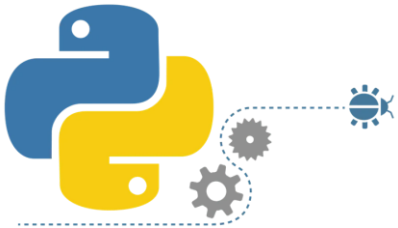
Curso FullStack Python

Codo a Codo 4.0



Clase 25

Python – Parte 3

Listas

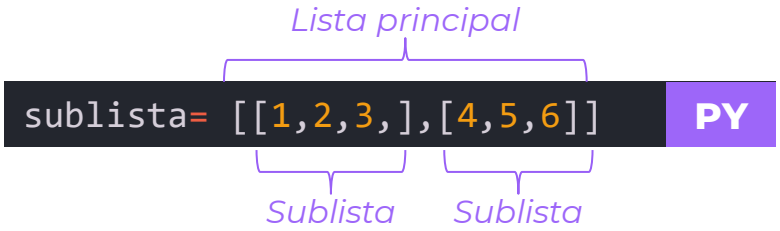
La lista es una secuencia ordenada de elementos. Pueden tener elementos homogéneos del mismo tipo de dato o combinar distintos tipos de datos, aunque por convención en las listas se guardan elementos homogéneos.

Crear listas

Se crean al asignar a una variable una secuencia de elementos encerrados entre corchetes [] y separados por comas. Los corchetes pueden estar juntos, creando una lista vacía.

```py
numeros= [1,2,3,4,5] #Lista de números
dias= ["Lunes", "Martes", "Miércoles"] #Lista de strings
elementos= [] #Lista vacía
```

Una lista puede contener, a su vez, otra lista:



Listas | Acceso por subíndice

- El acceso a los elementos se usa el subíndice, el primer elemento siempre lleva el subíndice cero.
- Usar un subíndice negativo hace que la cuenta comience desde atrás.
- Usar subíndices fuera de rango genera un error: *out of range*

5	7	4	4	5	6	2	6	1
0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
-9	*-8*	*-7*	*-6*	*-5*	*-4*	*-3*	*-2*	*-1*

Listas | Impresión

- Se pueden imprimir con un ciclo **while** o **for**.
- También pueden imprimirse directamente.

```py
numeros= [1,2,3,4,5]
print(numeros)
```

Listas | Operador In para iterar listas

- Podemos iterar utilizando el subíndice, utilizando un contador, **while** y **len** para generar la secuencia de índices:

PY

```
#Mostrar la lista, separando los elementos con un espacio
def MostrarLista(lista):
    i = 0
    while i < len(lista):
        print(lista[i], end=" ")
        i = i + 1
    print()

#Declaración de la lista y llamado a la función
lista=["A", "B", "C", "D", "E"]
MostrarLista(lista)
```

terminal

```
A B C D E
```

Listas | Operador In para iterar listas

- Podemos iterar utilizando el subíndice, for y range para generar la secuencia de índices:

```py
def SumarLista(lista):
    suma = 0
    for i in range(len(lista)):
        suma = suma + lista[i]
    return suma

#Declaración de la lista y llamado a la función
lista=[2,3,4,5,6]
print(SumarLista(lista))
```

```terminal
20
```

- Podemos iterar en forma directa los elementos de la lista, sin necesidad de generar la secuencia de subíndices. En este caso la variable ***i*** toma el elemento de la lista.

```py
vocales=['a','e','i','o','u']
for i in vocales:
    print(i, end="-")
```

```terminal
a-e-i-o-u-
```

Listas | Desempaquetado

- Consiste en asignar sus elementos a un conjunto de variables:

```py
dias= ["Lunes", "Martes", "Miércoles"]
d1, d2, d3 = dias
print(d1)
print(d2)
print(d3)
```

PY

terminal

```
Lunes
Martes
Miércoles
```

Listas | Operaciones

- Las listas pueden **concatenarse** con el operador **suma**:

```py
lista1= [1,2,3]
lista2= [4,5,6]
lista3= lista1 + lista2
print(lista3) #[1,2,3,4,5,6]
```

PY

Listas | Operaciones (continuación)

- La concatenación nos permite **agregar** elementos nuevos a la lista:

```py
lista=[3,4,5]
lista= lista + [6] #[3,4,5,6]
```

El elemento debe encerrarse entre corchetes para que sea una lista.

