**Clase 02**

**SINTAXIS Y TIPOS DE DATOS**

En esta lección revisaremos los tipos de datos, aprenderemos cómo se pueden almacenar los datos en las listas de Python y sobre el concepto de objetos en la programación.

**2.1 Tipos de datos**

Los tipos de variables posibles en Python son:

**Números:**

* string (str): Son caracteres, no tienen valor numérico
* integer (int): Números enteros o naturales
* float (float): Números flotantes o reales

In [0]:

a = 3 *# esto es un número entero [int].*

print("a es: ", type(a))

print(a)

b = 39.747 *# esto es un número de coma flotante (float)*

print("**\n**b es: ", type(b))

print(b)

z = 4 + 6j *# esto es un número complejo*

print("**\n**z es: ", type(z))

print(z)

str1 = 'hola'

str2 = int("123")

print("**\n**a+b=", a + b)

print("**\n**str2 + a = ", str2 + a)

print(type(str2))

**Cadena de texto:**

In [0]:

c1 = "Bienvenido año nuevo " *# esto es una cadena de caracteres.*

print("c1 es: ", type(c1))

print(c1)

c2 = "2019" *# esto es una cadena de caracteres.*

print("**\n**c2 es: ", type(c2))

print(c2)

c3 = c1 + c2 *# concatenación de dos cadenas de caracteres*

print("**\n**c3 es: ", type(c3))

print(c3)

print("**\n**c3 tiene ", len(c3), "caracteres") *# La función [len] devuelve el número de caracteres de una cadena.*

**Booleanos:**

El tipo booleano sólo puede tener dos valores: True (cierto) y False (falso). Estos valores son especialmente importantes para las expresiones condicionales y los bucles.

In [0]:

*# Conjunción*

**True** **and** **True**

*#True and False*

In [0]:

*# Disyunción*

*#True or True*

*#True or False*

**False** **or** **False**

**Ejemplos:**

In [0]:

dd= **not** **True**

print("dd es: ", dd)

In [0]:

**not** **not** **not** **not** **not** **True**

In [0]:

(4 < 5) **and** (5 < 6)

In [0]:

(4 < 5 ) **and** (9 < 6)

In [0]:

x1 = (1 == 2) **or** (2 == 2)

x2 = (1 == 2) **and** (2 == 2)

print("x1 es: ", x1)

print("x2 es: ", x2)

In [0]:

2 + 2 == 4 **and** **not** 2 + 2 == 5 **and** 2 \* 2 == 2 + 2

In [0]:

nombre = 'CHOSICA'

id = 202906

longitud = 76+41/60+23.8/3600 *# 76° 41'23.8*

latitud = 11+55/60+48.5/3600 *# 11° 55'48.5"*

tipo = 'Hidrometrica'

fecha = "28-02-1998"

Podemos explorar los diferentes tipos de datos almacenados en variables usando la type()función.

In [0]:

type(nombre)

In [0]:

type(id)

In [0]:

type(latitud)

In [0]:

type(tipo)

**Métodos de cadenas**

Recuerde que los tipos de datos son importantes porque algunos no son compatibles entre sí.

In [0]:

nombre + id

Aquí obtenemos una **TypeError** porque Python no sabe cómo combinar una cadena de caracteres (nombre) con un valor entero ( id).

En otros lenguajes de programación es necesario escribir o declarar variables antes que puedan almacenar cualquier cosa en ellas. Esto no es necesario cuando trabajas con cadenas de caracteres en Python.

In [0]:

cadena1 = "océano "

cadena2 = "atmósfera"

cadena = "Interacción "+cadena1 + cadena2

print(cadena)

In [0]:

cadena1 = "océano "

cadena2 = "atmósfera"

cadena = "Interacción "+cadena1 + cadena2

print(cadena)

print("--"\*2)

cadena1 = "océano **\n**"

cadena2 = "atmósfera"

cadena = "Interacción "+cadena1 + cadena2

print(cadena)

In [0]:

cadena = "123"

numero = str(123)

c = cadena + numero

print(c)

