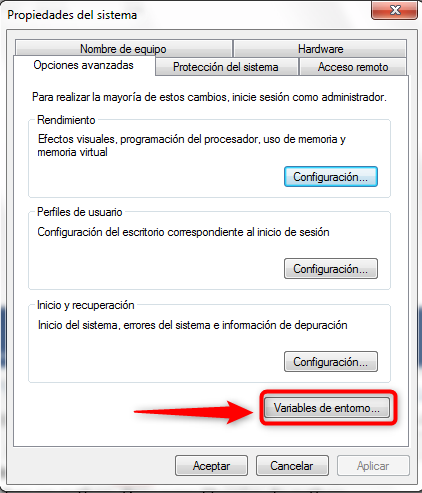
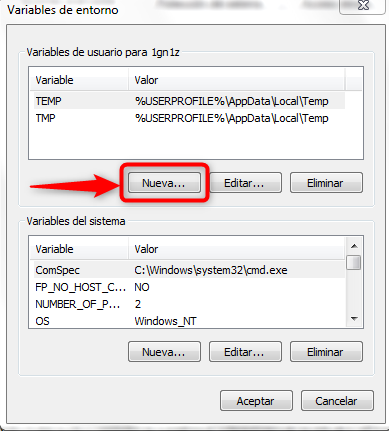
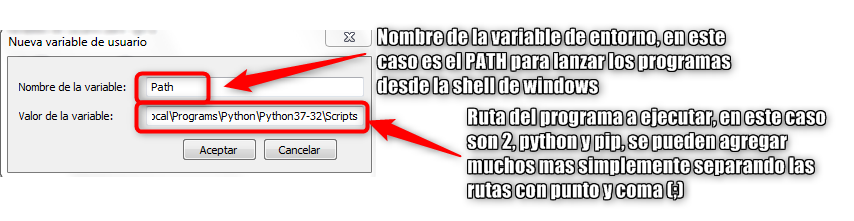
**NOTAS DEL CURSO DE PYTHON IMPARTIDO POR UDEMY**

**PIP** es un manejador de paquetes, lo único que hace PIP es descargar e instalar programas hechos en Python. Descargar librerías de Python

**VARIABLES DE ENTORNO**

En mi caso no tenía ninguna de las variables de entorno de Python, para agregarla simplemente escribí **variables de entorno** en el buscador de Windows y luego entrar a la opción **variables de entorno**:





Para añadir más de una variable de entorno al PATH de W7 simplemente la agregamos, separándola de la(s) otra(s) con un punto y coma (**;**). Ejemplo:

C:\Perl64\site\bin**;**C:\Python27

Ejemplo personal con Python 3.7.3 y PIP en la variable **Path**:

C:\Users\1gn1z\AppData\Local\Programs\Python\Python37-32\**;**C:\Users\1gn1z\AppData\Local\Programs\Python\Python37-32\Scripts

**NUMEROS Y OPERADORES ARITMÉTICOS**

En Python tenemos suma, resta, multiplicación y división. Además, como en la vida real tenemos lo que se denomina como **Precedencia Aritmética**. Por ejemplo, cual es el resultado de esta operación:

8 + 2 \* 10

La respuesta NO es 100, la respuesta a esto sería 10\*2+8= 28.

Primero que todo se toman las operaciones que estén entre paréntesis () si es que los hay, es lo primerísimo que se toma en cuenta.

Luego, se toma la multiplicación y la división conforme aparezcan de izquierda a derecha.

Y finalmente la suma y resta, igual conforme aparezcan de izquierda a derecha.

Python, al realizar una división **SIEMPRE** devuelve un flotante, ósea, **un numero decimal**.

Si queremos que la división en Python devuelva un **entero** en una división podemos usar **doble diagonal //**.

19 / 2 = 9.5 **19 // 2 = 9**

El número que devuelve Python al usar la doble diagonal, es redondeado, **siempre devuelve solo la parte entera del resultado**.

También tenemos el signo de **potencia**. Por ejemplo, si queremos hacer 6\*6\*6 = 216

Para usar la potencia en Python simplemente usamos **doble signo de asterisco \*\***

6\*6\*6 = 216 **6\*\*3= 216**

**VARIABLES**

En el mundo real, cuando vemos problemas matemáticos como:

X = 10 z = 5 velocidad = 30kmps

Están serían las variables, x, z, altura.

Se llaman variables porque su valor **varia**, la velocidad de un auto varia, el valor de x varia, etc.

En Python, para declarar una variable simplemente escribimos

El **nombre de la variable**, y el **valor de la variable**, por ejemplo:

X = 2

Ósea, el valor de la izquierda (2), se guarda en la variable de la derecha (X).

Con las variables, **el valor queda almacenado**, pudiendo acceder a esa variable en cualquier momento. A diferencia de las operaciones aritméticas simples que vimos anteriormente.

También podemos asignarle una operación aritmética a una variable, por ejemplo:

**>>>** y = 6 \*\* 3

**>>>** y

216

**>>>**

Ahora, no hay que confundir valor con **asignación**, por ejemplo:

x = 2 esto imprime 2

si ponemos x + 2 = 4

pero si volvemos a imprimir x, vale nuevamente 2. Ya NO vale 4.

Esto pasa ex hicimos una suma solamente, **no asignamos con un nuevo valor la variable**.

