Introducción a python

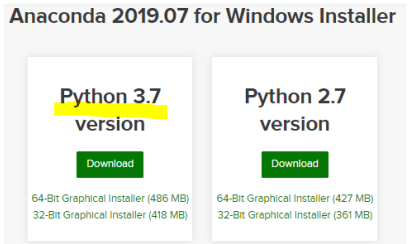
Dra. Ana Lidia Franzoni

Notas tomadas de la Mtra. Teresa Sóla

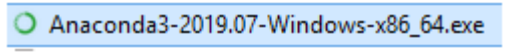
Instalación Anaconda Navigator

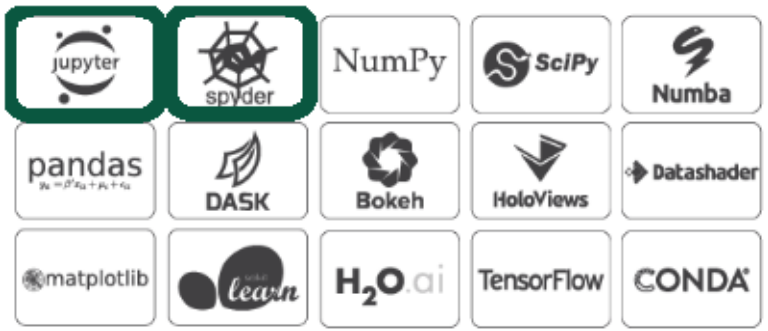
INSTALAR LA DISTRIBUCIÓN ANACONDA.
HTTPS://WWW.ANACONDA.COM/DISTRIBUTION/

SEGUIR LAS INDICACIONES DE LA PÁGINA, INSTALAR LA VERSIÓN 3 DE **PYTHON**.



EJECUTAR EL ARCHIVO DESCARGADO, SEGUIR LOS PASOS QUE INDICA EL INSTALADOR.

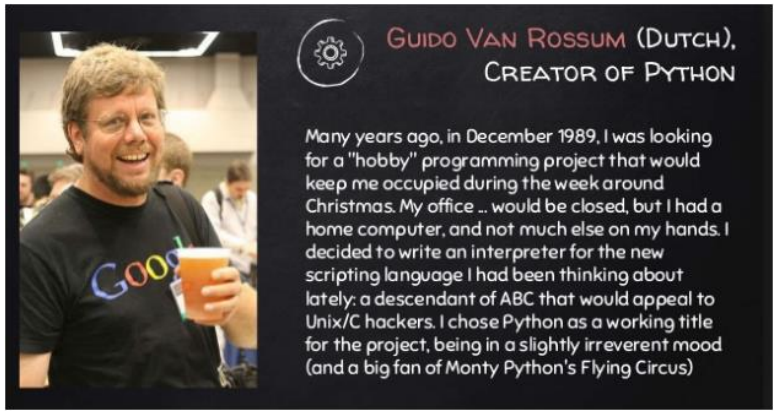




Introducción

- La distribución de Anaconda es utilizada por científicos de datos, ya que facilita la administración de entornos de **Python** y **R**, permite administrar varias versiones, cuenta con una gran colección de librería que permiten:
  - Procesamiento de grandes volúmenes de información
  - Análisis predictivo
  - Manejo de arreglos
  - Aprendizaje de máquina
  - Visualización

- En el curso utilizaremos los IDESs: Notebook jupyter y Spyder.
- Las librerías de matplotlib, pandas, seaborn.

