**Módulo de 8:**

* **Random**
* **Operadores ternarios**
  + **if**
* **Colecciones por comprensión (comprehension).**
  + **If else for**
    - **lists**
    - **sets**
    - **dict**
* **Funciones all y any.**
  + - **all**
    - **any**

**Random**

* Import random

Este Módulo implementa generadores de números “aleatorios” realmente pseudoaleatorios y realizar otras operaciones relacionadas con la “aleatoriedad”.

Realmente ninguna computadora puede generar números aleatorios pero los mas cercanos es este Módulo.

|  |
| --- |
| En la selección de enteros, existe una selección uniforme dentro de un rango.  Para las secuencias, existe una selección uniforme de un elemento aleatorio, una función para generar una permutación aleatoria de una lista in-situ y una función para el muestreo aleatorio sin reemplazo.  Para números reales, existen funciones para calcular distribuciones uniformes, normales (Gaussianas), log-normales, exponenciales negativas, gamma y beta. Para generar distribuciones angulares está disponible la distribución de von Mises.  La clase Random puede ser también subclaseada si quieres usar un generador básico diferente para tu propio diseño: en este caso, invalida los métodos random(), seed(), getstate() y setstate(). Opcionalmente, se puede substituir un método getrandbits() por un nuevo generador — esto permite a randrange() producir selecciones sobre un rango arbitrariamente amplio.  El Módulo random también proporciona la clase SystemRandom, la cuál usa la función del sistema os.urandom() para generar números aleatorios a partir de fuentes proporcionadas por el sistema operativo. |

·

|  |
| --- |
| * La cantidad de módulos de random es grande. Solo veremos 4 métodos   random  randint  choise  shuffle |

·

El Módulo random proporciona varios métodos.

El método random.randint ( inicio , fin ) genera números aleatorios entre inicio y fin (ambos inclusive).

|  |
| --- |
| import random  for contador in range (1,6):  ruleta = random.randint(0,36)  print (f"La salida Nº{contador} salio {ruleta}") |

Salida esperada por consola

|  |
| --- |
| La salida Nº1 salio **18#< --------------- estos valores serán diferentes**  La salida Nº2 salio 11**#< --------------- estos valores serán diferentes**  La salida Nº3 salio 8**#< ---------------- estos valores serán diferentes**  La salida Nº4 salio 17**#< --------------- estos valores serán diferentes**  La salida Nº5 salio 22**#< --------------- estos valores serán diferentes** |

·

El método random.random() generar un número de punto flotante (float) aleatorio en el rango (0.0, 1.0).

|  |
| --- |
| import random  for contador in range (1,6):  ruleta = random.randint(0,36)  print (f"La salida Nº{contador} salio {ruleta}") |

Salida esperada por consola

|  |
| --- |
| **Flotante Nº1 - 0.16379003716651208#< --------------- estos valores serán diferentes**  **Flotante Nº2 - 0.42712189631259456#< --------------- estos valores serán diferentes**  **Flotante Nº3 - 0.0179556551311989#< ----------------- estos valores serán diferentes**  **Flotante Nº4 - 0.841633219622988#< ------------------ estos valores serán diferentes**  **Flotante Nº5 - 0.06566822557938701#< --------------- estos valores serán diferentes** |

·

El método random.choice (colección) selecciona aleatoriamente un elemento de la colección.

|  |
| --- |
| import random  lista = ["A","B","C","D","E","F"]  for contador in range (1,6):  desde\_lista = random.choice(lista)  print (f"Desde lista Nº{contador} - {desde\_lista}") |

Salida esperada por consola

|  |
| --- |
| Desde lista Nº1 - **F#< --------------- estos valores serán diferentes**  Desde lista Nº2 - **A#< --------------- estos valores serán diferentes**  Desde lista Nº3 - **D#< --------------- estos valores serán diferentes**  Desde lista Nº4 - **F#< --------------- estos valores serán diferentes**  Desde lista Nº5 - **C#< --------------- estos valores serán diferentes** |

·

El método random.shuffle (colección) reorganiza los elementos de la colección en un orden aleatorio.

|  |
| --- |
| import random  lista = ["A","B","C","D","E","F"]  print (f"lista antes {lista}")  random.shuffle(lista)  print (f"lista despues {lista}") |

Salida esperada por consola

|  |
| --- |
| lista antes **['A', 'B', 'C', 'D', 'E', 'F']**  lista despues **['E', 'D', 'F', 'B', 'C', 'A']#< --------------- estos valores serán diferentes** |

·

**Operadores ternarios**

Los operadores ternarios u operadores condicionales, permiten expresar una condición ( if-else en una sola línea) y realizar una asignación o retorno de valores basados en esa condición.