Para que x valga 4, podemos hacerlo así:

**x = x + 2**

Ahora sí, x vale el valor de x (2) + 2. Esto se le conoce como asignación.

Ahora bien, si hacemos una operación como:

**z = 19/2 = 9.5**

A este resultado se le conoce como **Punto Flotante o FLOAT**, así se le conoce en Python a todo **numero decimal**.

También tenemos el tipo de dato **Cadena o String**, que simplemente son **cadenas de texto**, por ejemplo:

a = “Hola Mundo”

En Python, a diferencia de otros lenguajes, no tenemos que declarar que tipo de variable vamos a usar (String, Float, Int, etc.) Simplemente escribimos el nombre de la variable y la asignación o valor de la misma.

También podemos **asignar variables a variables**.

**>>> x = 10**

**>>> y = 20**

**>>> z = x + y**

**>>> z**

**30**

Podemos usar variables para operaciones aritméticas y asignarla a otras variables.

**CADENAS**

Las cadenas se usan para muchas cosas, por ejemplo, generar contraseñas y usarlas, guardar el nombre del servidor al que vamos a accesar, guardar nombres de tablas en bases de datos, para llaves de un diccionario, etc.

En Python NO hay ninguna diferencia entre usar comillas dobles o simples para asignar una cadena.

**>>> nombre = "1gn1z"**

**>>> nombre**

**'1gn1z'**

**>>> nombre = '1gn1z'**

**>>> nombre**

**'1gn1z'**

Pero, en algunos casos muy particulares, tal vez necesitemos usar ambas comas, por ejemplo:

**>>> frase = "Ella me dijo: 'Hola' "**

**>>> frase**

**"Ella me dijo: 'Hola' "**

**>>> frase = 'Ella me dijo; "Hola" '**

**>>> frase**

**'Ella me dijo; "Hola" '**

Si por ejemplo necesitamos obligatoriamente usar comillas dobles o simples, necesitamos usar el carácter conocido como **Carácter de escape (\)** podríamos hacer algo así:

>>> frase = "Ella me dijo: \"Hola\" "

>>> frase

'Ella me dijo: "Hola" '

Esto le indica a Python que **después de la diagonal invertida (Carácter de escape)** Las comillas las ignore y las tome como parte del texto que tenemos.

El carácter de escape, para indicar texto literal con comillas, ya sean dobles o sencillas se usa así:

variable = "cadena normal **\"Texto con comillas\"** "

Para indicar el texto entre comillas, siempre se escribe así:

a = "Texto normal" **\"**Texto con comillas**\"**"

Pero, además, el carácter escape se usa para otras cosas, por ejemplo, decirle a Python que queremos hacer algo especial con la letra que viene después, por ejemplo, para hacer un salto de línea.

>>> frase = "Ella me dijo: **\n**\"Hola\""

>>> print (frase)

Ella me dijo:

"Hola"

El salto de línea (\n) se usa para indicar que, **todo lo que esta después de la \n**, se imprima en una nueva línea. Este se muestra **siempre con la instrucción PRINT**.

También podemos usar el símbolo de suma (+), para unir 2 cadenas o 2 variables, a esto se le conoce como **concatenación**.

**>>> nombre = "1gn1z"**

**>>> saludo = "Kiubo! "**

**>>> print (saludo + nombre)**

**Kiubo! 1gn1z**

Aquí ya agregué yo el espacio en la variable saludo, pero se puede agregar el espacio desde el print:

>>> nombre = "1gn1z"

>>> saludo = "Kiubo!"

>>> print(saludo + **" "** + nombre)

Kiubo! 1gn1z

>>>

Simplemente debemos **concatenar un par de comillas con el espacio dentro de ellas**.

También podemos usar el símbolo de multiplicación para multiplicar una cadena, por ejemplo:

>>> print(nombre \* 3)

1gn1z1gn1z1gn1z

Si a la izquierda tiene una cadena, y a la derecha tiene un operador aritmético de suma o multiplicación, lo que hace Python es que va a sumar o multiplicar esa cadena el número de veces especificado.

**CONDICIONALES**

Las condiciones nos sirven para determinar si un valor de una variable se cumple o no se cumple, puede ser tipo booleana, entero o flotante, incluso cadenas.

Las condiciones en Python se escriben con la palabra reservada **IF** (Si), después lo que queremos testear (en este caso una variable), y después lo comparamos con algún signo de comparación, en este caso igual se escribe con doble signo de igual **==**.

El signo de igual, cuando es uno, sirve para **asignar un valor**. En este caso a la variable **edad** le asignamos el valor de **21**, es decir, el dato 21 se guarda en la variable edad ok.

edad = 21  
  
if edad == 21:  
 print("Puedes entrar a pistiar!")

Como vemos, al terminar la condición if, es necesario cerrarla con dos puntos:

Como vemos, la condición se cumple, así que se ejecuta el código siguiente.

¡OJO! El bloque de código que se ejecutara, es el código donde empieza con un **TAB**, después del if, ese código se ejecuta.

edad = 21  
  
if edad == 21:  
 print("Puedes entrar al bar")  
 print("Y también puedes beber")

**TODO LO QUE LLEVE TAB ANTES ES LO QUE SE EJECUTA EN LA CONDICIONAL IF.**

edad = 15  
  
if edad == 21:  
 print("Puedes entrar al bar")  
 print("Y también puedes beber")

Aquí, la condición NO se cumple, por lo tanto, no se imprime nada, pero que pasa si le quitamos el TAB a una línea de código?

edad = 15  
  
if edad == 21:  
 print("Puedes entrar al bar")  
print("Y también puedes beber")

**Y también puedes beber**

El código SI se ejecuta, ya que **al quitarle el TAB ya NO es parte de la condicional IF**.

