

Tema: Python

Contenido

El propósito de esta practica es aprender los fundamentos del lenguaje de programación Python, para ser aplicado posteriormente en la solución de problemas de aprendizaje automático y ciencias de datos.

Objetivo Especifico

- a) Instalar Python y entornos de desarrollo IDE.
- b) Aprender la sintaxis del lenguaje de programación Python.

Material y Equipo

- a) Virtual Box
- b) Linux Mint 21.3

Introduccion Teorica

Según la documentación oficial, Python es un lenguaje de programación potente y fácil de aprender. Tiene estructuras de datos eficientes de alto nivel y un enfoque simple pero efectivo para la programación orientada a objetos. Dispone de una elegante sintaxis y la escritura dinámica de Python, junto con su naturaleza de lenguaje interpretado, lo convierten en un lenguaje ideal para secuencias de comandos y desarrollo rápido de aplicaciones en muchas áreas y en la mayoría de las plataformas.

El intérprete de Python y su extensa biblioteca estándar están disponibles gratuitamente en formato fuente o binario para todas las plataformas principales desde el sitio web de Python, <https://www.python.org/>, y pueden distribuirse gratuitamente. El mismo sitio también contiene distribuciones y punteros a muchos módulos, programas y herramientas gratuitos de Python de terceros, y documentación adicional.

Procedimiento

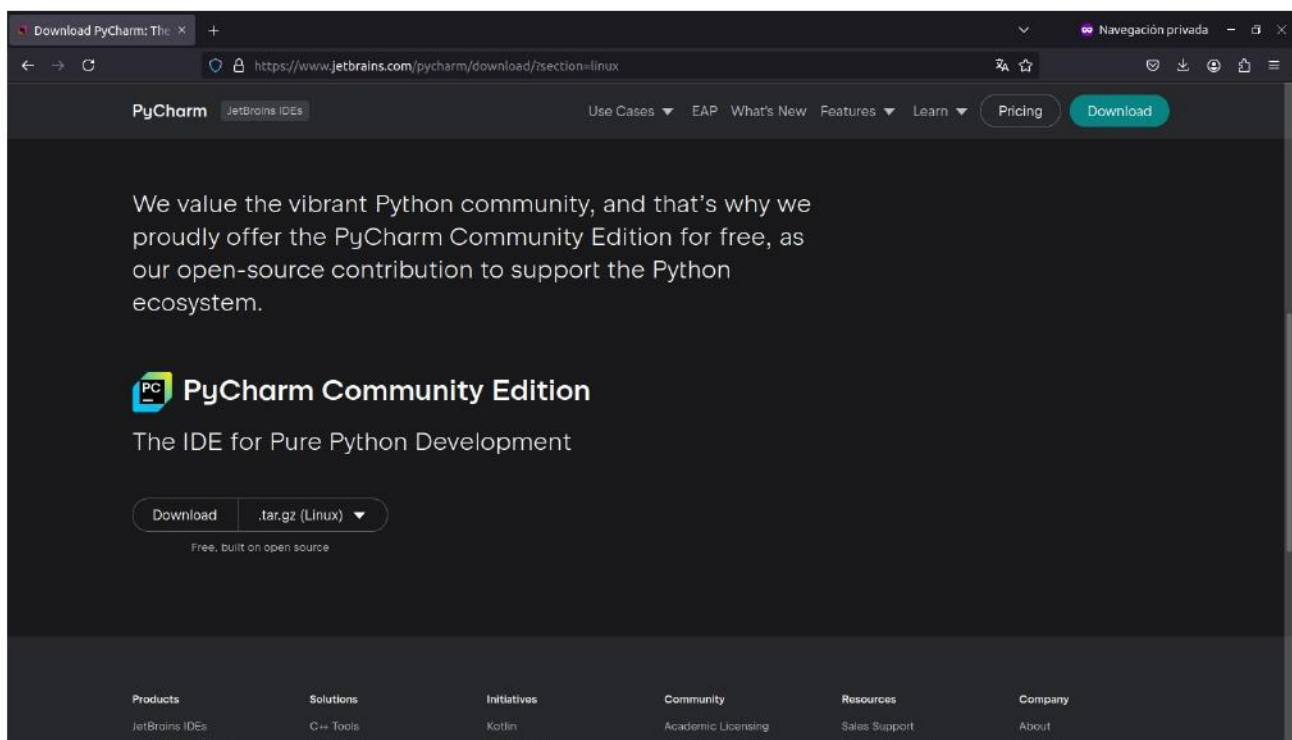
Instalación de Python

```
$ apt install software-properties-common -y  
$ add-apt-repository ppa:deadsnakes/ppa  
$ apt update  
$ apt install python3.12  
$ python3  
0  
$ python3.12 --version
```