**Indexación de cadenas**

In [0]:

cadena1 = "océano "

cadena2 = "atmósfera"

cadena = "Interacción "+cadena1 + cadena2

print(cadena)

print("**\n**")

print("índice [0]:", cadena[0])

print("índice [:] al [4]:", cadena[:4])

print("------------------------------------------------")

print("índice [-1]:", cadena[-1])

print("índice [-1] al [-4]:", cadena[-4:])

**Multiplicar cadena**

In [0]:

cadena1 = "océano "

cadena2 = "atmósfera"

cadena = "Interacción "\*3 + cadena1 + cadena2

print(cadena)

print("**\n**")

cadena = "Interacción "\*6 + cadena1 + cadena2

print(cadena)

**Añadir caracteres a una cadena**

In [0]:

cadena1 = "océano "

cadena2 = "atmósfera"

cadena = "Interacción " + cadena1 + cadena2

print(cadena)

print("**\n**")

cadena += " continente"

print(cadena)

**Determinar el número de caracteres de una cadena**

La función len devuelve el número de caracteres de una cadena. Esto es considerando la cantidad de caracteres y espacios en blanco.

In [0]:

cadena1 = "océano "

print(len(cadena1))

cadena2 = "atmósfera"

print(len(cadena2))

cadena = "Interacción " + cadena1 + cadena2

print(len(cadena))

**Buscar sub-cadenas**

Es posible recorrer todas las sub cadenas de una cadena, utilizando el método find, quién devuelve el índice de inicio de la sub-cadena.

In [0]:

cadena1 = "océano "

cadena2 = "atmósfera"

cadena = "Interacción " + cadena1 + cadena2

print(cadena)

buscarSC = cadena.find("atmósfera")

print(buscarSC)

Si la sub-cadena no se encuentra dentro de la cadena, el método **find** devolverá el valor -1

In [0]:

cadena1 = "océano "

cadena2 = "atmósfera"

cadena = "Interacción " + cadena1 + cadena2

print(cadena)

buscarSC = cadena.find("acción océ")

print(buscarSC)

**Convertir: minúsculas y mayusculas**

Los caracteres tienen diferentes métodos. Sigue de cerca las instrucciones para descubrir algunas de ellas. Si desea descubrirlos con más detalle, siempre puede escribir **help(str)**.

**upper():** devuelve una copia de la cadena en la que todos los caracteres como mayúsculas.

**lower():** devuelve una copia de la cadena en la que todos los caracteres como minúsculas.

In [0]:

cadena1 = "océano "

cadena2 = "atmósfera"

cadena = "Interacción " + cadena1 + cadena2

print(cadena)

print(cadena.upper())

print(cadena.lower())

**Reemplazar caracteres**

In [0]:

cadena1 = "océano "

cadena2 = "atmósfera"

cadena = "Interacción " + cadena1 + cadena2

print(cadena)

newCad = cadena.replace("o", "land")

print(newCad)

**Convertir datos de un tipo a otro (Coerción)**

No es el caso que variables como la ***nombre*** y ***id*** no puedan combinarse en absoluto, para combinar una cadena de caracteres con un número necesitamos realizar una conversión de tipo de datos para que sean compatibles. Por ejemplo, podríamos convertir el ***id***entero en una cadena de caracteres usando la función str().

In [0]:

idstr = str(id)

In [0]:

type(id)

In [0]:

print(id)

Como puede ver, **str()** convierte un valor numérico en una cadena de caracteres con los mismos números que antes.

Similar a usar **str()** para convertir números en cadenas de caracteres, **int()** se puede usar para convertir cadenas o números de punto flotante en enteros y **float()** se puede usar para convertir cadenas o números enteros en números de punto flotante.

**Combinando texto y números**

Aunque la mayoría de las operaciones matemáticas operan con valores numéricos, una forma común de combinar cadenas de caracteres es usar el operador de suma +.

In [0]:

nombreeid = nombre + ": " + str(id)

In [0]:

print(nombreeid)

Tenga en cuenta que aquí estamos convirtiendo **id** una cadena de caracteres utilizando la **str()** función dentro de la asignación a la variable **nombreeid**. Alternativamente, podríamos haber añadido simplemente **nombre** y **idstr**.