- Podemos **modificar** los elementos de una lista utilizando el subíndice:

```py
lista=[3,4,5]
lista[1]=7
print(lista) #[3,7,5]
```

- La función ***len()*** devuelve la cantidad de elementos de una lista:

```py
lista=[3,4,5,6]
print(len(lista)) # 4
```

Listas | Operaciones (continuación)

- La función ***max()*** devuelve el mayor elemento de una lista.
- La función ***min()*** devuelve el menor elemento de una lista.
- La función ***sum()*** devuelve la suma de los elementos de una lista:

```py
lista=[3,4,5,6]
print(max(lista)) # 6
print(min(lista)) # 3
print(sum(lista)) # 18
```

*Para utilizar **max(), min() y sum()** la lista debe ser homogénea*

- La función ***list()*** convierte cualquier secuencia a una lista. Se puede utilizar con rangos, cadenas y algunos más...

```py
lista= list(range(6))
print(lista) #[0,1,2,3,4,5]
```

- El operador ***in*** permite verificar la presencia de un elemento, mientras que la ausencia de un elemento se comprueba con ***not in***:

```py
lista=[3,4,5,6]
print(4 in lista) #True
print(8 in lista) #False
```

Devuelven True o False

Listas | Métodos

Un ***método*** es un procedimiento o función que pertenece a un objeto. Permiten manipular los datos almacenados en él y se escriben luego del nombre del objeto, separados por un punto.

- **append():** Agrega un elemento al final de la lista.

```py
lista=[3,4,5]
lista.append(6)
print(lista) #[3,4,5,6]
```

- **insert(<pos>, <elemento>):** Inserta un elemento en una posición determinada de la lista.

```py
lista=[3,4,5]
lista.insert(0,2) #Agrega en la posición 0 el número 2
print(lista) #[2,3,4,5]
lista.insert(3,25) #Agrega en la posición 3 el número 25
print(lista) #[2,3,4,25,5]
```

Listas | Métodos (continuación)

- **pop():** Elimina el último elemento de la lista.

```py
lista=[6,9,8]
lista.pop() # queda [6,9]
```

- **pop(<posición>):** Elimina un elemento en una posición determinada de la lista.

```py
lista=[3,4,5]
lista.pop(1) # queda [3,5]
```

Provoca un error si no existe

- **remove(<valor>):** Elimina un elemento en la lista, identificado por su valor.

```py
lista=[3,4,5]
lista.remove(3) # queda [4,5]
```

Provoca un error si no existe. Si está más de una vez elimina la primera aparición

- **index(<valor>):** Busca un valor y devuelve su posición.

```py
lista=[3,4,5]
print(lista.index(5)) #2
```

Provoca un error si no existe. Si está más de una vez devuelve la primera aparición

El método admite como argumento adicional un índice inicial a partir de donde comenzar la búsqueda (`lista.index(5,2)`), opcionalmente también el índice final (`lista.index(5,2,4)`).

Listas | Métodos (continuación)

- **count():** Devuelve la cantidad de repeticiones de un elemento, cero si no lo encuentra.

```py
lista=[3,4,5,3,5,8,5]
print(lista.count(5)) #El número 5 está 3 veces
print(lista.count(2)) #El número 2 no está en la lista
```

- **reverse():** Invierte el orden de los elementos de una lista.

```py
lista=[3,4,5]
lista.reverse()
print(lista) #[5,4,3]
```

Listas | Métodos (continuación)

- **sort ():** Ordena los elementos de la lista.

```py
lista=[5, 1, 7, 2]
lista.sort()
print(lista) #[1,2,5,7]
```

- **sort (reverse=True):** Con parámetro reverse en True, ordena la lista de mayor a menor.

```py
lista=[5, 1, 7, 2]
lista.sort(reverse=True)
print(lista) #[7,5,2,1]
```

- **clear():** Elimina todos los elementos de una lista.

```py
lista=[3,4,5]
lista.clear()
print(lista) #[]
```

Cadenas de caracteres

Una **cadena de caracteres (string)** es una secuencia de caracteres. Python soporta caracteres regionales como ñ y vocales con tilde.

