CURSO PYTHON

NOTAS ADICIONALES

Se puede añadir a printf un final, siendo que se añade el end= “que se añade”

Python permite la declaración de variables en una sola línea, siendo lo siguiente la sintaxis var1,var2=5,2

Concatenacion de cadenas: Unir dos cadenas o mas, en Python existen diversas formas de hacerlo.

El primero es el uso del operador +, prácticamente lo que hace es que lo agrega una al lado del otro con base en como se acomoden antes o después del operador.

La segunda forma es mediante el método join. Se invoca usando .join, la sintaxis de este método es la siguiente “”.join([cadena1, cadena2,….])

FORMATEO DE CADENAS

Python ofrece varias maneras de formatear cadenas, que incluyen la capacidad de concatenar texto, variables e incluso dar otro tipo de formateo por ej indica el numero de decimales a utilizar

El primer método es f-string: Esta es la opción mas rencomdada por ser la mas sencilla,rápida y recomendada, la sintaxis es f’hola {variable}

El segundo método es format: Es muy versátil y poderoso, permite construir cadenas muy completas, la sintaxis es ‘hola {}’.format(variable)

METODOS DE CADENAS (BASICO)

Las cadenas en Python vienen con una serie de métodos útiles que facilitan su manipulación. Por ejemplo:

Upper: Cambian a mayúsculas las cadenas

Lower: Cambia a minúsculas las cadenas

Strip: Eliminan espacios en blanco al inicio y final de la cadena

Se invocan estos métodos llamando a la cadena y luego el .metodo

OBTENER LARGO DE UNA CADENA

Para obtener la longitud de una cadena utilizamos la función len, su sintaxis seria la siguiente len(cadena)

SUBCADENAS EN PYTHON

Una subcadena es una parte de una cadena principal y hay varias maneras de extraer subcadenas en Python. Se pueden extraer, buscarlas, reemplazarlas, entre oteras operaciones.

* Extracción de cadenas (slicing): Permite indica el indice de inicio y final (sin incluir el ultimo carácter). Su sintaxis seria subcadena=cadena[inicio:final]. El ultimo numero es el índice que no se cuenta ese no se añade, solo es hasta donde se indica.
* Buscar subcadenas (find): El método find deuvelve el índice de la primera apareció de la subcadena. Si no lo encuentra la subcadena devuelve -1. La sintaxis seria posición=cadena.find(“cadena a encontrar”)
* Reemplazar subcadenas(replace): El método replace reemplaza una subcadena por otra dentro de una cadena principal. Su sintaxis es la siguiente nueva\_cadena=cadena.replace(“cadena que reemplaza”,”cadena nueva”)
* Extraer subcadenas por separadores(Split): La función Split permite dividir una cadena en una lista de subcadenas basadas en un carácter separador. Ej de su usoTexto

  Descripción generada automáticamente

La sintaxis de ella es cadena.split(“cadena o elementos que se usa como separador”)

MULTIPLICACION DE CADENAS

Si una cadena se quiere multiplicar varias veces solo se usa la siguiente sintaxis cadena\_multiplicada=cadena\*numero de veces a multiplicar

CONVERSION DE TIPOS DE DATOS

La conversión de tipos de datos, también conocida como casting, es una técnica para manipular datos que no están en el tipo requerido. Podemos hacer conversiones desde y hacia distintos tipos de datos

* Convertir a entero Función int()
* Convertir a flotante funcion float()
* Converitr a cadena funcion str()
* Convertir a booleano funcion bool()

Sintaxis de las funciones conversión=funcion(dato a cambiar)

Nota: En la función bool el valor es 0 cuando es cadena vacia o none, en caso contrario regresa true

ENTRADA DE DATOS POR CONSOLA

En Python la entrada de datos se realiza usando al funcion input(). Esta funcion pausa la ejecución del programa y espera a que el usuario ingrese un texto desde el cadena. Una vez que el usuario presenta enter se devuelve una cadena.

