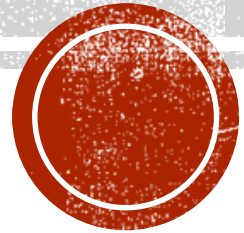
ESTRUCTURAS DE DATOS FUNDAMENTALES DE PYTHON

Juan Manuel Reyes



LISTAS

- Existen varios tipos de datos compuestos, los cuales son utilizados para agrupar varios valores en un solo lugar.
- En Pyhton el más versátil es la Lista, que puede escribirse como una secuencia de valores separados por coma encerrados por corchetes.

```
cuadrados = [1, 4, 9, 16, 25]
print (cuadrados)
```

- Las listas están indexadas, es decir, se puede acceder a cada posición utilizando un índice.

```
print (cuadrados[3])
print (cuadrados[0])
```

- Las listas pueden ser rebanadas (sliced), se puede obtener una porción de ellas con el operador :

```
print (cuadrados[1:3])
```

OPERACIONES SOBRE LISTAS

- Cuando se rebana una lista, no son obligatorios los índice inicial ni final

```
print (cuadrados[2:])
print (cuadrados[:3])
print (cuadrados[:])
```

- Concatenación

```
cuadrados = cuadrados + [36, 49, 63]
print (cuadrados)
```

- Modificación de valores en una posición

cuadrados [7] = 64

- Agregar nuevos valores a la lista con la operación append

```
cuadrados.append(81)
cuadrados.append(100)
print (cuadrados)
```

MAS OPERACIONES SOBRE LISTAS

- Es posible cambiar valores de porciones de una lista

```
letras = [‘a’,‘b’,‘c’,‘d’,‘e’,‘f’,‘g’]
print (letras)
letras[2:5] = [‘C’,‘D’,‘E’]
print (letras)
letras[1:4] = []
print (letras)
letras[:] = []
print (letras)
```

- De manera similar a las cadenas de texto (strings), podemos conocer la cantidad de elementos con la función ***len***

```
print (len(letters))
print (len(cuadrados))
```

- También es posible anidar listas

```
a = [0,1,2,3,4]
b = [5,6,7,8,9]
numbers = [a,b]
print (numbers)
```

EL CICLO FOR

- La clausula for es utilizada en Python para iterar sobre cualquier secuencia (lista o cadena de texto, por ejemplo)

```
animales = [“gato”,”perro”,”loro”]
for animal in animales:
    print (animal, len(animal))
```

- Se puede combinar su uso con la función ***range*** que genera progresiones aritméticas

```
for i in range(6):
  print (i)

for i in range (5,10):
  print (i)

print (list(range(3,2,12)))
```

EJERCICIOS

- Escriba un programa que permita leer n números enteros, luego lea un (1) número y diga cuántas veces se encuentra dicho número en el listado inicial

- Escriba un programa que dada una cadena de texto, imprima la misma cadena pero al revés

- Escriba un programa que imprima la siguiente sucesión de números:

0, 1, 1, 2, 3, 5, 8, 13, …, n

Para todos los números de la sucesión menor que n

- Escriba un programa que dado un número n, cree una matriz (lista de listas) con los valores de 1 a n^2

```
Por ejemplo si se lee n = 3

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

CADENAS DE TEXTO COMO LISTAS

- Las cadenas de texto pueden ser tratadas como listas

```
saludo = “Hola a todos”
print (saludo[0])
print (saludo[3])
print (saludo[len(saludo)-1])
print (saludo[1:5])
print (saludo[-3])
print (saludo[:-1])
```

- Sin embargo, a diferencia de las listas, las posiciones de una cadena son inmutables

```
frase = “Python nació en 1991”
frase[0] = “P”
frase[-1] = 2
frase[:6] = “C++”
```

TUPLAS - DEFINICIÓN

- Son tipos de datos de la categoría *secuencias* en Python, similares a las listas

```
tupla = 34,58,63
print (tupla)
```

- Son representadas rodeadas por paréntesis, en lugar de corchetes como las listas

```
persona = (“Ana”,”Paz”,35)
print (persona)
```

- Soportan la concatenación

```
tupla = tupla + (87,45)
```

- Los elementos de una tupla pueden ser otras tuplas o valores de cualquier otro tipo de dato