Instalación de Pycharm o Geany como IDE

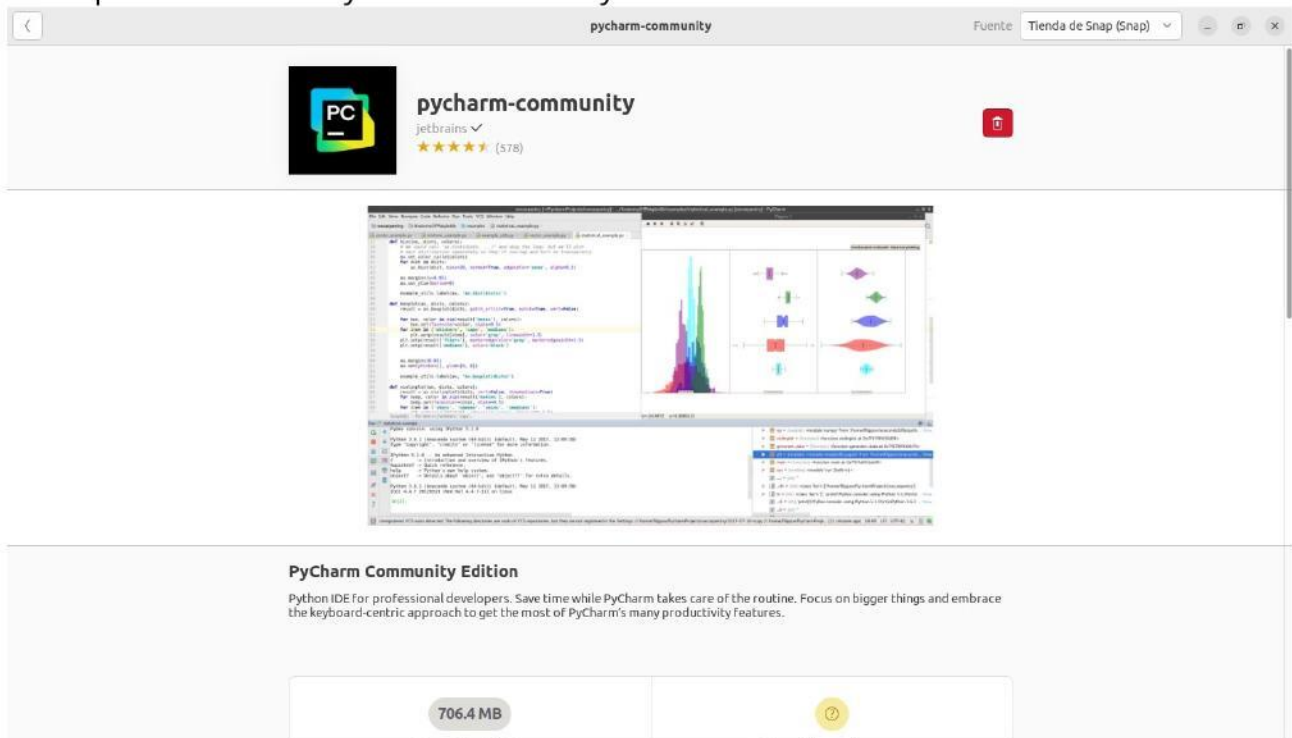
Descargar Pycharm Community Edition:

<https://www.jetbrains.com/pycharm/download/?section=linux>



3 ASI104 - Guía 4

Otra opción es instalar Pycharm Community Edition desde el Gestor de Software de Linux:

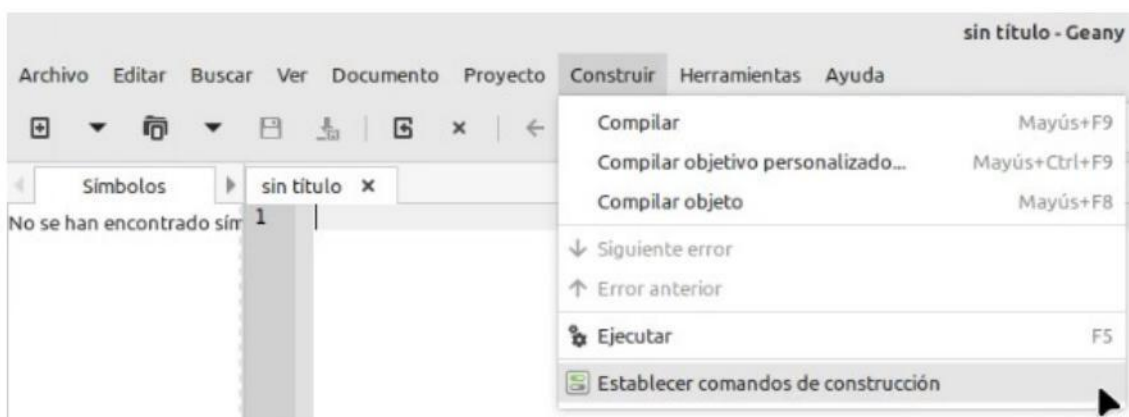


Otra opción más ligera es instalar Geany como IDE:

```
#sudo apt-get install geany
```

Instalado geany se debe configurar las opciones de compilación para que Geany pueda interpretar y ejecutar un programa en Python.