La invocación simplemente es usando input()

Pero si se le quiere añadir mas personalización se puede añadir una cadena a la funcion input ej input(“hola mundo”). Asimismo si queremos que la entrada sea de un tipo de dato especifico tenemos que hacer uso de las funciones de casteo con la de input siendo que se invoca primero a la funcion de casteo primero y dentro de los argumentos de esta al input, ej float(input(“Ingrese los datos”))

GENERACION VALORES ALEATORIOS

La funcion randint() que es parte del modulo random nos permite generar números aleatorios. Su sintaxis es la siguiente randint(a,b) siendo que devuelve un numero aleatorio entre a y b incluyendo los valores correspondientes de estos. Es necesario importar en primer lugar el modulo random antes de la funcion randint . Para su uso se hace lo siguiente

Import random o from random import randint

Dado=randint(1,6)

COLECCIONES

Una coleccion es un conjunto de datos. En Python existen varios tipos que se puede utilizar con el objetivo de almacenar, organizar y manipular múltiples conjuntos de datos, por ello también se les conoce como estructuras de datos. Los tipos mas comunes son: Listas, tuplas, set (Conjuntos) y diccionarios.

LISTAS

Son colecciones ordenadas y mutables de elementos que pueden ser de diferentes tipos. Las listas son dinámicas es decir pueden cambiar de tamaño, añadir modificar o eliminar datos. Su sintaxis es lista=[elemento1,elemento2,elemento3]. Ejemeplo

Números=[5,4,6, “cinco”, “seis”, cadena\_numeros]

También puede haber listas dentro de listas

Lista\_con\_lista=[5,5,[6,8]]

Se acceden a los valores usando los subíndices de la lista, recordemos empieza desde 0, si queremos leerlos de derecha a izquierda simplemente se usan valores negativos siendo que el ultimo elemento también seria el índice -1.

Operaciones con listas

Modificación de un elemento de la lista: Se realiza asignando un nuevo valor al elemento en el índice de esa lista. Ej lista[5]=nuevo\_valor

* Agregar elementos: Existen varios métodos el primero es .append() el cual añade un nuevo valor a la cadena, su sintaxis es lista.append(nuevo\_valor). El segundo método es el método .insert, el cual permite insertar en una posición especifica de la lista, su sintaxis es la siguiente lista.insert(indice\_donde\_seañade,nuevo\_valor), cabe mencionar que los demás valores se recorren una posición si hay mas elementos adelante, es decir el elemento que estaba en la posición 2 ahora esta en la 3.
* Eliminar elementos: Existen varios métodos, el primero que se va a ver es .remove el cual nos permite eliminar un valor específicamente buscado, su sintaxis es la siguiente lista.remove(valor\_eliminar), si hay elementos duplicados se elimina el primero en encontrarse. Si se quiere eliminar por índice se utiliza el método .pop, su sintaxis es la siguiente lista.pop(índice\_del\_elemento\_a\_borrar). Finalmente esta otro método mediante el índice , usando la palabra del (Delete) su sintaxis es la siguiente del lista[índice\_del\_elemento\_a\_borrar]. Recordemos que al eliminar se recorren los elementos así no dejando espacios en vacio.
* Obtener sublista de lista, se puede hacer de forma similar al uso de cadenas indicando los índices con dos puntos de los intervalos de donde se obtienen los elementos, la sintaxis seria algo así sublista=lista[0:5], recordemos que el índice 5 no se considera es decir va de 0 a 4.
* Iteracion: si queremos iterar uns lista podemos hacer un uso de for, siendo que se declara una variable donde se almacena el valor de cada elemento en cada posición y se itera en la lista, es decir seria algo así for nombres in lista, siendo que se va a iterar hasta que se acabe la lista.

TUPLAS

Son similares a la lisras, ya que también son una colección de datos ordenados, pero las tuplas son inmutables lo que significa que una vez creada no es posible modificar su tamaño ni sus elementos. Las tuplas se uselen usar para crear colecciones de datos que no cambian con el tiempo. Su sintaxis es la siguiente tupla= (elemento1,elemento2,elemento3) también se puede hacer sin paréntesis siendo que se puede hacer así tupla=elemento1,elemento2,elemento3

Si se quiere crear una tupla de un elemento solo se hace lo siguiente tupla\_unitaria=elemento,

También puede haber tuplas dentro de tuplas siendo la sintaxis tupla=(5,5,(6,6))

Desempaquetar tuplas: Permite asignar valores a diferentes variables en una sola operación. Siendo que primero se declaran las variables en una sola línea y posteriormente se igualan a la tupla, siendo que se asignan de manera automática de acuerdo al orden de creación e índice de la lista los correspondientes valores.

