# Curso de Python

Clase 02

# ¡Bienvenidxs!

# Contenido

- Estructuras de datos
  - Listas
  - Tuplas
  - Conjuntos
  - Diccionarios
- Programación modular
  - Funciones
  - Funciones Lambda
  - Listas de comprensión
- Módulos y paquetes
  - Librería estándar de Python y uso
  - Construcción de Módulos
  - Construcción de Paquetes

# CARPETAS DE TRABAJO SUGERIDA

Antes de iniciar a realizar programas, se recomienda crear una estructura de carpetas para esta clase de la siguiente manera:

```
Curso-de-Python
    └── Clase-02
        └── mi-programa.py
```

`Curso-de-Python`
: Carpeta que contendrá todo los archivos del curso

`Clase-02`
: Carpeta que contiene los archivos sólo para la Clase 02

`mi-programa.py`
: Archivo con terminación en .py que contendrá el programa en Python a ejecutar.

# Estructuras de datos

# Listas

Las listas son secuencias mutables normalmente usadas para almacenar colecciones de datos homogéneos o cuando menos del mismo tipo, pero si fuera necesario los datos pueden ser heterogéneos también.

# LISTAS

```
In [1]: # Creando una lista vacía

In [2]: l1 = []

In [3]: l2 = list()

In [4]: # Obteniendo el tipo de dato

In [5]: type(l1)
Out[5]: list

In [6]: type(l2)
Out[6]: list

In [7]: # Creando una lista de enteros

In [8]: l3 = [1, 2, 3, 4, 5]
```

```
In [9]: # Creando una lista de floats

In [10]: l4 = [1.5, 2.6, 3.7, 4.8]

In [11]: # Creando lista de string

In [12]: l5 = [“a”, “abc”, “3 letras”]

In [13]: # Lista de tipos mixtos

In [14]: producto = [“Lápiz”, 5.50, 3]

In [15]: # Y el tipo de ‘producto’ es

In [16]: type(producto)
Out[16]: list
```

# LISTAS DE ENTEROS

**Nombre del script:** `listas-de-enteros.py`

**Descripción:**
El script deberá de crear e imprimir las siguientes listas de números enteros: lista con 10 números, con 100 números, con 10 000 números, con 100 000 000 números,  con 10 000 000 000 000 000 números.

**Ejemplo de ejecución:**

```
Clase-02 $ python listas-de-enteros.py
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 2
2, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 4
2, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 6
2, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 8
2, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 2
2, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 4
...
998, 9999]
10000
```

## ACTIVIDAD

Modifica el script `listas-de-enteros.py` para que imprima una lista de 50 000 000 (50 millones) de números flotantes distintos.

**Script a crear:**
`lista-de-flotantes.py`

Ejemplo de ejecución:

```
Clase-02 $ python lista-de-flotantes.py
```

# OPERACIONES CON LISTAS Y EL MÓDULO RANDOM

```
In [1]: # Sumando listas

In [2]: l1 = [“a”, “b”, “c”]

In [3]: l2 = [“d”, “d”]

In [4]: l1 + l2
Out[4]: ['a', 'b', 'c', 'd', 'd']

In [6]: l1 += l2  # l1 = l1 + l2

In [7]: l1
Out[7]: ['a', 'b', 'c', 'd', 'd']

In [8]: # Convertir lista a cadena

In [9]: “”.join(l1)
Out[9]: 'abcdd'
```

```
In [10]: # Usando el módulo random
In [11]: import random

In [12]: # Elige al azar un elemento de
una secuencia, como una cadena o lista
In [13]: random.choice(“abc123”)
Out[13]: '1'

In [14]: # Desordena elementos de una
lista
In [15]: l = list(“abc12”)
In [16]: l
Out[16]: ['a', 'b', 'c', '1', '2']
In [17]: random.shuffle(l)

In [16]: l
Out[16]: ['c', 'a', '1', 'b', '2']
```

# LISTAS DE CADENAS

**Nombre del script:** `genera-claves.py`

**Descripción:**
El script deberá de crear e imprimir una lista de  n claves de longitud m incluyendo cuando menos una minúscula, una mayúscula y un dígito. Los valores de n y m serán solicitados al usuario y m deberá tener el valor de 8 en caso de que el usuario no proporcione ninguno.

**Ejemplo de ejecución:**

```
Clase-02 $ python genera-claves.py
Número de claves a generar: 5
Longitud de claves (8):

wpbWmE0M
jb7Sd6GU
R4V2uxsr
X6pC47Jf
SrVb1iDk

Clase-02 $
```

## ACTIVIDAD

Modifica el script `genera-claves.py` para que incluya cuando menos un símbolo ($%&/#@-_=+-*) como parte de la clave.

**Script a crear:**
`genera-claves.py`

Ejemplo de ejecución:

```
Clase-02 $ python genera-claves.py

Número de claves a generar: 4
Longitud de claves (8): 10

ngTApN0yv$
yspxsCj3+W
OhzBG57N*/
*d*@9OtN25

Clase-02 $
```

# PRODUCTOS CON LISTAS

**Nombre del script:** `productos-con-listas.py`

**Descripción:**
Copiar el script `lista-de-productos.py` de la clase anterior con el nombre solicitado y modificarlo para que cada producto se almacene en una variable de tipo lista.

