

Clase 3

Contenido

| | |
|--------------------------------------|----|
| 1.- Tipos de datos numéricos. | 1 |
| 2.- Sistemas de representación. | 2 |
| 3.- Operadores. | 3 |
| 4.- Funciones matemáticas. | 4 |
| 5.- Conjuntos. | 5 |
| 6.- Cadenas de texto. | 6 |
| 7.- Tuplas. | 11 |
| 8.- Listas | 14 |
| 9.- Diccionarios. | 23 |

1.- Tipos de datos numéricos.

Enteros.

Asignación:

```
clase3.py  ●
clase3.py > ...
1  num_entero = 8
2  num_negativo = -56
3  |
```

Clase 3

Reales.

Asignación:

```
clase3.py ×  
clase3.py > ...  
1 num_real = 4.7  
2
```

Complejos.

Asignación:

```
clase3.py ●  
clase3.py > ...  
1 num_complejo = 3.2 + 5j  
2  
3 #o también  
4 num_complejo2 = 7j + 3  
5
```

2.- Sistemas de representación.

Python puede representar los números enteros en los sistemas decimal, octal, binario y hexadecimal.

Binario.

Numero 7 representado en binario.

```
clase3.py ●  
clase3.py > ...  
1 num_binario = 0b11  
2
```

Clase 3**Octal.**

Numero 8 representado en octal.

```

clase3.py  X
clase3.py > ...
1  num_octal = 0o10
2

```

Hexadecimal.

Numero 255 representado en hexadecimal.

```

clase3.py  X
clase3.py > ...
1  num_hex = 0xf
2

```

3.- Operadores.

La Tabla 1 resume los principales operadores y operaciones numéricas a las que hace referencia.

| 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) |

Tabla 1. Principales operaciones y operadores numéricos.

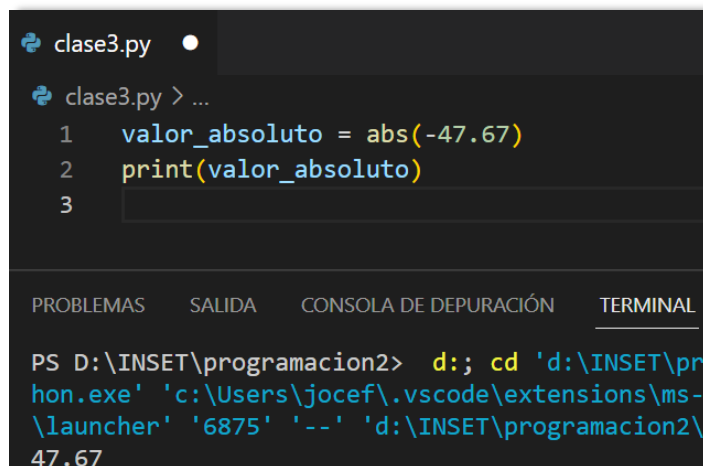
Clase 3

4.- Funciones matemáticas.

Aparte de las operaciones numéricas básicas, anteriormente mencionadas, Python nos permite aplicar otras muchas funciones matemáticas. Entre ellas, tenemos algunas como el valor absoluto, la raíz cuadrada, el cálculo del valor máximo y mínimo de una lista o el redondo para números reales. La mayoría de estas operaciones se encuentran disponibles a través de un módulo llamado [math](#).

Ejemplo:

Valor absoluto de un número:

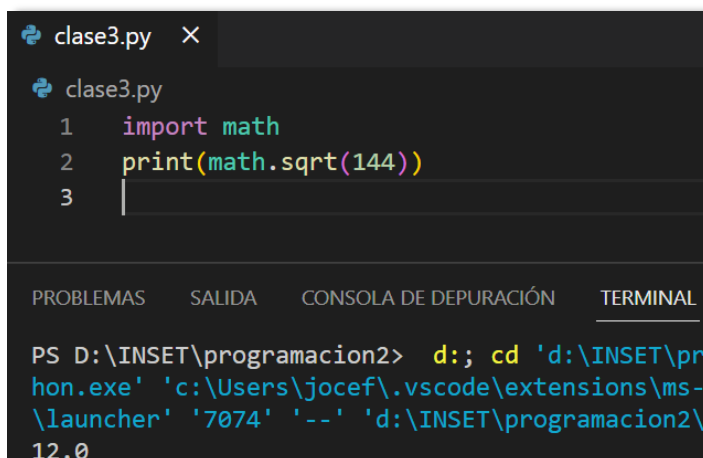


```
clase3.py
clase3.py > ...
1 valor_absoluto = abs(-47.67)
2 print(valor_absoluto)
3

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python.exe 'c:\Users\jocef\.vscode\extensions\ms-python.python\launcher' '6875' '--' 'd:\INSET\programacion2\clase3.py'
47.67
```

Para algunas funciones deberemos importar el módulo, por ejemplo, el siguiente código calcula la raíz cuadrada del número 144:



```
clase3.py X
clase3.py
1 import math
2 print(math.sqrt(144))
3

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python.exe 'c:\Users\jocef\.vscode\extensions\ms-python.python\launcher' '7074' '--' 'd:\INSET\programacion2\clase3.py'
12.0
```

Clase 3

5.- Conjuntos.