```
familia = (
    (“Luis”,”padre”,”amarillo”),
    ("Felipe","hijo",True),
    ("Doris","prima",[5,6,7,8])
)
```

- Puedes acceder a las posiciones de una tupla con corchetes igual que las listas y cadenas de texto

```
print(persona[1]); print(familia[2][2])
```

LAS TUPLAS SON INMUTABLES

- Al igual que las cadenas de texto, las tuplas también son inmutables

```
persona[0] = “Ana María”
familia[0][1] = “PADRE”
```

- Las posiciones en una tupla no se pueden modificar, pero si las posiciones de un elemento mutable (como las listas) al interior de una tupla

```
gastos = ([150,320,474],[42,86])
gastos[1][0] = 56
```

- Asignar tuplas simples sin paréntesis

```
x = 1,2,3,4 #packing
```

- Con un solo elemento

```
y = (8)
z = 8,
len(y)
len(z)
```

- Unpacking es posible también

```
a,b,c,d = x
```

EJERCICIOS

- Escriba un programa en Python que pregunte la cantidad de personas cuyos datos se leerán, luego los datos de cada persona así: nombre completo y fecha de nacimiento (esta información debe ser almacenada en una lista de tuplas), y luego calcule la edad de cada una de las personas en años, meses y días (esta información debe quedar también en una lista de tuplas).

- Escriba un programa en Python que permita tomar la asistencia en un salón de clase de la siguiente manera: pregunte por el nombre de cada estudiante y la hora a la que llegó a clase (hora y minuto), debe guardar esta información en una lista de tuplas y luego debe indicar el nombre del estudiante que llegó primero a clase y del estudiante que llegó de último a clase (si hay empate, del primero que fue registrado, en cada caso).

CONJUNTOS

- Es una colección de elementos no repetidos donde el orden no es importante

```
cesta = {'Limón','Naranja','Limón'}
```

- Verificar si un elemento está en un conjunto

```
'Naranja' in cesta
```

- Función ***set*** también es posible con algunos tipos (listas, cadenas, tuplas)

```
codigos = set("abcdefg")

codigos = set(range(20))
```

- Podemos utilizar las operaciones clásicas de conjuntos

```
a = set("abcdefghi")

b = set("acegijklm")

print (a|b); print (a&b)

print (a^b); print (a-b)
```

- Iterar sobre los elementos de un conjunto

```
for elem in a:

  print(elem)
```

DICCIONARIOS

- Son conocidos en otros lenguajes también como arreglos asociativos
- A diferencia de las secuencias, que son arreglos indexados por números, los diccionarios están indexados por una clave que puede ser de cualquier tipo inmutable, por ejemplo cadenas de texto

```
casa = {‘color’:’rojo’,’area’:500}
print (casa)
print (casa[‘color’])
print(list(casa))
```

```
‘color’ in casa
‘área’ not in casa
```

```
producto = dict([(‘nombre’,’silla’),
(‘precio’,24500.0),(‘cantidad’,12)])
```