Tambien podemos combinar más de una condición, con la palabra reservada **ELIF** (Else If, Si no). Esto se le llama **ANIDACION.**

NO es recomendable anidar demasiadas condiciones, ya que se hace ilegible el código ok.

if edad < 18:  
 print("No puedes entrar :(")  
elif edad >= 21:  
 print("Puedes entrar al bar")  
 print("Y también puedes beber")

Ahora, que pasa si queremos agregar un caso que cubra todas las demás posibles condiciones?

Pues agregaremos un **ELSE**. (RECORDAR QUE LAS CONDICIONES LLEVAN DOS PUNTOS AL FINAL (:))

edad = 20  
if edad < 18:  
 print("No puedes entrar :(")  
elif edad >= 21:  
 print("Puedes entrar al bar")  
 print("Y también puedes beber")  
else:  
 print("Puedes entrar, pero no beber")

En este caso NO se cumple ninguna condición, por lo tanto, salta al ELSE.

Para pedir datos al usuario, debemos especificar qué tipo de dato será:

**int(input(“Aquí va un número”))**

**float(input(“Aquí va un número decimal”))**

Para poner una cadena, se puede poner así:

**variable = input("Texto")**

**FUNCIONES**

Las funciones permiten encapsular código en pequeños bloques que podemos reusar en varios códigos sin tener que escribirlo una y otra vez.

Cuando usamos (llamamos) una función en el código, tenemos que escribir el nombre de la función, y después 2 paréntesis para llamarla.

Algunas funciones tienen **parámetros** que debemos especificar dentro de los paréntesis, como, por ejemplo, la función **print** debemos indicar el parámetro, es decir, la cadena que queremos imprimir.

print("Olakase")

Para definir nuestras propias variables en Python usamos la palabra reservada **def** de definition, luego el nombre de nuestra función y finalmente los paréntesis y 2 puntos al final, al igual que las condicionales if, elif y else.

Y al igual que las condicionales, debemos añadir un TAB para que Python sepa que todo lo que este con el tab es parte de la función.

def saludar():  
 print("Ola k ase")

Al ejecutarla no pasa nada, porque **no hemos llamado la función**, para llamarla simplemente debemos poner el nombre de esta en cualquier parte del código.

def saludar():  
 print("Ola k ase")  
  
saludar()

Pero las funciones también pueden tener parámetros.

Cuando tengamos código dentro de la función, por ejemplo, condicionales, **todo debe tener un TAB después de la función**, Para que Python reconozca que ese bloque de código pertenece a la función.

def confirmacion\_edad(edad):  
 if edad < 18:  
 print("No puedes entrar")  
 elif edad >= 21:  
 print("Puedes entrar al bar y también puedes beber")  
 else:  
 print("Puedes entrar al bar pero no puedes beber")

Para poner un parámetro que va a aceptar la función debemos ponerlo entre paréntesis, en este caso una variable llamada edad.

def confirmacion\_edad(edad):  
 if edad < 18:  
 print("No puedes entrar")  
 elif edad >= 21:  
 print("Puedes entrar al bar y también puedes beber")  
 else:  
 print("Puedes entrar al bar pero no puedes beber")  
  
edad = 21  
edad\_2 = 17  
  
confirmacion\_edad(edad)  
confirmacion\_edad(edad\_2)

Como vemos, poder llamar a la función muchas veces, además de que **podemos ejecutarla con distintos parámetros**.

En python se usan **guiones bajos**, por que, si usamos guion normal, pensara que es una resta y habrá errores.

Es convención en Python usar el **Snake case**:

def pedir\_pizza():

def confirmacion\_edad(edad):

**OBJETOS**

**Todo es un objeto en python!!!**. Pero que es un objeto? Pensemos en un objeto en el mundo real, por ejemplo un coche. Un coche es un objeto, y tiene cierto numero de **atributos**, como las placas, el color, el numero de puertas, el tipo de transmisión etc.

Y el coche tiene la **función** de tocar el clacson, encenderse o apagarse, etc.

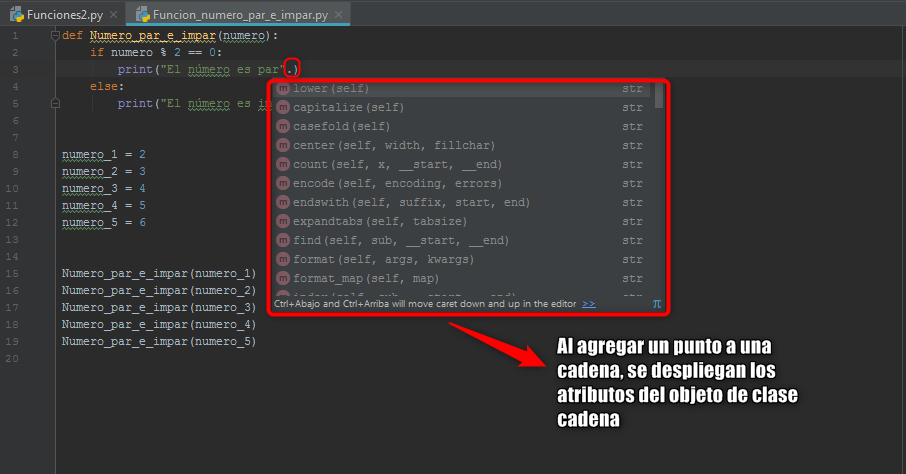
Los objetos en la programación son similares, se encargan de representar una forma abstracta de algo que puede **tener varios atributos y varias funciones.**

Por ejemplo un objeto llamado “Jugador” que tenga atributos, como por ejemplo el numero de vidas, numero de armas, tipo de armas. Y tiene funciones como caminar, correr, atacar, defenderse, etc.