In [0]:

In [0]:

**Tips**

**Ayuda**

Tal vez ya sabes el nombre de una función, pero todavía tiene que averiguar cómo usarlo.

Irónicamente, hay que pedir información sobre una función con otra función: **help()**.

También se puede usar **?** antes del nombre de la función.

In [0]:

help(max)

?max

**Clase 03**

**ESTRUCTURAS DE DATOS**

**Tuplas**

Al igual que las cadenas de caracteres, las tuplas son una secuencia de elementos ordenados. La diferencia de una tupla radica en que sus elementos no son necesariamente caracteres. Los elementos individuales pueden ser de cualquier tipo (str, int, float). Además, las tuplas son objetos inmutables.

[ ]



t1 = (1)

print("t1 \t= ",t1)

​

t2 = (1, 'a', 34.55, 3 + 4j, 'Hello world!', t1)

print("t2 \t= ", t2)

print(type(t1))

print(type(t2), "\n")

​

t3= ("two","rain")

t = t2 + t3

​

print("t \t= ", t)

print("t[3] \t= ", t[3])

print("t[2:5] \t = ",t[2:5])

​

print("---"\*20)

print("len(t)", len(t))

**Listas**

Al igual que una tupla, una lista es una secuencia de valores ordenados, donde cada valor es identificado por un índice. La sintaxis para expresar una lista es literalmente igual al de una tupla; la diferencia está en el uso de brackets **[ ]** en lugar de paréntesis y su mutabilidad.

[ ]



l1 = ["wind"]

print("l1 \t= ",l1)

​

l2 = [1, 'a', 34.55, 3 + 4j, 'Hello world!', l1]

print("l2 \t= ", l2)

l2[-1]="hola"

​

l3= ["two","rain"]

l = l2 + l3

print("l \t= ", l)

​

print("l[3] \t= ", l[3])

print("l[3:5] \t = ",l[3:5])

​

print("---"\*20)

print("len(l)", len(l))

**Mutabilidad de una lista**

**Algunos métodos de las listas**

* **lista[i]:** Devuelve el elemento que está en la posición i de la lista.
* **lista.pop(i):** Devuelve el elemento en la posición i de una lista y luego lo borra.
* **lista.append(elemento):** Añade elemento al final de la lista.
* **lista.insert(i, elemento)**: Inserta elemento en la posición i.
* **lista.extend(lista2)**: Fusiona lista con lista2.
* **lista.remove(elemento)**: Elimina la primera vez que aparece elemento.
* **lista.count(e)**: Devuelve el número de veces que se repite (e) dentro de lista
* **lista.sort()**: Ordena los elementos de la lista en orden ascendente
* **lista.reverse()**: Invierte el orden de los elementos de la lista

[ ]



l1 = ["wind"]

print("l1 \t= ",l1)

​

l2 = [1, 'a', 34.55, 3 + 4j, 'Hello world!', l1]

print("l2 \t= ", l2)

​

print("\nEjemplos de mutabilidad\n")

l2[0]=100

l2[-1].append("sst")

print("l2 \t= ", l2)

​

# Reemplazar el elemento "sst" por "prc"

l2[-1][1]="prc"

print("l2 \t= ", l2)

**Convertir una cadena de caracteres en una lista**

[ ]



​

cadena1 = "océano "

cadena2 = "atmósfera"

​

cadena = "Interacción " + cadena1 + cadena2

print(cadena)

​

lista1 = cadena.split(" ")

print(lista1)

​

lista2 = cadena.split("o")

print(lista2)

[ ]



​

elem1, elem2, elem3 = lista1

print(elem1)

print(elem2)

print(elem3)

**Diccionarios**

De la misma forma como sucede con un diccionario convencional, un diccionario en Python es un objeto que tiene asociado algo.