Desde el menú principal de Geany seleccionar la opción “Construir”, después seleccionar “Establecer comandos de construcción”



En la ventana de “Establecer los comandos de construcción”, asegurarse que las opciones “Comandos de Python” y “Comandos de ejecución” estén configurados con la cadena que se indica en el campo “comando”.

Establecer los comandos de construcción

#	Etiqueta	Comando	Directorio de trabajo	Reiniciar
Comandos de Python				
1.	Compile	python3 -m py_compile "%f"		
2.				
3.	Lint	pep8 --max-line-length=80 '		
Expresión regular de error:		(.+)::([0-9]+)::([0-9]+)		
Comandos independientes				
1.	Compilar	make		
2.	Compilar objetivo personalizado...	make		
3.	Compilar objeto	make %e.o		
4.				
Expresión regular de error:				
<i>Nota: El elemento 2 abre un diálogo y añade la respuesta al comando.</i>				
Comandos de ejecución				
1.	Execute	python3 "%F"		
2.				
<i>%d, %e, %f, %p y %l se sustituirán en los campos de comandos y directorios, consulte el manual para más información.</i>				
			Cancelar	Aceptar

Fundamentos Python

Tipos de datos

Los tipos de datos básicos en Python son cuatro: enteros, flotantes, números complejos, y booleanos:

Enteros	1, -3, 42, 355, 8888888888888888, -7777777777
Flotantes	3.0, 31e12, -6e-4
Números complejos	3 + 2j, -4- 2j, 4.2 + 6.3j
Booleanos	True, False

Numeros Enteros

```
walter@pc-walter: ~$ python
Python 2.7.18 (default, Jul 1 2022, 12:27:04)
[GCC 9.4.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 10 + 21
31
>>> 15 * 70
1050
>>> 7/2
3
>>> 7//2
3
>>> 7%2
1
>>> 2**3
8
>>> 10000000000**3
10000000000000000000000000000000000000000L
>>>
```

Números Flotantes

```
walter@pc-walter: ~  
walter@pc-walter:~$ python  
Python 2.7.18 (default, Jul 1 2022, 12:27:04)  
[GCC 9.4.0] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> 5.1**2.7  
81.36534597766715  
>>> 256.15/15.8  
16.212025316455694  
>>> 256.15*15.8  
4047.1699999999996  
>>> 1000000000.1**4  
1.0000000004000001e+36  
>>> 
```

Números complejos

```
walter@pc-walter: ~  
walter@pc-walter:~$ python  
Python 2.7.18 (default, Jul 1 2022, 12:27:04)  
[GCC 9.4.0] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> (12.7+14.25j) + (23.4-7.8j)  
(36.099999999999994+6.45j)  
>>> (12.7+14.25j) * (23.4-7.8j)  
(408.3299999999999+234.39j)  
>>> (12.7+14.25j) / (23.4-7.8j)  
(0.3057692307692307+0.7108974358974359j)  
>>> (12.7+14.25j) ** (23.4-7.8j)  
(-6.618784826260123e+32+9.220737912259308e+31j)  
>>> x=(12.7+14.25j) + (23.4-7.8j)  
>>> x  
(36.099999999999994+6.45j)  
>>> x.real  
36.099999999999994  
>>> x.imag  
6.45  
>>> 
```


Operadores aritméticos

En python están disponibles los siguientes operadores aritméticos:

Operador	Nombre	Ejemplo
+	adición	$x + y$
-	substracción	$x - y$
*	multiplicación	$x * y$
/	división	x / y
%	modulo	$x \% y$
**	exponencial	$x ** y$
//	división entera	$x // y$

Operadores de asignación

Operador	Nombre	Ejemplo
=	$x = 5$	$x = 5$
+=	$x += 3$	$x = x + 3$
-=	$x -= 3$	$x = x - 3$
*=	$x *= 3$	$x = x * 3$
/=	$x /= 3$	$x = x / 3$
%=	$x \% = 3$	$x = x \% 3$
//=	$x //= 3$	$x = x // 3$
**=	$x ** = 3$	$x = x ** 3$
&=	$x \& = 3$	$x = x \& 3$
=	$x = 3$	$x = x 3$
^=	$x \wedge = 3$	$x = x \wedge 3$
>>=	$x >> = 3$	$x = x >> 3$
<<=	$x << = 3$	$x = x << 3$