```
vuelo = dict(origen=‘Cali’,
destino=‘Bogotá’, distancia=350)
```

```
vuelo.items()
```

RECORRER DICCIONARIOS

```
salud = {‘peso’:62,
’altura’:1.72,’edad’:28}
```

```
for clave in list(salud):
  print(clave, salud[clave])
```

```
for clave, valor in salud.items():
  print(clave, valor)
```

```
for clave in salud:
  print(clave, salud[clave])
```

Funciones y Estructuras de Datos en Python

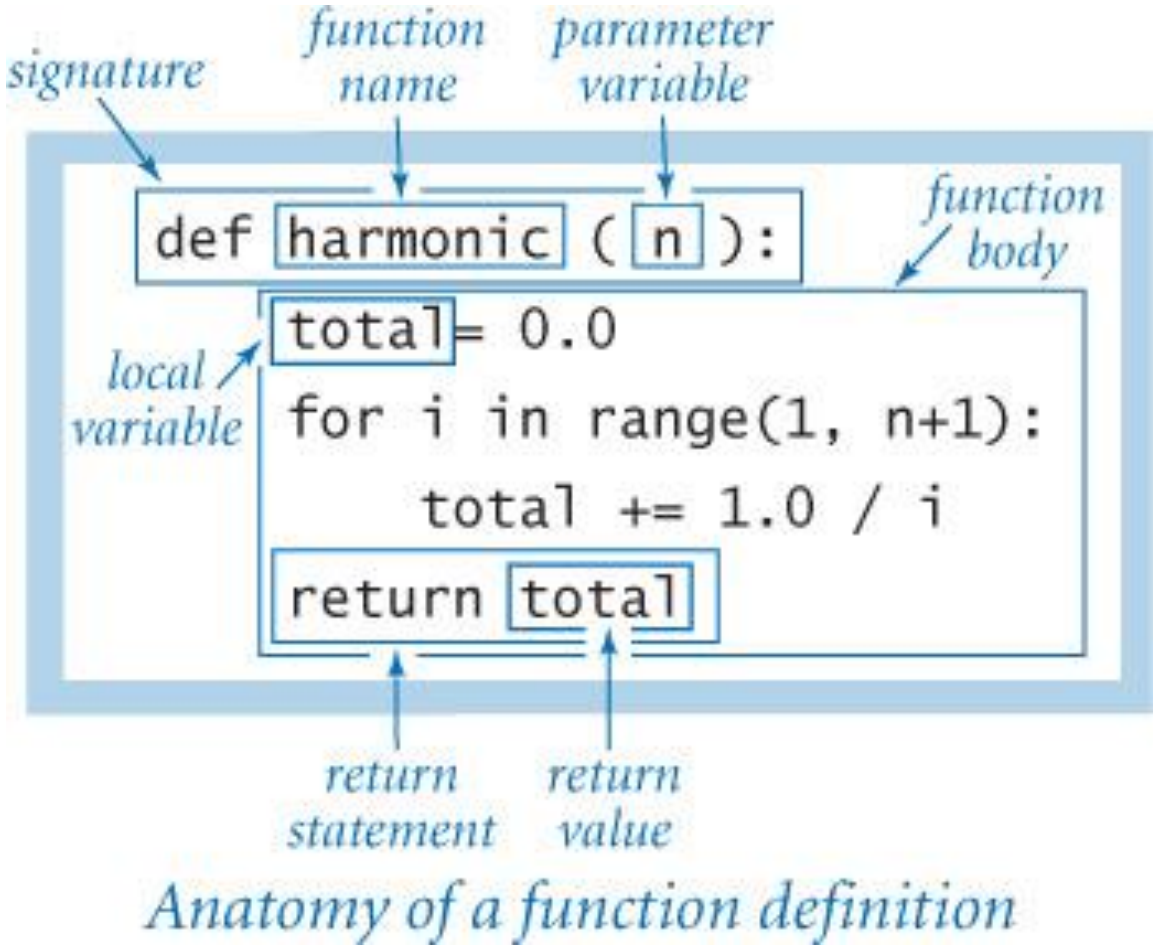
Declaración e invocación de funciones

```
def multiplicacion(x,y):
        z = x*y
        return z

a = multiplicacion(4,7)
print("El resultado es:",a)

def sumatoria(inicio, fin):
        suma = 0
        for valor in
range(inicio,fin+1):
                suma = suma+valor
        return suma

s = sumatoria(3,10)
print("La sumatoria es:",s)
```



Anatomy of a function definition

Funciones en Python


```
producto = multiplicacion
p = producto(9,3)
print(“El resultado es:”,p)
```

```
def suma(a,b): return a+b

def operarPrimerosN(op,n,valorI):
        resultado = valorI
        for i in range(1,n+1):
                resultado =
op(resultado,i)
        return resultado

x = operarPrimerosN(suma,5,0)
y = operarPrimerosN(producto,5,1)

print("Con suma: ",x)
print("Con producto: ",y)
```

Función map

```
def doblar(x): return 2*x

lista = [2,1,4,9,3,7,5]

dl = map(doblar,lista)

for x in dl:
        print(x,end=" ")
```

La función **map** aplica la función pasada como primer parámetro a cada uno de los elementos del iterable pasado como segundo parámetro.

El resultado retornado es un iterable map.

Para manejar con mayor facilidad el resultado del map, es común pasarlo a la función **list** para convertirlo en una lista.