Para acceder a los métodos de un objeto se usa la notación de punto. **Un método y una función son similares**.

Ya conocemos las funciones, que son construidas en python, sin embargo **la función pasa a llamarse Metodo, si esta dentro de un objeto.**

Por lo tanto **un método es una función que esta dentro de un objeto, y una función NO necesariamente es un método**.

TODO ES UN OBJETO EN PYTHON. Por ejemplo, la cadena es un objeto, como todo en python, y si queremos acceder a los métodos que tiene el objeto de la clase cadena, ponemos **punto .**

print("No puedes entrar".upper())

Por ejemplo en esta línea, el método de el objeto de clase cadena **upper** (que no acepta ningún parámetro.), hace que la cadena se muestre **toda en mayúsculas**.

Por ejemplo, ando viendo que onda xd y si ponemos la cadena con el método **lower**, toda la cadena se muestra en minúsculas.

print("Spam and Eggs".lower())

spam and eggs

Con **capitalize** se muestra solo la primera letra de la cadena en mayuscula.

print("Spam and Eggs".capitalize())

Spam and eggs

Con **center** se centra el texto, **requiere un parámetro NUMERICO dentro de los parentesis**

print("Spam and Eggs".center(100))

Spam and Eggs

Y asi hay muchos, poco a poco los iremos viendo.

**LISTAS**

Hasta el momento solo hemos visto variables que almacenan solo un dato, pero que pasa si queremos almacenar mas de un dato? Para eso tenemos las listas :D.

Las listas son **un conjunto de objetos**, similar a un **arreglo** en otros lenguajes, pero a diferencia, las listas en python puedes tener el tamaño que queramos, podemos cambiar el tamaño, y podemos guardar cualquier tipo de objetos y datos que queramos sin nungun problema.

Para usar las listas en python se definen poniendo **entre corchetes** lo que llevara nuestra lista, separando los datos u objetos por **comas**,.

Las listas **no se pueden concatenar con números enteros,** si deseamos agregar mas datos a una lista se hace **concatenando el valor actual de la lista mas los nuevos valores**:

**>>> lista = [1, 2, 3, 4, 5]**

**>>> lista**

**[1, 2, 3, 4, 5]**

**>>> lista = lista + [6, 7]**

**>>> lista**

**[1, 2, 3, 4, 5, 6, 7]**

Y también podemos abreviarlo asi:

***>>> lista += [8, 9]***

**>>> lista**

**[1, 2, 3, 4, 5, 6, 7, 8, 9]**

**El mas igual +=** abrevia **lista = lista +**

Al igual que las cadenas, podemos multiplicar los valores de una lista, simplemente escribiendo lista por (lista \*)

**>>> lista \* 2**

**[1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9]**

Otra forma de agregar elementos podemos usar un método del objeto de clase lista, escribiendo el **método append.**

**>>> lista.append(10)**

**>>> lista**

**[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]**

Podemos agregar cualquier tipo de dato a las listas, por ejemplo, con el método append, podemos agregar también una cadena.

**>>> lista.append("1gn1z")**

**>>> lista**

**[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, '1gn1z']**

Incluso podemos agregar listas dentro de listas. Esto se escribe con el método append, y dentro de los paréntesis, escribimos corchetes para indicar que se trata de otra lista:

**>>> lista.append([11, 12, 13])**

**>>> lista**

**[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, '1gn1z', [11, 12, 13]]**

Que pasa si queremos agregar mas de un elemento a nuestra lista?

**Con el argumento append solo se acepta UN ELEMENTO**. Si a la lista le queremos agregar mas de un elemento con un método usamos el elemento **EXTEND**.

**>>> lista\_2.extend([7,8])**

**>>> lista\_2**

**[1, 2, 3, 4, 5, 6, 7, 8]**

Ahora, si queremos borrar un elemento de nuestra lista, usaremos el método **REMOVE**:

**>>> lista\_2.remove(8)**

**>>> lista\_2**

**[1, 2, 3, 4, 5, 6, 7]**

Podemos eliminar cualquier elemento de la lista con el método remove, no necesariamente tiene que ser el ultimo ok.

**>>> lista\_2.remove(4)**

**>>> lista\_2**

**[1, 2, 3, 5, 6, 7]**

**SPLIT AND JOIN (Listas y cadenas).**

En una cadena, por ejemplo “1gn1z”, cada letra o numero es un caracter, asi que es muy similar a una lista de numeros solamente.

Podemos crear listas con cerchetes **[]**, pero también podemos crear una lista con **el metodo LIST** seguido de **paréntesis**. Pero si hacemos una lista de números, nos dara un error, ya que el método list toma a lo mucho 1 argumento.

