Módulo I Tipos y Colecciones de Datos

Conceptos Básicos

Objeto

Componente que se aloja en la memoria y que tiene asociados una serie de vectores y operaciones que pueden ser realizadas con él. Un objeto en Python puede ser una cadena de texto, un número real, un diccionario o un objeto propiamente dicho, según el paradigma OOP, creado a partir de una clase determinada.

Contamos con:

1. Números
2. Cadenas de texto
3. Listas
4. Tuplas
5. Diccionarios
6. Dataframes

Además, se cuenta con un tipo de objeto especial llamado None que se emplea para asignar un valor nulo.

Números

Tipos: enteros, reales y complejos

In [16]:
```
num_entero = 8
```

In [17]:
```
#en el contexto de los números enteros
num_negativo = -100
```

In [18]:
```
num_real = 4.5
```

In [19]:
```
#otra manera de expresar un número real es:
num_real = 0.5e-7
```

In [20]:
```
#números formados por una parte real y otra imaginaria
num_complejo = 3.2 + 7j
```

Operadores

Los operadores resultan importantes para trabajar con los números. Entre estos tenemos:

1. Operadores aritméticos: suma, resta, división entera y real y multiplicación.
2. Operadores de bajo nivel y entre bits: NOT, NOR, XOR y AND.
3. Operadores para comprobar igualdad y desigualdad y para realizar operaciones lógicas: AND y OR

Expresión con operador	Operación
A + B	Suma
A – B	Resta
A * B	Multiplicación
A % B	Resto
A / B	División real
A // B	División entera
A ** B	Potencia
A | B	OR(bit)
A ^ B	XOR(bit)
A & B	AND(bit)
A == B	Igualdad
A != B	Desigualdad
A or B	OR(Lógica)
A and B	AND(Lógica)
not A	Negación(Lógica)

Funciones matemáticas

In [29]:
```
#valor absoluto
abs(-47.67)
```

Out[29]: 47.67

Para algunas operaciones necesitamos importar el módulo math

In [30]:
```
import math as m
m.sqrt(144)
```

Out[30]: 12.0

Cadenas de texto

Las cadenas de texto *(strings)* son otro de los tipos de datos más utilizados en programación. Una Cadena de texto es un conjunto inmutable y ordenado de caracteres. Para su representación y definición se pueden utilizar tanto comillas dobles *(")*, como simples *(')*.

In [39]:
```
cadena = "esto es una cadena de texto"
```

In [40]:
```
cadena2 = """Esta cadena de texto
tiene más de una línea. En concreto, cuenta
con tres líneas diferentes"""
```

In [41]:
```
type(cadena)
```

Out[41]: str

Principales funciones y métodos

In [49]:
```
cad = "cadena de texto de ejemplo"
len(cad)
```

Out[49]: 26

In [50]:
```
cad = "xyza"
cad.find("a")
```

Out[50]: 3

In [51]:
```
cad = "Hola Mundo"
cad.replace("Hola", "Adiós")
```

Out[51]: 'Adiós Mundo'

In [52]:
```
#eliminar espacios en blanco
cad = " cadena con espacios en blanco "
```

In [53]:
```
cad.strip()
```

Out[53]:
```
'cadena con espacios en blanco'
```

In [54]:
```
cad.lstrip()
```

Out[54]:
```
'cadena con espacios en blanco '
```

In [55]:
```
cad.rstrip()
```

Out[55]:
```
' cadena con espacios en blanco'
```

In [56]:
```
#convertir a mayúsculas y minúsculas
cad2 = cad.upper()
print(cad2)
```

```
CADENA CON ESPACIOS EN BLANCO
```

In [57]:
```
print(cad2.lower())
```

```
cadena con espacios en blanco
```

In [58]:
```
#convertir primer carácter a mayúscula
cad = "un ejemplo"
cad.capitalize()
```

Out[58]:
```
'Un ejemplo'
```

In [59]:
```
#dividir una cadena de texto basándose en un carácter
cad = "primer valor; segundo; tercer valor"
cad.split(";")
```

Out[59]:
```
['primer valor', ' segundo', ' tercer valor']
```

Operaciones

In [61]:
```
#concatenar
cad_concat = "¡Hola" + " Mundo!"
print(cad_concat)
```

```
¡Hola Mundo!
```