**Ejemplo de ejecución:**

```
Clase-02 $ python productos-con-listas.py
-------------------------------------------------------------
PRODUCTO                                          | PRECIO
-------------------------------------------------------------
Automóvil                                         |  150000.00
Bicicleta                                         |   13000.00
Chamarra                                          |    3999.99
Laptop Thinkpad                                   |   21750.00
Gafas de realidad virtual Lenovo con sable laser  |    5000.00
-------------------------------------------------------------
Clase-02 $
```

# LISTAS DE LISTAS

```
In [1]: # Lista con elementos listas

In [2]: l1 = [[1, 2], [3, 4],
              [“a”, “b”]]

In [3]: # Obteniendo elementos de l1

In [4]: l1[0]
Out[4]: [1, 2]

In [5]: l1[2]
Out[5]: [‘a’, ‘b’]

In [6]: l1[2][0]
Out[6]: ‘a’
```

```
In [7]: # Imprimiendo listas de listas

In [8]: for lista in l1:
   ...:     print(lista)
   ...:
[1, 2]
[3, 4]
['a', 'b']

In [9]: for lista in l1:
   ...:     for valor in lista:
   ...:         print(valor, end=" ")
   ...:     print()
   ...:
1 2
3 4
a b
```

# ADICIONAR PRODUCTOS

**Nombre del script:** `adicionar-productos.py`

**Descripción:**
Usar el script anterior y modificarlo para que el usuario pueda agregar nuevos productos y que además la lista de productos se imprima en orden alfabético. El script termina cuando el usuario introduce un script vacío.

**Ejemplo de ejecución:**

```
Clase-02 $ python adicionar-productos.py
-----------------------------------------------------------------
PRODUCTO                                          | PRECIO
-----------------------------------------------------------------
Automóvil                                         |  150000.00
Bicicleta                                         |   13000.00
-----------------------------------------------------------------

Capturar nuevo producto (nombre, precio): Motocicleta, 20000.00

-----------------------------------------------------------------
PRODUCTO                                          | PRECIO
-----------------------------------------------------------------
Automóvil                                         |  150000.00
Bicicleta                                         |   13000.00
Motocicleta                                       |   20000.00
-----------------------------------------------------------------
```

## ACTIVIDAD

Modifica el script `adicionar-productos.py` para que imprima el precio total después de haber terminado de adicionar los nuevos productos.

**Script a crear:**
`adicionar-productos-total.py`

Ejemplo de ejecución:

```
Clase-02 $ python adicionar-productos-total.py
-----------------------------------------------------------------
PRODUCTO                                                | PRECIO
-----------------------------------------------------------------
Automóvil                                               |  150000.00
Bicicleta                                               |   13000.00
Chamarra                                                |    3999.99
-----------------------------------------------------------------

Capturar nuevo producto (nombre, precio): Moto,20000
-----------------------------------------------------------------
PRODUCTO                                                | PRECIO
-----------------------------------------------------------------
Automóvil                                               |  150000.00
Bicicleta                                               |   13000.00
Chamarra                                                |    3999.99
Moto                                                    |   20000.00
-----------------------------------------------------------------

Capturar nuevo producto (nombre, precio):
-----------------------------------------------------------------
                El costo total de los productos es: |  186999.99
-----------------------------------------------------------------
```

# Tuplas

Las tuplas son secuencias de objetos inmutables que normalmente se utilizan para almacenar datos heterogéneos y es ampliamente usado para pasar datos hacia y desde funciones o regresar datos de funciones o para crear variables cuyos datos no cambien en el tiempo de vida de la variable.

# TUPLAS

```
In [1]: # Creando una tupla vacía

In [2]: t1 = ()

In [3]: t2 = tuple()

In [4]: # Obteniendo el tipo de dato

In [5]: type(11)
Out[5]: tuple

In [6]: # Tupla de un elemento

In [7]: t3 = (1,)

In [8]: # Asignación múltiple

In [9]: (a, b) = (10, 20)
```

```
In [10]: a, b = 10, 20  # Es lo mismo
In [11]: a
Out[11]: 10
In [12]: # Intercambiando valores
In [13]: a, b = b, a
In [14]: a
Out[14]: 20
In [15]: # Modificando una tupla
In [16]: t = (10, 20)
In [16]: t[0] = 30
---------------------------------------
TypeError
<ipython-input-4-027a7f842b02>
----> 1 t[0] = 30

TypeError: 'tuple' object does not
support item assignment
```

# ORDENANDO LISTA DE PRODUCTOS

**Nombre del script:** `productos-ordenados.py`

**Descripción:**
Usar script `productos-con-listas-de-listas.py` como base e imprime la lista de productos ordenadas en base al precio.

**Ejemplo de ejecución:**

```
Clase-02 $ python productos-ordenados.py
-------------------------------------------------------------
PRODUCTO                                          | PRECIO
-------------------------------------------------------------
Chamarra                                          |    3999.99
Gafas de realidad virtual Lenovo con sable laser  |    5000.00
Bicicleta                                         |   13000.00
Laptop Thinkpad                                   |   21750.00
Automóvil                                         |  150000.00
-------------------------------------------------------------
Clase-02 $
```

# Conjuntos

Los conjuntos son colecciones mutables no ordenada de objetos únicos, usados principalmente en operaciones de lógica y matemáticas, pero se puede aplicar no sólo a números a cadenas o tuplas por ejemplo.