**>>> list(1, 2, 3, 4, 5)**

**Traceback (most recent call last):**

**File "<pyshell#35>", line 1, in <module>**

**list(1, 2, 3, 4, 5)**

**TypeError: list expected at most 1 arguments, got 5**

En este caso, dentro de los parentesis tenemos que agregar corchetes.

**>>> list([1,2,3,4,5])**

**[1, 2, 3, 4, 5]**

Si ponemos el método list con una cadena, nos divide la cadena a caracteres sueltos:

**>>> list("1gn1z")**

**['1', 'g', 'n', '1', 'z']**

Con cualquier cadena pasa lo mismo, ademas de que como vemos, no importa si hay numeros, ya que si esta especificada como cadena, las separa igualmente.

Tambien podemos guardar la lista en una variable, y aquí si podemos concatenar una nueva lista:

**>>> lista\_letras = list("youtube")**

**>>> lista\_letras**

**['y', 'o', 'u', 't', 'u', 'b', 'e']**

**>>> lista\_letras + list("es genial")**

**['y', 'o', 'u', 't', 'u', 'b', 'e', 'e', 's', ' ', 'g', 'e', 'n', 'i', 'a', 'l']**

Ahora bien, si queremos agregar un elemento de tipo **cadena**, pero que sea un elemento mas, no que divide las letras, se usa el método **SPLIT**. **EL METODO SPLIT SIRVE PARA CONVERTIR UNA SERIE DE PALABRAS (CADENAS) A UNA LISTA**

**>>> "1gn1z".split()**

**['1gn1z']**

Como vemos, al usar el método Split de la clase cadena, toma la cadena y usa el espacio como delimitador por defualt.

Como que?

Osea que separa las cadenas de texto según los espacios que usemos, por ejemplo en la frase “Youtube es genial”, pues el método Split separa las cadenas conforme los espacion que usamos:

**>>> "Youtube es genial".split()**

**['Youtube', 'es', 'genial']**

Pero también pdemos usar otros delimitadores, por ejemplo en una lista x, si no ponemos el espacio nos toma todas las cadenas como una sola, por que no pusimos el delimitador por default que es el espacio:

**>>> "pechuga,papitas,pizza".split()**

**['pechuga,papitas,pizza']**

Para delimitar la lista con cada uno de los elementos **los separamos con una coma,** y después le indicamos el delimitador al método Split.

**>>> comida\_favorita = "pechuga,papitas,pizza"**

**>>> comida\_favorita**

**'pechuga,papitas,pizza'**

**>>> comida\_favorita.split(",")**

**['pechuga', 'papitas', 'pizza']**

Ahora si creo la lista con los elementos separados, usando la coma como separador.

Otro método es el llamado **JOIN**. Es como el opuesto al Split, lo que hace es tomar una lista y tomarla en una sola cadena dependiendo del separador que hayamos usado. **Este es un método de la clase string, no de la clase list. El método join SIRVE PARA CONVERTIR UNA LISTA EN UNA CADENA DE TEXTO**

**>>> ", ".join(lista\_comida\_favorita)**

**'Pechuga, papitas, pizza'**

Como vemos, el método join se usa al principio, indicando primero el delimitador, luego un punto y finalmente el método join y la lista que queremos juntar.

**>>> ",".join(lista\_comida\_favorita)**

**'Pechuga,papitas,pizza'**

Como vemos en este otro ejemplo, ya no hay espacio entre los elementos de la lista por que ya que no lo especifiamos en el método join.

Podemos usar variedad de delimitadores, como en este otro ejemplo que pusimos un guion.

**>>> "-".join(lista\_comida\_favorita)**

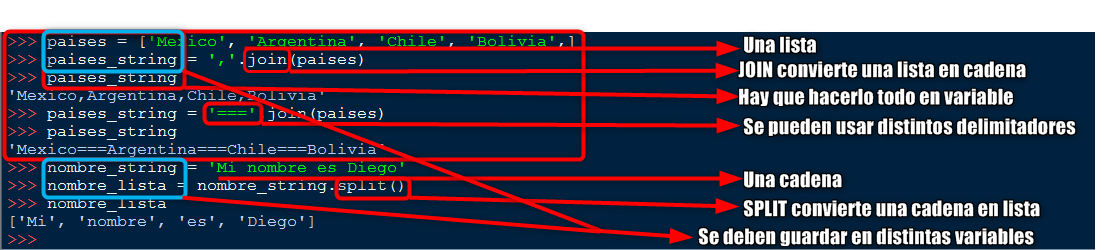
**'Pechuga-papitas-pizza'**

Estos métodos son muy útiles cuando trabajamos con muchas cadenas, y usándolas podemos simplificar el código, por ejemplo podemos hacer una cadena y concatenarle la lista con el método join, por ejemplo:

**>>> "Mi comida favorita es: " + ", ".join(lista\_comida\_favorita)**

'Mi comida favorita es: Pechuga, papitas, pizza'

**JOIN CONVIERTE UNA LISTA EN CADENA**

**SPLIT CONVIERTE UNA CADENA EN LISTA**

**>>> names = ['Luis', 'Juan', 'Pedro']**

**>>> names\_string = ','.join(names)**

**>>> "Mis amigos son: " + ','.join(names)**

**'Mis amigos son: Luis,Juan,Pedro'**

Como vemos, podemos escribir una cadena y concatenarle el método join, ya que el método join convierte una lista en cadena, asi que no hay problema :D

Si quisiéramos concatenar la cadena con la lista si arroja error

**>>> "Mis amigos son: " + names**

**Traceback (most recent call last):**

**File "<pyshell#66>", line 1, in <module>**

**"Mis amigos son: " + names**

**TypeError: can only concatenate str (not "list") to str**

Como vemos en la ultima linea de especifica claramente que una cadena solo se puede concatenar con otra, no con una lista.

