Unidad 4

Manejo de arreglos y funciones en Python

Cortes de Listas

Los **Cortes de Listas** ofrecen una manera más avanzada de obtener valores de una lista.

Se hace llamando una lista teniendo como argumento **dos enteros separado por dos puntos**.

Esto devuelve una lista que contiene todos los valores de la lista vieja entre los índices.

```
cuadrados = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
print (cuadrados[2:6])
print (cuadrados[3:8])
print (cuadrados[0:1])
```

```
>>>
[4, 9, 16, 25]
[9, 16, 25, 36, 49]
[0]
>>>
```

Cortes de Listas

¿Cuál es el resultado este código?

```
sqs = [0, 1, 4, 9, 16, 25, 36, 49, 64]
print(sqs[4:7])
```

- [25, 36, 49]
- [16, 25, 36]
- [16, 25, 36, 49]

Cortes de Listas

Si el primer número en un corte es omitido, toma el principio de la lista.

Si el segundo número es omitido, se toma el final de la lista.

```
cuadrados = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
print (cuadrados[:7])
print (cuadrados[7:])
```

```
>>>
[0, 4, 9, 16, 25, 36]
[49, 64, 81]
>>>
```

Cortes de Listas

Rellena los espacios en blanco para tomar los primeros dos elementos de la lista

```
lista = ["a", "b", "c", "d"]
a = lista [0 :2 ]
```

Cortes de Listas

Los cortes de lista también pueden tener un tercer número, representando el incremento, para incluir valores alternativos en el corte.

```
cuadrados = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
print (cuadrados[::2])
print (cuadrados[2:8:3])
```

```
>>>
[0, 4, 16, 36, 64]
[4, 25]
>>>
```

Cortes de Listas

¿Cuál es la salida de este código?

```
sqs = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
print (sqs[1::4])
```

- [ ] [1, 25]
- [x] [1, 25, 81]
- [ ] Ocurre un error
- [ ] [0, 1, 4]

Cortes de Listas

Los valores negativos pueden ser utilizados en un corte de lista.

Cuando los valores negativos son utilizados para el primer y segundo valor del corte, estos cuentan desde el final de la lista.

```
cuadrados = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
print (cuadrados[1:-1])
print (cuadrados[3:-1])
print (cuadrados[1:-3])
```

```
>>>
[1, 4, 9, 16, 25, 36, 49, 64]
[9, 16, 25, 36, 49, 64]
[1, 4, 9, 16, 25, 36]
>>>
```

Si un valor negativo es usado en un incremento, el corte es hecho al revés. Utilizando **[::-1]** como un corte es una manera común de invertir una lista.

Cortes de Listas

¿Cuál es la salida de este código?

```
sqs = [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
print (sqs[7:5:-1])
```

- [49, 36]
- [49]
- []

Listas por comprensión

Las **listas por compresión** son una manera útil de crear rápidamente listas cuyo contenido es una regla sencilla.

```
cubos = [i**3 for  i in range(5)]
print (cubos)
```

```
>>>
[0, 1, 8, 27, 64]
>>>
```

Listas por comprensión

¿Qué crea esta lista por comprensión?

```
nums = [i*2 for i in range(10)]
```

- ○ Una lista con todos los números entre 0 y 10
- ● Una lista con los números pares entre 0 y 18
- ○ Una lista con los números pares entre 0 y 10

Listas por comprensión

Una **lista por compresión** también puede contener una sentencia **if** para aplicar una condición en los valores de la lista.

```
nums = [i**2 for  i in range(10) if i**2 % 2 == 0]
print (nums)
```

```
>>>
[0, 4, 16, 36, 64]
>>>
```

Funciones de cadena

Python contiene muchas funciones integradas y métodos útiles que resultan para cumplir tareas comunes.

join – combina una lista de cadenas con otra cadena separador

```
print (", ".join(["spam","eggs","ham"]))
```

```
>>>
"spam, eggs, ham"
>>>
```

replace – reemplaza una subcadena de una cadena con otra

```
print ("Hola Bio".replace("Bio","mundo"))
```

```
>>>
"Hola mundo"
>>>
```