# CONJUNTOS

```
In [1]: # Creando una conjunto vacío

In [2]: s1 = set()

In [3]: # Obteniendo el tipo de dato

In [4]: type(s1)
Out[4]: set

In [5]: # Conjunto de varios elementos

In [6]: s2 = {1, 2, “tres”, (4, 5)}

In [7]: # Accediendo a elementos

In [8]: s2[0]
TypeError: 'set' object does not
support indexing
```

```
In [9]: # De listas a conjuntos
In [10]: s3 = set([1, 2, 2, 3])
In [11]: s3
Out[11]: {1, 2, 3}
In [12]: # Agregando elementos
In [13]: s3.add(3)
In [14]: s3
Out[14]: {1, 2, 3}
In [15]: s3.add(0)
In [16]: s3
Out[16]: {0, 1, 2, 3}
In [12]: # Intersección
In [13]: s3.intersection({1,2,4,5})
Out[16]: {1, 2}
```

# ELIMINANDO PRODUCTOS DUPLICADOS

**Nombre del script:** `productos-unicos.py`

**Descripción:**
Usar script `productos-con-listas-de-listas.py` agregar uno o dos registro duplicados y luego hacer uso de conjuntos para eliminarlos.

**Ejemplo de ejecución:**

```
Clase-02 $ python productos-unicos.py
-------------------------------------------------------------
PRODUCTO                                          | PRECIO
-------------------------------------------------------------
Chamarra                                          |    3999.99
Gafas de realidad virtual Lenovo con sable laser  |    5000.00
Bicicleta                                         |   13000.00
Laptop Thinkpad                                   |   21750.00
Automóvil                                         |  150000.00
-------------------------------------------------------------
Clase-02 $
```

# Diccionarios

Los diccionarios son objetos contenedores que utilizan una llave (key) para la búsqueda arbitraria de sus elementos (values) y por tal razón se necesitan de la pareja (llave, valor) para poder adicionar elementos al diccionario.

# DICCIONARIOS

```
In [1]: # Creando un diccionario vacío

In [2]: d1 = {}

In [3]: d2 = dict()

In [4]: # Obteniendo el tipo de dato

In [5]: type(d1)
Out[5]: dict

In [6]: type(d2)
Out[6]: dict

In [7]: # Diccionario de frecuencias

In [8]: d3 = {“Durango”: 5, “Puebla”:
7, “Queretaro”: 15, “Colima”: 3}
```

```
In [9]: # Diccionarios tipos mixtos

In [10]: producto = {“Nombre”:“Lápiz”,
“Precio”:5.50, “Descuento”:3]

In [11]: # Imprimiendo llaves

In [12]: producto.keys()
Out[12]: dict_keys([“Nombre”, “Precio”,
“Descuento”])

In [13]: # Imprimiendo items

In [14]: producto.items()
Out[14]:
dict_items([(“Nombre”,”Lápiz”),
(“Precio”, 5.50), (“Descuento”, 3)])
```

# DICCIONARIOS GRAFICAR REPETICIONES

**Nombre del script:** `diccionarios-de-enteros-repeticiones.py`

**Descripción:**
El script deberá generar una lista de 10 000 número enteros aleatorios, cada número puede estar en el rando de 0 a 99. Posteriormente encontrar el número de veces que se repite cada número e imprimir la lista de números ordenada crecientemente y las veces que se repite cada uno y de forma gráfica.

**Ejemplo de ejecución:**

```
Clase-02 $ python diccionarios-de-enteros-repeticiones.py
0 ---------------------------------------------------------- 103
1 ------------------------------------------------------ 97
2 ------------------------------------------------------------ 109
3 ------------------------------------------------------------- 110
4 ---------------------------------------------------------- 107
…
97 ------------------------------------------------------------ 114
98 ------------------------------------------------- 99
99 ------------------------------------------------- 97
```

# PRODUCTOS CON DICCIONARIOS

**Nombre del script:** `productos-con-diccionarios.py`

**Descripción:**
Usar el script `productos-con-listas.py` como base y modificarlo para usar diccionarios en lugar de listas para cada variable de un producto.

**Ejemplo de ejecución:**

```
Clase-02 $ python productos-con-diccionarios.py
-------------------------------------------------------------------------
NOMBRE                                                | PRECIO     | DESCUENTO
-------------------------------------------------------------------------
Automóvil                                             |  150000.00 |       0.00
Bicicleta                                             |   13000.00 |       0.00
Chamarra                                              |    3999.99 |       0.00
Laptop Thinkpad                                       |   25000.00 |      13.00
Gafas de realidad virtual Lenovo con sable laser      |    5000.00 |       0.00
-------------------------------------------------------------------------
```

# LISTAS DE DICCIONARIOS

```
In [1]: # Lista con un diccionario
In [2]: l1 = [
    {“Nombre”:”Lapiz”, “Precio”:5.50}
]

In [3]: # Agregando otro diccionario

In [4]: l1.append(
    {“Nombre”:”Goma”, “Precio”:6.50}
)

In [5]: type(l1)
Out[5]: list

In [6]: type(l1[0])
Out[6]: dict
```

```
In [9]: # Imprimiendo lista de
        # nombres de productos

In [10]: for p in l1:
    ...:     print(p["Nombre"])
    ...:
Lápiz
Goma
```

# PRODUCTOS CON LISTAS DE DICCIONARIOS

**Nombre del script:** `productos-con-lista-de-diccionarios.py`

**Descripción:**
Usar el script `productos-con-diccionarios.py` como base y modificarlo para usar una lista de diccionarios como variable donde se almacenarán todos los productos.