**>>> caracteres = ['a','b','c','d']**

**>>> caracteres\_string = ', '.join(caracteres)**

**>>> caracteres\_string**

**'a, b, c, d'**

En este caso al delimitador le agregue un espacio, el cual se respeta.

**>>> "Las primeras 4 letras del alfabeto son: " + ', '.join(caracteres)**

**'Las primeras 4 letras del alfabeto son: a, b, c, d'**

Y asi convertimos una lista en cadena con el método join. Simplemente escribiendo la cadena, concatenándole el método join partiendo por el delmitador, el método join y dentro de los paréntesis la variable que contiene la lista que será convertida en cadena.

**>>> letras = ['x','y','z']**

**>>> letras\_string = '==='.join(letras)**

**>>> "Las últimas letras del alfabeto son: " + .join(letras)**

**SyntaxError: invalid syntax**

**>>> "Las últimas letras del alfabeto son: " + '=='.join(letras)**

**'Las últimas letras del alfabeto son: x==y==z'**

Tenemos que especificar que delimitador usaremos. OBLIGATORIAMENTE ok ;).

Con el método **Split** no podremos concatenar cadenas, por que el método **Split**

**NOTA. No es obligatorio que guardemos en variables distintas, pero es practico hacerlo xq asi podemos acceder a la lista o a la cadena. Por ejemplo:**

**>>> lista\_cadena = 'Convertir esta cadena a lista con el método split'**

**>>> lista\_convertida = lista\_cadena.split()**

**>>> lista\_convertida**

**['Convertir', 'esta', 'cadena', 'a', 'lista', 'con', 'el', 'método', 'split']**

**>>> lista\_cadena**

**'Convertir esta cadena a lista con el método split'**

**>>> c = ['pechuga', 'papitas', 'pizza']**

**>>> c2 = ''.join(c)**

**>>> 'Mi comida favorita es: ' + ''.join(c)**

**'Mi comida favorita es: pechugapapitaspizza'**

**>>> c2 = ' '.join(c)**

**>>> 'Mi comida favorita es: ' + ' '.join(c)**

**'Mi comida favorita es: pechuga papitas pizza'**

**NOTA2. Con el método JOIN se pueden agregar delimitadores varios, como espacios, comas, puntos, guiones, etc.**

**Con el método SPLIT NO se puede, los delimitadores tienen que estar en la cadena.**

**>>> c3 = 'pechuga papitas pizza'**

**>>> c4 = c3.split()**

**>>> c4**

**['pechuga', 'papitas', 'pizza']**

**NOTA3. Con el método SPLIT, al dejar los paréntesis en blanco, agrega una coma y un espacio para separar los elementos que serán convertidos en la lista. PERO! Los elementos en la cadena tienen que estar separados previamente por un espacio. En realidad la COMA se agrega obligatoriamente por que el método SPLIT convierte la cadena en lista, y en las listas en Python van cada elemento separado por una coma.**

**>>> c3 = 'pechuga papitas pizza'**

**>>> c4 = c3.split()**

**>>> c4**

**['pechuga', 'papitas', 'pizza']**

**Si esta toda la cadena junta, la lista convertida con el método Split se convierte en una lista de un solo elemento:**

**>>> c3 = 'pechugapapitaspizza'**

**>>> c4 = c3.split()**

**>>> c4**

**['pechugapapitaspizza']**

**Con este método, escrita la cadena asi sin espacion, NO se le pueden agregar espacios desde el método Split. Obligatoriamente deben estar separados los elementos desde la cadena, como en el ejemplo de arriba.**

Las listas y las cadenas en Python se parecen mucho xq ambas son literales, esto quiere decir que **contienen varios elementos y ambas se pueden recorrer utilizando Ciclos.**

**INDICES**

Los índices indican la posición de un elemento en una lista o una cadena.

Pero los indicen empiezan a contar **a partir del 0**. Pero como sabemos el índice de un elemento? Simplemente **escribimos el nombre de la variable donde esta la lista, y agregamos el método INDEX, esto es, agregamos punto, el método index y entre paréntesis agregamos como parámetro el elemento que deseamos saber su posición**.

**>>> alfabeto\_lista**

['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'x']

**>>> alfabeto\_lista.index("a")**

0

**>>> alfabeto\_lista.index("b")**

1

**>>> alfabeto\_lista.index("z")**

25

Como vemos, podemos consultar cualquier posición de un elemento con el método index.

**Tambien podemos consultar la posición de un elemento en una Cadena.**

>>> alfabeto = "abcdefghijklmnopqrstuvwxyzx"

**>>> alfabeto.index("a")**

0

**>>> alfabeto.index("b")**

1

**>>> alfabeto.index(1)**

Traceback (most recent call last):

File "<pyshell#9>", line 1, in <module>

alfabeto.index(1)

**TypeError: must be str, not int**

Como vemos, nos indica que hay error porque debemos buscar una cadena, no un entero.