La función para crear una función se llama set() y acepta como argumentos una serie de valores pasados entre comillas simples, como si fuese una cadena de texto.

Ejemplo:

```
clase3.py ●
clase3.py > ...
1 conjunto = set ('846')
2
3 #También puede ser definido empleando llaves:
4 conjunto = {8, 4, 6}
5
```

Operaciones como unión, intersección, creación de subconjuntos y diferencia están disponibles para conjuntos en Python. Algunas operaciones se pueden hacer directamente a través de operadores o bien, llamando al método en

cuestión de cada instancia creada. Un ejemplo de ello es la operación intersección. Creemos un nuevo conjunto, utilicemos el operador & y observemos el resultado:

```
clase3.py X
clase3.py > ...
1 conjunto = {'8', '4', '6'}
2 conjunto_2 = set('785')
3 interseccion = conjunto & conjunto_2
4 print("Intersección usando &:",interseccion)
5
6 #También podríamos usar:
7 otra_forma = conjunto.intersection(conjunto_2)
8 print("Intersección usando metodo intersection:",interseccion)
9

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; & 'C:\Use
hon.exe' 'c:\Users\jocef\.vscode\extensions\ms-python.python-2023.6.0\
\launcher' '8574' '--' 'd:\INSET\programacion2\clase3.py'
Intersección usando &: {'8'}
Intersección usando metodo intersection: {'8'}
PS D:\INSET\programacion2>
```

Clase 3

6.- Cadenas de texto.

Básicamente, una cadena de texto o string es un conjunto inmutable y ordenado de caracteres. Para su representación y definición se pueden utilizar tanto comillas dobles ("), como simples ('). Por ejemplo, en Python, la siguiente sentencia crearía una nueva variable de tipo string:

```
clase3.py •
clase3.py > ...
1  cadena = "esto es una cadena de texto"
2  |
3
```

Si necesitamos declarar un string que contenga más de una línea, podemos hacerlo utilizando comillas triples en lugar de dobles o simples:

```
clase3.py •
clase3.py > ...
1  cad_multiple = """Esta cadena de texto
2  tiene más de una línea. En concreto, cuenta
3  con tres líneas diferentes"""
4  |
```

Principales funciones y métodos.

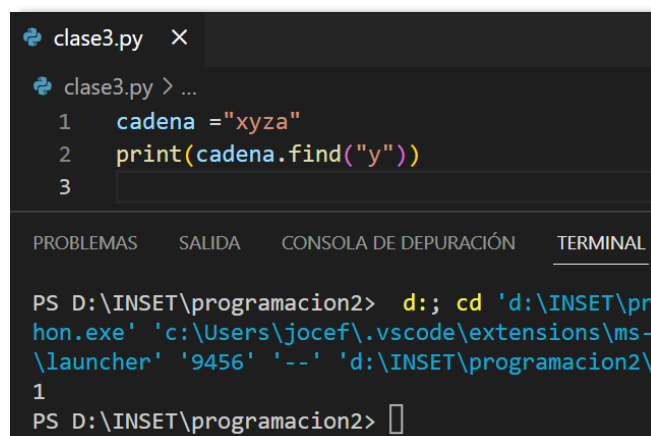
Una de las funciones más comunes que podemos utilizar sobre strings es el cálculo del número de caracteres que contiene. El siguiente ejemplo nos muestra cómo hacerlo:

```
clase3.py X
clase3.py > ...
1  cad = "Cadena de texto de ejemplo"
2  print(len(cad))
3

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d;; cd 'd:\INSET\pro
hon.exe' 'c:\Users\jocef\.vscode\extensions\ms-p
\launcher' '9124' '--' 'd:\INSET\programacion2\c
26
PS D:\INSET\programacion2> |
```

La función find() devuelve el índice correspondiente al primer carácter de la cadena original que coincide con el buscado:

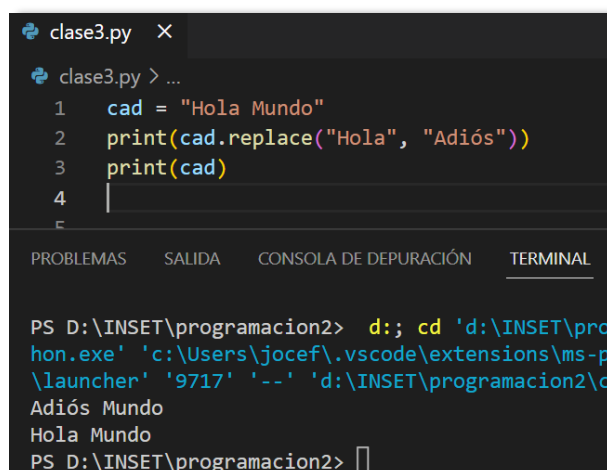
Clase 3

```
clase3.py X
clase3.py > ...
1  cadena = "xyza"
2  print(cadena.find("y"))
3
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python clase3.py
1
PS D:\INSET\programacion2> 
```

Si no lo encuentra, devolverá -1.

Para reemplazar una serie de caracteres por otros, contamos con el método `replace()`. En el siguiente ejemplo, sustituiremos la subcadena "Días" por "Adiós":



```
clase3.py X
clase3.py > ...
1  cad = "Hola Mundo"
2  print(cad.replace("Hola", "Adiós"))
3  print(cad)
4
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python clase3.py
Adiós Mundo
Hola Mundo
PS D:\INSET\programacion2> 
```

Obsérvese que `replace()` no altera el valor de la variable sobre el que se ejecuta. Así pues, en nuestro ejemplo, el valor de la variable `cad` seguirá siendo "Hola Mundo".

Los métodos `strip()`, `lstrip()` y `rstrip()` nos ayudarán a eliminar todos los espacios en blanco, solo los que aparecen a la izquierda y solo los que se encuentran a la derecha, respectivamente:

Clase 3

```
clase3.py X
clase3.py > ...
1  cad = " cadena con espacios en blanco "
2  print(cad.strip())
3
4  print(cad.lstrip())
5
6  print(cad.rstrip())
7  |

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2' & python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python\launcher' '9915' '--' 'd:\INSET\programacion2\clase3.py'
cadena con espacios en blanco
cadena con espacios en blanco
cadena con espacios en blanco
PS D:\INSET\programacion2> |
```

El método upper() convierte todos los caracteres de una cadena de texto a mayúsculas, mientras que lower() lo hace a minúsculas. Veamos un sencillo ejemplo:

```
clase3.py X
clase3.py > ...
1  cad = " cadena con espacios en blanco "
2  cad2 = cad.upper()
3  print(cad2)
4  print(cad2.lower())
5  |

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2' & python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python\launcher' '10678' '--' 'd:\INSET\programacion2\clase3.py'
CADENA CON ESPACIOS EN BLANCO
cadena con espacios en blanco
PS D:\INSET\programacion2> |
```

Otro método, llamado capitalize(), el cual solo convierte el primer carácter de un string a mayúsculas:

```
clase3.py X
clase3.py > ...
1  cad = "un ejemplo"
2  print(cad.capitalize())
3  |

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2' & python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python\launcher' '10836' '--' 'd:\INSET\programacion2\clase3.py'
Un ejemplo
PS D:\INSET\programacion2> |
```


Clase 3

En ocasiones puede ser muy útil dividir una cadena de texto basándonos en un carácter que aparece repetidamente en ella. Esta funcionalidad es la que nos ofrece `split()`. Supongamos que tenemos una cadena con varios valores separados por `;` y que necesitamos una lista donde cada valor se corresponda con los que aparecen delimitados por el mencionado carácter. El siguiente ejemplo nos muestra cómo hacerlo:

```
clase3.py X
clase3.py > ...
1  cad = "primer valor;segundo;tercer valor"
2  print(cad.split(";"))
3
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'
PS D:\INSET\programacion2> python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python\launcher' '11065' '--' 'd:\INSET\programacion2\c
['primer valor', 'segundo', 'tercer valor']
PS D:\INSET\programacion2> 
```

`join()` es un método que devuelve una cadena de texto donde los valores de la cadena original que llama al método aparecen separados por un carácter pasado como argumento:

```
clase3.py X
clase3.py > ...
1  cadena = "abc"
2  cadena_comas = ",".join(cadena)
3
4  print(cadena_comas)
5
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'
PS D:\INSET\programacion2> python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python\launcher' '11565' '--' 'd:\INSET\programacion2\c
a,b,c
PS D:\INSET\programacion2> 
```

Clase 3

Operaciones

El operador + nos permite concatenar dos strings, el resultado puede ser almacenado en una nueva variable:

```
clase3.py X
clase3.py > ...
1  cad_concat = "Hola" + " Mundo!"
2  print(cad_concat)
3
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python.python-2023.6.4\launcher' '11696' '--' 'd:\INSET\programacion2\clase3.py'
Hola Mundo!
PS D:\INSET\programacion2> 
```