**Ejemplo de ejecución:**

```
Clase-02 $ python productos-con-lista-de-diccionarios.py
-------------------------------------------------------------------------
NOMBRE                                                | PRECIO     | DESCUENTO
-------------------------------------------------------------------------
Automóvil                                             |  150000.00 |       0.00
Bicicleta                                             |   13000.00 |       0.00
Chamarra                                              |    3999.99 |       0.00
Laptop Thinkpad                                       |   25000.00 |      13.00
Gafas de realidad virtual Lenovo con sable laser      |    5000.00 |       0.00
-------------------------------------------------------------------------
```

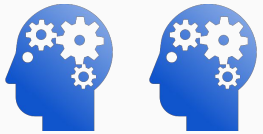
DICCIONARIOS, CICLOS Y SUBTOTALES

## ACTIVIDAD

Modifica el script `productos-con-listas-de-diccionarios.py` para que imprima una columna adicional con título Subtotal y los valores se obtienen al aplicar el descuento al precio del producto

**Script a crear:**
`productos-subtotales.py`

**Sugerencia:**
El valor del descuento está expresado en porciento, así que toca convertirlo a decimal para poder usarlo.



Ejemplo de ejecución:

```
Clase-02 $ python productos-subtotales.py

-------------------------------------------------------------------------
NOMBRE                                    | PRECIO     | DESCUENTO  | SUBTOTAL
-------------------------------------------------------------------------
Automóvil                                 |  150000.00 |       0.00 |  150000.00
Bicicleta                                 |   13000.00 |       0.00 |   13000.00
Chamarra                                  |    3999.99 |       0.00 |    3999.99
Laptop Thinkpad                           |   25000.00 |      13.00 |   28250.00
Gafas de realidad virtual Lenovo con ...  |    5000.00 |       0.00 |    5000.00
-------------------------------------------------------------------------
```

## ACTIVIDAD

Modifica el script `productos-subtotales.py` para que imprima una línea adicional en la tabla con el total que corresponde a la suma de todos los subtotales.

**Script a crear:**
`productos-totales.py`

Ejemplo de ejecución:

```
Clase-02 $ python productos-totales.py

-------------------------------------------------------------------------------
NOMBRE                                    | PRECIO     | DESCUENTO  | SUBTOTAL
-------------------------------------------------------------------------------
Automóvil                                 |  150000.00 |       0.00 |  150000.00
Bicicleta                                 |   13000.00 |       0.00 |   13000.00
Chamarra                                  |    3999.99 |       0.00 |    3999.99
Laptop Thinkpad                           |   25000.00 |      13.00 |   28250.00
Gafas de realidad virtual Lenovo con ...  |    5000.00 |       0.00 |    5000.00
-------------------------------------------------------------------------------
                                                              Total: |  200249.99
-------------------------------------------------------------------------------
```

# PRODUCTOS CON DICCIONARIOS A PARTIR DE LISTAS

**Nombre del script:** `listas-a-diccionarios.py`

**Descripción:**
Usar el script `productos-con-listas-de-listas.py` como base y agregar código para convertir la lista de listas a una lista de diccionarios usando ciclos

**Ejemplo de ejecución:**

```
Clase-02 $ python listas-a-diccionarios.py
-------------------------------------------------------------------------
NOMBRE                                                | PRECIO     | DESCUENTO
-------------------------------------------------------------------------
Automóvil                                             |  150000.00 |       0.00
Bicicleta                                             |   13000.00 |       0.00
Chamarra                                              |    3999.99 |       0.00
Laptop Thinkpad                                       |   25000.00 |      13.00
Gafas de realidad virtual Lenovo con sable laser      |    5000.00 |       0.00
-------------------------------------------------------------------------
```

# Programación Modular

# Funciones

Las funciones son piezas de código delimitadas y a las que se les puede asignar un nombre con el que pueden ser invocadas, con el objetivo de modularizar (segmentar de lo complejo a lo simple) o de reutilizar (código que se usa repetidamente).

# FUNCIONES

```
In [1]: # Definición de una función
In [2]: def nombre():
   ...:     print("Soy nombre!")
   ...:

In [3]: # Uso

In [4]: nombre()
Soy nombre!

In [4]: # Con parámetros
In [5]:  def saludos(nombre):
   ...:     print("Hola {}!"
   ...:         .format(nombre))
   ...:
In [6]: saludos(“Neo”)
Hola Neo!
```

```
In [7]: # Regresando valores

In [8]: def lee_coordenada():
    ...:     resp = input("x,y,x = ")
    ...:     return resp.split(",")
    ...:

In [9]: x, y, z = lee_coordenada()
x,y,x = 3,5,7

In [10]: x
Out[12]: ‘3’

In [10]: y
Out[12]: ‘5’

In [10]: z
Out[12]: ‘7’
```

# LISTAS DE ENTEROS CON FUNCIONES

**Nombre del script:** `listas-de-enteros-funciones.py`

**Descripción:**
El script deberá de crear e imprimir la lista de números enteros  con N elementos, el valor de N será proporcionado por el usuario, por lo que se sugiere hacer uso de dos funciones.

**Ejemplo de ejecución:**

```
Clase-02 $ python listas-de-enteros-funciones.py
Cantidad de números a generar ? 10
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

Clase-02 $ python listas-de-enteros-funciones.py
Cantidad de números a generar ? 1000
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, …, 999]

Clase-02 $ python listas-de-enteros-funciones.py
Cantidad de números a generar ? 100000000
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, …, 99999999]
```

## ACTIVIDAD

Modifica el script `listas-de-enteros-funciones.py` para al crear la lista con 50 millones de número o más muestre una advertencia de que el script podría bloquear el sistema operativo y pedir confirmación al usuario de si desea o no continuar.

