**Python**

Python es un programa de programacion utilizado para diferentes tareas. Fue creado por Guido van Rossum, y lanzado en 1991.

Es usado para:

* Desarrollo web (servers).
* Desarrollo software.
* Matemáticas.
* Scripting del sistema.

**\***En visual studio code para abrir una ventana interactiva de python debemos colocar **# %%** al inicio del código, y de esta forma se creara una celda interactiva.

**\***El **INDENT** en Python es muy importante y forma parte de la funcionalidad de tal, sin este no correría de la forma que debería. Ejemplo:

**Funciona**

if 5 > 2:  
  print("5 es mas grande")

**No funciona**

if 5 > 2:  
print("5 es mas grande")

De la misma forma el código bajo la anterior línea tiene que estar a los mismos espacios que el anterior o sino no funciona. Ejemplo:

**Funciona**

if 5 > 2:  
  print("5 es mas grande")

print("Hola Mundo")

**No funciona**

if 5 > 2:  
 print("5 es mas grande")

print("Hola Mundo")

**\***Los comnetarios se establecen con **#**. Dado que python no tiene una forma de comentar codigo en muchas líneas se puede usar:

**“””**

**This is a comment**

**“””**

Dado que python ignora todo texto no asignado a variable.

Funciones Principales

**print(“**text**”** or **variable)**: Imprime el valor o texto que le hayas ingresado.

Variables

En python las variables no necesitan ser declaradas y solo pueden ser escritas de esta forma:

**X = 1**

**Y = 2**

**--**Los nombres de las variables no pueden empezar con un numero, no pueden tener un **-** y son case sensitive, es decir **HOLA** y **hola** pueden ser diferentes variables.

**--**En este lenguaje se pueden declarar varias variables al mismo tiempo de la siguiente forma:

**x, y, z = “hola”, “adiós”, “bienvenido”**

De esta manera la primera variable tiene el primer valor, la segunda el segundo y así siguiendo una relación aritmética.

**--**Además varias variables pueden tener el mismo valor:

**x = y = z = “Hola”**

**\***Todas contienen **“Hola”**

**--**Las variables de texto y el texto en si se concatenan con un **+**. Esto también puede utilizarse con dos variables que contengan texto. Ejemplo:

**x = “hola”**

**y = “mundo”**

**z= x + y**

**print(z)**

Resultado: **“Hola mundo”**

**--**Las variables dentro de funciones no son globales y por lo tanto solo pueden ser referidas dentro de ellas. Para hacer una variable global en una función se debe usar el comando **global**.

def myfunc():  
  global x  
  x = "fantastic"  
myfunc()

**\***Si una variable local tiene el mismo nombre que una global, no la va a sobrescribir fuera de su función. Esto cambia cuando se usa **global** dentro de la función.

Data types

Existen varios tipos de datas en python:

Text type = str

Numeric type = int, float, complex

Sequence type = list, tuple, range

Mapping type = dict

Set type = set, frozenset

Boolean type = bool

Binary type = bytes, bytearray, memoryview

Dado que no se necesita especificar que datos se colocan al completar una variable, tales se identifican manualmente cuando se crean:

String:

**x = “Hello world”**

Int:

**x = 1**

Float:

**x = 1.5**

Complex:

**x = 1j**

List:

**x = [“1”, “2”, “3”]**

Tuple:

**x = (“1”, “2”, “3”)**

Range:

**x = range(5)**

Dict:

**x = {“name” : “Antonio”, “lastname” : “Glorioso”}**

Set:

**x = {“1”, “2”, “3”}**

Frozenset:

**x = frozenset({“1”, “2”, “3”})**

Bool:

**x = true**

Bytes:

**x = b”hello”**

Bytearray:

**x = bytearray(5)**

Memoryview:

**x = memoryview(bytes(5))**

**\***Se puede saber que tipo de dato es un valor o una variable con **type(x)**.

Numbers

Existen tres tipos de números en python: int (enteros), float (decimales o con e), complex (complejos).

Podemos transformar números de una de estas categorías a otra (menos los complejos)

**#convert from int to float:**

**x = float(1)**

**#convert from float to int:**

**y = int(2.8)**

**#convert from int to complex:**

**z = complex(x)**

**\***Estas funciones también cambian strings de números a sus respectivas clases.

**Funciones o Imports especiales**

**Import random**

**print(random.randrange(n1, n2))**

Nos imprime un número random desde el **n1** al **n2**.

Strings

Strings de una sola línea se establecen con dos **“**text**”**. Strings de muchas líneas se establecen con seis **“””**text**“””.**

Como vimos en las variables los trings se concatenan con un **+**. Para agregar un espacio entre ellos se puede poner **+ “ ” +**.

