TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE

- Variables.
- Scope de variables.
- Tipos de datos: números, decimales, cadenas de caracteres, listas, diccionarios, tuplas, conjuntos y booleans.

VARIABLES

Piensa en una variable como un nombre asociado a un objeto concreto. En Python, las variables no necesitan ser declaradas o definidas de antemano, como ocurre en muchos otros lenguajes de programación. Para crear una sólo tienes que asignarle un valor, y luego empezar a utilizarla. La asignación se hace con un signo de igualdad (=):

```
1 n=100
```

Esto se lee o interpreta como "asigna a n el valor 100". Una vez hecho esto, se puede utilizar en una sentencia o expresión, y su valor podría ser sustituido.

```
1 n=300
2 print(n)
```

Obteniendo como resultado:

```
300
```

Python también permite la asignación encadenada, que hace posible asignar el mismo valor a varias variables simultáneamente.

```
1 a = b = c = 300
2 print(a,b,c)
```

Obteniendo como resultado:

```
300 300 300
```

En el ejemplo anterior se le asignó el valor 300 a tres variables simultáneamente.

SCOPE DE VARIABLES

Es el lugar donde podemos encontrar una variable, y también acceder a ella si es necesario. De esta manera hay variables locales y globales.

Las variables globales son las que se definen y declaran fuera de cualquier función, y no se especifican a ninguna función. Pueden ser utilizadas por cualquier parte del programa.

Una variable local es aquella que es declarada dentro de una función, por lo tanto, existe solo dentro de ella, y no se puede acceder a ésta fuera.

```
#variable global
s = "esta es una variable global"
def f():
        #variable local
s = "esta es una variable local"
```

En el ejemplo arriba escrito hemos declarado dos variables, ambas llamadas "s". La primera fue declarada fuera de la función "f", y la otra dentro de ella.

Si fuera de la función escribimos `print(s)`, el resultado que obtendremos será:

```
Esta es una variable global
```

TIPOS DE DATOS

Python maneja diversos tipos de datos por defecto. Entre ellos tenemos: datos numéricos, lógicos, de tipo cadena, e incluso diversos tipos de estructuras de datos. Esto sin tomar en cuenta los tipos de datos creados por el usuario.

Entre los numéricos tenemos los **números enteros**. Estos son números tanto positivos como negativos que carecen de parte decimal. Pueden ser tan largos como lo permita la memoria del sistema que esté corriendo el programa.

También contamos con los datos de **punto flotante**. Estos son números tanto positivos como negativos que incluyen valores decimales. Dependiendo de su tamaño, pueden ser escritos en notación tradicional o en notación científica, es decir:

```
#4.2 escrito en notación tradicional
4.2
#0.00042 escrito en notación científica
```

```
4 4.2e-4
```

Los datos de tipo **cadena de caracteres** en Python, incluyen cualquier cosa pueda ser definida como una cadena, si al declarar la variable encerramos el texto que deseamos entre comillas, bien sea dobles o simples. La cadena puede estar vacía, siempre y cuando se abran y cierren las comillas.

Con comillas dobles:

```
1 print("Soy una cadena")
```

Resultado:

```
Soy una cadena
```

Con comillas simples:

```
1 print('También soy una cadena')
```

Resultado:

```
También soy una cadena
```

Cadena vacía:

```
1 print("")
```

Resultado:

```
""
```

Los datos tipo cadena pueden tener tantos caracteres como se desee, el limite será impuesto por la memoria del sistema que corra el programa.

Si se desea incluir comillas dentro de la cadena de caracteres, la forma más sencilla de hacerlo es envolver la cadena en las comillas contrarias. Es decir, si deseo incluir comillas sencillas, debo envolver el texto en comillas dobles, y viceversa.

```
1 print("Esto es una comilla sencilla ( ' ) ")
```

Resultado:

```
Esto es una comilla sencilla ( ' )
```

Las **listas** de Python son uno de los tipos de datos más versátiles que nos permiten trabajar con múltiples elementos a la vez. Una lista se crea colocando elementos dentro de corchetes [ ], separados por comas.

```
#Una lista de enteros
mi_lista[1, 2, 3]
```

Una lista puede tener cualquier cantidad de elementos, y estos elementos cada uno puede ser de un tipo distinto, incluso puede contener otra lista.

```
#Otra lista
otra_lista[4, 5.28, "tercer elemento", [1, 2, 3]]
```

Hay varias formas de acceder a los datos de la lista. La más común de ellas es haciendo uso del operador corchetes [ ], con el cual podemos acceder a cada dato según su índice. El índice es la posición en la que está guardado el dato, tomando en cuenta que el índice de la primera posición es cero (0).

Haciendo uso de la lista del ejemplo anterior.

```
print(otra_lista[2])
```

Obtenemos como resultado:

```
Tercer elemento
```

Python permite la indexación negativa para sus secuencias. El índice -1 se refiere al último elemento, -2 al penúltimo elemento, y así sucesivamente.

```
lista2[1, 2, 3, 4, 5]
print(lista2(-1))
```

Resultado:

```
5
```

Otra estructura de datos de uso común en Python es el **diccionario**. Este es una colección desordenada de elementos. Cada elemento de un diccionario tiene una dupla clave/valor.