In [62]:
```
print("Hola Mundo " * 4)
```

```
Hola Mundo Hola Mundo Hola Mundo Hola Mundo
```

In [63]:
```
cad = "Nueva cadena de texto"
"h" in cad
```

Out[63]:
```
False
```

In [64]:
```
cad = "Cadenas"
print(cad[2])
```

```
d
```

In [65]:
```
print(cad[:3])
```

```
Cad
```

In [66]:
```
cad[-3]
```

Out[66]:
```
'n'
```

In [67]:
```
cad[3:]
```

Out[67]:
```
'enas'
```

Listas

Las listas contienen valores de cualquier tipo simple (numérico o no numérico), y podrían ser estructuras compuestas (lista de listas). Si usamos como referencia a una hoja de calculo con datos sobre individuos, una lista podria ser una fila que tiene los datos de los individuos.

```
In [68]:  Estudiante=["Manuel Ponte",23,"False"]
```

Los nombres pueden contener letras del alfabeto y números (y algunos caracteres de puntuación), pero no debe comenzar con un número.

```
In [69]:  Estudiante
```

```
Out[69]: ['Manuel Ponte', 23, 'False']
```

Nota que hay varios tipos de datos en la lista:

- *Nombre y apellido* es texto o caracteres.
- *edad* es un numero
- *False es un valor lógico*

Para acceder a cada uno de los elemento tu lista:

```
In [70]:  Estudiante[0] # primer elemento comienza con indice '0'
```

```
Out[70]: 'Manuel Ponte'
```

```
In [71]:  Estudiante[:2] # todo antes de índice 2
```

```
Out[71]: ['Manuel Ponte', 23]
```

```
In [72]:  Estudiante[-1] # ultimo elementos
```

```
Out[72]: 'False'
```

Si quieres cambiar algun valor:

```
In [73]:  Estudiante[0]='Omar Escobedo'
          Estudiante
```

```
Out[73]: ['Omar Escobedo', 23, 'False']
```

Para añadir elementos:

```
In [74]:  Estudiante.append('UNMSM')

          # Ahora tienes:
          Estudiante
```

```
Out[74]: ['Omar Escobedo', 23, 'False', 'UNMSM']
```

Para borrar, se puede hacer por...

- posición
- valor

Así, ante estas listas:

```
In [75]:  elementsA=[11,22,33,44]
          elementsB=[11,22,33,44]
```

Entonces:

```
In [76]:  ## borrar tercer elemento
          del elementsA[2]
          # luego:
```

```
elementsA  #
```

```
Out[76]: [11, 22, 44]
```

```
In [77]:  #  borrar valor '22'
          # se puede eliminar por posición o por valor
          elementsB.remove(22)
          elementsB
```

```
Out[77]: [11, 33, 44]
```

Si lo que querias era dejar una lista de 4 elementos, pero el último dejarlo vacío:

```
In [78]:  #None es como NA en R
          Estudiante[3]=None
          Estudiante
```

```
Out[78]: ['Omar Escobedo', 23, 'False', None]
```

Para eliminar la lista:

```
In [79]:  lista=['a','b']
          lista
```

```
Out[79]: ['a', 'b']
```

```
In [80]:  del lista #del elimina, en este caso el objeto newList
          #newList #si se corre debe arrojar error porque ya el objeto se ha eliminado
```

A veces queremos eliminar valores repetidos:

```
In [81]:  dias=['Lunes','Martes','Miércoles','Jueves','Viernes','Sábado','Domingo','Domingo']
          dias
```

```
Out[81]: ['Lunes',
          'Martes',
          'Miércoles',
          'Jueves',
          'Viernes',
          'Sábado',
          'Domingo',
          'Domingo']
```

Para ello tenemos la función *set*:

```
In [82]:  dias=list(set(dias)) #función set es conjunto (función matemática). Los conjuntos no admiten repetidos
          dias
```

```
Out[82]: ['Miércoles', 'Martes', 'Domingo', 'Sábado', 'Lunes', 'Viernes', 'Jueves']
```

Tuplas

En Python una tupla es una estructura de datos que representa una colección de objetos, pudiendo estos ser de distintos tipos. Internamente, para representar una tupla, Python utiliza un *array* de objetos que almacena referencias hacia otros objetos.