Funciones de cadena

startswith y **endswith –** determina si hay una subcadena al principio o al final de una cadena, respectivamente.

```
print ("Esta es una oración ".startswith("Esta"))
```

```
>>>
True
>>>
```

```
print ("Esta es una oración ".endswith("oración"))
```

```
>>>
True
>>>
```

Funciones de cadena

Para cambiar una cadena de mayúsculas a minúsculas y viceversa, de utilizan **lower** y **upper** respectivamente.

```
print ("ESTO ES UNA ORACIÓN ".lower())
```

```
>>>
esto es una oración
>>>
```

```
print ("Esto es una oración ".upper())
```

```
>>>
ESTO ES UNA ORACIÓN
>>>
```

split – este método es el opuesto de *join*, convirtiendo una cadena con un determinado separador de una lista.

```
print ("spam, eggs, ham".split(", "))
```

```
>>>
['spam', 'eggs', 'ham']
>>>
```

None

El objeto **None** es utilizado para representar la ausencia de un valor. Es similar al a **null** en otros lenguajes de programación.

```
>>> None == None
True
>>>print (None)
None
```

El objeto **None** es devuelto por cualquier función que NO retorne (return) algo explícitamente.

```
def alguna_func():
      Print (“Hola Mundo!”)

var=alguna_func()
print(var)
```

```
>>>
Hola Mundo!
None
>>>
```

Diccionarios

Los **diccionarios** son estructuras de datos utilizadas para mapear claves arbitrarias a valores.

Las listas pueden ser consideradas como diccionarios con clave de números enteros dentro de un cierto rango.

Los diccionarios pueden ser indexados de la misma manera que las listas, utilizando corchetes que contengan la clave.

```
edades = {"Andrés":24, "María":42, "Jorge":58}
print(edades["Andrés"])
print(edades["Jorge"])
```

Cada elemento de un diccionario es representado por un par **key:value**

```
>>>
24
58
>>>
```

Diccionarios

Tratar de indexar una clave que no es parte del diccionario devuelve un **KeyError**.

```
colores = {"rojo":[255,0,0], "verde":[0,255,0], "azul":[0,0,255]}
print(colores["rojo"])
print(colores["amarillo"])
```

```
>>>
[255,0,0]
KeyError: ‘amarillo’
>>>
```

Un diccionario vacío está definido como **{}**

Diccionarios

Al igual que las listas, las claves de un diccionario pueden ser asignadas a distintos valores.

A una nueva clave de un diccionario se le puede también asignar un valor, no solo a las ya existentes.

```
raiz = {1:1, 2:2, 3:"Bio", 4:16}
raiz[8] = 64
raiz[3] = 9
print (raiz)
```

```
>>>
{1: 1, 2: 2, 3: 9, 4: 16, 8: 64}
>>>
```

Diccionarios

¿Cuál es el resultado de este código?

```
primos = {1:2, 2:3, 4:7, 7:17}
print (primos[primos[4]])
```

17

Diccionarios

Para determinar si una clave está en un diccionario, se puede utilizar **in** o **not in**, al igual que en una lista.

```
nums = {
   1:"Uno",
   2:"Dos",
   3:"Tres"
}
print (1 in nums)
print (“Tres” in nums)
print (4 not in nums)
```

```
>>>
True
False
True
>>>
```

Diccionarios

Completar los espacios en blanco para imprimir “**Si**” si una llave **112** está presente en el diccionario llamado “**pares**”.

```
if  112   in   pares   :
    print ("Si")
```

Diccionarios

Un método útil de diccionario es **get**. Hace lo mismo que indexar pero si una clave no es encontrada en el diccionario, devuelve None

```
pares = { 1: "manzana",
   "naranja": [2,3,4],
   None: "True"
}
print (pares.get("naranja"))
print (pares.get(7))
print (pares.get(12345,"No está en el diccionario"))
```

```
>>>
[2,3,4]
None
No está en el diccionario
>>>
```

Diccionarios

¿Cuál es el resultado de este código?

```
fib = {1:1, 2:1, 3:2, 4:3, 5:5, 6:8}
print (fib.get(4,0) + fib.get(7,5))
```

8