Se crean poniendo sus elementos entre llaves:

**dic = {"key":"value", ...}** **Donde:** **"key" : son numeros enteros o cadenas de caracteres.** **"value": numeros enteros, cadenas de caracteres, tuplas, listas, arrays.**

[ ]



lima = {"region":"costa","superficie":"2672km2","altura": 154, "lat": -12, "lon":-77}

print(lima)

print(type(lima))

#print("\n\n")

**Agregar elementos en diccionarios**

[ ]



print(lima)

[ ]



lima["idioma"]= "español"

lima["fundacion"]="18 de enero de 1535"

lima["poblacion"]= [8574974,"3878928 mujeres", "3726814 hombres"]

​

print(lima)

**Estructura de un diccionario**

[ ]



keys = lima.keys()

values= lima.values()

items = list(lima.items())

print(keys)

print("\n")

print(values)

print("\n")

print(items)

print(type(items))

print(items[0])

**Operaciones con diccionarios**

* **diccionario.get('key')**: Devuelve el valor que corresponde con la key introducida.
* **diccionario.pop('key')**: Devuelve el valor que corresponde con la key introducida, y luego borra la key y el valor.
* **diccionario.update({'key'**:'valor'})\*\*: Inserta una determinada key o actualiza su valor si ya existiera.
* **"key" in diccionario**: Devuelve verdadero (True) o falso (False) si la key (no los valores) existe en el diccionario.
* **"definicion" in diccionario.values()**: Devuelve verdadero (True) o falso (False) si definición existe en el diccionario (no como key).

[ ]



dic=lima

print(dic)

idioma = dic.get("idioma")

print("\nidioma de lima -> ",idioma)

[ ]



dic=lima

print(dic)

idioma = dic.pop("idioma")

print("\nidioma de lima -> ",idioma)

print(dic)

**Índices de listas**

Anteriormente, hemos visto algunos datos relacionados con una de varias estaciones hidrométricas del SENAMHI en el área de Chosica. En lugar de tener variables individuales para cada una de esas estaciones, podemos almacenar muchos valores relacionados en una colección . El tipo de colección más simple en Python es una **lista**.

**Creando una lista**

Creamos una lista de varias estaciones.

[ ]



nombresEstaciones = ['Chosica', 'San Mateo', 'Rio Blanco', 'Puente Los Angeles']

[ ]



print(stationNames)

[ ]



type(stationNames)

Aquí tenemos una lista de 4 nombres de estaciones en una lista llamada **nombresEstaciones**. Como puede ver, la función **type()** reconoce esto como una lista. Las listas se pueden crear utilizando los corchetes ( [ ] ), con comas que separan los valores de la lista.

**Índice de valores**

Para acceder a un valor individual en la lista necesitamos usar un valor de índice. Un valor de índice es un número que se refiere a una posición dada en la lista. Veamos el primer valor en nuestra lista como ejemplo:

[ ]



print(nombresEstaciones[1])

¿Espera, qué? Este es el segundo valor en la lista que hemos creado, ¿qué está mal? Resulta que Python (y muchos otros lenguajes de programación) inician los valores almacenados en colecciones con el valor de **índice 0**. Por lo tanto, para obtener el valor del primer elemento de la lista, debemos usar el índice 0.

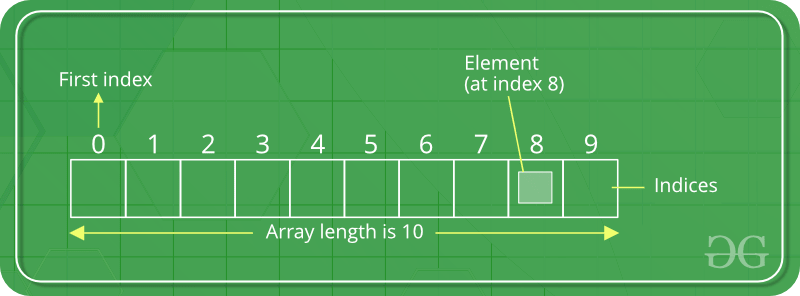
[ ]