Las strings de python se comportan como arrays, teniendo cada carácter un numero al cual podemos ingresar. De tal forma:

**a = “hello”**

**print(a[0])**

Nos devuelve **h**.

Tambien podemos hacer un slice del texto con:

**print(a[0:3])**

Esto nos devolvería **hel** dado que el **3** no esta incluido.

**\***Estos números puedenser negativos y la cuenta va de derecha a izquierda:

**print(a[-3:-1])**

Nos da como resultado **ll**, **-3** no incluido.

Dado que no se pueden concatenar números y string se utiliza el método **format()**. Este toma en el string los valor **{}** y los remplaza por los números que querramos. Ejemplo

**a = “Tengo {} años”**

**b = 17**

**print(a.format(b))** #Devuelve “Tengo 17 años”

\*Esto se puede hacer con cuantas variables y números quieras aplicándose en una regla de fila, es decir **a** en **format(a)** con 1er **{}**, **b** en **format(a, b)** con 2do **{}**, etc.

\*Se puede usar un placeholder dentro de los **{}** para indicar que variable se les asigna, es decir **{2}** iria con el **c** en **format(a, b, c)** dado que el **a** comienza por 0.

**Insertar caracteres ilegales en string**

Estos caracteres nos ayudan a poder insertar caracteres o espacios ilegales en un string normal. Ellos son:

**\”** Coloca una doble comilla.

**\’** Coloca una simple comilla.

**\\** Coloca un solo \.

**\n** Divide el string en otra línea.

**\t** Coloca un espacio de tab.

**\b** Elimina un espacio de tab.

**\ooo** Escribir texto con valores octagonales.

**\hex** Escribir texto con valores hex.

**Funciones especiales**

**varN = “example”**

**len(varN):** Devuelve la longitud de un string.

**str(varN)** Cambia un número a una string.

**varN.strip():** Remueve cualquier espacio en blanco al inicio de la string.

**varN.lower():** Cambia la string a lowercase.

**varN.upper():** Cambia la string auppercase.

**varN.replace(“**str**”, “**new str**”):** Cambia la string indicada por una nueva (puede ser un solo caracter).

**varN.split (“**str**”):** Separa el string en el caracter o string que le indiquemos, y luego borra tal.

**in** or **not in**: Estos métodos nos dejan saber si un string existe en tal otro string. Ejemplo:

**a = “hola mundo”**

**b = “mundo” in a**

**print(b)** #Nos devuelve true.

**a = “hola mundo”**

**b = “mundo” not in a**

**print(b)** #Nos devuelve false.

**\***Ver <https://www.w3schools.com/python/python_strings.asp> para más métodos o funciones

Listas

Hay cuatro tipos de datos de recopilación en el lenguaje de programación Python:

* **List** es una colección ordenada y modificable. Permite miembros duplicados.
* **Tuple** es una colección ordenada e inmutable. Permite miembros duplicados.
* **Set** es una colección que no está ordenada ni indexada. No hay miembros duplicados.
* **Diccionario** es una colección desordenada, modificable e indexada. No hay miembros duplicados.

Se pueden acceder a sus valores con **[]**. Estos empiezan por 0 si va de izquierda a derecha, si es al revés comienzan por -1.

Podemos tener varios valores con la función **[2:4]** Se toma el 2 y el 3 pero se deja afuera el 4. Si no colocamos el primer número **[:4]** tomara todos hasta el 4 (no incluido), y si no colocamos el ultimo **[2:]** tomara todos desde el 2 hasta el último.

**Funciones o metodos de listas**

**varN = [“x0”, “x1”]**

**len(varN):** Retorna la longitud de la lista.

**varN.append(“x2”):** Ingresa un nuevo valor a la lista en el último lugar.

**varN.insert(n°, “x3”):** Inserta un valor a la lista en la posición elegida.

**varN.remove(“x0”):** Elimina el valor especificado.

**varN.pop(n°):** Elimina el valor según su index. Si no se especifica elimina el ultimo.

**del varN[n°]:** Elimina un valor según su index.

**del varN:** Elimina la lista completamente.

**varN.clear():** Limpia la lista.

**varN2 = varN.copy():** Copia la lista a una nueva variable.

**varN3 = varN2 + varN:** Junta dos listas en una misma creando una nueva variable.

**varN.extend(varN2):** Incluye los valores de **varN2** al final de la lista **varN**.

**varN.update(varN2):** Incluye los valores de **varN2** al final de la lista **varN**. Solo funciona con **sets** (elimina los duplicados)

**Tuples**

Los valores de los **tuples** no pueden ser cambiados ni tampoco se puede agregar otros. Se puede cambiar el **tuple** a una **list** para poder hacer cambio de sus valores y luego volver a hacerlo un **tuple**.

