon)

1. **float** : this type of data refers to numbers with decimals

global\_mean\_sea\_level\_2018=21.36

print(global\_mean\_sea\_level\_2018)

1. **str** : standing for string, an str type of variable keeps data as a text string.

my\_reason\_for\_coding='PRUEBA'

print(my\_reason\_for\_coding)

1. **bytearray**
2. **datetime**
3. **boolean**

staying\_alive=True

print(staying\_alive)

**print(type(staying\_alive))**

## STRINGS

Ejemplo: str = str.upper()

**Metodos:**

* **.replace ():** se usa para cambiar subcadenas en una cadena
* **.lower ():** se utiliza para cambiar los caracteres de una cadena a caracteres inferiores
* **.upper ():** se usa para cambiar los caracteres de una cadena a letras mayúsculas
* **.capitalize ():** se utiliza para convertir el primer carácter de la cadena en letra mayúscula.
* **.startswith ():** se usa para verificar si la cadena comienza con una letra específica
* **.endswith (): se usa para verificar si la cadena termina con una letra específica**
* **.join (): un método de cadena muy útil que se utiliza para unir una cadena especificada a la cadena**
* **.split (): otro método muy útil que se utiliza para dividir una cadena en función de los caracteres especificados**
* **.strip (): se usa para quitar una cadena de caracteres especificados, incluidos los espacios vacíos**
* **.index (): se usa para identificar el índice de un carácter en una cadena. Si no encuentra index devuelve error**
* **.find (): también se usa para encontrar el índice de un carácter en una cadena. Si no encuentra find devuelve -1.**
* **.count (): se usa para contar una subcadena en una cadena**
* **type (): función utilizada para identificar el tipo de datos, se puede utilizar en cadenas.**
* **Elemento de la cadena:  print(msg[0])**

# CONVERSIÓN DE DATOS

EJEMPLO 1

men\_stepped\_on\_the\_moon=12

answer\_1=type(men\_stepped\_on\_the\_moon)

print(answer\_1)

EJEMPLO 2

my\_grade="10"

answer\_5=int(my\_grade)

print(answer\_5)

EJEMPLO 3

my\_temp=97.70

answer\_6=int(my\_temp)

print(answer\_6)

EJEMPLO 4

gross\_world\_product=84.84

gwp\_str=str(gross\_world\_product)

answer\_8="In 2018 gross product of the world (GWP) was " + gwp\_str + " in trillion US dollars."

print(answer\_8)

# DATOS ESTRUCTURADOS

**list : []  
tuple : ()  
dictionary : {}**

## LIST

gift\_list=["prueba 1",3,3.43,"prueba 2"]

print(gift\_list)

print(gift\_list[3]) -> “prueba 2”

lst=[1,2,3,4]

for i in lst:

print(i)

import random

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

random.shuffle(lst)

print(lst)

**Características:**

* **Indexing:** Tiene Index (comienza con 0), en cambio los diccionarios tienen llaves. También los elementos se pueden llamar por el index inverso (comienza con -1).
* **Mutability:** puede agregar y eliminar sus elementos como desee, en cambio en las tuplas no se puede.
* **Múltiples tipos de datos:** Las listas también pueden contener integers, floats, strings, dictionaries, tuples o incluso otras listas o una combinación de cualquiera de ellos.
* **Sintaxis:** [,].Es necesario seguir la sintaxis con respecto a los tipos de datos dentro de las listas.

**Metodos:**

* **.append():** agregar elementos a la lista. Ej: p\_datatypes.append(“Tribeca”)
* **.insert():** insertar un elemento con un índice particular de la lista. Ej: p\_datatypes.insert(0, “Tribeca”)
* **.index():** mostrar el índice de un elemento.
* **.clear():** borrar todos los elementos de la lista.
* **.remove():** eliminar un elemento de la lista
* **.reverse():** reversar el orden de la lista
* **.count():** cuenta los elementos de la lista
* **sum():** suma los elementos de la lista
* **min():** muestra el elemento de menor valor de la lista
* **max():** muestra el elemento de máximo valor de la lista
* **.sort():** Ordena la lista. Ejemplos:
  + **lst.sort(reverse = False)**
  + **key\_list = list(dict.keys())**

**key\_list.sort()**

* **.pop():** Eliminará el elemento especificado de la lista (o diccionario) y devolverá el elemento eliminado al mismo tiempo.