Permite en una forma concisa de expresar una estructura condicional en una sola línea. A diferencia de una estructura if-else tradicional, los operadores ternarios permiten evaluar una expresión y retornar un resultado basado en una condición.

|  |
| --- |
| Operadores ternarios solo se trabaja con   * + if   + else |

|  |
| --- |
| Solo se trabaja con if else |

La sintaxis básica del operador ternario en Python es la siguiente:

**salida ="**valor o expresión verdadero**" if condición else "**valor o expresión falsa**"**

|  |
| --- |
| edad = 18  comprar\_alcohol = True if edad >= 18 else False  print(f"puedes comprar alcohol? {comprar\_alcohol}") |

Salida esperada por consola

|  |
| --- |
| puedes comprar alcohol? True |

·

|  |
| --- |
| edad = 15  comprar\_alcohol = True if edad >= 18 else False  print(f"puedes comprar alcohol? {comprar\_alcohol}") |

Salida esperada por consola

|  |
| --- |
| puedes comprar alcohol? False |

·

salida = ("valor o expresión verdadera","valor o expresión falsa")[booleana]

|  |
| --- |
| anda = True  estado = ("si: el codigo corre", "bug: no, el código no corre")[anda]  print("estado del programa ", estado) |

Salida esperada por consola

|  |
| --- |
| estado del programa bug: no, el código no corre |

·

|  |
| --- |
| Los operadores ternarios son útiles para expresar de manera concisa estructuras condicionales simples en una sola línea de código.  Según la comunidad, no son métodos Python estándar , es importante utilizarlos con moderación y asegurarse de que la expresión sea legible y comprensible para otros programadores que lean el código.  Como en varios elementos (el mas conocido operador walrus :=) la comunidad asegura que confunde el código y que resta mas de lo que suma |

**Colecciones por comprensión (comprehension).**

Python permite construir de con una sintaxis compacta y de forma simple y rápida listas, conjuntos y diccionarios (list,str & dict). Son una forma elegante y eficiente de crear colecciones en base a iteraciones y condiciones.

·

|  |
| --- |
| Colecciones por comprensión solo se trabaja con   * + for   + if   + else   (aunque pueden ser varios anidados) |

La sintaxis básica de comprensiones de colecciones en Python es la siguiente:

iterables: un iterable es un objeto que se puede recorrer o iterar elemento por elemento.

Básicamente, cualquier objeto que pueda devolver uno a la vez sus elementos cuando se utiliza en un bucle for se considera un iterable.

Los iterables pueden ser objetos como listas, tuplas, conjuntos, diccionarios, cadenas de texto y otros.

Cuando se utiliza un bucle for para iterar sobre un iterable, se obtiene acceso a cada elemento del iterable en sucesión. Esto permite realizar operaciones en cada elemento o realizar algún tipo de procesamiento secuencial.

En todos estos ejemplos, se utiliza un bucle for para iterar sobre los elementos de los objetos iterables. En cada iteración, se obtiene un elemento del iterable y se realiza alguna operación con él.

Es importante tener en cuenta que un objeto iterable se puede recorrer una vez. Después de que se haya agotado su secuencia de elementos, no se puede volver a recorrer a menos que se cree un nuevo iterable o se reinicie de alguna manera.

En resumen, un iterable en Python es cualquier objeto que se puede recorrer o iterar elemento por elemento, lo que permite realizar operaciones en cada elemento de forma secuencial.