COMBINACION LISTAS Y TUPLAS

Se puede hacer la creación de listas conteniendo tuplas, siendo que la sintaxis seria la siguiente lis\_tup=[(tupla),(tupla),(tupla)]

SETS

Los sets son colecciones de datos no ordenadas de elementos únicos. Son útiles cuando se debe asegurar que no haya elementos duplicados en la colección. La sintaxis es set={elemento1,elemento2,elemento3}, el orden en que se almacenan lo es seguro pero si hay algún elemento repetido este no se agrega, se descarta. Es decir si se añade 1 otra vez, no se añadira ya que si esta no cumple con que sea único. Aquí no hay índices.

Operaciones con set:

* Agregar elementos: Se hace utilizando el método .add, el cual su sintaxis es la siguiente set.add(nuevo\_valor), se añade y puede variar.
* Eliminar un elemento: Se hace utilizando el método .remove, su sintaxis es la siguiente set.remove(valor\_a\_eliminar)
* Iterar elemento: Se realiza de la firma forma que una tupla o lista. Es decir for elemento in set, iterara sobre cada elemento.
* Comprobar si un elemento existe: Se puede realizar tanto con listas o tuplas, siendo que se hace usando la siguiente sintaxis valor\_cierto= valor\_buscado in set

Operaciones de conjuntos

Union: Su sintaxis es la siguiente unión= set1 | set2, siendo que si se imprime dicho set nuevo será la unión de esos dos, los valores repetidos igual no se repiten.

Intersección: Su sintaxis es set1 & set2, lo que hace es que da los valores similares del set1 y set2.

Diferencia: Su sintaxis es set1-set2 permite quitar los elementos del set2 que se encuentran en el set1.

NOTA: Recuerde en los set no se usa ciclo for para checar la existencia de un elemento sino que se usa el if elemento in set.

AÑADIDO: SI queremos crear alguna colección vacia, se usa las funciones de set(), list(), tuple() y dict() para crearlos.

DICCIONARIOS

Los diccionarios son una colección de datos ordenada. Y los elementos se almacenan en forma de llave:valor o clave:valor. Es una estructura muy utul cuando neceseitan asociar un conjunto de valores con un conjunto de claves o llaves que sirven como índices únicos. Su sintaxis es la siguiente diccionario={clave:valor,clave2:valor}. Cabe mencionar que las claves tienen que ser strings ósea palabras.

Nota: Los diccionarios no permiten agregar elementos duplicados

Para acceder a los elementos de un diccionario mediante su clave de manera independiente, se utilizan dos métodos, la primera mediante la clave que es el índice asociado al valor es decir diccionario[clave]. Otra forma es mediante el método .get el cual su sintaxis es la siguiente diccionario.get(clave)

Operaciones de diccionarios

Modificar valor: Se hace de la forma como se haría en una lista usando la clave para modificar el valor asociado a dicha clave, la sintaxis seria diccionario[clave]=nuevo\_valor

Agregar elementos: Se puede utilizar la sintaxis de diccionario[clave\_nueva]=nuevo dato, siempre y cuando esa clave no este registrada.

Eliminar elementos: Se utiliza la palabra del, siendo al sintaxis del diccionario[clave], eliminándose con ello la clave y su valor asociado. También existe el método .pop siendo que en este caso su sintaxis es la siguiente diccionario.pop(clave)

Iteracion sobre elementos: En este caso cada elemento se considera como una tupla (Clave, valor) por lo que si se itera sobre los elementos se pueden declarar dos valores y ciclar sobre el diccionario, y para obtener los elementos se usa el método .item con sintaxis diccionario.items() donde se hace un unpacking como una tupla. Pero también podemos obtener solo los valores o claves. Para solo los valores se usa el método .values donde se va ir obteniendo los valores de cada tupla correspondiente es decir solamente cada valor, y si solo se quieren obtener las llaves se usa el método .keys de sintaxis diccionario.keys() donde solo se nos devuelve las llaves de diccionario.

Se puede dar la combinación de diferentes tipos de colecciones ej listas con diccionarios, entre otras.

Enumarate en for ayuda a que en cada impresión se enumere cada ciclo.

COMPRESION DE LISTAS

La compresión de listas es una forma concisa y eficiente de crear listar a partir de otros iterables, listas tuplas, etc. Permite filtrar elementos y aplicar expresiones a cada elemento de un iterable de manera muy legible y en una sola línea de código.La sintaxis es [nueva\_expresion for elemento in iterable if condición]

Nueva expresión: Es la expresión que define como se modifica o procesa cada elemento iterable

Elemento: Variable que repsenta cada elemento del iterable

Iterable: Donde se itera

Condición: Es una condición para la filtración. Ejemplo

Números=[1,2,3,4,5]

Pares=[x for x in numeros if x%2==0]

FUNCIONES

Las funciones son bloques de código para realizar una tarea en particular. SE pueden reutilizar siempre.