## DICTIONARY

grocery\_list={"prueba 1":1,"prueba 2":2}

print(grocery\_list)

**Características:**

* **No tiene índice, cada item cuenta con una llave y un valor**
* **No tiene orden**
* **Sintaxis:** {LLAVE:VALOR,}.Es necesario seguir la sintaxis con respecto a los tipos de datos dentro.

**Metodos:**

* **.keys ():** se usa para mostrar las claves en un diccionario
* **.items ():** se utiliza para crear una tupla de clave-valor (más sobre esto más adelante)
* **.get ():** se usa para devolver el valor de una clave

p\_ages = {"Tom": 32, "Jess": 29, "Pete": 18}

print ( p\_ages.get ("Jonah", "Persona no encontrada") )

* **.clear ():** borra todo el diccionario
* **.copy ():** usado para copiar un diccionario
* **len ():** se usa para obtener la longitud de un diccionario
* **type ():** usado para decir el tipo
* **min ():** se usa para identificar la clave con el valor más alto
* **max ():** se usa para identificar la clave con el valor más bajo
* **Actualizar Diccionario:** dict["born"] = -428
* **Agregar ítem:** dict["work"] = ["Apology", "Phaedo", "Republic", "Symposium"]
* **Combinar Diccionarios: dict3 =  dict1 | dict2**
* **Modificar Diccionarios: dict2 | = dict1**
* **.pop():** Eliminará el elemento especificado de la lista (o diccionario) y devolverá el elemento eliminado al mismo tiempo.

## TUPLE

bucket\_list=("PRUEBA",3,4.56)

print(bucket\_list)

**Características:**

* **Es igual a las listas pero no es modificable. No se pueden realizar inserciones, modificaciones, y reordenamiento.**
* **Sintaxis:** (,).Es necesario seguir la sintaxis con respecto a los tipos de datos dentro de las listas.

**Metodos:**

* **.index():** mostrar el índice de un elemento.
* **.count():** cuenta los elementos de la lista
* **sum():** suma los elementos de la lista
* **min():** muestra el elemento de menor valor de la lista
* **max():** muestra el elemento de máximo valor de la lista

# FUNCIONES GENERALES

## INPUT

name = input("Please enter your name.")

print(name)

La entrada devuelve una cadena.

EJEMPLO 1

birthday\_list={'mom': '1/5/1862', 'dad': '5/20/1860',

'son': '8/12/1930', 'sister': '10/15/1885'}

answer=input("Family member please")

if answer in birthday\_list:

print("{}'s birthday is {}".format(answer.capitalize(), birthday\_list[answer]))

else:

print("No record")

EJEMPLO 2

message = input("How many miles would you like to convert?")

result = int(message)\*1.609

print(result)

## RANGE

range(start, stop, step)

Sin embargo, si escribe solo un número dentro de la función range (), asumirá inteligentemente que el inicio es 0 (cero), el paso es 1 y la parada es el número que ha especificado.

Rango es un tipo de dato.

Ejemplo:

rng = range(0,10,2)

print(list(rng))

# ERROR HANDLING

## NAMEERROR

Obtendrá un NameError cuando no se pueda encontrar un objeto en Python.

## SYNTAXERROR

Obtendrá un SyntaxError cuando cometa un error de sintaxis en Python. Puede ser una cita faltante o un paréntesis.

## TYPEERROR

Obtendrá un TypeError cuando aplique una operación o función al tipo incorrecto de datos, como aplicar operaciones aritméticas a cadenas.

## INDEXERROR

Obtendrá un IndexError cuando intente alcanzar un índice fuera de los límites de sus datos.

## KEYERROR

KeyError es como IndexError para diccionarios. Si intenta acceder a una clave que no está incluida en su diccionario, obtendrá un KeyError.

## ATTRIBUTEERROR

El error de atributo ocurre cuando intenta utilizar un atributo o método que no se aplica a los datos específicos en los que está trabajando. Por ejemplo, intentar aplicar el método .reverse () en una cadena.