Pero, si por ejemplo, buscamos un símbolo o carácter que NO este en la cadena o lista, obviamente también nos dara un error:

**>>> alfabeto.index("%")**

Traceback (most recent call last):

File "<pyshell#6>", line 1, in <module>

alfabeto.index("%")

**ValueError: substring not found**

Como vemos, nos indica que el carácter no fue encontrado, esto es asi por que el carácter ingresado en el index no existe en la cadena, de ahí que ponga **substring**, subcadena no encontrada.

En el caso de que sea una lista:

**>>> alfabeto\_lista.index("%")**

Traceback (most recent call last):

File "<pyshell#7>", line 1, in <module>

alfabeto\_lista.index("%")

**ValueError: '%' is not in list**

Nos indica en la ultima linea que el character ingresado NO esta en la lista.

Si por ejemplo, ahora queremos saber que elemento esta en determinada posición, simplemente escribimos la variable que queremos consultar, y **entre corchetes la posición que deseamos buscar**:

**>>> alfabeto\_lista[10]**

**'k'**

Y funciona exactamente igual con las cadenas:

**>>> alfabeto[10]**

**'k'**

En Python también existen **índices negativos**, que se empiezan desde el **menos 1 (-1)**, y empiezan a buscar **desde el ultimo elemento de la lista o cadena, siendo el ultimo elemento el menos 1**.

**>>> alfabeto\_lista[-1]**

**'z'**

**>>> alfabeto[-1]**

**'z'**

**>>> alfabeto[25]**

**'z'**

Los índices comunes empiezan de **izquierda a derecha**.

Los índices negativos empiezan de **derecha a izquierda**.

**DEL**

Otra palabra reservada en Python es la palabra DEL. Del nos permite elmininar variables enteras o para borrar elementos en nuestras listas. **UNA VEZ BORRADO ALGO NO SE PUEDE RECUPERAR.**

Simplemente escribimos la palabra reservada DEL y el nombre de la variable que queremos eliminar:

**variable\_basura = "Spam and Eggs"**

**>>> del variable\_basura**

>>> varible\_basura

Traceback (most recent call last):

File "<pyshell#27>", line 1, in <module>

varible\_basura

**NameError: name 'varible\_basura' is not defined**

Y para eliminar elementos en una lista, simplemente debemos escribir la palabra reservada **DEL** y **entre corchetes** escribimos el numero de la posición que deseamos eliminar:

**>>> lista\_vocales**

['a', 'e', 'i', 'o', 'u']

**>>> del lista\_vocales[0]**

**>>> lista\_vocales**

['e', 'i', 'o', 'u']

Tambien podemos usar números negativos para eliminar algún elemento de la lista:

**>>> del lista\_vocales[-1]**

**>>> lista\_vocales**

['e', 'i', 'o']

En el caso de las cadenas, al intentar borrar un elemento de la cadena, nos dara un error:

**>>> del vocales[0]**

Traceback (most recent call last):

File "<pyshell#34>", line 1, in <module>

del vocales[0]

**TypeError: 'str' object doesn't support item deletion**

Como vemos en la ultima linea, el error nos indica que el objeto cadena NO soporta borrado de items, ya que en python **las cadenas son inmodificables**.

Si queremos modificar una cadena, ya sea agregándole o quitándole elementos, lo mas fácil seria hacerlo mediante los métodos JOIN y SPLIT.

Por ejemplo podemos agregar una variable con una cadena:

**>>> var = "agregar una letra: "**

Ya que las cadenas son **inmutables** debemos convertirla en una lista con el método **SPLIT**:

**>>> var2 = var.split()**

**>>> var2**

**['agregar', 'una', 'letra:']**

Ahora ya que nuestra cadena es una lista, podemos agregarle elementos con el método **append**:

**>>> var2.append("Z")**

**>>> var2**

**['agregar', 'una', 'letra:', 'Z']**

Como vemos la letra se añadió a la lista satisfactoriamente, ahora lo ultimo que necesitamos hacer es converitir de nuevo la lista a cadena, con el método **JOIN**:

**>>> var = ''.join(var2)**

**>>> var**

**'agregarunaletra:Z'**

Recordemos que el método **JOIN** si permite poner los delimitadores como queramos, asi que simplemente agregamos un espacio al delimitador:

**>>> var = ' '.join(var2)**

**>>> var**

**'agregar una letra: Z'**

Tarea lista :)

**IN**

La palabra reservada **in** en Python, nos permite verificar si **algo esta dentro de otro algo**, por ejemplo si una letra es parte de una palabra, o si un elemento es parte de una lista.

Y como el **in** nos devuelve un valor de **cierto o falso** lo podemos usar **con nuestros condicionales IF**.

**>>> vocales = "aeiou"**

**>>> lista\_vocales = list(vocales)**

**>>> "a" in vocales**

**True**

**>>> "z" in vocales**

**False**

**>>>**

Como vemos, nos indica con **True** si esta el elemento, si no arroja **False**.

Asi que, como ya mencionamos anteriormente, podemos usar el **in** en conjunción con el **if**

**>>> if "a" in vocales:**

**print("el elemento a buscar esta en la cadena")**

**else:**

**print("El elemento a buscar NO esta en la cadena")**

**el elemento a buscar esta en la cadena**

La primera línea dice, **si** **a** esta **en** vocales entonces:

**NOT**

Si queremos saber si algo NO esta en una lista o cadena podemos usar la palabra reservada **not**, para verificar si algo no esta dentro de algo.