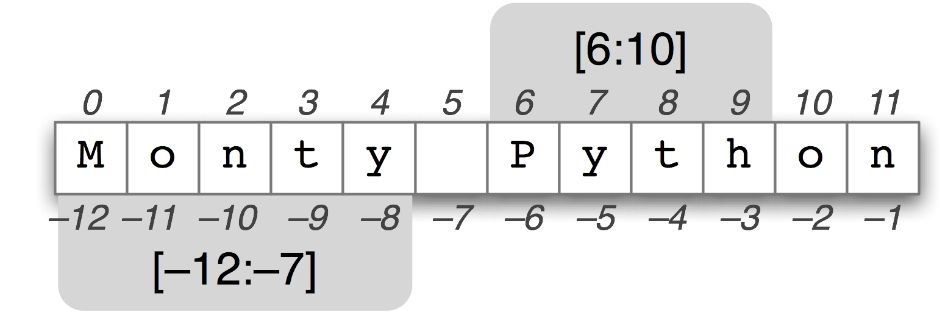
print(nombresEstaciones[0])

Haz doble clic (o ingresa) para editar

**Estructura de una lista:**



**Subsetting:**



**Consejos de valor de índice**

Si conocemos la longitud de la lista, ahora podemos usarla para encontrar el valor del último elemento de la lista.

[ ]



print(nombreEstaciones[4])

**¿IndexError?!!?**

Así es, ya que nuestra lista comienza con el índice 0 y tiene 4 valores, el índice del último elemento de la lista es 3. Eso no es lo ideal, pero afortunadamente hay un buen truco en Python para encontrar el último elemento en una lista.

**len(nombreEstaciones) - 1**

[ ]



print(nombreEstaciones)

[ ]



print(nombreEstaciones[-1])

[ ]



print(nombreEstaciones[-4])

Sí, en Python puede ir hacia atrás a través de listas utilizando valores de índice negativos. El índice -1da el último valor en la lista y el índice -len(nombreEstaciones) daría el primero. Por supuesto, todavía es necesario mantener los valores de índice dentro de sus rangos.

[ ]



print(nombreEstaciones[-5])

**Modificar los valores de la lista**

Otra característica de las listas es que son mutables , lo que significa que los valores en una lista que se ha definido pueden modificarse. Considere una lista de los tipos de estaciones de observación correspondientes a los nombres de estaciones en la lista **nombreEstaciones**.

[ ]



tiposEstaciones = ['Estacion climatica', 'Estacion climatica', 'Estacion climatica', 'Estacion climatica']

print(tiposEstaciones)

Ahora, como vimos antes, el tipo de estación para Chosica debería ser **'Estacion hidrometrica'**, no **'Estacion climatica'**. Afortunadamente, esta es una solución fácil. Simplemente reemplazamos el valor en la ubicación correspondiente en la lista con la correcta.

[ ]



tiposEstaciones[2] = 'Estacion hidrometrica'

print(tiposEstaciones)

**Tipos de datos en listas**

Las listas también pueden almacenar más de un tipo de datos. Consideremos que además de tener una lista de cada nombre de estación, ID, latitud, etc. nos gustaría tener una lista de todos los valores para una fecha específica de la estación Chosica.

[ ]



nombre = 'CHOSICA'

id = 202906

longitud = 76.689

latitud = 11.930

tipo = 'Hidrometrica'

fecha = "28-02-1998"

​

estacionChosica = [nombre, id, longitud, latitud, tipo, fecha]

print(estacionChosica)

Aquí tenemos una lista con 3 tipos diferentes de datos en ella. Podemos confirmar esto utilizando la función **type()**.

[ ]



type(estacionChosica)

[ ]



type(estacionChosica[0])

[ ]



type(estacionChosica[1])

[ ]



type(estacionChosica[3])

**Agregar y eliminar valores de las listas**

Finalmente, podemos agregar y eliminar valores de las listas para cambiar sus longitudes. Consideremos que ya no queremos incluir el primer valor en la **estacionChosica**.

[ ]



print(nombresEstaciones)

[ ]



del nombresEstaciones[0]

[ ]



print(nombresEstaciones)

La función **del** permite eliminar valores en listas. También se puede utilizar para eliminar valores de la memoria en Python. Si en su lugar nos gustaría agregar algunas muestras a la lista de nombres de estaciones, podemos hacerlo de la siguiente manera.

[ ]



nombresEstaciones.append('Yanacoto')

nombresEstaciones.append('Puente Huachipa')

[ ]



print(nombresEstaciones)

Como puede ver, agregamos valores uno a la vez usando **nombresEstaciones.append()**. list.append() se llama método en Python, que es una función que funciona para un tipo de datos determinado (una lista en este caso).