## VALUEERROR

El error de valor se produce cuando aplica una función a un tipo de datos correctamente, pero el contenido no es adecuado para esa operación.

# FUNCIONES DEFINIDAS

## EJEMPLO SIMPLE

* **Sintaxis:**

def helloworld():

print("Hello World!")

* **Llamado:**

helloworld()

## EJEMPLO CON PARAMETRO

* **Sintaxis:**

def per\_hundred(x):

print(x/100)

* **Llamado:**

per\_hundred(5000)

* **Resultado:**

50

## EJEMPLO CON RETURN

* **Sintaxis:**

def per\_hundred(x):

return x/100

* **Llamado:**

my\_answer = per\_hundred(5000)

print (my\_answer)

* **Resultado:**

50

## EJEMPLO CON DOBLE LLAMADO

* **Sintaxis:**

def plus\_two(x):

print (x+2)

return (x+2)

* **Llamado:**

a, b = plus\_two(2), plus\_two(6)

print(a, b)

* **Resultado:**

4, 8

## EJEMPLO CON PARAMETROS OPCIONALES

* **Sintaxis:**

def fave\_color(name, color="azure"):

msg = name+"'s favorite color is: "+color

return msg

* **Llamado 1:**

print(fave\_color("Pete"))

* **Resultado 1:**

El color favorito de Pete es: azul

* **Llamado 2:**

print(fave\_color(name="Olivia", color="violet"))

* **Resultado 2:**

El color favorito de Olivia es: violeta

## EJEMPLO COMPLEJO

**Function life expectancy calculation**

def new\_life (name, age):

if smoker(name) == True:

life\_exp = 150

else:

life\_exp = 200

life\_remaining = life\_exp - age

msg = "Hi "+name+"! Your new expected remaining life is: "+str(life\_remaining)+" years."

return msg

**Function for smoker database check**

def smoker (name='', smoke=False):

if name.capitalize() in ["Jane", "Zack", "Melissa"]:

smoke = True

return smoke

**Execution**

print(new\_life("jane", 30))

# SLICING

**Sintaxis: [start: stop: step].**

EJEMPLO 1

* **Código:**

wrd="Toscana"

ans\_1=wrd[:4]

print(ans\_1)

* **Resultado:** Tosc

EJEMPLO 2

* **Código:**

ans\_1=wrd[3::]

print(ans\_1)

* **Resultado:** Cana

EJEMPLO 3

* **Código:**

ans\_1=wrd[3:6]

print(ans\_1)

* **Resultado:** can

EJEMPLO 4

* **Código:**

ans\_1=wrd[::2]

print(ans\_1)

* **Resultado:** tsaa

# OPERATORS

## Assignment Operators

**= ->**La variable va en el lado izquierdo del operador (signo igual) y el valor que se asigna va a la derecha del signo igual.

\*=           a\*=b       instead of       a=a\*b

/=           a/=b        instead of        a=a/b

%=           a%=b      instead of       a=a%b

//=          a//=b      instead of       a=a//b

\*\*=        a\*\*=b      instead of        a=a\*\*b

&=          a&=b        instead of         a=a&b

|=           a|=b         instead of         a=a|b

^=           a^=b        instead of         a=a^b

>>=        a>>=b      instead of         a=a>>b

<<=        a <<= b     instead of        a= a << b

## Arithmetic Operators

+             **Suma**: suma los valores a + b

-              **Resta**: resta valores a - b

\*             **Multiplicación**: multiplica los valores a \* b

/              **División** **float**): divide el primer valor por el segundo valor y devuelve el valor con decimales (flotante) a / b

//            **División** **floor**): igual que arriba, pero devuelve enteros sin decimales, redondea el resultado a // b

%             **Módulo**: da el resto de un proceso de división a% b

\*\*            **Potencia**: eleva el primer valor a la potencia del segundo valor a \*\* b

## LOGICAL OPERATORS

and          **AND lógico**: devuelve verdadero cuando ambos operandos son verdaderos

or            **OR lógico**: devolverá verdadero si uno de los operandos es verdadero

not          **NOT lógico**: devolverá verdadero si el (los) operando (es) es falso