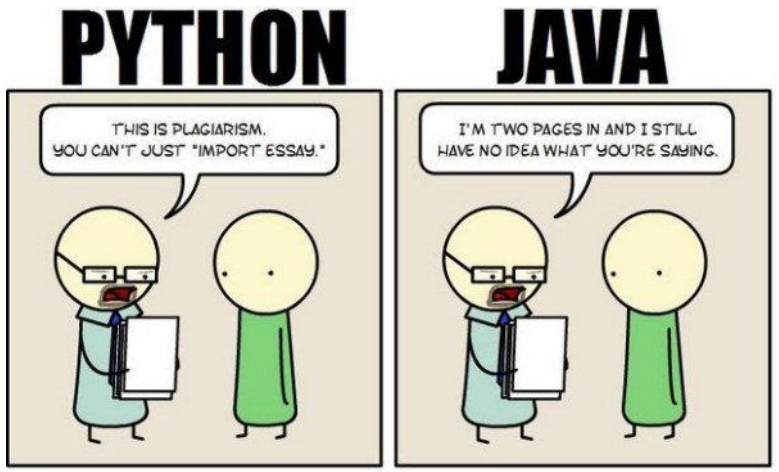


Introducción

- **Python** es un lenguaje de programación de alto nivel, creado por Guido van Rossum en los 90’s.

Introducción

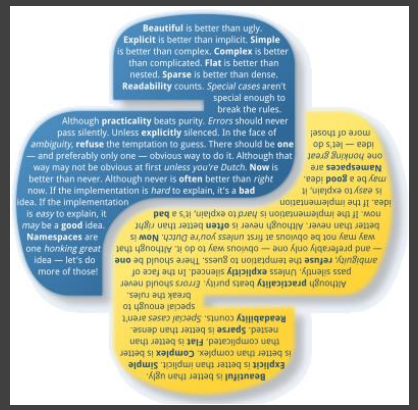
- **Python** es un lenguaje de programación:
  - de código abierto,
  - intuitivo,
  - que enfatiza la legibilidad,
  - limpieza del código,
  - interpretado,
    - interactivo,
    - script,
  - multiparadigmas (POO, funcional, imperativa),
  - multiplataforma.



Introducción

- Python utiliza:
  - sangrías de espacios blancos,
  - dos puntos **:** para delimitar el inicio de un bloque,
  - distingue entre mayúsculas y minúsculas
  - los objetos que maneja son:
    - mutables e inmutables.
- Python dejó de utilizar:
  - las llaves **{}** como delimitadores de bloques,
  - el punto y coma **;** como terminación de sentencias.

Introducción



La documentación del lenguaje, su implementación y de los módulos de la biblioteca estándar pueden consultarse en las ligas.

https://docs.python.org/3.6/reference/

https://docs.python.org/3.6/library/

La filosofía central se resume en “*The Zen of Python*”

https://www.python.org/dev/peps/pep-0020/

También lo puedes desplegarlo así:

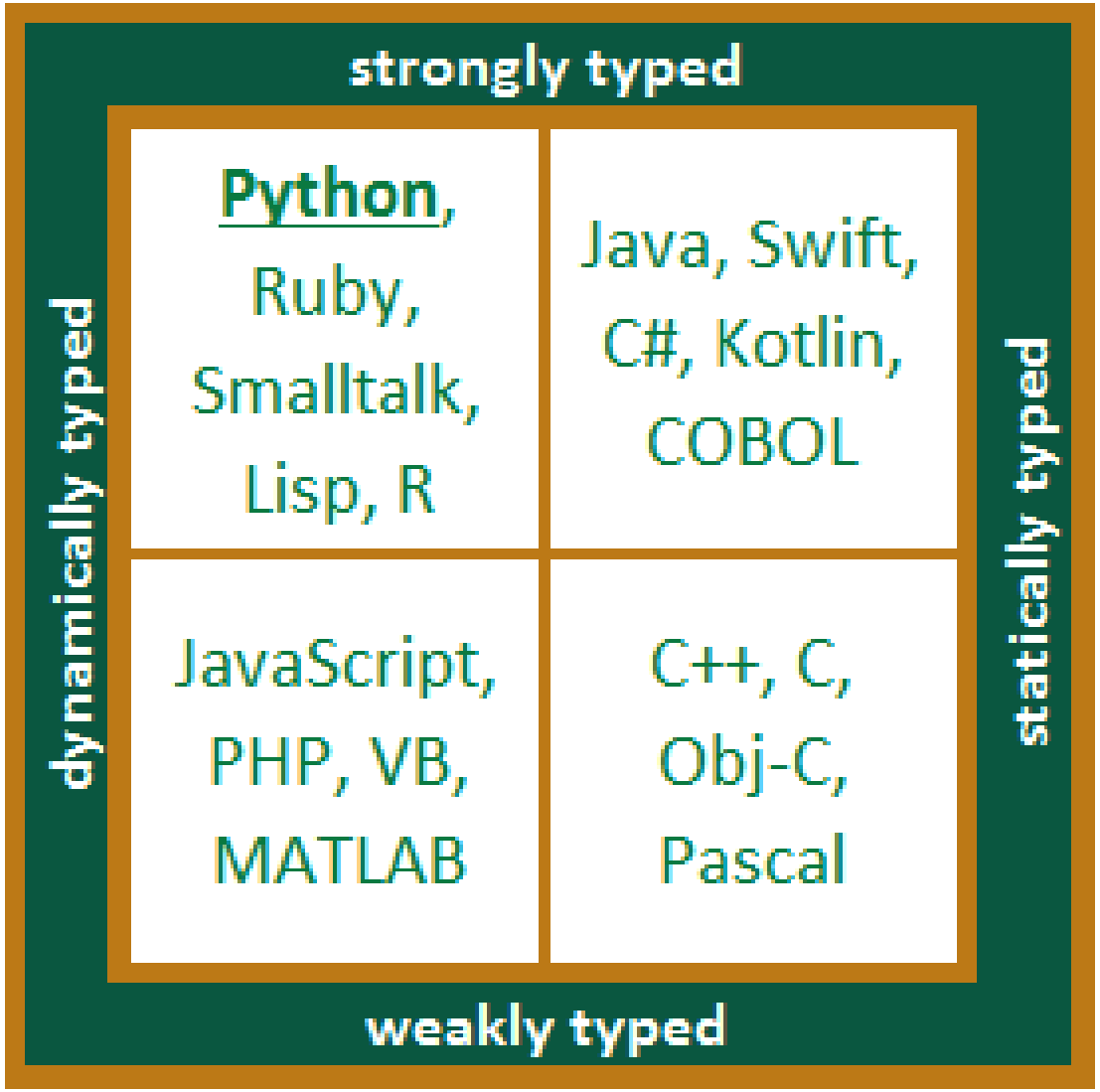
```
import this
```

Introducción

- **Desventajas**
  - Errores en tiempo de ejecución, ya que es un lenguaje interpretado.
  - Es lento en ejecución (comparado con lenguajes compilados).
  - Alto consumo de memoria.
  - Capa de acceso a BD menos desarrolladas que para otros lenguajes.
  - No es el lenguaje ideal para aplicaciones multiproceso.

- **Ventajas**
  - Lenguaje de alto nivel, fácilmente integrable con código en C, C++.
  - Extensa colección de librerías.
  - Fácil de aprender y programar.
  - Portable.
  - Se puede programar fácilmente prototipos que puede crecer a producción.
  - Es un lenguaje popular, cuenta con buen soporte de la comunidad.
  - Gratuito.
  - Estructuras de datos fáciles de usar.
  - IoT con Raspberry Pi.
  - Grandes compañías que utilizan **Python**:
    - DropBox, Google, Yelp, Spotify, Netflix, Instagram, etc.