La sintaxis básica de un objeto por comprensión en Python es la siguiente:

**listas y sets:**

**[**"valor o expresión" for elemento in iterable **]**

**[**"valor o expresión verdadero" for elem in iterable **if condición ]**

**[**"valor/expresión verdadero" **if condición else** "**valor/expresión falso**" for elem in iterable**]**

·

|  |
| --- |
| numeros = [2, 9, 6, 25, 8, 49, 10, 81, 12 ]  salidas = [cada\_numero \*\* 2 for cada\_numero in numeros]  print (f"{salidas=}")  #--------------------------------------------------------  salidas = [cada\_numero \*\* 2 for cada\_numero in numeros if cada\_numero % 2 == 0]  print (f"{salidas=}")  #--------------------------------------------------------  salidas = [cada\_numero \*\* 2 if cada\_numero % 2 == 0 else cada\_numero \*\*(1/2) for cada\_numero in numeros]  print (f"{salidas=}")  #-------------------------------------------------------- |

Salida esperada por consola

|  |
| --- |
| salidas=[4, 81, 36, 625, 64, 2401, 100, 6561, 144]  salidas=[4, 36, 64, 100, 144]  salidas=[4, 3.0, 36, 5.0, 64, 7.0, 100, 9.0, 144] |

·

**diccionario:**

**{**clave: valor **for** clave, valor **in** diccionario.items() }

**{**clave: valor **for** clave, valor **in** diccionario.items() if condición }

**{**clave: valor True **if condición else** "**valor falso**" **for** clave,valor **in** diccionario.items()}

·

|  |
| --- |
| numeros\_dic={"1ro" : 6 , "2do" : 49 , "3ro" : 25 , "4to" : 4, "5to" : 9}  salidas = {cada\_clave : cada\_numero \*\* 2 for cada\_clave,cada\_numero in numeros\_dic.items()}  print (f"{salidas=}")  #--------------------------------------------------------  salidas = {cada\_clave : cada\_numero \*\* 2 for cada\_clave,cada\_numero in numeros\_dic.items() if cada\_numero % 2 == 0}  print (f"{salidas=}")  #--------------------------------------------------------  salidas = {cada\_clave : cada\_numero \*\* 2 if cada\_numero % 2 == 0 else cada\_numero \*\*(1/2) for cada\_clave,cada\_numero in numeros\_dic.items() }  print (f"{salidas=}")  #-------------------------------------------------------- |

Salida esperada por consola

|  |
| --- |
| salidas={'1ro': 36, '2do': 2401, '3ro': 625, '4to': 16, '5to': 81}  salidas={'1ro': 36, '4to': 16}  salidas={'1ro': 36, '2do': 7.0, '3ro': 5.0, '4to': 16, '5to': 3.0} |

·

En este ejemplo, se utiliza una comprensión de diccionario para generar un diccionario donde las claves son los nombres y los valores son números enteros.

**Funciones all y any.**

**all()**

**Devuelve verdadero**

si todas las condiciones o los elementos son verdaderos (o si el iterable está vacío).

**Devuelve False**

si está vacío o si todos son falsos.

Todo se puede considerar como una secuencia de operaciones AND en los iterables proporcionados.

·

|  |
| --- |
| todo= all([True, True, True, True])  print (f"{todo=}")  #--------------------------------------------------------  todo= all([False, True, True, False])  print (f"{todo=}")  #--------------------------------------------------------  todo= all([False, False, False])  print (f"{todo=}")  #-------------------------------------------------------- |

Salida esperada por consola

|  |
| --- |
| todo=True  todo=False  todo=False |

·

|  |
| --- |
| a=1  b=2  c=3  d=4  e=5  f=0  if all ([(a<b) ,( b<c) ,( c<d ) ,(d<e)]):  print("todas son True")  print ([(a<b) ,( b<c) ,( c<d ) ,(d<e)])  else:  print ("al menos una condicion no es True")  print([(a<b) ,( b<c) ,( c<d ) ,( d<e )]) |

Salida esperada por consola

|  |
| --- |
| todas son True  [True, True, True, True] |

·

|  |
| --- |
| a=1  b=2  c=3  d=4  e=5  f=0  if all ([(a<b) ,( b<c) ,( c<d ) ,( d<e ) ,(e<f)]):  print("todas son True")  print([(a<b) ,( b<c) ,( c<d ) ,( d<e ) ,(e<f)])  else:  print ("al menos una condicion no es True")  print([(a<b) ,( b<c) ,( c<d ) ,( d<e ) ,(e<f)]) |

Salida esperada por consola

|  |
| --- |
| al menos una condicion no es True  [True, True, True, True, False] |

· **any()**

**Devuelve verdadero** si todas las condiciones o los elementos son verdaderos (o si el iterable está vacío).

**Devuelve False** si está vacío o si todos son falsos.

Todo se puede considerar como una secuencia de operaciones AND en los iterables proporcionados.