Al principio parace que fueran listas:

```
In [7]:  Estudiantetupla=("Omar Escobedo",26,"False")
         Estudiantetupla
```

```
Out[7]: ('Omar Escobedo', 26, 'False')
```

Para crearlas puedes usar '( )', el comando *tuple()* o nada:

```
In [8]:  Estudiantetupla='Karolayne Pacherres',23,'True'
         Estudiantetupla
```

```
Out[8]: ('Karolayne Pacherres', 23, 'True')
```

```
In [9]:  #Se puede realizar la consulta a través del índice que ocupa en la misma.
         #exactamente igual que un array
         Estudiantetupla[0]

Out[9]: 'Karolayne Pacherres'
```

Dado que una tupla puede almacenar distintos tipos de objetos, es posible anidar diferentes *tuplas*; veamos un sencillo ejemplo:

```
In [10]:  t = (1, ("a", 3), 5.6)
```

Concatenar dos tuplas es sencillo, se puede hacer directamente a través del operador +. Otros de los operadores que se pueden utilizar es , *que sirve para crear una nueva tupla donde los elementos de la original se repiten* n* veces.

```
In [12]:  ("r", 2) * 3

Out[12]: ('r', 2, 'r', 2, 'r', 2)
```

Los principales métodos que incluyen las tuplas son index() y count(). El primero de ellos recibe como parámetro un valor y devuelve el índice de la posición:

```
In [13]:  t = (1, 3, 7)
          t.index(7)

Out[13]: 2
```

El método count() sirve para obtener el número de ocurrencias de un elemento en una tupla:

```
In [14]:  t = (1, 3, 1, 5, 1)
          t.count(1)

Out[14]: 3
```

Sobre las tuplas también podemos usar la función integrada len(), que nos devolverá el número de elementos de la misma. Obviamente, deberemos usar la variable tupla como argumento de la mencionada función.

```
In [97]:  len(t)

Out[97]: 5
```

Nota: La gran diferencia es que una vez creadas, ninguno de sus valores se puede modificar.

```
In [15]:  # genera error
          Estudiantetupla[1]=50
          # arroja error porque la tuple no es modificable
          #R no tiene tuplas en sus paquetes básicos

          ---------------------------------------------------------------------------
          TypeError                                 Traceback (most recent call last)
          <ipython-input-15-3c3bebd6b480> in <module>
                1 # genera error
          ----> 2 Estudiantetupla[1]=50
                3 # arroja error porque la tuple no es modificable
                4 #R no tiene tuplas en sus paquetes básicos

          TypeError: 'tuple' object does not support item assignment
```

Diccionario

Los Diccionarios, superficialmente, son lo más similares a las listas de R:

```
In [17]:  # creando diccionarios:
          EstudianteDict={'Nombres':"Manuel Ponte",
                          'edad':23,
                          'femenino':False}
          #en los diccionarios se deben indicar los nombres de los atributos #
          seeing it:
          EstudianteDict
```

Out[17]: {'Nombres': 'Manuel Ponte', 'edad': 23, 'femenino': False}

Pero no tienen índices:

In [18]:
```
EstudianteDict[0]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-18-9f38b07bc1c2> in <module>
----> 1 EstudianteDict[0]


KeyError: 0
```

Pare ver un elemento, tienes que saber el nombre de su campo ('key'):

In [19]:
```
EstudianteDict['edad']
```

Out[19]: 23

A partir de ahi, puedes hacer las operaciones comunes:

In [21]:
```
EstudianteDict['edad']=24
EstudianteDict
```

Out[21]: {'Nombres': 'Manuel Ponte', 'edad': 24, 'femenino': False}

Detalles importantes para estructuras básicas

In [104…
```
type(EstudianteDict)
```

Out[104… dict

In [105…
```
type(Estudiante)
```

Out[105… list

In [106…
```
type(Estudiantetupla)
```

Out[106… tuple

B) Asegúrate que las funciones se pueden compartir

In [107…
```
listTest=[1,2,3,3]
tupleTest=(1,2,3,4,4)
dictTest={'a':1,'b':2,'c':2}
len(listTest), len(tupleTest), len(dictTest)

#len ve la cantidad de elementos en cada tipo de estructura de datos.
```

Out[107… (4, 5, 3)