Veremos un poco más sobre esto a continuación.

**El concepto de objetos**

Python es un lenguajes de programación dentro de los que se denominan **'lenguajes orientados a objetos (POO)'**.

Puede llevar bastante tiempo entender lo que esto significa, pero la explicación simple es que podemos considerar las variables que definimos como **'objetos'** que pueden contener datos conocidos como **atributos** y un conjunto específico de **funciones** conocidas como **métodos** . La oración anterior también puede tomar algún tiempo para entenderse por sí misma, pero usando un ejemplo, el concepto de **"objetos"** es mucho más fácil de entender.

**Un (mal) ejemplo de métodos**

Consideremos nuestra lista **nombresEstaciones**. Como sabemos, ya tenemos datos en la lista **nombresEstaciones**, y podemos modificarlos utilizando métodos incorporados como **nombresEstaciones.append()**.

En este caso, el método **append()** es algo que existe para las listas, pero no para otros tipos de datos. Es intuitivo que le gustaría agregar (o adjuntar) elementos a una lista, pero tal vez no tenga sentido agregar a otros tipos de datos.

[ ]



nombresEstacionesLength = len(nombresEstaciones)

[ ]



print(nombresEstacionesLength)

[ ]



type(nombresEstacionesLength)

[ ]



nombresEstacionesLength.append(1)

Aquí obtenemos un **AttributeError** porque no hay ningún método integrado en el tipo **int** de datos para adjuntar a los datos **int**. Si bien **append()** tiene sentido para los **listdatos**, no es sensato para los datos **int**, por lo que no existe tal método para los datos **int**.

**Invertir una lista**

Hay otros dos métodos comunes para listas que necesitamos ver. Primero, está el método **.reverse()**, utilizado para invertir el orden de los elementos en una lista.

[ ]



nombresEstaciones.reverse()

[ ]



print(nombresEstaciones)

Un error común al ordenar las listas es hacer algo así nombresEstaciones = nombresEstaciones.reverse().

**¡No hagas esto!** Al revertir las listas con **.reverse()** el **valor** es devuelto (es por esto que no hay salida de pantalla cuando se ejecuta **nombresEstaciones.reverse()**). Si luego asigna la salida de **nombresEstaciones.reverse()** a **nombresEstaciones**, revertirá la lista, pero luego sobrescribirá su contenido con el valor **None** . Esto significa que ha eliminado el contenido de la lista (!)

**Ordenar una lista**

El método **.sort()** funciona de la misma manera.

[ ]



nombresEstaciones.sort()

[ ]



print(nombresEstaciones)

Como puede ver, la lista se ha ordenado alfabéticamente utilizando el método **.sort()**, pero cuando esto ocurre no hay salida de pantalla. De nuevo, si tuviera que asignar esa salida a la lista **nombresEstaciones** , se ordenaría, pero el contenido se asignaría **None**.

*Como habrás notado, aparece antes en la lista ordenada. Esto se debe a que la ordenación alfabética en Python coloca las mayúsculas antes de las minúsculas.*

**Múltiples argumentos**

Echa un vistazo a la documentación de sorted() al escribir help(sorted)en el shell de IPython. La función **sorted()** toma tres argumentos: **iterable**, **key** y **reverse**.

**key = None** significa que si no especificas el argumento **key**, será None. **reverse=False** significa que si no especificas el argumento **reverse**, será False.

En este ejercicio, solo tendrás que especificar **iterable** y **reverse**, no **key**.

La primera entrada que pase **sorted()** coincidirá con el argumento **iterable**

[ ]



​

primera = [11.25, 18.0, 20.0]

segunda = [10.75, 9.50] # Crear las listas primera y segunda

​

full = primera + segunda # Paste together first and second: full

full\_sorted = sorted(full, reverse = True) # Sort full in descending order: full\_sorted

​

print(full\_sorted) # Print out full\_sorted

**Enumerar atributos**

No discutiremos ningún atributo de lista porque, por lo que sabemos, no hay ninguno, pero encontraremos algunos atributos muy útiles de otros tipos de datos en el futuro.

[ ]



​

[ ]



​