## RELATIONAL OR COMPARISON

>             **Mayor que**: devuelve Verdadero si el lado izquierdo es mayor que el lado derecho x> y

<             **Menor que**: devuelve Verdadero si el lado izquierdo es menor que el lado derecho x <y

==           **Igual**: devuelve Verdadero si el lado izquierdo es igual al lado derecho x == y

!=           **No igual**: devuelve Verdadero si el lado izquierdo no es igual al lado derecho x! = Y

>=           **Mayor o igual a**: devuelve Verdadero si el lado izquierdo es mayor o igual que el lado derecho x> = y

<=           **Menor o igual a**: devuelve Verdadero si el lado izquierdo es menor o igual que el lado derecho x <= y

## MEMBERSHIP OPERATORS

in               Devolverá True si un objeto está en otro objeto

not in       Devolverá True si un objeto no está en otro objeto

## IDENTITY OPERATORS

is                    Dice si dos objetos son idénticos

is not            Dice si dos objetos no son idénticos

## BITWISE OPERATORS

&             Bitwise AND       a & b

|              Bitwise OR          a | b

~             Bitwise NOT       ~a

^             Bitwise XOR        a ^ b

>>           Bitwise RIGHT SHIFT             a>>

<<           Bitwise LEFT SHIFT                a<<

# .format()

Se usa para formatear cadenas de Python y se aplica en cadenas de Python para:

* Colocar convenientemente valores o variables en una cadena

Ejemplo:

name=input("Please enter your name.")

str="Hello!, {}".format(name)

str = "One year has {} months, {} weeks and {} days.".format(12, 52, 365)

str = "One year has {2} months, {0} weeks and {1} days.".format(52, 365, 12)

* Formateo de números.

# .join()

Une el listado con el conector que le pongas

**\n -> salto de linea**

Ejemplo:

lst=["Hawaii", "Phuket", "Aruba", "Keys"]

joined="+++".join(lst)

print(joined)

poem\_lst=["Not enjoyment, and not sorrow,", "Is our destined end or way;", "But to act, that each tomorrow", "Find us farther than today."]

poem\_str = "\n".join(poem\_lst)

print(poem\_str)

# .split()

Separa las palabras en un listado.

str="Hello World!"

lst = str.split(" ")

print(lst)

suntime = “Aruba | Keys | Miami | Myrtle | Daytona”  
print(suntime.split(“|”, maxsplit=2))

# .strip()

Borra los espacios en blanco o los caracteres que se indiquen.

str=" Hello World! "

str=str.strip()

print(str)

str="#$^&#@%$& Babylon #@$&@#"

str = str.strip("#$^&@% ")

print(str)

**rstrip -> borra los de la derecha**

**lstrip -> borra los de la izquierda**

# dir()

La función dir () le mostrará los métodos y propiedades (atributos) de un objeto.

# DATOS ANIDADOS

nested\_lst1 = [[1,2], ["Venus","Mars"], [True, False]]

print(nested\_lst1[2][0]) ->**True**

nested\_lst1 = [[1,2], ["Venus","Mars"], [True, False]]

print(\*lst1) -> **[1,2], [“Venus”,”Mars”], [True, False]**

nested\_lst1 = [[1,2], ["Venus","Mars"], [True, False]]

lst\_T = zip(\*lst1)

print(list(lst\_T)) -> **[(1, ‘Venus’, True), (2, ‘Mars’, False)]**

nested\_lst2 = [{"camera":2, "phone":1}, {"car":1, "van":0}]

print(nested\_lst2[1]["car"]) -> **1**

# IF-ELIF-ELSE

EJEMPLO:

treepersqkm = {"Finland": 90652, "Taiwan": 69593, "Japan": 49894, "Russia": 41396, "Brazil": 39542, "Canada": 36388, "Bulgaria": 24987, "France": 24436, "Greece": 24323, "United States": 23513, "Turkey": 11126, "India": 11109, "Denmark": 6129, "Syria": 534, "Saudi Arabia": 1}

def moretrees(dict):

lst = []

for i in dict:

if dict[i]>20000:

lst.append(i)

else:

pass

return lst

print(moretrees(treepersqkm))