Data Frames

Los Data Frames pueden interpretarse como estructuras compuestas en base a las simples. Python requiere que llamemos al paquete pandas para usar DFs:

In [23]:
```
import pandas as pd

# estas son columnas:
#listas de columnas

nombres=["Manuel", "Karen", "Karolayne", "Omar"]
edad=[23,29,23,26]
pais=["Ecuador", "Perú", "Brasil", "Argentina"]
educacion=["Bach", "Lic", "Bach", "PhD"]
```

In [24]:
```
# las llevamos a diccionario:
data={'nombres':nombres, 'edad':edad, 'pais':pais, 'educacion':educacion}
```

...y de dict a DF:

```
In [25]:  estudiantes=pd.DataFrame(data)
          # check it:
```

Out[25]:

	nombres	edad	pais	educacion
0	Manuel	23	Ecuador	Bach
1	Karen	29	Perú	Lic
2	Karolayne	23	Brasil	Bach
3	Omar	26	Argentina	PhD

También podría ser a partir de filas:

```
In [26]:  # Listas por filas
          row1=["Manuel", 23,"Ecuador", "Bach"]
          row2=["Karen", 29, "Perú", "Lic"]
          row3=["Karolayne",  23,  "Brasil",  "Bach"]
          row4=["Omar", 26, "Argentina", "PhD"]

          #Lista de listas de filas
          listOfRows=[row1,row2,row3,row4]

          #creandolo:
          DF_lists=pd.DataFrame(listOfRows,columns=['nombres','edad','pais','educacion'])
          DF_lists
```

Out[26]:

	nombres	edad	pais	educacion
0	Manuel	23	Ecuador	Bach
1	Karen	29	Perú	Lic
2	Karolayne	23	Brasil	Bach
3	Omar	26	Argentina	PhD

Tendremos un data frame? Recuerda que type te lo dice:

```
In [116…  type(estudiantes)
```

```
Out[116… pandas.core.frame.DataFrame
```

Pero Pandas da más detalles con 'dtypes':

```
In [117…  estudiantes.dtypes # es como la función str en R
          # object es un texto
```

```
Out[117… nombres      object
         edad          int64
         pais         object
         educacion    object
         dtype: object
```

Además de *dtypes()*, es bueno saber que puedes usar:

```
In [118…  # cuantas filas y columnas tienes:
          estudiantes.shape #en R es el dim
```

```
Out[118… (4, 4)
```

No hay función específica para saber cuantas filas o columnas hay independientemente, pero len ayuda:

```
In [119…  len(estudiantes.index) # or students.shape[0]
```

```
Out[119… 4
```

```
In [120…  len(estudiantes.columns) # or students.shape[1]
```

```
Out[120… 4
```

También son muy útiles las función head(), que te permite ver las filas al inicio del DF:

```
In [121…  estudiantes.head(1) # función head de R
```

Out[121…

	nombres	edad	pais	educacion
0	Manuel	23	Ecuador	Bach

Y su antónimo también está disponible:

```
In [122…  estudiantes.tail(2)
```

Out[122…

	nombres	edad	pais	educacion
2	Karolayne	23	Brasil	Bach
3	Omar	26	Argentina	PhD

Claro que es bueno saber qué variables tenemos: In

```
[123… estudiantes.columns
```

```
Out[123… Index(['nombres', 'edad', 'pais', 'educacion'], dtype='object')
```

Lo anterior no es una lista, pero si la necesitas:

```
In [124…  list(estudiantes)
```

```
Out[124… ['nombres', 'edad', 'pais', 'educacion']
```

Tu DF es la tabla de datos a la que estás acostumbrado a utilizar. Si quieres ver algun elemento en particular:

```
In [125…  # una columna
          estudiantes.nombres
```

```
Out[125… 0       Manuel
         1        Karen
         2    Karolayne
         3         Omar
         Name: nombres, dtype: object
```

```
In [126…  # o
          estudiantes['nombres']
```

```
Out[126… 0       Manuel
         1        Karen
         2    Karolayne
         3         Omar
         Name: nombres, dtype: object
```

```
In [127…  # # varias columnas (con posiciones)
          # # ANTES DE LA COMA FILAS, DESPUES DE LA COMA COLUMNAS
          estudiantes.iloc[:,[1,3]]
```