```
def cuadrado(x): return x*x
c = list(map(cuadrado,range(10)))
print(c)
```

```
vals = ['1','5','8','4']
vals = list(map(int,vals))
print(vals)
```

Ejercicio 1

Escriba un programa en Python que lea de la entrada estándar una secuencia de n valores enteros separados por espacios e indique cuál es el mayor de todos.

Ejercicio 2

Escriba un programa en Python que lea de la entrada estándar una secuencia de n valores enteros separados por espacios e imprima en una sola línea separados por un espacio los elementos ingresados en la secuencia pero sin repetir. También imprima la cantidad de elementos impresos. El orden de salida no es importante.

Utilice **conjuntos** de Python para resolver este problema.

Ejemplo:

Entrada:

2 4 2 6 6 8 5 3 6 3 1 4 3 5 1 4 6 3 0 4

Salida:

2 4 6 8 5 3 1 0

8

Ejercicio 3

Escriba un programa en Python que lea de la entrada estándar una secuencia de n valores enteros separados por espacios e imprima sin repetir una línea con el par de valores: número y cantidad de veces que se repite. También imprima la cantidad de líneas impresas.

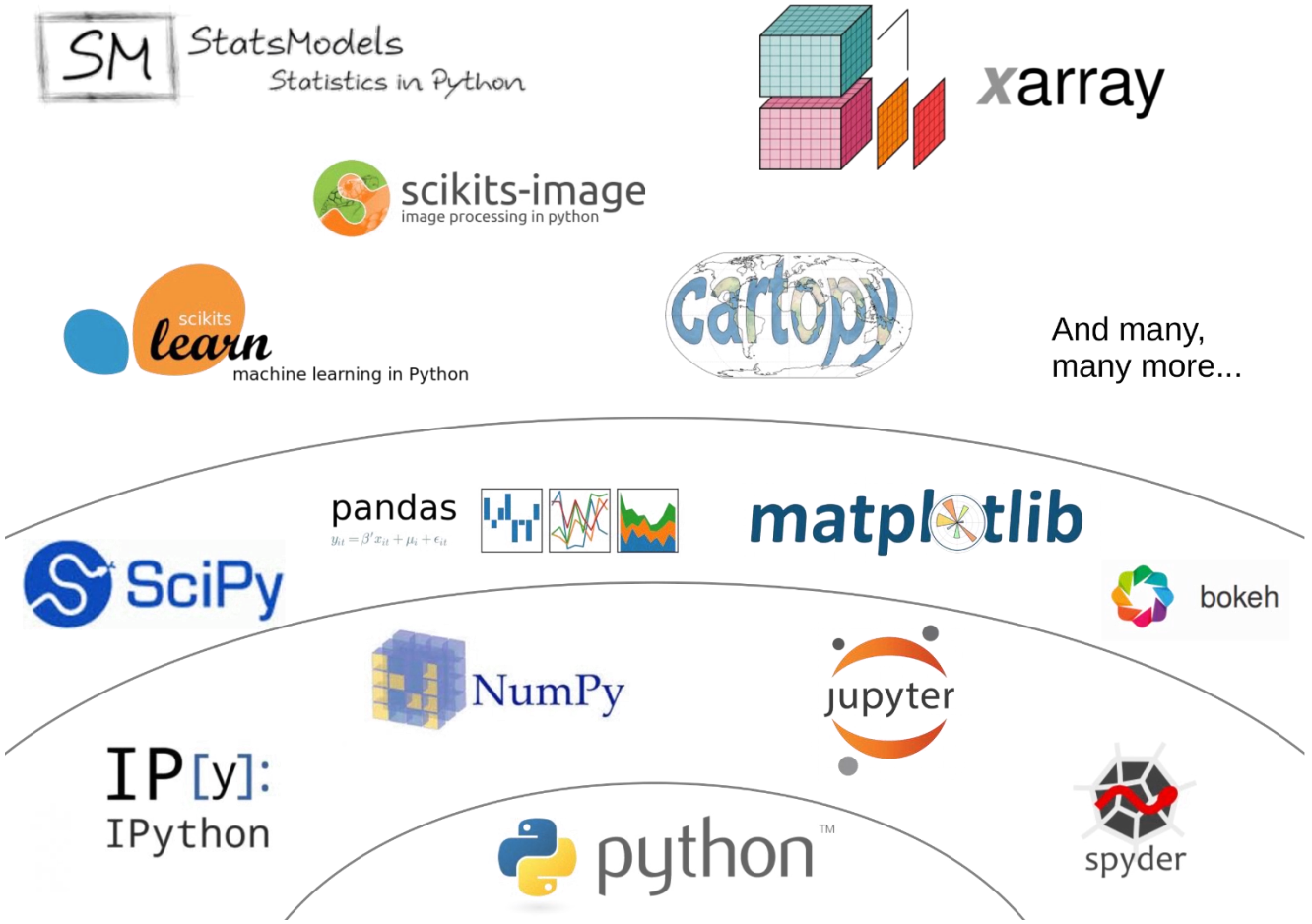
Utilice **diccionarios** de Python para resolver este problema.

Ejemplo:
Entrada:
2 4 2 6 6 8 6 4 4 6 4 4

Salida:
2 2
4 5
6 4
8 1
4

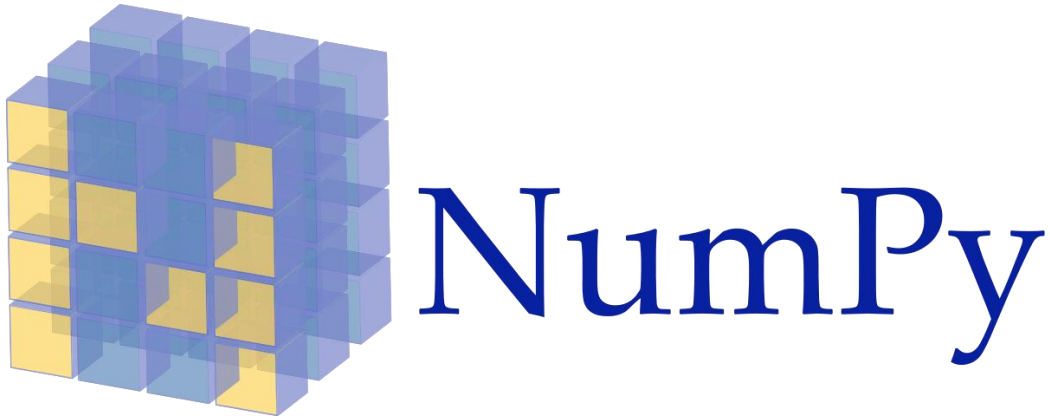
NumPy

Ecosistema SciPy



NumPy

- Es el paquete fundamental para computación científica de datos con Python



NumPy

- El principal objeto de NumPy es el **arreglo multidimensional homogéneo**.
- Este arreglo es una tabla de elementos (usualmente números), todos del mismo tipo, indexados por una tupla de enteros no negativos.
- En NumPy, las dimensiones son llamadas ejes.
- Por ejemplo, el arreglo `[8, 2, 5]` tiene un eje. Ese eje tiene 3 elementos en él, por tanto decimos que tiene un largo de 3.
- Ahora otro ejemplo de dos dimensiones, o mejor 2 ejes. El primer eje tiene largo 2 y el segundo largo 3.