Operadores de comparación

Operador	Nombre	Ejemplo
==	Igual	$x == y$
!=	Desigual	$x != y$
>	Mayor que	$x > y$
<	Menor que	$x < y$
>=	Mayor o igual que	$x >= y$
<=	Menor o igual que	$x <= y$

Listas

Una lista puede contener una combinación de varios tipos de datos como parte de sus elementos, incluidas cadenas, tuplas, listas, diccionarios, funciones, objetos de archivo.

Una lista se puede indexar desde el frente (inicio) o desde el reverso (final).

También se puede hacer referencia a un subsegmento o sector de una lista utilizando la notación de sectores:

```
walter@pc-walter: ~
python3
Python 3.8.10 (default, Nov 22 2023, 10:22:35)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> x = ["primero", "segundo", "tercero", "cuarto"]
>>> x[0]
'primero'
>>> x[2]
'tercero'
>>> x[-1]
'cuarto'
>>> x[-2]
'tercero'
>>> x[1:-1]
['segundo', 'tercero']
>>> x[0:3]
['primero', 'segundo', 'tercero']
>>> x[-2:-1]
['tercero']
>>> x[:3]
['primero', 'segundo', 'tercero']
>>> x[-2:]
['tercero', 'cuarto']
>>>
```

x=	["primero",	"segundo",	"tercer",	"cuarto"]
Indices positivos		0	1	2	3	
Indices negativos		-4	-3	-2	-1	

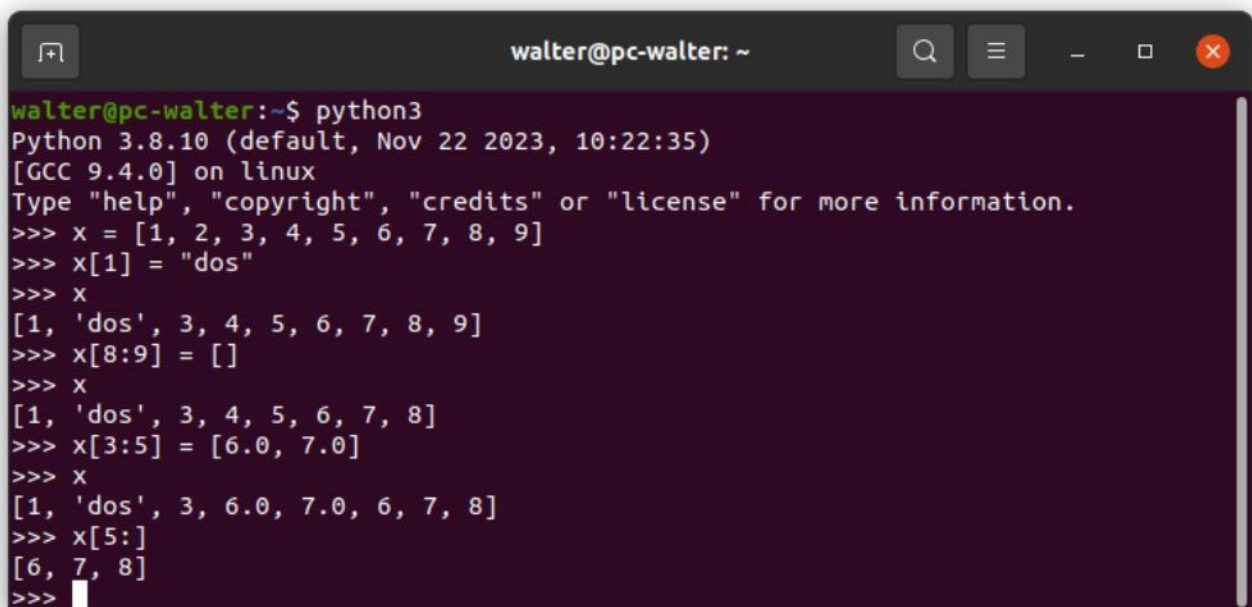

```

>>> x = ["primero", "segundo", "tercero", "cuarto"]
>>> x[0]
'primero'
>>> x[2]
'tercero'
>>> x[-1]
'cuarto'
>>> x[-2]
'tercero'
>>> x[1:-1]
['segundo', 'tercero']
>>> x[0:3]
['primero', 'segundo', 'tercero']
>>> x[-2:-1]
['tercero']
>>> x[:3]
['primero', 'segundo', 'tercero']
>>> x[-2:]
['tercero', 'cuarto']