Introducción



```
help("keywords")
```

"There are only two hard things in Computer Science: cache invalidation and naming things"

– Phil Karlton

Reglas de nombres de variables

- Los nombres de las variables y funciones son case sensitive.
- Los caracteres permitidos para nombrar variables son:
  - mayúsculas (**A** a la **Z**),
  - minúsculas (**a** a la **z**),
  - dígitos de (**0** al **9**) y
  - el guión bajo (_).
- No utilizar símbolos especiales como:
  - !, @, #, #, {, }, [,], (,), +, -, *, /, etc., ni espacio en el nombre de la variable.
- El primer carácter no puede ser numérico.
- La lista de palabras reservadas se pueden despliegar con la siguiente instrucción:

Operadores y precedencia

- Aritméticos: +, -, *, /, //, %, **
- Concatenación: *, +
- Comparación o igualdad: <, >, <=, >=, ==, !=
- Lógicos: and, or, not
- Pertenencia: in, not in
- Identidad: is, is not
- Asignación aumentada: +=, -=, *=, /=, %=, //=

Precedencia de operadores

- (), [], {}
- **
- - + (unario)
- *, /, %, //
- +, -
- <, >, <=, >=
- ==, !=
- +=, -=, *=, /=, %=, //=
- is, is not
- in, not in
- not
- and
- or

```
# -*- coding: utf-8 -*-
"""
Editor de Spyder

Este es un archivo temporal.
"""
print("Hello Anaconda")

x=10
y=12.5
z='100'
type(x)
```

Nombre	Tipo	Tamaño	
a	int	1	17
b	int	1	3
cociente	int	1	5
division	float	1	5.66666
residuo	int	1	2

```
a = 17
b = 3
division = a/b
cociente = a//b
residuo= a%b
```

Todo es un objeto en Python

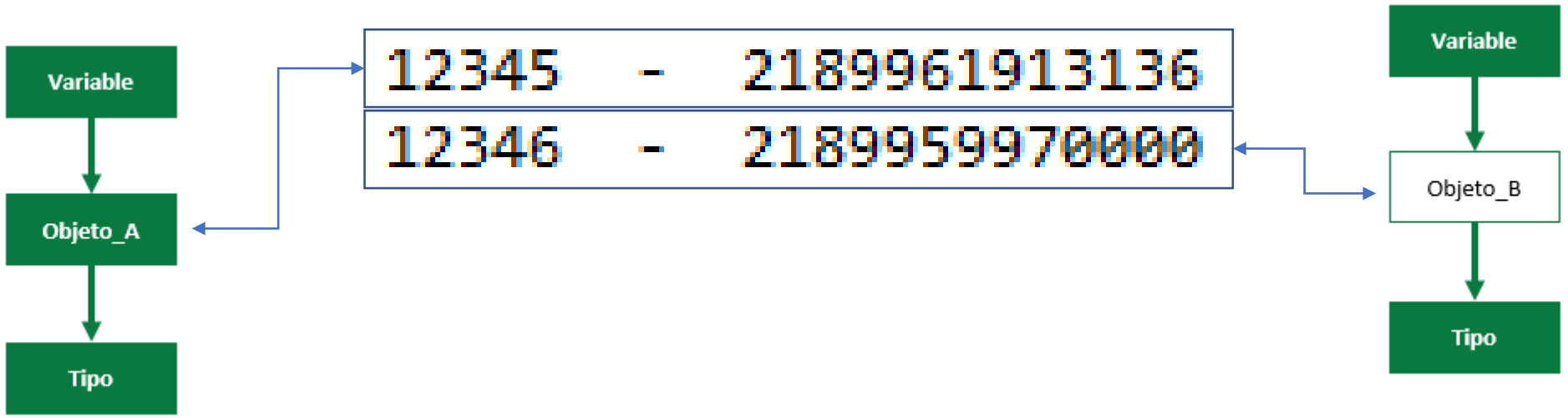
Todos las variables de **Python** son objetos de alguna clase, por lo tanto poseen métodos y propiedades.

Cada vez que se modifica el valor se crea un objeto nuevo.