Podemos declarar una cadena de caracteres con comillas simples o comillas dobles:

```py
cad1= "Lunes"
cad2= 'Martes'
```

Una constante de cadena extensa puede ser distribuida en varias líneas con barra invertida ****

```py
marca="Alfa Romeo"
frase= 'Dijo "me encanta programar" y comenzó a' \
    "estudiar ingeniería en sistemas"
```

También se permite utilizar otro tipo de comillas en la misma cadena

Las cadenas pueden concatenarse con el operador **+ (suma)**:

```py
nombre= input("Ingrese su nombre: ")
saludo= "Hola "+ nombre
print(saludo) #Hola Pedro
```

Cadenas de caracteres (continuación)

Para concatenar un numero a una cadena, primero se debe convertir a cadena **str(valor)**

```py
edad= 25
mensaje= "La edad es: "+ str(edad) # La edad es: 25
altura= 1.75
mensaje= "La altura es: "+ str(altura) # La altura es: 1.75
```

Replicación: Podemos replicar una cadena con ***

```py
risa='ja'
carcajada= risa*5
print(carcajada) #jajajajaja
asteriscos= "*"*10
print(asteriscos) #**********
```

Cadenas de caracteres (continuación)

Comparación: Se pueden comparar como cualquier otra variable con **==**

PY
```
pais= input("Ingrese su pais de nacimiento: ")
if pais == "Argentina":
    print("La nacionalidad es Argentino")
else:
    print("La nacionalidad NO es Argentino")
```

terminal
```
Ingrese su pais de nacimiento: argentina
La nacionalidad NO es Argentino
```

*Es **case sensitive**: distingue entre mayúsculas y minúsculas*

Subíndices: Se pueden acceder usando subíndices:

PY
```
cad="Martes"
print(cad[2])
```

terminal
```
r
```

*No se puede modificar usando subíndices, las cadenas **son inmutables**.*

Cadenas de caracteres (continuación)

Rebanadas: Se pueden manipular usando rebanadas:

```py
cad="Miércoles"
print(cad[1:5])
```

```terminal
iérc
```

Cadenas de caracteres: Funciones

Longitud: Se puede utilizar **len** para saber la longitud:

```py
cad="Miércoles"
print(len(cad))
```

```terminal
9
```

Cadenas de caracteres: Funciones (continuación)

In / Not In: Se pueden utilizar para saber si una subcadena se encuentra o no en una cadena:

PY

```
cad="Jueves"
if "ve" in cad:
    print("se encuentra en la cadena")
else:
    print("no se encuentra en la cadena")

if "iércol" in cad:
    print("se encuentra en la cadena")
else:
    print("no se encuentra en la cadena")
```

terminal

```
se encuentra en la cadena
no se encuentra en la cadena
```

Cadenas de caracteres: Funciones (continuación)

Iteraciones: Se puede iterar sobre la cadena:

```py
cad="Aprendé Python"
for letra in cad:
    print(letra)
```

```
A
p
r
e
n
d
é

P
y
t
h
o
n
```

terminal

Max / Min: Se puede utilizar **max()** o **min()**:

```py
cad="Python"
print(max(cad)) #y
print(min(cad)) #P
```

Todas las letras mayúsculas van antes de las letras minúsculas

Count / index: Se puede utilizar **count()** o **index()**:

```py
cad="Programación"
print(cad.count("a")) #2
print(cad.index("grama")) #3
```

Cadenas de caracteres: Métodos

<separador>.join(<secuencia>): Devuelve una cadena con el separador entre cada carácter:

```py
cad="12345"
cad='-'.join(cad)
print(cad) #1-2-3-4-5
```

<cadena>.split(<sep>): Divide una cadena en una lista, buscando **sep** como separador:

```py
cad="Programando en Python"
lista= cad.split(' ')
print(lista) #['Programando', 'en', 'Python']
```

<cadena>.replace(<viejo>,<nuevo>,<max>): Reemplaza una cadena por otra hasta un máximo. Si se omite max reemplaza todas las apariciones.

```py
cad="Hoy es un día frío. Qué frío está!"
cad=cad.replace('frío', 'húmedo')
print(cad) # Hoy es un día húmedo. Qué húmedo está!
```

Cadenas de caracteres: Métodos (continuación)

<cadena>.isalpha(): Devuelve True si ***todos*** los caracteres de <cadena> son alfabéticos. Sino, devuelve False. Reconoce caracteres regionales.

```py
cad="Python"
cad2="Python3"
print(cad.isalpha()) # True
print(cad2.isalpha()) # False
```

<cadena>.isdigit(): Devuelve True si ***todos*** los caracteres de <cadena> son dígitos numéricos.

```py
cad="1234"
cad2="1234a"
print(cad.isdigit()) # True
print(cad2.isdigit()) # False
```

<cadena>.isalnum(): Devuelve True si ***todos*** los caracteres de <cadena> son alfabéticos o numéricos.

```py
cad=""
cad2="12"
cad3="ab"
print(cad.isalnum()) # False
print(cad2.isalnum()) # True
print(cad3.isalnum()) # True
```

Cadenas de caracteres: Métodos (continuación)

<cadena>.isupper(): Devuelve True si ***todos*** los caracteres de <cadena> están en mayúsculas. Ignora los caracteres no alfabéticos.