Pantalla de computadora con letras

Descripción generada automáticamente con confianza mediaImagen de la pantalla de una computadora

Descripción generada automáticamente con confianza baja

Para declarar una funcion se hace la siguiente sintaxis def nombre\_funcion(parámetros)

MANEJO DE PARAMETROS EN FUNCIONES

Cuando declaramos las funciones existen dos formas de poner los parámetros, siendo que la primera es solo escribir el nombre del parámetro, pero si quieremos especificar el tipo de dato a usar se tiene que usar la siguiente sintaxis mensaje:str

Formas de usar parámetros

Def suma(valor1,valor2)

Def suma(valor1:int,valor2:int)

Recordemos que en el cuerpo de la funcion se usan los nombres de los parámetros, no se usa lo que colocamos cuando la invocamos.

MODULOS EN PYTHON

Es un archivo con la definición de variables y funciones. Es una biblioteca como en C. Para crear estas se crea un archivo de extensión py y apartir de ella solo se coloca lo que quiere que almacene esta. En dado caso de querer usarla solo se usa el import nombre\_archivo y también se puede usar el from nombre\_archivo import funcion. Si en el modulo queremos hacer una prueba de la funcion se añade el siguiente línea

If \_\_name\_\_ == “\_\_main\_\_” y luego la prueba de la funcion, la ejecución respectiva que se quiere realizar.

ARGUMENTOS POR NOMBRE

Si queremos pasar los argumentos por así decirlo en desorden a la hora de la invocación pero que se coloquen de forma correcta al ya pasarse a la función, lo que se hace es que cuando llamamos a la funcion se tiene que seguir la siguiente sintaxis

Funcion(nombre\_parametro= valor, nombre\_parametro= valor)

Pero que pasaría si queremos que haya valores por default cuando invocamos a la funcion, ósea cuando solo queremos pasar un parámetro y los otros estén por default, en dicho caso tenemos que modificar la declaración de la funcion siendo que en la declaración se haría así def funcion(parámetro= valor\_defualt, parametro2= valor\_default2)

REGRESAR UNA TUPLA DE VALORES DESDE UNA FUNCION

En este caso lo que se realiza es que en el return se colocan los valores de retorno separándolos por comas, en otro caso también podrían los valores envolverse en un paréntesis pero recordemos que esto es opcional. La sintaxis quedaría algo así:

Return (parametro1,parametro2,parametro3)

ALCANCE DE VARIABLES

Las variables pueden tener un alcance global o local dependiendo de donde y como se declaren. Las variables que globales son aquellas que están disponibles a lo largo de todo el programa mientras que las locales solo están disponibles dentro del bloque de código o donde fueron declaradas. Para visualizar esto se puede ver en el tap de las líneas de código siendo que generalmente las variables globales no tienen tap, y una local de una funcion se encuentra dentro de esta. Si queremos usar una variable global en una funcion se utiliza la siguiente sintaxis dentro de esta global nombre\_variable, con ello ya podremos utilizarla en la funcion.

ARGUMENTOS VARIABLES

En Python los argumentos variables nos permiten que una funcion acepte un numero cambiante de elementos. Hay dos tipos principales

Argumentos posicionales variables \*arg: Permiten pasar múltiples argumentos posicionales a una funcion recibiéndolos como una tupla dentro de la funcion.

Argumentos con palabra clave \*\*kwargs: Recibe los argumentos en forma de diccionario. Llave-valor.

\*ARGS (Arguments)

Se colocan en la declaración de la funcion después de colocar los parámetros bases, es decir los que sabemos que son solo uno. Ej def suma(valor1,valor2, \*args)

Cuando se usan los argumentos adicionales se reciben como una tupla, ademas de que ya no en el cuerpo de la funcion se coloca el \* sino solo el nombre de los parámetros. En el caso de usar esto se puede incluso no enviar valores, pero recuerde los \*args se ponen al final de la declaración de los parámetros.

KWARGS (keyword arguments)

Se colocan al final final de la declaración incluso si hay args de la funcion siendo la sintaxis: def funcion (parámetro,\*args,\*\*kwargs). Y de igual forma a args no se usa el \*\* dentro del cuerpo de la funcion y cuando se invoca la funcion simplemente sigue la siguiente forma funcion(valor1, valores adicionales…., clave=valor,clave=valor). Se pueden dejar vacio igual este parámetro. Aquí los datos se envían en forma de diccionario.