Todo es un objeto en Python

```
36 # todo es un objeto
37 miX = 12345
38 print( miX, ' - ', id(miX) )
39
```

```
40 miX = miX + 1
41 print( miX, ' - ', id(miX) )
42
```



Tipos de datos iterables - (str)

- Textos (str) delimitador comillas simples o compuestas ‘’ o “”

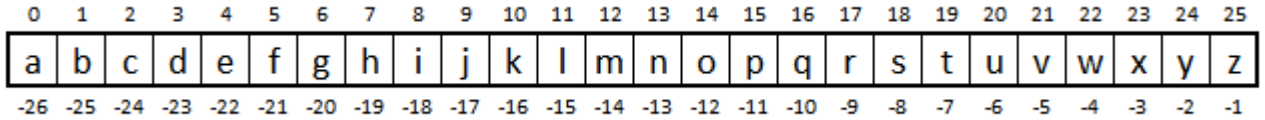
  También conocido como cadena de caracteres. Es una secuencia de caracteres utilizada para almacenar caracteres alfanuméricos y símbolos.

  - Es **inmutable**, no permite hacer modificaciones de los elementos directas a los elementos.
  - Contiene métodos asociados para la manipulación de strings. Se puede acceder a ellos fácilmente desde jupyter tecleando un punto seguido de un tab.

```
44 #cadenas STR
45 miStringA = 'hola ITAM'
46 miStringB = miStringA
47
48 miStringA = 'hola de Nuevo!'
49
50
51 miStringA =miStringA.replace('h','¡H')
52 print(miStringA)
53
```

Nombre	Tipo	Tamaño	
miStringA	str	1	¡Hola de Nuevo!
miStringB	str	1	hola ITAM

Tipos de datos iterables - (str, list, tuple)




```
1  cad = 'abcdefghijklmnopqrstuvwxyz'
2  print( cad   )                  abcdefghijklmnopqrstuvwxyz
3  print( len(cad)  )              26
4  print( cad[5]  )       # [5]    f
5  print( cad[2:5]  )     # [2,5)  cde
6  print( cad[:5]  )      # [0:5)  abcde
7  print( cad[5:]  )      # [5:final]  fghijklmnopqrstuvwxyz
8  print( cad[1::2]  )    # [1:final:step 2]  bdfhjlnprtvxz
9
10 print( cad[-26] )      # [long-26] == [0]  a
11 print( cad[-1]  )      # [long-1]  == [25]  z
12 print( cad[-2:]  )     # [long-2:final] == [24:final]  yz
13
14 print( cad[:-2]  )     # [inicio:long-2]  == [0:24)  abcdefghijklmnopqrstuvwx
15 print( cad[::-1]  )    # [0:final] orden inverso  zyxwvutsrqponmlkjihgfedcba
16 print( cad[5:2:-1]  )  # (2:5] orden inverso  fed
17 print( cad[-2:-5:-1] ) # (long-5:long-2] == (21:24] orden inverso  yxw
18 print( cad[-5:-2] )    # [long-5:long-2) == [21:24)  vwx
19 print( cad[::-2] )     # orden inverso step 2  zxvtrpnljhfdb
```

- Permite el acceso a porciones (str, list, tuple) usando slicing. **cad[start:stop -1:step]**
- La (str, list, tuple) puede ser recorrida hacia adelante (índices positivos) o hacia atrás (índices negativos).
- Inicia en el índice 0.

(list) - Tipos de datos iterables

- Listas (list) delimitador []
  - Es una secuencia ordenada de elementos heterogéneos.
  - Es **mutable**; puede ser modificado sin crear un nuevo objeto.
  - Sus elementos se pueden acceder por el índice.

    - El tamaño de las lista no es fijo.
    - Usa el “slicing” para acceder a los elementos.
    - Contiene métodos asociados para la manipulación de listas

```
strA = 'AAAA'
print('1)', strA)
lstA = ['a','e','i','o','u']
print('2) ', lstA)
lstB = [1000, lstA, 10, 20, strA]
print('3) ', lstB)


strA = '------->ZZ'
lstA[3] = 'Python'
print('4)', strA)
print('5) ', lstA)
print('6) ', lstB)
```

```
1) AAAA
2)  ['a', 'e', 'i', 'o', 'u']
3)  [1000, ['a', 'e', 'i', 'o', 'u'], 10, 20, 'AAAA']
4) ------->ZZ
5)  ['a', 'e', 'i', 'Python', 'u']
6)  [1000, ['a', 'e', 'i', 'Python', 'u'], 10, 20, 'AAAA']
```