**Sets**

Dado que los **sets** no tiene un index ordenado debes recorrerlos con un **for**:

**for x in varN:**

**print(x)**

Tampoco puedes cambiar sus valores pero si agregar nuevos con **varN.add(“x3”)** o **varN.update(“x4”, “x5”)** si deseas poner varios.

**Dict**

Para acceder a los valores de un **dict** debemos referirnos a sus key values, es decir **varN[“key1”]** retornara el **value1** de **ke1**. Se puede recorrer sus valores con un **for** pero solo dara sus keys, para aceder a sus valores tendremos que:

**for x in varN:**

**print(varN[x])**

Para tener sus dos valores esta:

**for x, y in varN.items():**

**print(x, y)**

Para agregar nuevos valores podemos:

**varN(“keyN”) = valueN**

Condicionales

**If and else**

El **if** nos sirve para ver si tal declaración es verdadera y realizar una acción determinada. El **else** se realiza cuando la declaración es falsa.

Python utiliza el **elif** para realizar otra declaración que no haya sido verdadera en el **if**.

**and** es para realizar dos o más declaraciones y se ejecute mientras todas ellas sean verdaderas.

**or** es para realizar dos o más declaraciones y se ejecute mientras una de ellas sea verdadera.

Short Hand **If** and **If else**

if a > b: print("a is greater than b")

a = 2  
b = 330  
print("A") if a > b else print("B")

\*Los **if** no pueden ir vacios pero el método **pass** nos deja realizar tal función.

Loops

**While Loop**

El **while loop** se utiliza para realizar repetidamente una acción concreta. Este se establece con:

**varN = n°**

**while varN >/</= n°:**

**//Code**

**varN += 1**

\*Este loop contiene la declaración **break** que la podemos utilizar para romperlo cuando suceda algo (por ejemplo con un **if**)

**varN = n°**

**while varN >/</= n°:**

**//Code**

**if varN > n°:**

**//Code**

**break**

**varN += 1**

\*Este loop contiene la declaración **continue** que la podemos utilizar para evitar una iteración cuando llegue a tal numero (por ejemplo con un **if**)

**varN = n°**

**while varN >/</= n°:**

**//Code**

**if varN = n°:**

**continue**

**varN += 1**

#El cogido evitara tal numero al llegarlo, pero seguirá con sus iteraciones.

\*Este loop contiene la declaración **else** que la podemos utilizar cuando la condición de while sea falsa

**varN = n°**

**while varN >/</= n°:**

**//Code**

**varN += 1**

**else:**

**//Code**

**For Loops**

Un bucle **for** se utiliza para iterar sobre una secuencia (que es un **list**, un **tuple**, un **dict**, un **conjunto** o una **string**).

Esto es menos parecido a la palabra clave **for**en otros lenguajes de programación y funciona más como un método iterador como se encuentra en otros lenguajes de programación orientados a objetos.

Ejemplo:

**fruits = ["apple", "banana", "cherry"]  
for x in fruits:**

**print(x)**

#Imprimira cada element en **fruits**

\*Los **for loops** contienen las mismas cualidades que los **whiles**, con sus mismos funcionamientos: **break, continue, else**.

\*En el **for loop** podemos encontrar una nueva función que nos ayuda a imprimir cierta cantidad exacta de cosas, el **range()**. Su funcionamiento es:

-Si le colocamos un número solo, este ira desde 0 hasta un número antes del que pusimos.

-Si le colocamos dos, este ira desde el primer número, contándolo, hasta uno antes del segundo número que colocamos.

-Si le colocamos tres, tendrá el mismo funcionamiento que el segundo, pero su suma (que normalmente es de +1) será del número que hayamos puesto.

\*Puede haber **for** en **for** haciéndolo un **nested loop**.

Funciones

Una función es un bloque de código que solo se ejecuta cuando se llama. Puede pasar datos, conocidos como parámetros, a una función. Una función puede devolver datos como resultado. Esta se escribe:

**def funN():**

**//codigo**

Y se llama:

**funN()**

Puede tener parámetros que luego serán rellenados por sus argumentos y que la función procesara:

**def funN(numN):**

**result = numN + 1**

**return result**

**funN(2)**

#Retorna 3

\*Estos parámetros pueden ser más de 1, pero sus argumentos deben estar de acuerdo a la posición que están en los parámetros.

\*Los parámetros pueden tener un valor estándar, que se utilizara si no colocamos nada al llamar la función.