<cadena>.islower(): Devuelve True si ***todos*** los caracteres de <cadena> están en minúsculas. Ignora los caracteres no alfabéticos.

<cadena>.upper(): Devuelve una cadena convertida a mayúsculas. Ignora los caracteres no alfabéticos.

<cadena>.lower(): Devuelve una cadena convertida a minúsculas. Ignora los caracteres no alfabéticos.

PY

```python
cad="Python"
cad2="python"
print(cad.isupper()) # False
print(cad2.islower()) # True
print(cad2.upper()) # PYTHON
print(cad.lower()) # python
```

Cadenas de caracteres: Métodos (continuación)

<cadena>.capitalize(): Devuelve una cadena convertida a mayúscula sólo el primer caracter de la primer palabra y el resto en minúsculas.

<cadena>.title(): Devuelve una cadena convertida a mayúscula el primer caracter de cada palabra y el resto en minúsculas.

```py
cad="Aprendiendo programación en Python"
print(cad.capitalize()) # Aprendiendo programación en python
print(cad.title()) # Aprendiendo Programación En Python
```

<cadena>.center(<ancho>,[relleno]): Devuelve una cadena en el ancho especificado. El resto de la cadena se rellena con espacios o con el caracter relleno.

```py
cad1="Hola"
cad2=cad1.center(10,"*")
print(cad2) # ***Hola***
```

Cadenas de caracteres: Métodos (continuación)

<cadena>.ljust(<ancho>,[relleno]): Devuelve una cadena alineada a la izquierda en el ancho especificado. El fin de la cadena se rellena con espacios o con el caracter relleno.

<cadena>.rjust(<ancho>,[relleno]): Devuelve una cadena alineada a la derecha en el ancho especificado. El comienzo de la cadena se rellena con espacios o con el caracter relleno.

```py
cad1="Hola"
cad2=cad1.ljust(10, '-')
cad3=cad1.rjust(10, '-')
print(cad2) # Hola------
print(cad3) # ------Hola
```

<cadena>.zfill(<ancho>): Devuelve una cadena alineada a la derecha en el ancho especificado. El comienzo de la cadena se rellena con ceros.

```py
n=3
cad= str(n).zfill(5)
print(cad) # 00003
```

Cadenas de caracteres: Métodos (continuación)

<cadena>.lstrip(<str>): Devuelve una cadena sin los caracteres indicados en str al inicio de la cadena.

<cadena>.rstrip(<str>): Devuelve una cadena sin los caracteres indicados en str al final de la cadena.

```py
cad="---Hola-Mundo----"
cad2="---Hola-Mundo----"
cad= cad.lstrip("-")
print(cad) #Hola-Mundo----
cad2= cad2.rstrip("-")
print(cad2) #---Hola-Mundo
```

<cadena>.strip(<str>): Devuelve una cadena sin los caracteres indicados en str al inicio y al final de la cadena.

```py
cad="---Hola-Mundo----"
cad= cad.strip("-")
print(cad) #Hola-Mundo
```

Cadenas de caracteres: Métodos (continuación)

<cadena>. find(<str>,[[inicio],[fin]]): Devuelve la posición donde encuentra str en la cadena. Si no lo encuentra devuelve -1 (index devuelve error). Se puede indicar los subíndices desde y hasta a buscar.

<cadena>. rfind(<str>,[[inicio],[fin]]): Similar al find pero busca la última aparición.

PY

```
cad="---Hola-Mundo----"
pos= cad.find("Mundo")
print(pos) #8
cad="---Hola-Mundo--Mundo--"
pos= cad.rfind("Mundo")
print(pos) #15 (última aparición de Mundo)
```

Cadenas de caracteres: Formato

%formato: Disponible desde el inicio de Python.