```
[[ 1., 0., 0.],
 [ 0., 1., 2.]]
```

ndarray

- Lo primero que debemos hacer para utilizar NumPy es importar la librería que ya viene instalada como parte del ScyPy stack en Anaconda.

```
import numpy as np
a = np.array([2,3,4])
print(a)
print(a.dtype)
print(a.dtype.name)
print(type(a))
print(a.itemsize)
print(a.size)
print(a.ndim)
print(a.shape)
```

Secuencias de Secuencias

- La función **array** de la librería NumPy transforma secuencias de secuencias en arreglos de dos dimensiones. Secuencias de secuencias de secuencias en arreglos de tres dimensiones y así.

```
b = np.array([(3.2, 8, 2), (5, 9.4, 6.1)])
print(b)

c = np.array([[3,2,1],[9,8,7]], dtype='float64')
print(c)
d = np.array([(2,4,6),(1,3,5)], dtype=complex)
print(d)
```

Arreglos con Valores por Defecto

- Podemos crear un arreglo de las dimensiones que indiquemos, inicializado en zeros, unos, vacío o valores aleatorios.

```
e = np.zeros((3,4))
print(e)
f = np.ones((3,2,4), dtype=np.int16)
print(f)
g = np.empty((5,4))
print(g)
h = np.random.random((2,4))
print(h)
```

Modificar dimensiones de un arreglo

```
l = np.array([3,9,1,4,10,6,12,8,2,5,11,7])
m = l.reshape((3,4))
print("l=%s"%l)
print("m=%s"%m)


o = m.reshape((2,6))
print(o)


p = m.reshape((2,3,2))
print(p)
```

arange

```
q = np.arange(15)
print(q)
r = np.arange(20).reshape(4,5)
print(r)
s = np.arange(24).reshape(2,6,2)
print(s)
t = np.arange(10000)
print(t)
u = np.arange(10000).reshape(100,100)
print(u)
```

```
import sys
v = np.arange(10000)
print(v)
np.set_printoptions(threshold=sys.maxsize)
print(v)
```

arange y linspace

```
w = np.arange(10,30,5)
print(w)
x = np.arange(0,2,0.3)
print(x)
```

```
y = np.linspace(0,2,9)
print(y)
z = np.linspace(0,2*np.pi,100)
print(z)
```

```
ss = np.sin(z)
print(ss)
```

Operaciones Básicas

```
a = np.array([100,200,300,400,500])
b = np.array([500,400,300,200,100])
c = a-b
print(c)

d = np.arange(10)**2
print(d)

e = 10*np.cos(a)
print(e)

f = a<35
print(f)
```

Mas Operaciones

```
A = np.array([[1,1],[0,1]])
B = np.array([[2,0],[3,4]])


C = A * B
print(C)


D = A @ B
print(D)


E = A.dot(B)
print(E)
```

*=

+=

-=

```
a = np.ones((2,3), dtype=int)
a *= 3
print(a)

b = np.random.random((2,3))
print(b)
b += a
print(b)

a += b


c = a + b
print(c)
```

Operaciones Unarias Frecuentes

```
a = np.random.random((3,4))
print(a)

print(a.sum())

print(a.min())

print(a.max())
```

```
z = np.arange(4)
print(np.exp(z))
print(np.sqrt(z))
```

Operaciones por Ejes

```
b = np.arange(30).reshape(5,6)
print(b)

print(b.sum(axis=0))

print(b.min(axis=1))

print(b.cumsum(axis=1))

print(b.cumsum(axis=0))
```

Indexado y Slicing

```
a = np.arange(10)**3
print(a)

print(a[2:5])
print(a[3:7:2])

a[1:4] = -1
print(a)

print(a[::-1])

a[2:9:3] = 1010
print(a)
```

Iterando los arreglos

```
a = np.arange(0,20,2)

for elem in a:
  print(elem**2, end=“, ”)


b = np.arange(3,301,3).reshape(10,10)
for fila in b:
  print(“fila: %s”%fila)


for elem in b.flat:

  print(elem)


print(b.flat)

print(b.ravel())
```

Indexado de arreglos multidimensionales

```
a = np.random.random(15).reshape(5,3)
print(a)
print(a[2,1])
print(a[1:4,0:2])
```

```
def f(x,y):
  return x+y
```

```
b = np.fromfunction(f,(6,7),dtype=int)
print(b)
print(b[:,2])
```

Ejercicio

- Escriba un programa en Python, utilizando la función fromfunction de NumPy para generar una matriz (o arreglo bidimensional) de n filas x m columnas, que intercale los valores 1 y 0 de la siguiente forma:

0 1 0 1 0 1 …
1 0 1 0 1 0 …
0 1 0 1 0 1 …
…

El valor de n y m deben ser leídos a través de input.