Escuela: EPET N° 1 de Caucete.

Docente: Luis Sesé.

Grado: 6to año, ciclo orientado.

Turno: Tarde.

Área Curricular: Practicas profesionalizantes I.

Título de la propuesta: Listas

Listas son otro tipo de objeto en Python. Son utilizadas para almacenar una lista indexada de objetos. Una lista se crea utilizando corchetes con comas separando a los objetos. Un cierto objeto en la lista puede ser accediendo al utilizar un índice dentro de corchetes.

Por ejemplo:

```
words = [“Hello” , “Word” , “!”]
print(words[0])
print(words[1])
print(words[2])
```

La salida seria:

```
>>>
Hello
Word
!
>>>
```

Lista Vacía

Una lista vacía es creada con un par vacío de corchete.

```
Empty_list = [ ]
```

```
print(empty_list)
```

La salida seria:

```
>>>
[ ]
>>>
```

Normalmente una lista contiene objetos de un solo tipo, pero es también posible incluir varios tipos diferentes.
Las listas también pueden ser anidadas dentro de otras listas.

```
number = 3
things = [“string” , 0 , [1 , 2 , number] , 4.56 ]
print(things[1])
print(things[2])
print(things[2] [2])
```

La salida seria:

```
>>>
0
[1 , 2 , 3]
3
>>>
```

Indexar fuera de los límites de los valores posibles de una lista genera un IndexError. Algunos tipos, como las cadenas, pueden ser indexados como listas. La indexación de cadenas se comporta como si estuviese indexando una lista que contiene los caracteres de la cadena. La indexación de otros tipos, tales como enteros, no es posible y ocasiona un TypeError. Ejemplo:

```
str = “Hello Word!”
print(str[6])
```

La salida seria:

```
>>>
w
>>>
```

Operaciones de listas

El objeto en un determinado índice de una lista puede ser reasignados.
Por ejemplo:

```
nums = [7 , 7 , 7 , 7 , 7 ]
nums[2] = 5
print(nums)
```

Las listas pueden ser sumadas y multiplicadas de la misma manera que las cadenas.

Por ejemplo:

```
nums = [1 , 2 , 3]
print(nums + [4 , 5 , 6])
print(nums * 3)
```

La salida seria:

```
>>>
[1 , 2 , 3 , 4 , 5 , 6]
[1 , 2 , 3 , 1 , 2 , 3 , 1 , 2 , 3]
>>>
```

Para verificar si un elemento está en una lista, se puede utilizar el operador **in**. Este devuelve **True** si un elemento está una o más veces en una lista y **False** si no.

```
words = [“spam” , “egg” , “spam” , “sausage”]
print(“spam” in words)
print(“egg” in words)
print(“tomato” in words)
```

La salida seria:

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

Primera actividad:

Determine la salida del siguiente código:

```
nums = [5,4,3,2,1]
print(nums[1])
```

Segunda actividad:

¿Cuántos objetos hay en esta lista? [2,]

Tercera actividad:

Completa los espacios en blanco _ para crear una lista e imprimir el tercer elemento.

```
list = _42 , 55 , 67]
print[list[_])
```

Cuarta actividad:

¿Qué línea de código dará error?

```
num = [5 , 4 , 3 , [2] , 1]
print(num[0])
print(num[3][0])
print(num[5])
```

¿Linea 2? ¿Linea 3? ¿Linea 4?

Quinta actividad:

Determine el resultado de este código:

```
nums = [1 , 2 , 3 , 4 , 5]
nums[3] = nums[1]
print(nums[3])
```

Sexta actividad:

Completa los espacios en blanco _ para crear una lista, reasignar su segundo elemento e imprimir la lista completa.

```
nums = [33 , 42 , 56_
nums[_] = 22
print(____)
```

Séptima actividad:

Determine el resultado de este código:

```
nums = [10 , 9 , 8 , 7 , 6 , 5]
nums[0] = nums[1] – 5
if 4 in nums:
  print(nums[3])
else:
  print(nums[4])
```

Director: Mario Gomez.