```

Reemplazo de elementos de una lista

También se puede hacer referencia a un subsegmento o sector de una lista utilizando la notación de sectores:

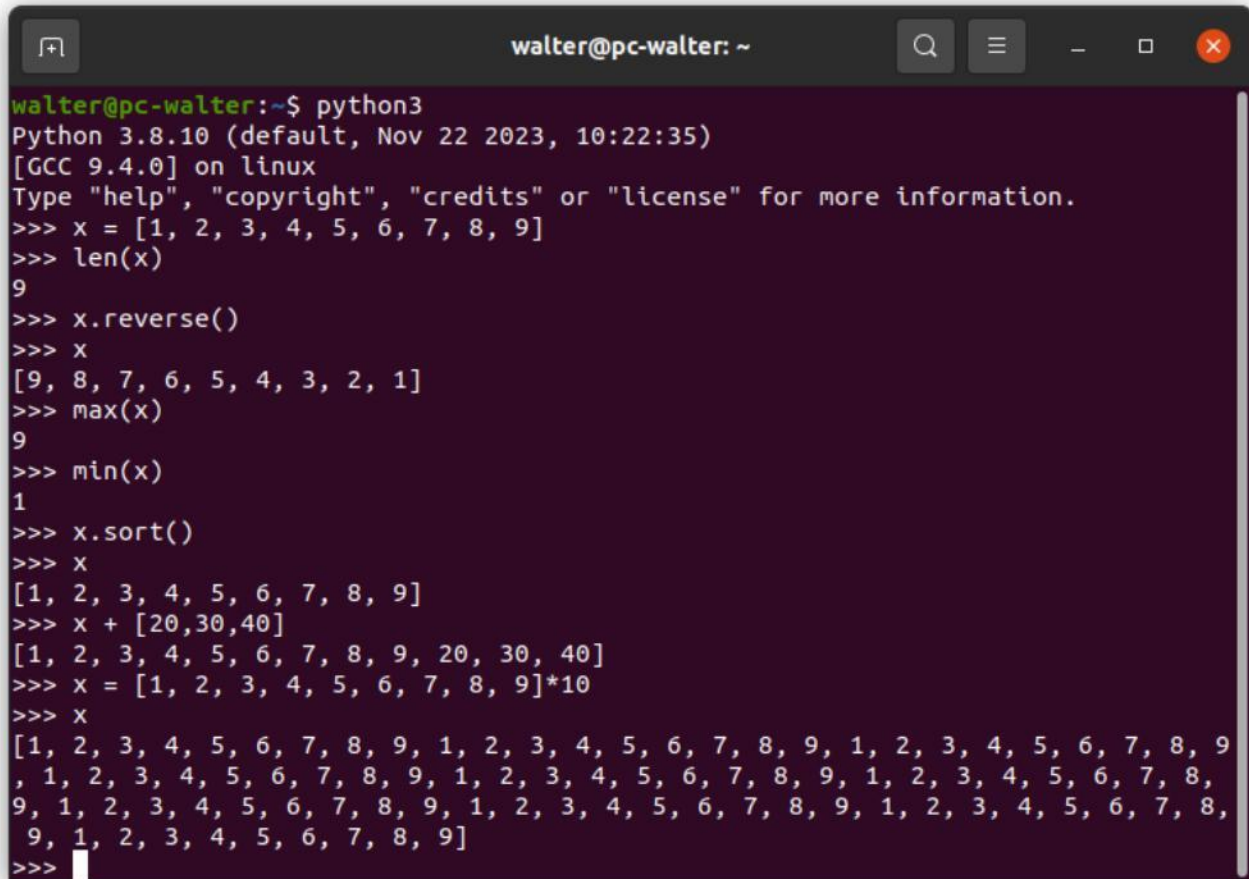


```

walter@pc-walter: ~
walter@pc-walter:~$ python3
Python 3.8.10 (default, Nov 22 2023, 10:22:35)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> x = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> x[1] = "dos"
>>> x
[1, 'dos', 3, 4, 5, 6, 7, 8, 9]
>>> x[8:9] = []
>>> x
[1, 'dos', 3, 4, 5, 6, 7, 8]
>>> x[3:5] = [6.0, 7.0]
>>> x
[1, 'dos', 3, 6.0, 7.0, 6, 7, 8]
>>> x[5:]
[6, 7, 8]
>>>