Los diccionarios están optimizados para recuperar valores cuando se conoce la clave.

Crear un diccionario es tan sencillo como colocar los elementos dentro de llaves { } separadas por comas.

Un elemento tiene una clave y un valor correspondiente, el cual se expresa como un par (clave: valor).

Mientras que los valores pueden ser de cualquier tipo de datos y pueden repetirse, las claves deben ser de tipo inmutable (cadena, número o tupla con elementos inmutables), y deben ser únicas.

```
# Diccionario con claves tipo entero
mi_diccionario = {1: 'manzana', 2: 'pelota'}

# Diccionario con claves mixtas
mi_ diccionario2 = {'nombre': 'Juan', 1: [2, 4, 3]}
```

Para acceder a los datos de un diccionario se utilizan las claves. Las claves pueden usarse dentro de corchetes [ ], o con el método `get()`.

```
mi_diccionario = {'nombre': Daniel, 'edad': 26}

print(mi_ diccionario['nombre'])
```

Resultado:

```
Daniel
```

Otro ejemplo para obtener la edad:

```
print(mi_ diccionario.get('edad'))
```

Resultado:

```
26
```

Una **tupla** en Python es similar a una lista. La diferencia entre ambas es que no podemos cambiar los elementos de una tupla una vez asignada, mientras que sí podemos cambiar los elementos de una lista.

Una tupla se crea colocando todos los elementos dentro de paréntesis ( ), separados por comas. Los paréntesis son opcionales, sin embargo, es una buena práctica utilizarlos.

Una tupla puede tener cualquier número de elementos, y éstos pueden ser de diferentes tipos (entero, flotante, lista, cadena, entre otros).

Tupla con números enteros:

```
# Tupla de numeros enteros
mi_tupla = (1, 2, 3)
```

```
print(mi_tupla)
```

Resultado:

```
(1, 2, 3)
```

Tupla con tipos de datos mixtos:

```
# tupla con tipos de datos mixtos
mi_tupla = (1, "Hola", 3.4)
print(mi_tupla)
```

Resultado:

```
(1, 'Hello', 3.4)
```

Al igual que en las listas, la forma de acceder a los datos de la tupla es haciendo uso del operador corchetes [ ], con el cual podemos acceder a cada dato según su índice. El índice es la posición en la que está guardado el dato, tomando en cuenta que el índice de la primera posición es cero (0).

```
mi_tupla = ('p','e','r','m','i','t')

print(mi_tupla[0])
```

Resultado:

```
p
```

Accediendo al índice 5:

```
print(mi_tupla[5])
```

Resultado:

```
t
```

Y también al igual que en las listas, Python permite la indexación negativa para las tuplas. El índice -1 se refiere al último elemento, -2 al penúltimo elemento, y así sucesivamente.

```
tupla2 = (1, 2, 3, 4, 5)
print(tupla2(-1))
```

Resultado:

```
5
```

Un **conjunto** (set) es una colección desordenada de elementos. Cada elemento del conjunto es único (no hay duplicados), y debe ser inmutable (no puede ser modificado). Sin embargo, un conjunto es mutable, pues podemos añadir o eliminar elementos de él.

Los conjuntos también pueden utilizarse para realizar operaciones matemáticas con ellos, como la unión, la intersección, la diferencia simétrica, entre otros.

Un conjunto se crea colocando todos los elementos dentro de llaves { }, separados por comas, o utilizando la función incorporada `set()`.

Puede tener cualquier número de elementos, y éstos pueden ser de diferentes tipos (entero, flotante, tupla, cadena, entre otros). Pero un conjunto no puede tener elementos mutables como listas, conjuntos o diccionarios.

Conjunto de enteros:

```
# conjunto de enteros
mi_conjunto = {1, 2, 3}
print(mi_conjunto)
```

Resultado:

```
{1, 2, 3}
```

Conjuntos de datos mixtos:

```
# conjunto de tipos de datos mixtos
mi_conjunto = {1.0, "Hola", (1, 2, 3)}
print(mi_conjunto)
```

Resultado:

```
{1.0, "Hola", (1, 2, 3)}
```

Los conjuntos son mutables. Sin embargo, como no están ordenados, la indexación no tiene sentido. No podemos acceder o cambiar un elemento de un conjunto utilizando la indexación. El tipo de datos set no lo soporta.

Podemos añadir un solo elemento utilizando el método `add()`, y múltiples elementos utilizando el método `update()`. El método `update()` puede tomar como argumento tuplas, listas, cadenas, u otros conjuntos. En todos los casos, se evitan los duplicados.

Inicializar **mi_conjunto.**

```
mi_conjunto = {1, 3}
print(mi_conjunto)
```

Resultado:

```
{1, 3}
```

Añade un elemento.

```
mi_conjunto.añadir(2)
print(mi_conjunto)
```

Resultado:

```
{1, 2, 3}
```

Añadir varios elementos.

```
mi_conjunto.actualizar([2, 3, 4])
print(mi_conjunto)
```

Resultado:

```
{1, 2, 3, 4}
```

El tipo **booleano** de Python sólo tiene dos valores posibles: True (verdadero) y False (falso). El tipo bool está incorporado, lo que significa que siempre está disponible en Python, y no necesita ser importado.