EJEMPLO 2:

str = "Aranges and lemons, Say the bells of St. Clement's. You owe me three farthings, Say the bells of St. Martin's"

def count\_l(a):

lst=a.split()

c=0

for i in lst:

if i.upper().startswith("A")==True:

c+=1

else:

pass

return c

print(count\_l(str))

# FOR LOOP

EJEMPLO:

lst = [[1,2,3], ["land", "sea", "sky"]]

for i in lst:

for j in i:

print(j)

# WHILE LOOP

EJEMPLO:

dict\_1 = {"Jingyi": 24, "Rishaan": 21, "Cloe": 35, "Amir": 32, "Mayumi": 18, "Peter": 42}

lst = list(dict\_1.keys())

i=0

dict\_2 = {}

while i < len(lst):

if 25 < (dict\_1[lst[i]]) < 55:

dict\_2[lst[i]] = True

else:

dict\_2[lst[i]] = False

i+=1

print(dict\_2)

EJEMPLO 2:

def name\_adder(list):

i = 0

new\_list = []

while i < len(list):

if list[i] != "":

new\_list.append(list[i])

else:

break

i = i+1

return new\_list

# BREAK & CONTINUE

**break** se utiliza habitualmente para:

* Terminando el ciclo prematuramente.

**continue** se utiliza habitualmente para:

* Saltarse la iteración y continuar con el ciclo.

# LAMBDA

* Como alternativa rápida a una función normal definida por el usuario.
* Como argumento para algunas otras funciones:
  + .sort()
  + sorted()
  + filter()
  + map()
  + reduce()
  + .apply()

Ejemplo 1:

a = lambda x: x+2  
print(a(1)) -> 3

Ejemplo 2:

a = lambda x: x%5==0  
print(a(555)) -> True

Ejemplo 3:

a = lambda x, y: x\*y  
a(11, 6) -> 66

Ejemplo 4:

lst = [1, 5, 66, 7]  
lst.**sort** (key = lambda x: x)  
lst\_**sorted** = **sorted**(lst, key=lambda x: x)  
print (lst) -> [1, 5, 7, 66]  
lst\_mapped = **map**(lambda x: x+5, lst)  
print(list(lst\_mapped)) -> [6, 10, 71, 12]

lst\_filtered = **filter**(lambda x: x>5, lst)  
print(list(lst\_filtered)) -> [66, 7]

Ejemplo 5:

a = lambda x: print(“Aloha!”) if x==”Hawaii” else print(“Ciao!”)

Ejemplo 6:

lst = **sorted**(lst, key=lambda x: x[1][-1], reverse = True)

# ZIP()

Toma dos o más listas y las combina.

Ejemplo 1:

names = ["Sofia Goggia", "Mikaela Shiffrin", "Wendy Holdener", "Lindsey Vonn", "Frida Hansdotter", "Michelle Gisin", "Ragnhild Mowinckel", "Federica Brignone", "Tina Weirather", "Ester Ledecka"]

medals =[3, 10, 7, 11, 6, 2, 3, 2, 2, 6]

zipped\_lst = zip(names,medals)

dict = {i[0].lower().replace(" ", "") : i[1] for i in zipped\_lst}

print(dict) -> {‘sofiagoggia’: 3, ‘mikaelashiffrin’: 10, ‘wendyholdener’: 7, ‘lindseyvonn’: 11, ‘fridahansdotter’: 6, ‘michellegisin’: 2, ‘ragnhildmowinckel’: 3, ‘federicabrignone’: 2, ‘tinaweirather’: 2, ‘esterledecka’: 6}

Ejemplo 2:

lst1=[19542209, 4887871, 1420491, 626299, 1805832, 39865590]

lst2=["New York", "Alabama", "Hawaii", "Vermont", "West Virginia", "California"]

lst3=zip(lst1,lst2)

print(lst3)

# MAP()

Ejemplo 1:

lst\_mapped = **map**(lambda x: x+5, lst)  
print(list(lst\_mapped)) -> [6, 10, 71, 12]

Ejemplo 2:

lst1=[100, 200, 300, 400, 500]

lst2=[1,10,100,1000,10000]

lst3=map(lambda x,y:x+y,lst1,lst2)

print(lst3) -> [101, 210, 400, 1400, 10500]