**Script a crear:** `lista-de-enteros-segura.py`

Ejemplo de ejecución:

```
Clase-02 $ python lista-de-enteros-segura.py

Cantidad de números enteros a generar? 50000000

El número proporcionado podría bloquear su sistema!
Desea continuar (s/n)? n

La creación de la lista ha sido cancelada por el usuario!

Clase-02 $
```

# GENERA CLAVES REHUSANDO CÓDIGO CON FUNCIONES

**Nombre del script:** `generando-claves-funciones.py`

**Descripción:**
El script deberá de crear e imprimir la lista de números enteros  con N elementos, el valor de N será proporcionado por el usuario, por lo que se sugiere hacer uso de dos funciones.

**Ejemplo de ejecución:**

```
Clase-02 $ python generando-claves-funciones.py
Número de claves a generar: a

Error: lo que has escrito no es un entero, intenta de nuevo!

Número de claves a generar: 5
Longitud de claves (8):
ZaagIKT0
22D5g9pM
9cuSjbXj
WnmdIrk6
fZc9j4zM
```

# Funciones Lambda

Las lambdas son funciones anónimas que suelen ser usadas cuando se necesita una función una sola vez. Normalmente se crea funciones lambda con el único propósito de pasarlas a funciones de orden superior.

# FUNCIONES LAMBDA

```
In [1]: # Definición de una lambda
In [2]: nombre = lambda: print("Soy una lambda!")
In [3]: nombre()
Soy una lambda!

In [4]: # Con parámetros
In [5]: nombre = lambda nom:print("Sal de la matrix {}!".format(nom))

In [6]: nombre("Platon")
Sal de la matrix Platon!

In [7]: # Como parámetro de otra función
In [8]: nombre = lambda nom: "Sal de la matrix {}!".format(nom)
In [9]: print(nombre(“Morpheus”))
Sal de la matrix Morpheus!
```

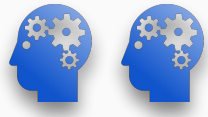
# TABLA DEL 3 CON FUNCIONES LAMBDA

**Nombre del script:** `tabla-del-3-lambda.py`

**Descripción:**
Modificar el `script tabla-del-3.py` creado en la clase 1 y usar una función lambda para simplificar y optimizar el código.

**Ejemplo de ejecución:**

```
Clase-02 $ python tabla-del-3-lambda.py
TABLA DEL 3
-------------
3 x  1 =  3
3 x  2 =  6
3 x  3 =  9
3 x  4 = 12
3 x  5 = 15
3 x  6 = 18
3 x  7 = 21
3 x  8 = 24
3 x  9 = 27
3 x 10 = 30
------------
```



**ACTIVIDAD**

Imprimir la tabla del n en la pantalla usando funciones lambda para leer n y para crear cada renglón de la tabla. Si es necesario se puede hacer uso de funciones convencionales.

**Script a crear:**
`tabla-del-n-lambda.py`

**Basado en el script:**
`tabla-del-3-lambda.py`

Ejemplo de ejecución:

```
Clase-02 $ python tabla-del-n-lambda.py

Escribe el número de la tabla a imprimir n= 11
TABLA DEL 11
--------------
11 x  1 =  11
11 x  2 =  22
11 x  3 =  33
11 x  4 =  44
11 x  5 =  55
11 x  6 =  66
11 x  7 =  77
11 x  8 =  88
11 x  9 =  99
11 x 10 = 110
--------------
```

# Listas de comprensión

Las listas de comprensión, del inglés *list comprehensions*, es una capacidad de las listas que permite crear nuevas listas en una misma línea de código.

# LISTAS DE COMPRENSIÓN

```
In [1]: # Lista de 10 enteros forma tradicional
In [2]: lista = []
In [3]: for i in range(10):
  .....     lista.append(i)
In [4]: lista
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [5]: # Lista de 10 enteros con listas de comprensión
In [6]: lista = [i for i in range(10)]
In [7]: lista
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

In [8]: # Lista de los renglones de la tabla del 3
In [9]: tabla_del_3 = ["3 x {0:2} = {0:2}".format(i) for i in range(1, 11)]
In [10]: tabla_del_3
Out[10]:
['3 x  1 =  1',
 '3 x  2 =  2',
 '3 x  3 =  3',
 '3 x  4 =  4',
 '3 x  5 =  5',
 '3 x  6 =  6',
 '3 x  7 =  7',
 '3 x  8 =  8',
 '3 x  9 =  9',
 '3 x 10 = 10']
```

# TABLA DEL 3 CON FUNCIONES LAMBDA Y LISTAS DE COMPRENSIÓN

**Nombre del script:** `tabla-del-3-comprension.py`

**Descripción:**
Modificar el script `tabla-del-3-lambda.py` haciendo uso de exclusivamente de funciones lambda y listas de compresión.

**Ejemplo de ejecución:**

```
Clase-02 $ python tabla-del-3-compresion.py
TABLA DEL 3
-------------
3 x  1 =  3
3 x  2 =  6
3 x  3 =  9
3 x  4 = 12
3 x  5 = 15
3 x  6 = 18
3 x  7 = 21
3 x  8 = 24
3 x  9 = 27
3 x 10 = 30
------------
```

# TABLA DEL 3 CON FUNCIONES LAMBDA

**Nombre del script:** `tabla-del-3-lambda.py`

**Descripción:**
Modificar el `script tabla-del-3.py` creado en la clase 1 y usar una función lambda para simplificar y optimizar el código.

