Programación avanzada.

7 de septiembre:

Tema 1: Manejo de errores:

El manejo de errores en Python se inventó con la finalidad de que el programa pueda continuar sin necesidad de que explote (o sea un error que detenga todo el código), uno de los ejemplos más comunes seria la división por 0, transformar un string a entero (si es texto).

Para evitar este tipo de cosas, entran estas funciones claves:

Try: acá ponemos el error que puede fallar.

Except: Que hace si el error ocurre.

Else: Que hacer si NO hubo error.

Finally: Que hacer siempre (con o sin error).

Con todo esto podemos armar códigos que sean útiles, aunque intentemos hacer que fallen, pero debemos tener en claro todo tipo de cosas que puedan suceder, o sea, tener presente todos los errores que puedan existir, si en dado caso existe uno y falla, se llama BUG, y pues tenemos que cambiarlo.

Un ejemplo es este:  
try:

x = int("hola") # Esto va a fallar porque "hola" no es número

except ValueError:

print("Error: no se puede convertir a número")

else:

print("Todo salió bien")

finally:

print("Esto siempre se ejecuta")

De esta manera ya tenemos la base para empezar con manejo de errores de manera formal y fija.

Tema 2: Bucles con manejo de errores: brake, cotinue y pass.

Brake:  
Esta función sirve para cuando sucede algo y tenemos que romper el código inmediatamente, ejemplo no pusimos en el bucle un manejo de errores porque eso sería algo bastante complicado de implementar, entonces rompemos el bucle de una vez y así podemos manejar el error de forma más general.

Nos puede servir de mucha ayuda con try/excetp.

Continue:  
Sirve para saltar la iteración actual del bucle y pasar directamente a la siguiente, esto no rompe el bucle solo ignora el resto del código en la vuelta que este haciendo.

Nos puede servir con try/excetp.

Pass:  
No hace nada, es una instrucción vacía y ya, esto puede ser útil si no tenemos código aun implementado en esa parte, pero queremos que no se rompa el bucle ahí.

Lo podemos usar en funciones, clases o condiciones que todavía no hemos implementado.

Muy útil durante el desarrollo de algo, pero después ya es mejor quitarlo.

Nos puede servir con try/except/finally.

¿Cómo lo podemos conectar con el manejo de errores que vimos arriba?  
Se puede combinar con programar más robustos.

Ejemplos:

* Brake: salir del bucle si ocurre un error con try.
* Continue: saltar a la siguiente iteración si se detecta un error, pero hay que seguir con el bucle.
* Pass: dejar preparado un bloque de except para manejar errores después.
* Finally: siempre hace algo.

Tema 3: Funciones built – in, útiles en Python.

* Len(): Da la longitud de un objeto, ya sea lista, string, tupla, diccionario, etc.
* Sum (): Devuelve la suma de los elementos numéricos de una lista o interables.
* Bin(): convierte un numero entero a binario (ejemplo: bin(10) = “0b1010”)
* Hex (): convierte un numero entero a hexadecimal (ejemplo: hex(255) = “0xff”)
* Enumerate (): Añade un contador automático al recorrer un iterable, ejemplo: for i, v in enemumarate ([“a”, “b”,…]): …
* Zip(): combina 2 o mas listas en pares, regresa siempre una tupla uniendo los mismos elementos, muy útil en for. Ejemplo: zip ([1,2], [“a”,” b”]) = [(1,” a”), (2,” b”]).
* Map(): aplica una función a todos los elementos de un iterable, ejemplo: map(str.upper, [“hola”, “mundo”] = [“HOLA”, “MUNDO”]).
* Filter (): filtra los elementos de un iterable en una forma booleana, ejemplo: filter(lambda x: x>0, [-2,-1,0,1,2]) → [1,2]
* Sorterd (): devuelve una nueva lista ordenada a partir de cualquier iterable, puede ordenar desde números hasta strigns, tiene parámetros como reverse 0 true o key= para personalizar. Ejemplo: sorted([3,2,1]) → [1,2,3]

Tema central e importante:

Programación orientada a objetos

En programación el concepto de clases funciona como un molde, donde entrara nuestro objeto, ¿Qué es un objeto?, un objeto es una instancia de esa clase, ejemplo: persona (Luis, 19), el primer elemento haciendo referencia al nombre y el segundo a su edad, aquí ya tendríamos creado el objeto.

Cosas importantes para tener en cuenta:

* Atributos: variables que solo pertenecen al objeto.
* Métodos: funciones dentro de la clase que actúan sobre esos atributos.
* \_\_int\_\_: constructor base, inicializa los atributos cuando se crea un objeto.
* Self: referencia al propio objeto (esto nos sirve como ejemplo para java, cuando veamos el método this).

Ejemplo base:

class Persona:

def \_\_int\_\_(self, nombre, edad):

self.nombre = nombre

self.edad = edad

def saludar (self):

return f”Hola, soy {self.nombre} y tebgo { self.edad} años.”

Ahora, dentro de POO las funciones que vimos arriba tienen más sentido.

* Len()
* Sum()
* Enumerate()
* Zip()
* Sorted()