```

Algunas funciones integradas pueden ser utilizadas para gestionar listas (len, max y min), algunos operadores como (in, + y *), también la instrucción del y los métodos de lista (append , count , extend , index , insert , pop , remove , reverse , and sort):



```
walter@pc-walter: ~  
walter@pc-walter:~$ python3  
Python 3.8.10 (default, Nov 22 2023, 10:22:35)  
[GCC 9.4.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> x = [1, 2, 3, 4, 5, 6, 7, 8, 9]  
>>> len(x)  
9  
>>> x.reverse()  
>>> x  
[9, 8, 7, 6, 5, 4, 3, 2, 1]  
>>> max(x)  
9  
>>> min(x)  
1  
>>> x.sort()  
>>> x  
[1, 2, 3, 4, 5, 6, 7, 8, 9]  
>>> x + [20,30,40]  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 20, 30, 40]  
>>> x = [1, 2, 3, 4, 5, 6, 7, 8, 9]*10  
>>> x  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9,  
1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8,  
9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8,  
9, 1, 2, 3, 4, 5, 6, 7, 8, 9]  
>>>
```

Tuplas

Las tuplas son similares a las listas pero son inmutables, es decir, no se pueden modificar una vez creadas. Los operadores (in, + y *) y las funciones integradas (en, max y min) operan en Tuplas de la misma manera que en las listas, porque ninguno de ellos modifica el original. La notación de índice y de corte funciona de la misma manera para obtener elementos o sectores, pero no se pueden utilizar para agregar, eliminar o reemplazar elementos. También hay sólo dos métodos de tupla, count y index. Las tuplas son más eficientes de usar cuando no se necesita realizar modificación de los datos.

```
walter@pc-walter: ~  
walter@pc-walter:~$ python3  
Python 3.8.10 (default, Nov 22 2023, 10:22:35)  
[GCC 9.4.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> x = (1, "dos", 3, 4.0, ["a", "b"], (5, 6))  
>>> x  
(1, 'dos', 3, 4.0, ['a', 'b'], (5, 6))  
>>> y = [1, 2, 3, 4]  
>>> y  
[1, 2, 3, 4]  
>>> tuple(y)  
(1, 2, 3, 4)  
>>> list(x)  
[1, 'dos', 3, 4.0, ['a', 'b'], (5, 6)]  
>>> z = x.index(4.0)  
>>> z  
3  
>>> x.index(4.0)  
3  
>>> y.count(3)  
1  
>>> y = [1, 2, 3, 4, 3, 5, 10, 12, 3, 15, 20]  
>>> y.count(3)  
3  
>>> y = [1, 2, 3, 4, 3, 5, 10, 12, 3, 15, 20, 3, 50, 60]  
>>> y.count(3)  
4  
>>> 
```

Diccionarios

El tipo de datos diccionario de Python proporciona una funcionalidad de matriz asociativa. La función `len()` devuelve el número de pares clave-valor en un diccionario. La declaración `del` se puede utilizar para eliminar un par clave-valor. Como es el caso de las listas, varios métodos de diccionario están disponibles: `clear`, `copy`, `get`, `has_key`, `items`, `keys`, `update`, and `values`.

```
walter@pc-walter: ~
Python 3.8.10 (default, Nov 22 2023, 10:22:35)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> x = {"uno": 1, "dos": 2, "tres": 3, "cuatro": 4}
>>> x
{'uno': 1, 'dos': 2, 'tres': 3, 'cuatro': 4}
>>> y = {"SI0104": "Sistemas Operativos", "PRE104": "Programacion estructurada",
"ACD104": "Aprendizaje Automatico y Ciencia de Datos"}
>>> y
{'SI0104': 'Sistemas Operativos', 'PRE104': 'Programacion estructurada', 'ACD104': 'Aprendizaje Automatico y Ciencia de Datos'}
>>> y["PRE104"]
'Programacion estructurada'
>>> y.get("DSM104", "no disponible")
'no disponible'
>>> y.get("ACD104", "no disponible")
'Aprendizaje Automatico y Ciencia de Datos'
>>> y["P00104"] = "Programacion Orientado a Objetos"
>>> y
{'SI0104': 'Sistemas Operativos', 'PRE104': 'Programacion estructurada', 'ACD104': 'Aprendizaje Automatico y Ciencia de Datos', 'P00104': 'Programacion Orientado a Objetos'}
>>>
```

Sets

Un Set en Python es una colección no ordenada de elementos, que se utiliza en situaciones en las que la pertenencia y la unicidad del conjunto son lo principal. Se puede pensar en los Sets como una colección de claves de diccionario sin ningún valor asociado.

```
walter@pc-walter: ~
Python 3.8.10 (default, Nov 22 2023, 10:22:35)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> x = set([1, 2, 3, 1, 3, 5])
>>> x
{1, 2, 3, 5}
>>> 1 in x
True
>>> 7 in x
False
>>>
```