**Ejemplo de ejecución:**

```
Clase-02 $ python tabla-del-3-lambda.py
TABLA DEL 3
-------------
3 x  1 =  3
3 x  2 =  6
3 x  3 =  9
3 x  4 = 12
3 x  5 = 15
3 x  6 = 18
3 x  7 = 21
3 x  8 = 24
3 x  9 = 27
3 x 10 = 30
------------
```

# ADICIONA PRODUCTOS CON PROGRAMACIÓN MODULAR

**Nombre del script:** `adicionar-productos-modular.py`

**Descripción:**
El script deberá:

1. Imprimir la lista de productos actual
2. Preguntar si se desea agregar un producto
3. SI, leer nombre y precio del nuevo producto
4. Agregar el nuevo producto a la lista de productos
5. Ordenar lista de productos
6. Regresar al paso 1.
7. NO, entonces terminamos

**Sugerencias:**

- Crear una función por cada punto del 1 al 5
- Usar siempre funciones lambda antes de funciones siempre que sea posible
- Usar listas de comprensión antes que ciclos convencionales siempre que sea posible
- Si un código se repite 2 o más veces, entonces posiblemente deba ser una función

**Basado en el script:** `adicionar-productos.py`

# ADICIONA PRODUCTOS CON PROGRAMACIÓN MODULAR

**Ejemplo de ejecución:**

```
Clase-02 $ python adicionar-productos-modular.py
-----------------------------------------------------------------
PRODUCTO                                                | PRECIO
-----------------------------------------------------------------
Automóvil                                               |   150000.0
Bicicleta                                               |    13000.0
-----------------------------------------------------------------
Desea agregar un nuevo producto (Si/No)? s
Escribe nuevo producto (nombre, precio): Tux, 0

-----------------------------------------------------------------
PRODUCTO                                                | PRECIO
-----------------------------------------------------------------
Automóvil                                               |   150000.0
Bicicleta                                               |    13000.0
Tux                                                     |        0.0
-----------------------------------------------------------------
Desea agregar un nuevo producto (Si/No)? n
```

# Módulos y Paquetes

# Librería estándar de Python

https://docs.python.org/3/library/index.html

Ofrece un amplio rango de servicios como se puede ver en la referencia y en general ésta librería de módulos (muchos de ellos escritos en C otros en Python) está incluída con cualquier instalación de Python y provee desde interacción con el sistema operativo hasta soluciones estandarizadas a muchos problemas que ocurren en el día a día de la programación.

¡Siéntete libre de usarla!

# EL MÓDULO OS

https://docs.python.org/3/library/os.html

```
In [1]: # Importando o usando o incluyendo el módulo
In [2]: import os
In [3]: # Obteniendo ayuda del módulo
In [4]: os?
File:        ~/miniconda3/lib/python3.7/os.py
Docstring:
OS routines for NT or Posix depending on what system we're on.

This exports:
  - all functions from posix or nt, e.g. unlink, stat, etc.
  - os.path is either posixpath or ntpath
...
In [5]: # Obteniendo el directio de trabajo actual
Out[5]: '/home/user/Curso-de-Python'
In [6]: # Obteniendo la lista de archivo del directorio actual
In [7]: archivos = os.listdir()
In [8]: archivos
Out[8]: ['Clase-02', 'Clase-01', 'README.md']

In [9]: # Obteniendo el tamaño en bytes de un archivo
In [10]: tamanio = os.path.getsize("README.md")
In [11]: tamanio
Out[11]: 465
```

# LISTA DE ARCHIVOS DEL DIRECTORIO ACTUAL - MÓDULOS

**Nombre del script:** `lista-archivos.py`

**Descripción:**
El script deberá de imprimir en la salida estándar la lista de archivos del directorio actual ordenados en base al tamaño en bytes.

**Ejemplo de ejecución:**

```
Clase-02 $ python lista-archivos.py
--------------------------------------------------
README.md                                      225
tabla-del-3-lambda.py                          419
tabla-del-3-comprension.py                     490
productos-con-listas-de-listas.py             1085
...
listas-de-enteros.py                          2144
genera-claves.py                              2560
genera-claves-funciones.py                    3182
Soluciones                                    4096
--------------------------------------------------
```

## ACTIVIDAD

Agrega el código necesario para que en el script anterior imprima el final de la tabla el total de bytes haciendo uso de funciones lambda y la ruta completa del directorio actual en la parte superior de la tabla

**Script a crear:**
`lista-archivos-total.py`

**Basado en el script:**
`lista-archivos.py`

Ejemplo de ejecución:

```
Clase-02 $ python lista-archivos-total.py

/home/user/Curso-de-Python/Clase-02
-------------------------------------------------
tabla-del-n-lambda.py                          770
listas-de-flotantes.py                         985
lista-archivos.py                             1449
adicionar-productos-total.py                  1806
productos-subtotales.py                       1840
productos-totales.py                          1888
adicionar-productos-modular.py                2559
genera-claves.py                              2880
listas-de-enteros-segura.py                   2926
-------------------------------------------------
                                             17103


Clase-02 $
```