<str>.format(<datos>): A partir de Python 2.6

```py
legajo= 12212
nombre= "María"
nota= 10
#%-formato
print("Legajo: %d Nombre: %s Nota: %d" %(legajo,nombre,nota))
print("Legajo: {} Nombre: {} Nota: {}".format(legajo,nombre,nota))

# En ambos casos devuelve:
# Legajo: 12212 Nombre: María Nota: 10
```

f-string (PEP498): El prefijo f permite dar formato a una cadena de caracteres. Ejemplo:

```py
#f-string
print(f"Legajo: {legajo} Nombre: {nombre} Nota: {nota}")
```

El resultado es el mismo que en el caso anterior

Tuplas

Conjunto de elementos separados por comas y encerrados entre paréntesis. Los paréntesis no son obligatorios. Estas son ***inmutables*** y en general contienen una secuencia heterogénea de elementos. Los elementos pueden ser mutables. Ejemplos:

```
()                                                  # tupla vacía
'un valor',                                         # tupla con un valor
('uno', 'dos', 'tres')                              # cadenas
('Palotes, Juan de', (1930, 11, 13), 3000936)       # datos de persona
'Palotes, Juan de', (1930, 11, 13), 3000936         # datos de persona
```

Creación: Por extensión. Los elementos se enumeran. Ejemplo:

```
tupla  = ('uno', 'dos', 'tres')
```

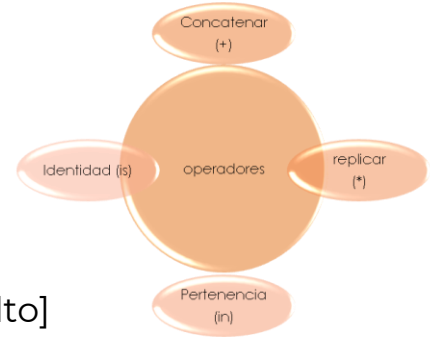
Empaquetado (*zip*): Se asigna a la tupla una enumeración de variables o valores. Ejemplo:

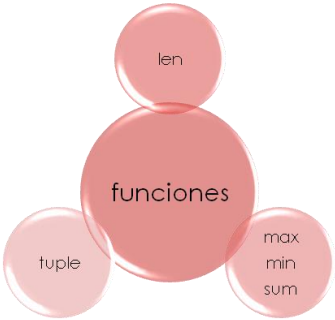
```
tupla  = 'Palotes, Juan de', (1930, 11, 13), 3000936
```

Desempaquetado (*unpack*): Se asigna los valores de una tupla a un conjunto de variables. una enumeración de variables o valores. requiere que la cantidad de variables a la izquierda del signo igual sea el tamaño de la tupla. Ejemplo:

```
nombre,  nacimiento, dni  =( 'Palotes, Juan de', (1930, 11, 13), 3000936)
```

Tuplas: Accesos





Puede accederse:

- Desempaquetando
- Con un índice, comenzando de la posición 0
- Por método rebanada o *Slicing* tupla[comienzo:fin:salto]

Ejemplos:

```
tupla = ('Palotes, Juan de', (1930, 11, 13), 3000936)
nombre, fecha, dni = tupla
print('Nombre: ', nombre, '. Fecha nac.: ', fecha, '. DNI: ', dni)
print('Nombre: ', tupla[0], '. Fecha nac.: ', tupla[1], '. DNI: ', tupla[2])
print(tupla[::])
```

PY

```
Nombre:  Palotes, Juan de . Fecha nac.:  (1930, 11, 13) . DNI:  3000936
Nombre:  Palotes, Juan de . Fecha nac.:  (1930, 11, 13) . DNI:  3000936
('Palotes, Juan de', (1930, 11, 13), 3000936)
```

terminal

Conjuntos

Conjunto de elementos únicos y no ordenados separados por comas y encerrados por llaves. Los elementos no pueden ser mutables. Ejemplos:

```
set()                                          # conjunto vacío
{'un valor'}                                   # conjunto con un valor
{'uno', 'dos', 'tres'}                         # conjunto de cadenas
{'Palotes, Juan de', (1930, 11, 13), 3000936}  # datos de persona
```

Creación: Por extensión. Los elementos se enumeran. Ejemplo:

```
conjunto  = { 1, 9, 0 }
```

Accesos: Puede accederse:

- Iterando
- Utilizando el método pop
- No puede accederse a través de un subíndice pues sus elementos no están ordenados