Estructuras condicionales

If Simple

Ejemplo 1 - if

```
if_simple.py x
1  año = int(input("Ingrese año: "))
2  if (año % 400 == 0) or (año % 4 == 0) and (año % 100 != 0):
3      print("El año ingresado es Bisiesto")
4  else:
5      print("El año No es Bisiesto")
6
```

Salida:

```
Run if_simple x
/home/walter/PycharmProjects/ClaseTeoriaPython/.venv/bin/python /home/walter/
ACD104 - Walter Sanchez
Ingrese año: 2052
El año ingresado es Bisiesto
Process finished with exit code 0
```

```
Run if_simple x
/home/walter/PycharmProjects/ClaseTeoriaPython/.venv/bin/python /home/walter/
ACD104 - Walter Sanchez
Ingrese año: 2057
El año No es Bisiesto
Process finished with exit code 0
```

Ejemplo 2 - if

```
if_simple_cuadratica.py x
1  a = int( input("Ingresar coeficiente a: "))
2  b = int( input("Ingresar coeficiente b: "))
3  c = int( input("Ingresar coeficiente c: "))
4
5  d = b**2 - 4*a*c
6
7  if d > 0 :
8      x1 = ( (-b) + d**0.5 )/ 2*a
9      x2 = ( (-b) - d**0.5 )/ 2*a
10     print("x1 = ", x1)
11     print("x2 = ", x2)
12 else:
13     print("La Solución con Números Complejos")
14
```

Salida:

```
Run if_simple_cuadratica x
/home/walter/PycharmProjects/ClaseTeoriaPython/.venv/bin/python /home/walter/
ACD104 - Walter Sanchez
Ingresar coeficiente a: 1
Ingresar coeficiente b: -5
Ingresar coeficiente c: 2
x1 = 4.561552812808831
x2 = 0.4384471871911697

Process finished with exit code 0
```


Ejemplo 3 – If en Escalera

```
if_escalera.py x
1  cum = float(input("Ingresar CUM: "))
2
3  if cum >= 7.5:
4      print("Puede cursar un maximo de 32 UV")
5  elif cum >= 7.0 and cum < 7.5:
6      print("Puede cursar un maximo de 24 UV")
7  elif cum >= 6.0 and cum < 7.0:
8      print("Puede cursar un maximo de 20 UV")
9  elif cum < 6.0:
10     print("Puede cursar un maximo de 16 UV")
11
```

Salida:

```
Run if_escalera x
/home/walter/PycharmProjects/ClaseTeoriaPython/.venv/bin/python /home/walter/ACD104 - Walter Sanchez
Ingresar CUM: 9.5
Puede cursar un maximo de 32 UV
Process finished with exit code 0
```

```
Run if_escalera x
/home/walter/PycharmProjects/ClaseTeoriaPython/.venv/bin/python /home/walter/ACD104 - Walter Sanchez
Ingresar CUM: 7.2
Puede cursar un maximo de 24 UV
Process finished with exit code 0
```

Ejemplo 4 – Estructura condicional múltiple

```
EstructurasCondicionales.py x
1  Meses = {
2      1: "Enero",
3      2: "Febrero",
4      3: "Marzo",
5      4: "Abril",
6      5: "Mayo",
7      6: "Junio",
8      7: "Julio",
9      8: "Agosto",
10     9: "Septiembre",
11    10: "Octubre",
12    11: "Noviembre",
13    12: "Diciembre"
14 }
15
16 entrada = int(input("Ingrese número de mes: "))
17 Mes = Meses.get(entrada, "Número de mes invalido")
18 print(Mes)
```

Salida:

```
Run EstructurasCondicionales x
/home/walter/PycharmProjects/ClaseTeoriaPython/.venv/bin/python /home/walter/
ACD104 - Walter Sanchez
Ingrese número de mes: 7
Julio
Process finished with exit code 0
```

```
Run EstructurasCondicionales x
/home/walter/PycharmProjects/ClaseTeoriaPython/.venv/bin/python /home/walter/
ACD104 - Walter Sanchez
Ingrese número de mes: 11
Noviembre
Process finished with exit code 0
```

Estructuras Repetitivas

Estructura for

Ejemplo 1 - for

```
for_ejemplo01.py x
1  numero = int(input("Ingresar un número: "))
2
3  contador = 0
4  for i in range(2, numero):
5      if numero % i == 0:
6          contador = contador + 1
7
8  if contador == 0:
9      print(numero, " es un número primo")
10 else:
11     print(numero, " no es un número primo")
12
```