FUNCIONES RECURSIVAS

Es una funcion que se llama a si misma. Tienen reglas para trabajar con ellas siendo la primera que se llama a si misma siempre, y la segunda es que se debe de avanzar siempre poco a poco hacia un caso base para no caer en un ciclo infinito.

LIMPIAR CONSOLA

Para limpiar la consola en Python, puedes usar diferentes comandos dependiendo del sistema operativo en el que estés ejecutando tu script. Aquí te muestro cómo hacerlo:

**En Windows**

Puedes usar el comando **cls**:

python

VerifyOpen In EditorRunCopy code

1import os

2

3os.system('cls')

CLASES Y OBJETOS

Un objeto es una representación de una entidad de la vida real en nuestro programa. Para crear un objeto primero necesitamos crear una clase o plantilla. Una clase representa las características en común de los objetos. Si queremos construir un edificio primero se quiere una clase y apartir de esa clase se crean varios objetos.

ELEMENTOS DE UNA CLASE

Se compone de atributos y métodos.

Los atributos son las características de nuestros objetos. Los métodos son las acciones que pueden realizar nuestros objetos. En si las acciones son funciones pero cuando se asocian con una clase se les llama métodos. Una vez que se define una clase se pueden crear objetos a esto se le llama instanciar una clase.

La sintaxis de una clase:

Clase persona:

Atributos: Nombre, email, celular

Métodos: Agregar nombre, mostrar persona

Imagen que contiene Diagrama

Descripción generada automáticamente

La sintaxis para crear una clase es la siguiente

Class nombre\_clase:

Def nombre\_metodo(self,parametro1,parametro2):

Self.nombreAtributNuevo=parametro1

Self\_nombreAtribut\_Nuevo=parametro2

#CREA LOS METODOS

Def mostrar\_metodo(self):

Printf(f’self.nombre’)

Self: sirve para acceder al atributo de esa clase.

CREACION DE OBJETOS

Para crear a un objeto, se define una variable y posteriormente se iguala a el nombre de la clase añadiendo un paréntesis después, La sintaxis seria algo así variable=Clase(), creandose así un objeto vacio en memoria, una vez ya inicializado se pueden acceder a los atributos y métodos de la clase, es decir ya podemos trabajar con el objeto y hacer uso de las funciones como en el anterior ejemplo de inicializar los valores. En los métodos aunque como parametro esta el self este es como si no existiera, ya que Python lo pasa en automático.

CONSTRUCTORES EN PYTHON

Es un método especial que se utiliza para crear un objeto. Sirve para crear e inicializar los atributos de un nuevo objeto. Su sintaxis es:

Class nombre\_clase

Def \_\_init\_\_(self,parametro1,parametro2):

Self.parametro1=parametro1

Self.parametro2=parametro2

Init=inicializar, se conoce como método mágico o de tipo dunder, doble guion bajo. Realizan varias tareas porque parece que hacen magia. Un ejemplo de este es: Imagen que contiene texto, grande, firmar, azul

Descripción generada automáticamente

Estos se mandan a llamar de manera automática en cuanto se crea un objeto de clase, por lo que para utilizarlos al crear un objeto se requiere en el paréntesis que se encuentra después de la clase añadir los parámetros de ese constructor.

DIRECCION DE MEMORIA DE OBJETOS

Si queremos saber la dirección de memoria cuando estamos con objetos se utiliza la funcion id devolviéndonos el valor numérico de la dirección de memoria de este, pero si queremos el hex hacemos uso del casteo hex(id(variable)))

FUNCIONES DE LIMPIADO Y PARADO DE CONSOLA

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

FLEXIBILIDAD DE CONSTRUCTORES

Si en el constructor no queremos meter parámetros al crear un objeto. En Python tenemos la oportunidad de no tener sobrecarga de constructores es decir poner varias veces \_\_init\_\_ tenemos problema. Para solucionar ello podemos hacer uso de valores default al declarar los valores, de esta manera nos ahorramos el problema de no querer meter ningún valor. Y si queremos usar los demás métodos y no pusimos valores validos solamente accedemos a los atributos del objeto y los inicializamos.