# EL MÓDULO HTTP.SERVER

https://docs.python.org/3/library/http.server.html#module-http.server

```
In [1]: # Importando o usando o incluyendo el módulo
In [2]: import http
In [3]: # Obteniendo ayuda del módulo
In [4]: http.server?
HTTP server classes.

Note: BaseHTTPRequestHandler doesn't implement any HTTP request; see
SimpleHTTPRequestHandler for simple implementations of GET, HEAD and POST,
and CGIHTTPRequestHandler for CGI scripts.
...
In [5]: # Obteniendo ayuda de la clase SimpleHTTPRequestHandler
In [6]: http.server.SimpleHTTPRequestHandler?
Init signature: http.server.SimpleHTTPRequestHandler(*args, directory=None, **kwargs)
Docstring:
Simple HTTP request handler with GET and HEAD commands.
...
In [6]: exit()

Clase-02 $ # Iniciando un mini servidor web, después de ejecutar el comando abrir el
          # navegador en la dirección http://0.0.0.0:8000/
Clase-02 $ python -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
```

# CREANDO UN MINI SERVIDOR WEB

**Nombre del script:** `mi-servidor-web.py`

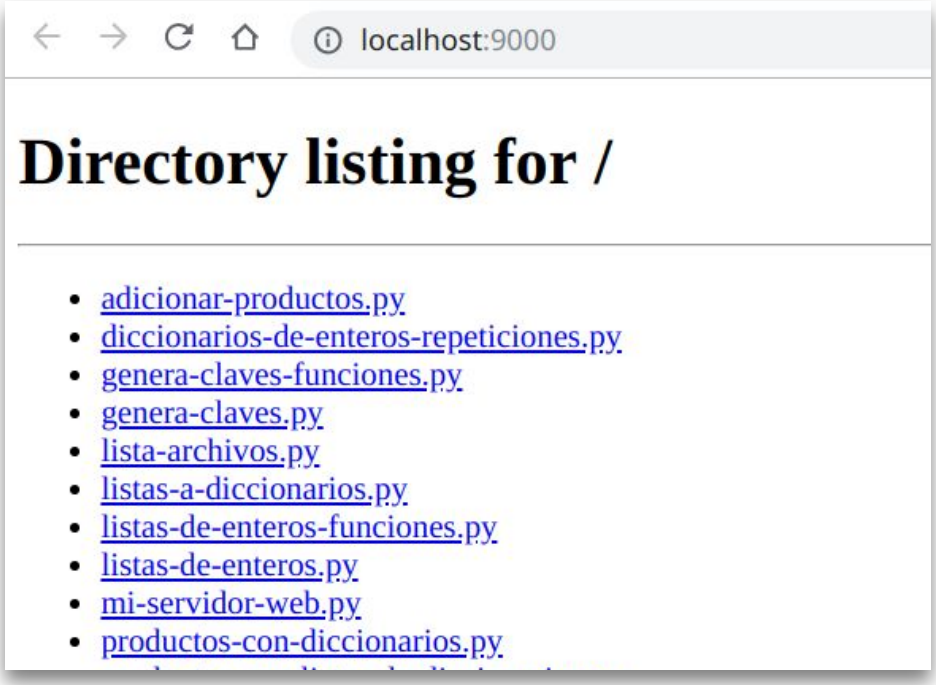
**Descripción:**
Este script permite mostrar como es posible crear un mini servidor web usando unas pocas líneas en Python. Después de ejecutar el script agregar un archivo index.html en la carpeta actual con algo de html y actualizar el navegador para ver los resultados. Usar el puerto 9000.

**Ejemplo de ejecución:**

```
Clase-02 $ python mi-servidor-web.py
Nuestro servidor está corriendo en http://localhost:9000
127.0.0.1 - - [29/Nov/2018 09:11:41] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [29/Nov/2018 09:11:41] code 404, message File not found
127.0.0.1 - - [29/Nov/2018 09:11:41] "GET /favicon.ico HTTP/1.1" 404 -
...
```

# CREANDO UN MINI SERVIDOR WEB

**Ejemplo de ejecución:**




# Creando módulos en Python

Los módulos en Python, son archivos con extensión .py, así que prácticamente cualquier script en Python se puede usar como módulo, sin embargo la idea de usar módulos está con agrupar pedazos de código para poder usarlo posteriormente.

En un módulo se puede definir funciones, clases y variables, también puede incluir código que se ejecuta inmediatamente.

# CREANDO MÓDULOS QUE EJECUTAN CÓDIGO

**Nombre del script:** `mi_primer_modulo.py`

**Descripción:**
Este script debe escribir en la salida estándar el mensaje “Soy un módulo sin control!”

**Ejemplo de ejecución:**

```
Clase-02 $ python mi_primer_modulo.py
Soy un módulo sin control!

Clase-02 $ ipython

In [1]: import mi_primer_modulo
Soy un módulo sin control!

In [2]: # Después de agregar un segundo mensaje al script
In [3]: exit()

Clase-02 $ ipython
In [1]: import mi_primer_modulo
Soy un módulo sin control!
Sigo haciendo lo que quiera!
```

# CREANDO MÓDULOS QUE NO SON SCRIPTS

**Nombre del script:** `tablas.py`

**Descripción:**
Este script debe contener una función llamada `tabla_del_3()` y lo que hace es imprimir la tabla de multiplicar del 3, sin embargo sólo deberá estar la definición de la función y nunca deberá ser llamada la función.