Salida:

```
Run for_ejemplo01 x
:
/home/walter/PycharmProjects/ClaseTeoriaPython/.venv/bin/python /home/walter/
ACD104 - Walter Sanchez
Ingresar un número: 41
41 es un número primo
Process finished with exit code 0
```

```
Run for_ejemplo01 x
:
/home/walter/PycharmProjects/ClaseTeoriaPython/.venv/bin/python /home/walter/
ACD104 - Walter Sanchez
Ingresar un número: 40
40 no es un número primo
Process finished with exit code 0
```

Ejemplo 2 - for

```
for_ejemplo02.py x
1  inicio = 2
2  ultimo = 20
3  incremento = 2
4  for i in range(inicio, ultimo+1, incremento):
5      c = i**3
6      print("El cubo de ", i, "es ", c)
7
```

Salida:

```
Run for_ejemplo02 x
/home/walter/PycharmProjects/ClaseTeoriaPython/.venv/bin/python /home/walter/
ACD101 - Walter Sanchez
El cubo de 2 es 8
El cubo de 4 es 64
El cubo de 6 es 216
El cubo de 8 es 512
El cubo de 10 es 1000
El cubo de 12 es 1728
El cubo de 14 es 2744
El cubo de 16 es 4096
El cubo de 18 es 5832
El cubo de 20 es 8000

Process finished with exit code 0
```

Ejemplo 3 - for

```
for_ejemplo03.py ×  
1 print ("Serie: 1 - (1/2) + (1/3) - (1/4) + (1/5) .... + (1/N)")  
2 print ("Ingrese número de términos de la serie:")  
3 N = int(input())  
4  
5 S = 0  
6 for x in range(1, N+1):  
7     if (x % 2) == 0:  
8         S = S - (1/x)  
9     else:  
10        S = S + (1/x)  
11 print ("La suma de la serie es:", S)
```

Salida:

```
Run for_ejemplo03 ×  
/home/walter/PycharmProjects/ClaseTeoriaPython/.venv/bin/python /home/walter/PycharmProjects/ClaseTeoriaPython/for_ejemplo03.py  
ACD101 - Walter Sanchez  
Serie: 1 - (1/2) + (1/3) - (1/4) + (1/5) .... + (1/N)  
Ingrese número de términos de la serie:  
20  
La suma de la serie es: 0.6687714031754279  
Process finished with exit code 0
```

Estructura while

Ejemplo 1 - while

```
while-ejemplo01.py x
1  n = 0
2  while n < 10:
3      n += 1
4      if n % 2 == 0:
5          continue
6      print(n)
7
```

Salida:

```
Run while-ejemplo01 x
/home/walter/PycharmProjects/ClaseTeoriaPython/.venv/bin/python /home/walter/
ACD101 - Walter Sanchez
1
3
5
7
9

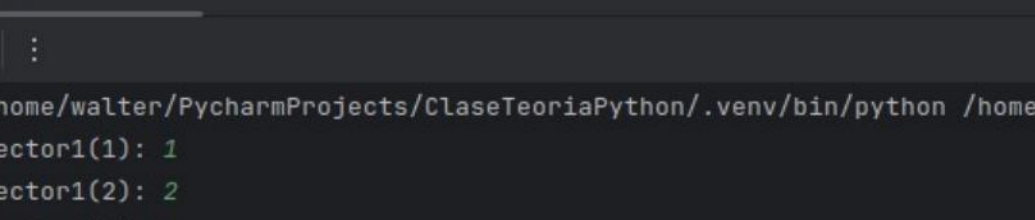
Process finished with exit code 0
```


Vectores

Ejemplo 1 – vectores

```
matriz01.py x
1 Vector1 = 3*[0]
2 Vector2 = [0]*3
3 for k in range(3):
4     Vector1[k] = int(input("Vector1({}): ".format(k+1)))
5 for k in range(3):
6     Vector2[k] = int(input("Vector2({}): ".format(k+1)))
7 suma = 0
8 for k in range(3):
9     W = Vector1[k]*Vector2[k]
10    suma = suma + W
11 print("El producto escalar es:", suma)
12 x = Vector1[1]* Vector2[2] - Vector1[2]* Vector2[1]
13 y = -( Vector1[0]* Vector2[2] - Vector1[2]* Vector2[0] )
14 z = Vector1[0]* Vector2[1] - Vector1[1]* Vector2[0]
15 print("El producto vectorial es: {}i {}j {}k".format(*args: x, y, z))
16
```

Salida:



```
Run matriz01 x
/home/walter/PycharmProjects/ClaseTeoriaPython/.venv/bin/python /home/walter/
Vector1(1): 1
Vector1(2): 2
Vector1(3): 3
Vector2(1): -1
Vector2(2): 1
Vector2(3): 2
El producto escalar es: 7
El producto vectorial es: 1i -5j 3k

Process finished with exit code 0
```