Tipos de datos iterables - (tuple)

- Tupla (tuple) delimitador (), aunque basta con la coma para construirlas.
  - Es **inmutable** y está compuesta por una secuencia de elementos ordenados e inmutables o listas.
  - Se pueden acceder sus elementos usando el índice o mediante “slicing”, pero no pueden ser modificados.
  - Una lista dentro de la tupla es mutable.

```
tupla = (1, 'python', True)
print( type(tupla) )
print( tupla[1] )
print(tupla)
```

```
<class 'tuple'>
python
(1, 'python', True)
```

```
otraLista = list(tupla)
print(type(otraLista))
```

```
<class 'list'>
```

```
miX = 123
miLista = [1,2,3]
miTupla = ('a', 789)
miTupla_2 = (miX, 'abcde', 10, miLista, miTupla)
print( miTupla )
print( miTupla_2 )
```

```
('a', 789)
(123, 'abcde', 10, [1, 2, 3], ('a', 789))
```

```
miX = 10
miLista[1] = 100
miTupla = (1,2,3,4,5)
print( miTupla )
print( miTupla_2 )
```

```
(1, 2, 3, 4, 5)
(123, 'abcde', 10, [1, 100, 3], ('a', 789))
```

(set) - Tipos de datos iterables

- Conjunto (set) delimitador {}
  - Estructura de datos, no ordenados que no permite elementos repetidos, los datos son **mutables** (set), solo acepta objetos inmutables como elementos.
  - Permite la operaciones entre conjuntos mediante operadores o métodos.

```
conj2 = set(range(2,21,2))
conj3 = set(range(3,21,3))

print( conj2 )
print( conj3 )
```

```
{2, 4, 6, 8, 10, 12, 14, 16, 18, 20}
{3, 6, 9, 12, 15, 18}
```

```
lstA = [1,2,3,1,5,3,3,2,2]
set(lstA)
```

```
{1, 2, 3, 5}
```

```
15 in conj2
```

```
False
```

```
conj3 | conj2 # union
```

```
{2, 3, 4, 6, 8, 9, 10, 12, 14, 15, 16, 18, 20}
```

```
conj3  & conj2 #intersección
```

```
{6, 12, 18}
```

```
conj3 - conj2 # diferencia
```

```
{3, 9, 15}
```

```
conj2 - conj3 # diferencia
```

```
{2, 4, 8, 10, 14, 16, 20}
```

```
conj2 ^ conj3 # diferencia simétrica
```

```
{2, 3, 4, 8, 9, 10, 14, 15, 16, 20}
```

Tipos de datos iterables - (dict)

```
miDicc = {'texto':'llave texto', True:'llave booleana', 123:'llave numérica',(1,2):'llave tupla'}
miDicc
```

```
{'texto': 'llave texto',
 True: 'llave booleana',
 123: 'llave numérica',
 (1, 2): 'llave tupla'}
```

```
miDicc[True]
```

```
'llave booleana'
```

```
miDicc_2 = {'lista':[1,2,3,4,[10,100,1000]], 'conj':{1,2,3}, 'num':1000, 'bool':False, 'dicc':{'a':1, 'e':2}}
miDicc_2
```

```
{'lista': [1, 2, 3, 4, [10, 100, 1000]],
 'conj': {1, 2, 3},
 'num': 1000,
 'bool': False,
 'dicc': {'a': 1, 'e': 2}}
```

```
print(miDicc_2['lista'])
print(miDicc_2['lista'][3])
print(miDicc_2['lista'][4][2])
```

```
[1, 2, 3, 4, [10, 100, 1000]]
4
1000
```

```
print(miDicc_2['dicc'])
print(miDicc_2['dicc']['e'])
```

```
{'a': 1, 'e': 2}
2
```

- Diccionario (dict) delimitador { llave: valor }
  - Estructura iterable, donde la llave es **inmutable** y el valor **mutable**, tiene mecanismos de acceso utilizando la llave. Las llaves pueden número, cadenas de caracteres, booleanos o tuplas.
  - Las llaves son única en el diccionario, ya que son el mecanismo de acceso.