·

|  |
| --- |
| algun= any([True, True, True, True])  print (f"{algun=}")  #--------------------------------------------------------  algun= any([False, True, True, False])  print (f"{algun=}")  #--------------------------------------------------------  algun= any([False, False, False])  print (f"{algun=}")  #-------------------------------------------------------- |

Salida esperada por consola

|  |
| --- |
| algun=True  algun=True  algun=False |

·

|  |
| --- |
| a=1  b=2  c=3  d=4  e=5  f=0  if any ([(a>b) ,( b<c) ,( c>d ) ,(d<e)]):  print("Alguno es True")  print ([(a>b) ,( b<c) ,( c>d ) ,(d<e)])  else:  print ("ninguno True")  print([(a>b) ,( b<c) ,( c>d ) ,( d<e )]) |

Salida esperada por consola

|  |
| --- |
| Alguno es True  [False, True, False, True] |

·

|  |
| --- |
| a=1  b=2  c=3  d=4  e=5  f=0  if any ([(a>b) ,( b>c) ,( c>d ) ,( d>e ) ,(e<f)]):  print("Alguno es True")  print([(a>b) ,( b>c) ,( c>d ) ,( d>e ) ,(e<f)])  else:  print ("ninguno True")  print([(a>b) ,( b>c) ,( c>d ) ,( d>e ) ,(e<f)]) |

Salida esperada por consola

|  |
| --- |
| ninguno True  [False, False, False, False, False] |

·

Estas funciones son especialmente útiles cuando necesitas verificar rápidamente si todos o al menos uno de los elementos de una secuencia cumplen una determinada condición.

Puedes utilizarlas en combinación con expresiones booleanas o con funciones lambda para realizar comprobaciones más complejas.

|  |
| --- |
| #se deberia validar el dato  nota\_matematica = int(input ("que nota tenes en matemática:"))  nota\_lengua = int(input ("que nota tenes en lengua:"))  nota\_fisica = int(input ("que nota tenes en física:"))  nota\_quimica = int(input ("que nota tenes en química:"))  nota\_gimnasia = int(input ("que nota tenes en gimnasia:"))  nota\_dibujo = int(input ("que nota tenes en dibujo:"))  lista\_materias = ["nota\_matemática", "nota\_lengua", "nota\_física", "nota\_química",  "nota\_gimnasia", "nota\_dibujo"]  estado\_toda\_las\_materias = [ nota\_matematica >= 7,  nota\_lengua >= 7,  nota\_fisica >= 7,  nota\_quimica >= 7,  nota\_gimnasia >= 7,  nota\_dibujo >= 7 ]  for estado\_materia in estado\_toda\_las\_materias:  print (f"Estado materia aprobado {estado\_materia}")  for materia, estado\_materia in zip(lista\_materias, estado\_toda\_las\_materias):# zip  print (f"Estado de {materia} aprobado {estado\_materia}")  if False in estado\_toda\_las\_materias:  print ('"IN" al menos desaprobada una')  if False not in estado\_toda\_las\_materias:  print ('"NOT IN" ni una desaprobada una')  if all (estado\_toda\_las\_materias):  print ('"ALL" felicitaciones todo aprobado')  if any (estado\_toda\_las\_materias):  print ('"ANY" al menos aprobada una')  if not all (estado\_toda\_las\_materias):  print ('"NOT ALL" al menos desaprobada una')  print ('"NOT ALL" te quedo algo colgado para rendir')  if not any (estado\_toda\_las\_materias):  print ('"NOT ANY"todo mallllllll') |