Ejemplo 3:

lst=[99.3890,-3.5, 5, -0.7123, -9, -0.003]

new\_lst=map(lambda x:**abs**(x),lst)

ans\_1=sum(new\_lst)

print(ans\_1) -> 117.6043

# FILTER()

Ejemplo 1:

lst1=[12, -1, 9, 8, -0.5, -0.2, -100]

lst2=filter(lambda x:x<0,lst1)

print(lst2) -> [-1, -0.5, -0.2, -100]

Ejemplo 2:

str1="Winter Olympics in 2022 will take place in Beijing China"

lst = list(filter(lambda x: True if x.lower() in "aeiou" else False, str1))

print(lst) ->['i', 'e', 'O', 'i', 'i', 'i', 'a', 'e', 'a', 'e', 'i', 'e', 'i', 'i', 'i', 'a']

Ejemplo 3:

lst1=[-1000, 500, -600, 700, 5000, -90000, -17500]

lst2=filter(lambda x:True if x>0 else False,map(lambda x:x\*-1,lst1))

print(lst2) -> [1000, 600, 90000, 17500]

# SORTED()

Ejemplo 1:

lakes1=["Malawi", "Medicine", "Tahoe", "Moraine", "Upper Arrow", "Plitvice", "Edith", "Emerald"]

lakes2=sorted(lakes1,key=lambda x: len(x),reverse=True)

print(lakes2) -> ['Upper Arrow', 'Plitvice', 'Medicine', 'Emerald', 'Moraine', 'Malawi', 'Edith', 'Tahoe']

# List Comprehension

Ejemplo 1:

lst = [1,2,3,4,5]

lst2 = [i for i in lst]

print(lst2) -> [1,2,3,4,5]

Ejemplo 2:

lst = [1,2,3,4,5,6,7,8,9]  
lst2 = [i\*\*2 for i in lst if i<4]  
print(lst2) -> [1,4,9]

Ejemplo 3:

lst = [(12, 100), (13, 100), (14, 200), (15, 200)]  
a = [i[0] for i in lst if i[1] > 100]

* + [14,15]

Ejemplo 4:

dict={"Sedan": 1500, "SUV": 2000, "Pickup": 2500, "Minivan": 1600, "Van": 2400, "Semi": 13600, "Bicycle": 7, "Motorcycle": 110}

lst=[i.upper() for i in dict if dict.get(i) < 5000]

print(lst) -> ['SEDAN', 'SUV', 'PICKUP', 'MINIVAN', 'VAN', 'BICYCLE', 'MOTORCYCLE']

# Dict Comprehension

Ejemplo 1:

lst = [1,2]

dict = {i:i for i in lst}

print(dict) -> {1: 1, 2: 2}

Ejemplo 2:

a = {"a":10, "b":20, "c":30}

dict = {a[i]:i for i in a}

print(dict) -> {10: ‘a’, 20: ‘b’, 30: ‘c’}

Ejemplo 3:

dict = [1,2,3,4,5,6,7,8,9]

dict2 = {i:i\*\*2 for i in dict if i<4}

print(dict2) -> {1: 1, 2: 4, 3: 9}

Ejemplo 4:

lst = [[1,2,3], ["land", "sea", "sky"]]

dict = {i:j for i,j in zip(lst[0], lst[1])}

print(dict) -> {1: ‘land’, 2: ‘sea’, 3: ‘sky’}

Ejemplo 5:

lst = [[73,1995,-5], ["land", "sea", "sky"]]

dict = {i:j for i,j in enumerate(lst[1], 1)}

print(dict) -> {1: ‘land’, 2: ‘sea’, 3: ‘sky’}

Ejemplo 6:

dict1={"NFLX":4950,"TREX":2400,"FIZZ":1800, "XPO":1700}

dict2={i:dict1.get(i) for i in dict1 if dict1.get(i) > 2000}

print(dict2) -> {'NFLX': 4950, 'TREX': 2400}

# HELP()

Se puede utilizar para comprender otras funciones o tipos de datos siempre que sea necesario.