(dict) - Tipos de datos iterables

```
diccA = {'Camila':1.65, 'Xavier':1.85, 'Lola':1.70}
print(diccA)
print(diccA['Lola'])
```

```
{'Camila': 1.65, 'Xavier': 1.85, 'Lola': 1.7}
1.7
```

```
mean(list(diccA.values()))
```

```
1.7333333333333334
```

```
llaves = diccA.keys()
print(llaves)

valores = diccA.values()
print(valores)
```

```
dict_keys(['Camila', 'Xavier', 'Lola'])
dict_values([1.65, 1.85, 1.7])
```

```
diccB = {'Hugo':{'ayp':7}, 'Paco':{'hca':9, 'ideas':10, 'ayp':9}, 'Luis':{'hca':10, 'mate':9}}
diccB
```

```
{'Hugo': {'ayp': 7},
 'Paco': {'hca': 9, 'ideas': 10, 'ayp': 9},
 'Luis': {'hca': 10, 'mate': 9}}
```

```
print( diccB['Hugo'] )
print( diccB['Paco']['hca'])
```

```
{'ayp': 7}
9
```

```
[mean( list(diccB[llave].values()) ) for llave in diccB.keys()]
```

```
[8.5, 9.333333333333334, 10.0]
```

resumen - Tipos de datos iterables

Nombre del tipo de dato	Tipo en Python	Elementos	Mutable	Acceso
texto	str	alfanuméricos	no	sí (índices)
lista	list	de cualquier tipo	sí (índices, métodos)	sí (índices)
tupla	tuple	inmutables y listas	no, sólo sus elementos mutables	sí (índices)
conjunto	set	elementos inmutables	sí (métodos)	no
diccionario	dict	llave (tipo básico) o tupla valor (cualquier tipo de dato)	sí	por las llaves

Estructuras de control

1) Condicionales:

- Permiten ir eligiendo bloques de código a ser ejecutados, dependiendo de la condición
- **Nota**: respetar la indentación de los bloques es **MUY IMPORTANTE**, ya que **Python** dejó de usar las llaves **{}** como delimitadores de bloques. Los espacios en blanco adquieren importancia.

```
x = 100
if x < 10:
    print('menor a diez')
else:
    print('mayor o igual a diez')
```

```
mayor o igual a diez
```

```
x = 100
if x < 10:
    print('menor a diez')
elif x <= 100:
    print('mayor o igual a diez y menor o igual 100')
else:
    print('mayor a 100')
```

```
mayor o igual a diez y menor o igual 100
```

```
x = 100
if x < 10:
    print('menor a diez')
else:
    if x <= 100:
        print('mayor o igual a diez y menor o igual 100')
    else:
        print('mayor a 100')
```

```
mayor o igual a diez y menor o igual 100
```

Estructuras de control

2) Cíclicas:

- Permiten ejecutar bloques de código varias veces, dependiendo de la condición.
  - while (mientras la condición sea verdadera)
  - for (para cada elemento del iterador)
    - list comprehensions

```
x = 1
while x < 4:
    print('hola')
    x = x + 1
```

```
hola
hola
hola
```

```
lstB = [7,99,4,50]
[x*2 for x in lstB]
```

```
[14, 198, 8, 100]
```

```
lstA = list(range(1,13,1))
lstB = [7,99,4,50]

solA = [x for x in lstA if x in lstB]
print(solA)

solB = []
for x in lstA:
    if x in lstB:
        solB.append(x)
print(solB)
```

```
[4, 7]
[4, 7]
```