**Ejemplo de ejecución:**

```
Clase-02 $ python tablas.py
Clase-02 $ ipython
In [1]: # Se importa el módulo y ahora no pasa nada (aparentemente)
In [2]: import tablas
In [3]: tablas.tabla_del_3?
Signature: tablas.tabla_del_3()
Docstring: Imprime la tabla del 3
File:      ~/repos/bedu/Curso-de-Python/Clase-02/tablas.py
Type:      function

In [3]: # Ahora a ejecutar la función tabla_del_3()
In [4]: tablas.tabla_del_3()
TABLA DEL 3
-------------
3 x  1 =  3
...
3 x  9 = 27
3 x 10 = 30
-------------
```

# CREANDO MÓDULOS QUE SON SCRIPT Y MÓDULOS A LA VEZ

**Nombre del script:** `archivos.py`

**Descripción:**
Modificar el script `lista-archivos.py` para que se pueda usar como script y como módulo.

**Ejemplo de ejecución:**

```
Clase-02 $ python archivos.py
--------------------------------------------------
README.md                                     225
mi_primer_modulo.py                           228
...
Soluciones                                   4096
__pycache__                                  4096
--------------------------------------------------
Clase-02 $ ipython
In [1]: # Se importa el módulo
In [2]: import archivos
In [3]: archivo?
...
Docstring:
Documentación del módulo archivos

Este es un ejemplo de documentación para un módulo y describe de forma
general su contenido, finalidad o incluso hasta ejemplos de uso.
```

# CREANDO MÓDULOS QUE SON SCRIPT Y MÓDULOS A LA VEZ

**Ejemplo de ejecución:**

```
In [4]: archivos.linea(40)
----------------------------------------

In [5]: lista = archivos.obtiene_archivos()  # lista de archivos del directorio actual
In [3]: len(lista)
Out[9]: 23
In [4]: archivos.imprime(lista)
README.md                                    225
mi_primer_modulo.py                          228
listas-de-enteros-funciones.py              1734
listas-de-enteros.py                        2144
...
genera-claves.py                            2560
genera-claves-funciones.py                  3182
Soluciones                                  4096
__pycache__                                 4096
```

# Creando paquetes en Python

Los paquetes en Python, son directorios que pueden contener módulos o otros paquetes, la única condición es que deben contener un archivo llamado `__init__.py` y puede estar vacio.

# EL HOLA MUNDO DE LOS PAQUETES

**Nombre del paquete:** `utilerias`

**Descripción:**
Crear un paquete llamado `utilerias` dentro de la carpeta de la Clase-02, importarlo desde IPython y mostrar la ayuda.

**Ejemplo de ejecución:**

```
In [1]: import utilerias
In [2]: utilerias?
Type:        module
String form: <module 'utilerias' from
'/home/rctorr/repos/bedu/Curso-de-Python/Clase-02/utilerias/__init__.py'>
File:        ~/repos/bedu/Curso-de-Python/Clase-02/utilerias/__init__.py
Docstring:   Hola mundo desde los paquetes!

In [3]: utilerias.<presionamos tab>
```

# PAQUETES Y MÓDULOS

**Nombre del paquete y módulo:** `utilerias.archivos`

**Descripción:**
Usar el paquete utilerías creado en el ejercicio anterior y agregar el módulo archivos creado anteriormente.

**Ejemplo de ejecución:**

```
utilerias/
├── archivos.py
├── __init__.py
└── __pycache__
    └── __init__.cpython-37.pyc

In [1]: import utilerias
In [2]: utilerias?
...
PAQUETE utilerias
    Contiene módulos que resuelven pequeñas tareas

MÓDULOS

archivos
El módulo archivos contiene funciones que permiten realizar operaciones
con el sistema de archivos del sistema operativo.
```

# PAQUETES Y MÓDULOS

**Ejemplo de ejecución:**

```
In [4]: # Importando un módulo de un paquete
In [5]: from utilerias import archivos
In [6]: archivos?
…
Docstring:
MÓDULO: archivos

Contiene funciones que permiten realizar operaciones con el sistema de
archivos del sistema operativo.

In [4]: archivos.obtiene_archivos()
Out[8]:
[('README.md', 225),
 ('mi_primer_modulo.py', 228),
 ('tabla-del-3-lambda.py', 419),
 ('tabla-del-3-comprension.py', 490),
 ...
 ('utilerias', 4096),
 ('__pycache__', 4096)]
```

SCRIPT FINAL

## ACTIVIDAD FINAL

**Lista de scripts a crear:**

```
notas-de-venta.py
notas/
|- __init__.py
|- entrada.py
|- salida.py
```

**Descripción:**
Modificar el script nota-de-venta.py para que las funciones sean realizadas por los scripts solicitados.

Ejemplo de ejecución:

```
Clase-01 $ python notas-de-venta.py

Desea iva desglosado (Si/No) ? 16
    Error: 16 no es una respuesta válida, intenta de nuevo.

Desea iva desglosado (Si/No) ? s
-------------------------------------------------------------------------
PRODUCTO                                    | PRECIO    | CANT | SUBTOTAL
-------------------------------------------------------------------------
Automóvil                                   | 150000.00 |    1 | 150000.00
Bicicleta                                   |  13000.00 |    2 |  26000.00
Chamarra                                    |   3999.99 |    2 |   7999.98
Laptop Thinkpad (Descuento incluido)        |  21750.00 |    1 |  21750.00
Gafas de realidad virtual Lenovo con sable  |   5000.00 |    2 |  10000.00
-------------------------------------------------------------------------
                                                        Subtotal | 215749.98
-------------------------------------------------------------------------
                                                             IVA |  34520.00
                                                           Total | 250269.98
-------------------------------------------------------------------------
```