Al igual que **in** se puede usar en conjunto con las condicionales **if**.

**>>> if "z" not in vocales:**

**print("El elemento a buscar NO esta en la cadena")**

**El elemento a buscar NO esta en la cadena**

Tanto el **in** como el **not** funcionan igual con las listas:

**>>> if "a" in lista\_vocales:**

**print("El elemento a buscar esta en la cadena")**

**El elemento a buscar esta en la cadena**

**CICLOS**

Los ciclos nos sirven para realizar tareas repetitivas, sin tener la necesidad de escribir, copiar o pegar un bloque de código una y otra vez.

En Python tenemos 2 ciclos, los ciclos **FOR** y los ciclos **WHILE**.

El ciclo **while** sirven para **repetir un bloque de código mientras una condición sea cierta**.

El ciclo **For** sirve para **recorrer elementos en una colección de cosas, como caracteres en una cadena o elementos de una lista**, etc.

El ciclo while se escribe con la palabra reservada **while (mientras)**, seguido de la **Condicional** y al final **dos puntos**.

**while manzanas > 0:** (Mientras manzanas sea mayor que 0)

Despues, ponemos el código que se ejecutara, ya que la condición SI se esta cumpliendo:

**manzanas = manzanas – 1**

y podemos simplificar esta línea, simplemente agregando -= (menos igual), justo asi:

**manzanas -= 1**

que quiere decir exactamente lo mismo, **manzanas es igual a manzanas**

**Me estoy comiendo una manzana**

**Me estoy comiendo una manzana**

**Me estoy comiendo una manzana**

**Me estoy comiendo una manzana**

**Me estoy comiendo una manzana**

**Me estoy comiendo una manzana**

**Me estoy comiendo una manzana**

**Me estoy comiendo una manzana**

**Me estoy comiendo una manzana**

**Me estoy comiendo una manzana**

Como vemos, esto imprime la línea que le especificamos, y la imprime 10 veces, ya que la variable manzanas vale 10, y el ciclo nos dice que mientras manzanas sea mayor que 10 imprima la línea indicada.

Y finalmente la ultima línea del ciclo indica que a cada iteración le reste 1 a la variable manzanas.

Si quisiéramos ver el numero de manzana, agregamos una concatenación y la palabra **str** y entre paréntesis la variable.

Tenemos que escribir str por que queremos concatenar un string con un numero, pero como no podemos concatenar un string con un int, lo que debemos hacer es convertir el int de la variable manzanas a un string, con la función **str**:

**while manzanas > 0:**

**print('Me estoy comiendo una manzana # ' + str( manzanas)) manzanas -= 1**

Ciclos **FOR**

Los ciclos for, sirven para recorrer los elementos en una colección.

El ciclo for se escribe con la palabra reservada **for**, seguido de **una variable que usaremos para referirnos al elemento en ese momento** y después la **colección que vamos a recorrer**.

***lista\_nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]***

***for numero in lista\_nums:***

En este caso, la variable número en la primera iteración almacenara el primer elemento de la colección, es decir el numero 1, a la siguiente el 2 y asi sucesivamente.

En la colección, por ejemplo en nuestra lista, no necesariamente tiene que ser un numero, podemos agregar una cadena sin ningún problema.

**lista\_nums = [1, '1gn1z', 3, 4, 5, 6, 7, 8, 9, 10]**

**CONTINUE y BREAK**.

En los ciclos FOR tenemos también 2 palabras reservadas llamadas **continue** y **break**, que como sus nombres indican, sirven para continuar o detener el ciclo for.

Por ejemplo, si queremos imprimir los números del 1 al 5. Podemos hacerlo de varias maneras, pero en este caso usaremos la palabra reservada **break**, **el break rompe el ciclo cuando se llega a dicha palabra**.

***for numero in lista\_nums:***

***if numero > 5:***

***break***

***print(numero)***

Este código lo que hace es guardar el primer elemento de la colección (1 de la lista) en la variable numero, luego entra a la condicional if que dice **si numero es mayor o igual a 5** entonces ahí termina (**break**), como 1 no es mayor que 5 entonces imprime 1 en la primera iteración, luego 2 tampoco es mayor que 5 y lo imprime, luego 3 y asi sucesivamente hasta que llega al 5, 5 es mayor 5, no, entonces imprime el numero 5, y finalmente se pregunta si 6 es mayor que 5, que da true, asi que ahí entra el break y termina el programa.

Si quisiéramos **omitir** un elemento de la colección, en este caso podemos usar la palabra reservada **continue**.

**Continue**, a diferencia del break, **no rompe todo el ciclo**, **lo único que hace es SALTARSE esa iteración.**

Asi que, si por ejemplo, queremos que se impriman todos los números menos el 5, entoces el código quedaría asi:

***lista\_nums = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]***

***for numero in lista\_nums:***

***if numero == 5:***

***continue***

***print(numero)***

**1**

**2**

**3**

**4**

**6**

**7**

**8**

**9**

**10**

Como vemos, la condición indica que si numero es igual a 5 entre el continue, es decir, se salte ese numero y continue el ciclo for, por eso arroja los números del 1 al 10 omitiendo el numero especificado para el continue.