Out[127…

	edad	educacion
0	23	Bach
1	29	Lic
2	23	Bach
3	26	PhD

```
In [128…  # varias columnas (con nombres)
          estudiantes[['edad','educacion']]
```

Out[128…

	edad	educacion
0	23	Bach
1	29	Lic
2	23	Bach
	26	PhD

In [129…
```
# o
estudiantes.loc[:,['edad','educacion']]
```

Out[129…

	edad	educacion
0	23	Bach
1	29	Lic
2	23	Bach
3	26	PhD

In [130…
```
# tambien:

estudiantes.loc[:,'edad':'educacion']
```

Out[130…

	edad	pais	educacion
0	23	Ecuador	Bach
1	29	Perú	Lic
2	23	Brasil	Bach
3	26	Argentina	PhD

In [28]:
```
# varias columnas (con posiciones)
## SIEMPRE NO TE MUESTRA LA ULTIMA COLUMNA QUE ESPECIFICAS EN ESTE CASO
## LA COLUMNA 4 NO SE VA A MOSTRAR
estudiantes.iloc[:,1:4]
```

Out[28]:

	edad	pais	educacion
0	23	Ecuador	Bach
1	29	Perú	Lic
2	23	Brasil	Bach
3	26	Argentina	PhD

In [132…
```
# sino:

estudiantes.iloc[:,[1,3]]
```

Out[132…

	edad	educacion
0	23	Bach
1	29	Lic
2	23	Bach
3	26	PhD

In [133…
```
# una fila
estudiantes.iloc[1,]
```

```
Out[133… nombres      Karen
         edad            29
         pais          Perú
         educacion      Lic
         Name: 1, dtype: object
```

```
In [134…  # varias filas
          estudiantes.iloc[[2,3],]
```

Out[134…

	nombres	edad	pais	educacion
2	Karolayne	23	Brasil	Bach
3	Omar	26	Argentina	PhD

Nótese en los casos anteriores, que si no indicabas filas, tenías toda la fila; y que si no indicas columnas, vienen todas las columnas. Si solo quieres un valor:

```
In [135…  estudiantes.iloc[1,3]
```

```
Out[135… 'Lic'
```

CONSULTAS **en Data Frames:**

```
In [142…  # ¿Quién es el más viejo del grupo?
          estudiantes[estudiantes.edad==max(estudiantes.edad)].nombres
```

```
Out[142… 1    Karen
         Name: nombres, dtype: object
```

```
In [143…  # ¿Quién es el más joven del grupo?
          estudiantes[estudiantes.edad==min(estudiantes.edad)].nombres
```

```
Out[143… 0       Manuel
         2    Karolayne
         Name: nombres, dtype: object
```

```
In [144…  # ¿Quién tiene más de 25 y es de Perú?
          estudiantes[(estudiantes.edad>25) & (estudiantes.pais=='Perú')].nombres # parentesis requeridos
```

```
Out[144… 1    Karen
         Name: nombres, dtype: object
```

```
In [145…  # ¿Quién no viene de Perú?
          estudiantes[estudiantes.pais!="Perú"].nombres
```

```
Out[145… 0       Manuel
         2    Karolayne
         3         Omar
         Name: nombres, dtype: object
```

```
In [146…  # ¿Quién no viene de estos lugares?

          lugares=["Perú", "Brasil"]
          estudiantes[~estudiantes.pais.isin(lugares)]
```

Out[146…

	nombres	edad	pais	educacion
0	Manuel	23	Ecuador	Bach
3	Omar	26	Argentina	PhD

```
In [147…  # Muestrame el DF ordenado decrecientemente por edad**

          estudiantes.sort_values(by=toSort,ascending=Order)
```

```
Out[147…
```

	nombres	edad	pais	educacion
1	Karen	29	Perú	Lic
3	Omar	26	Argentina	PhD
0	Manuel	23	Ecuador	Bach
2	Karolayne	23	Brasil	Bach

In [148…

```
# Muestrame el DF ordenado crecientemente, por educacion y luego por edad

toSort=["educacion","edad"]
Order=[True,True]
estudiantes.sort_values(by=toSort,ascending=Order)
```

Out[148…

	nombres	edad	pais	educacion
0	Manuel	23	Ecuador	Bach
2	Karolayne	23	Brasil	Bach
1	Karen	29	Perú	Lic
3	Omar	26	Argentina	PhD