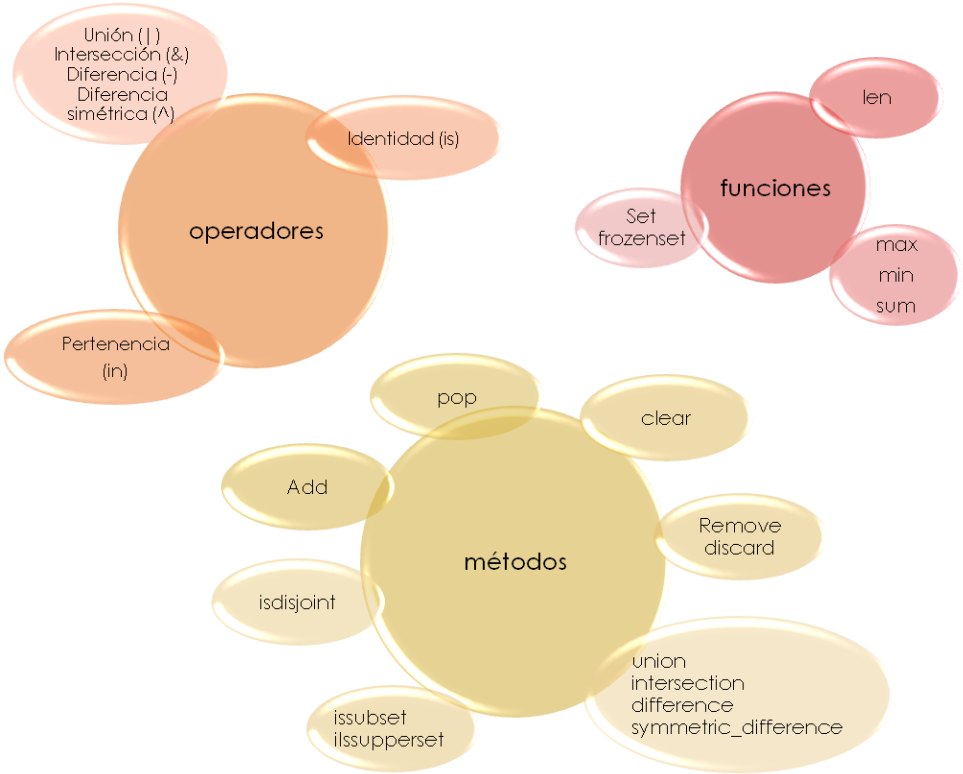
```
conjunto = {'Palotes, Juan de', (1930, 11, 13), 3000936}
a = set(conjunto)
print(a)
[print (elem) for elem in a]
print(a)
```

PY

terminal

```
{3000936, 'Palotes, Juan de', (1930, 11, 13)}
3000936
Palotes, Juan de
(1930, 11, 13)
{3000936, 'Palotes, Juan de', (1930, 11, 13)}
```

Conjuntos


Unión (|)
Intersección (&)
Diferencia (-)
Diferencia simétrica (^)
Identidad (is)
operadores
Pertenencia (in)
len
funciones
Set
frozenset
max
min
sum
pop
clear
Add
Remove
discard
métodos
isdisjoint
union
intersection
difference
symmetric_difference
issubset
ilssupperset

Diccionarios

Conjunto no ordenado de pares -*clave: valor*-. Las claves son únicas. Si se quiere guardar un valor a una clave ya existente se pierde dicho valor. Las claves pueden ser cualquier elemento de tipo inmutable (cadenas, números o tuplas si estas sólo contienen cadenas, números o tuplas). Se representa como una lista de pares clave:valor separados por comas encerrados entre llaves.

Para acceder a las claves Python utiliza un método de hash. Ejemplo:

```
{}                          # diccionario vacío
{'Juan': 56}                # diccionario de un elemento
{'Juan': 56, 'Ana': 15}     # diccionario de dos elementos
```

Creación:

- Por extensión. Los elementos se enumeran.
- Por compresión.

```
# Creación: Por extensión
diccionario  = {'Juan': 56, 'Ana': 15}
# Creación: Por compresión
diccionario  = {x: x ** 2 for x in (2, 4, 6)}
```

Diccionarios: Accesos

Puede accederse:

- A las claves, utilizando método keys()
- A los valores, utilizando la clave como índice
- A la clave-valor, utilizando método ítems()
- No es posible obtener porciones de un diccionario usando [:], ya que las claves no tienen.

PY

```
diccionario = {1: 'uno', 2:'dos', 3:'tres'}
print(diccionario.keys())
for i in diccionario.keys():
    print(diccionario[i])
for clave, valor in diccionario.items():
    print(clave, ':', valor, end= '; ')
```

terminal

```
dict_keys([1, 2, 3])
uno
dos
tres
1 : uno; 2 : dos; 3 : tres;
```