Puede buscar la construcción y la sintaxis, así como los argumentos y parámetros de otras funciones y tipos de datos.

* help(str)
* help(dict)
* help(list)
* help(tuple)
* help(keywords)

# DEBUGGING

# PASS STATEMENT

* Como marcador de posición cuando desea ejecutar su código en un estado incompleto y terminarlo más tarde en funciones, declaraciones condicionales o bucles.
* Al procesar excepciones después de la instrucción <except>.

Ejemplo 1:

if 1==1:

pass

# JSON

JSON es un formato muy conveniente que utilizan muchas estructuras de datos, servicios web y, en particular, soluciones de API REST.

Funciones:

* **json.loads(): Carga datos json en una estructura de datos de Python, como una lista o un diccionario desde una cadena.**

Ejemplo:

 data = “[{\”word\”:\”etiquette\”,\”score\”:1648,\”numSyllables\”:3},{\”word\”:\”aesthetic it\”,\”numSyllables\”:4},{\”word\”:\”arithmetic it\”,\”numSyllables\”:5},{\”word\”:\”geomagnetic cut\”,\”numSyllables\”:6},{\”word\”:\”pathetic it\”,\”numSyllables\”:4}]”

>>> import json  
>>> data = json.loads(data):  
>>> print(data):

[{'word': 'etiquette', 'score': 1648, 'numSyllables': 3},

{'word': 'aesthetic it', 'numSyllables': 4},

{'word': 'arithmetic it', 'numSyllables': 5},

{'word': 'geomagnetic cut', 'numSyllables': 6},

{'word': 'pathetic it', 'numSyllables': 4}]

* **json.dumps(): Vuelca datos json en una cadena.**

data = [{“word”:”etiquette”,”score”:1647,”numSyllables”:3},{“word”:”aesthetic it”,”numSyllables”:4},{“word”:”arithmetic it”,”numSyllables”:5},{“word”:”geomagnetic cut”,”numSyllables”:6},{“word”:”pathetic it”,”numSyllables”:4}]

>>> import json

>>> data = json.dumps(data, indent=4):

>>> print(data)

"[

{

"word": "etiquette",

"score": 1647,

"numSyllables": 3

},

{

"word": "aesthetic it",

"numSyllables": 4

},

{

"word": "arithmetic it",

"numSyllables": 5

},

{

"word": "geomagnetic cut",

"numSyllables": 6

},

{

"word": "pathetic it",

"numSyllables": 4

}

]"

# WEB REQUESTS

**requests.get(): se utilizará para realizar solicitudes HTTP**

**Sintaxis:**

**1) Recuerde importar primero el módulo de solicitudes.**

**import requests**

**2) request.get ("url") le ayudará a realizar una solicitud HTTP.**

**Ejemplos:**

**>>> import requests  
>>> f = r”https://www.doctorswithoutborders.org”  
>>> data = requests.get(f)  
>>> print(data) -> <Response [200]>**

**Metodos:**

**1) request.post(): se utiliza para enviar datos al servidor para tareas como: llenar formularios, cargar archivos, etc.**

**2) request.put(): solicita al servidor que guarde los datos actuales en una ubicación específica, como una URL predefinida.**

**3) request.head(): solicita solo la sección de encabezado de la página.**

**4) request.delete(): elimina todas las representaciones actuales de la fuente.**

**5) request.patch(): se puede usar para actualizar solo un dato parcial a diferencia del método .put () que se usa para una actualización completa.**

**6) requests.options():**

**7) request.connect() : establece un túnel al servidor.**

# TRY/EXCEPT

EJEMPLO 1:

>>> a = 5  
>>> b = 0  
>>> try:  
>>>     print(a/b)  
>>> except ZeroDivisionError:  
>>>     print(“You can’t divide with 0”)

Resultado: “You can’t divide with 0“

>>> except:  
>>>     print(“You can’t divide with 0”)

Resultado: “You can’t divide with 0“

EJEMPLO 2:

>>> a = “Hello World!”  
>>> try:  
>>>     a + 10  
>>> except BaseException as error:  
>>>     print(‘Exception occurred: {}’.format(error))

Resultado: Exception occured: can only concatenate str (not “int”) to str.