ENCAPSULAMIENTO

Nos permite ocultar la información que almacena un objeto también conocido como el estado del objeto. Para aplicar el concepto del encapsulamiento se deben aplicar las dos características

1. Atributos protegidos o privados: Se sigue la siguiente sintaxis
   1. Self.\_atributo ----atributo protegido
   2. Self.\_\_atributo -----atributo privado
2. Crear los métodos conocidos como get (leer) y set(modificar)
3. Entre protegido y privado depende de nuestro proyecto

Permite acceder a los valores de los estados los valores de los atributos.

Diagrama, Esquemático

Descripción generada automáticamente

Los métodos get y set se trabajan afuera de la clase, no dentro de ella. Si queríamos acceder a un atributo privado o protegido ya no se da la ayuda de acceder a estos en el código programándolo, aunque si deja Python no es correcto por lo que se debe de hacer sus correspondientes funciones get y set. Si hiciéramos en un protegido la modificación si se haría pero si esta en privado ya no deja hacerlo pues es como si fuera constante, si quisiera modificar podría ser objeto.\_clase\_\_atributo.

Pero ya al hacer los métodos o usar los atributos se requiere de que a ellos se acceda poniendo también su guion correspondiente es decir si queremos imprimir o leer algo tenemos que poner el self.\_atributo o self.\_\_atributo. Igual en los métodos pues se hace ello para cambiarlos o modificarlos.

MEJORAS DE ENCAPSULAMIENTO

Para mejorar las cuestiones de encapsulamiento y que se vea mejor el código, en lugar de definir el método get y set de manera explicita se utiliza el decorador de @property para indicar que ese atributo es una propiedad de nuestra clase y luego se define el método sin utilizar get\_ y solo el nombre del atributo al que se accede y luego ya el funcionamiento de ello. Esto permite que se vea mejor siendo que es como si accediéramos a la propiedad de la clase y en el de set hacemos uso de @nombreAtributoAModificar.setter y se vuelve a definir la funcion solo colocando el atributo que se modifica.

* Get: @property def atributo
  + Se invoca solo poniendo objeto.atributo
* Set: @atributo.setter def atributo
  + Se invoca solo poniendo objeto.atributo= nuevo\_valor

En Python, el decorador @property se utiliza para definir métodos que actúan como accesores (getters) de atributos, pero permiten usarlos de una forma más natural y pythonica, como si fueran atributos en lugar de métodos. Esto mejora la encapsulación y el control sobre cómo se acceden y modifican los atributos.

El decorador @marca.setter se usa para definir cómo se debe modificar el atributo (en este caso, self.\_marca).

ATRIBUTOS DINAMICOS

Para agregar un atributo a un objeto de manera dinámica ósea solo a ese objeto, se usa el método setattr(), siendo que su sintaxis es setattr(objeto, nuevo\_atributo, valor\_new\_atributo) pero tiene inconveniente pues los métodos ya establecidos no trabajaran con este. O también se puede añadir con objeto.nuevo\_atributo=valor.

Si queremos preguntarle los atributos de un objeto a Python se usa el atributo mágico \_\_dict\_\_ el cual nos dira que atributos tiene dicho objeto.

ATRIBUTOS DE CLASE Y ESTANCIA

Atributo de clase: Se asocian a la clase en si misma. Cualquier mod sobre esta clase se vera reflejada en cualquier objeto. Para declararse se hace fuera de cualquier método sino en la propia clase. Ademas para acceder a ellos se usa la siguiente sintaxis clase.atributo pero también se puede desde un objeto usando la sintaxis de objeto.atributo\_de\_clase

Atributo de instancia: Se asocian a los objetos en si mismo. Cualquier mod sobre este solo se hace en el objeto dado. Se hacen declaran dentro de un método principalmente en el \_\_init\_\_

METODOS DE CLASE

Se pueden crear métodos para asociarse con la clase en si misma. Recordemos método de estancia trabaja con los atributos de estancia. Para trabajar con los de clase existen dos métodos.

@staticmethod: Su sintaxis es @staticmethod def nombre\_metodo() return clase.atributo y su invocación es clase.nombre\_metodo().

@classmethod: Su sintaxis es @classmethod def nombre\_metodo(cls) return cls.atributo y su invocación es clase.nombre\_metodo()

cls hace referencia a los atributos de clase

NOTA ADICIONAL: Cuando se traba de atributos de clase o métodos de clase se asocian al contexto estático y cuando ya se crean objetos de la clase es el contexto dinámico. Los objetos si pueden acceder al contexto estático pero no al revés.