**def funN(numN = “1”):**

**result = numN + 1**

**return result**

**funN()**

#Retorna 2

\*Se puede colocar el parámetro dentro del argumento, así la colocación de estos no importa en cuanto a similitud de espaciamiento:

**def funN(numN, numN2, numN3)**

**result = numN + 1 \* numN2 / numN3**

**return result**

**funN(numN3 = 3, numN = 4, numN2= 1)**

\*Python permite la recursividad dentro de sus funciones, haciendo posible poder llamarla dentro de esta:

**def tri\_recursion(k):  
  if(k > 0):  
    result = k + tri\_recursion(k - 1)  
    print(result)  
  else:  
    result = 0  
  return result  
  
print("\n\nRecursion Example Results")  
tri\_recursion(6)}**

#Esto imprimirá: 1, 3, 6, 10, 15, 21. Esto dado a que cuando llamamos al 6, se llama la función al 5, después al 4 y así hasta el 1. Cuando ya no puede recursar mas (dado que el parámetro es 1 – 1 = 0, evitando el **if**), se imprime el 1. Así se van imprimiendo cada otra en forma de cadena, como un efecto domino, sumándose el resultado de la anterior a sí misma (1 + 2 = 3 + 3 = 6 + 4 = 10 + 5 = 15 + 6 = 21).

**Lambda Funciones**

Las Lambda son funciones recortadas o resumidas cuando se utilizan expresiones minimas. Ejemplo:

**x = lambda a : a + 10  
print(x(5))**

Esta puede tener mas de un argumento:

**x = lambda a, b : a \* b  
print(x(5, 6))**

Las lambda functions nos sirven para hacer funciones dentro de funciones. Es decir tenemos más espacio para el libre manejo y el re uso.

Ejemplo:

**def myfunc(n):  
  return lambda a : a \* n  
  
mydoubler = myfunc(2)  
  
print(mydoubler(11))**

Gracias a su fácil manipulación podemos modificarla para que triplique el número, en vez de duplicarlo:

**def myfunc(n):  
  return lambda a : a \* n  
  
mydoubler = myfunc(3)  
  
print(mydoubler(11))**

Class

Las clases son creadores de objetos que podemos manipular con metodos y funciones. Estos se definen con:

**class MyClass:  
  x = 5**

Y se crea el objeto al asignarle un nombre:

**p1 = MyClass()  
print(p1.x)**

\*Podemos cambiar o eliminar valores del objeto ingresando a través de sus llaves:

**p1.x = 1**

**del p1.x**

**\_\_init\_\_() Function**

La función ­­**\_\_init\_\_()** se establece al inicio de una clase, y sirve para que cuando queramos crear un objeto este sea mucho más manipulable en cuanto a sus valores y llaves. Este se utiliza en esta forma:

**class Person:  
  def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
  
p1 = Person("John", 36)  
  
print(p1.name)  
print(p1.age)**

\*El parametro self es la instancia propia del objeto, y que cuando se crea, toma el valor que le hayamos nombrado (en este caso p1).

\*Los parámetros pueden tener el nombre que quieras, pero el primero siempre se utiliza para darle valor a las keys.

**Metodos en clases**

Los métodos enals clases son funciones colocadas dentro de estas, diferenciadas de **\_\_init\_\_()**

Ejemplo:

**class Person:  
  def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
  
  def myfunc(self):  
    print("Hello my name is " + self.name)  
  
p1 = Person("John", 36)  
p1.myfunc()**

**\***Siempre primero debemos crear el objeto y luego llamar la función a través de la llave p1.

**Herencia**

En python podemos objetos a otros, convirtiéndolos así en hijos del primero. Esto nos sirve para poder crear diferentes acciones o métodos reutilizables y diferentes entre sí.

Primero lo que obviamente debemos hacer, es crear la clase padre (esta posee un método propio):

**class Person:  
  def \_\_init\_\_(self, fname, lname):  
    self.firstname = fname  
    self.lastname = lname  
  
  def printname(self):  
    print(self.firstname, self.lastname)**

Luego debemos crear la clase secundaria, utilizando en el parametro el nombre de la clase primaria

**class Student(Person):  
  def \_\_init\_\_(self, fname, lname, year):  
    super().\_\_init\_\_(fname, lname)  
    self.graduationyear = year**

**x = Student("Mike", "Olsen", 2019)**

\*Debemos colocar el \_\_init\_\_() dentro de la misma para poder modificar sus parámetros.

\*La función super() se utiliza dado que al crear el anterior init estamos borrando todo el contenido del padre, por lo que de esta forma se pasa su totalidad.

Así cuando creamos un objeto llamamos a la clase hijo que usa sus métodos, llama a la clase padre y esta a su vez utiliza sus propias funciones.