También podemos hacer los siguiente:

```
clase3.py X
clase3.py > ...
1  cad2 = "buenas"
2  cad3 = "gente"
3
4  print ("Hola " + cad2 + " noches " + cad3)
5  print("Hola {0}. Otra {1}".format(cad2, cad3))
6  print("Hola {cad2}. Otra {cad3}".format(cad2=cad2,cad3=cad3))
7
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; & 'C:\Users\jocef\.vscode\extensions\ms-python.python-2023.6.4\launcher' '12176' '--' 'd:\INSET\programacion2\clase3.py'
Hola buenas noches gente
Hola buenas. Otra gente
Hola buenas. Otra gente
PS D:\INSET\programacion2> 
```

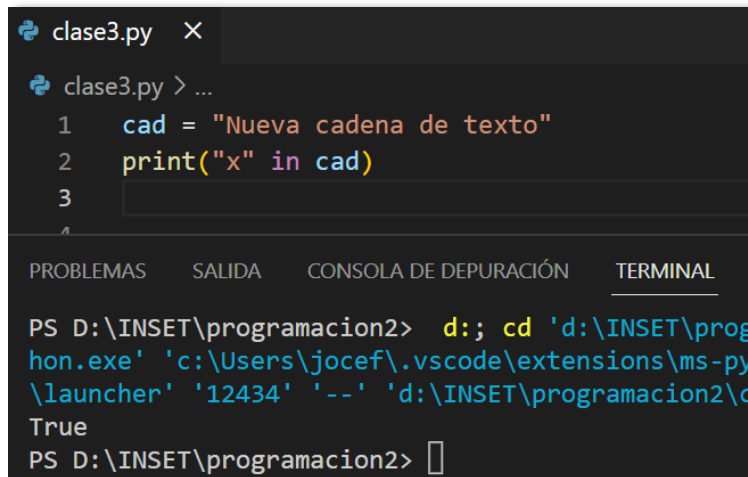
La concatenación entre strings y números también es posible, siendo para ello necesario el uso de funciones como int() y str(). El siguiente ejemplo es un caso sencillo de cómo utilizar la función str():

```
clase3.py X
clase3.py > ...
1  num = 3
2  print("Número: " + str(num))
3
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python.python-2023.6.4\launcher' '12280' '--' 'd:\INSET\programacion2\clase3.py'
Número: 3
PS D:\INSET\programacion2> 
```

Clase 3

Gracias al operador `in` podemos averiguar si un determinado carácter se encuentra o no en una cadena de texto. Al aplicar el operador, como resultado, obtendremos `True` o `False`, en función de si el valor se encuentra o no en la cadena. Comprobémoslo en el siguiente ejemplo:

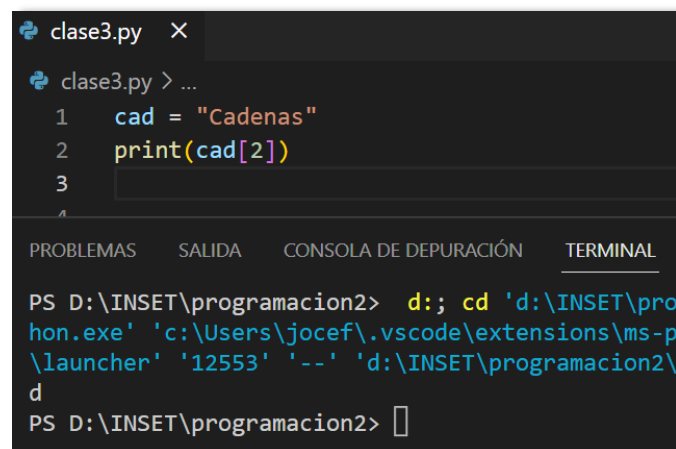


```
clase3.py X
clase3.py > ...
1  cad = "Nueva cadena de texto"
2  print("x" in cad)
3
4

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2' & python clase3.py
True
PS D:\INSET\programacion2> 
```

Un string es inmutable en Python, pero podemos acceder, a través de índices, a cada carácter que forma parte de la cadena:



```
clase3.py X
clase3.py > ...
1  cad = "Cadenas"
2  print(cad[2])
3
4

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2' & python clase3.py
d
PS D:\INSET\programacion2> 
```

7.- Tuplas

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.

Para declarar una tupla se utilizan paréntesis, entre los cuales deben separarse por comas los elementos que van a formar parte de ella.

Ejemplo:

Clase 3

```
clase3.py
clase3.py > ...
1  t = (1, 'a', 3.5)
2
```

Creamos una tupla de 3 elementos diferentes.

Los elementos son accesibles a través del índice que ocupan en la misma, similarmente a un array.

```
clase3.py
clase3.py > ...
1  t = (1, 'a', 3.5)
2
3  print(t[1])
4

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\pro
hon.exe' 'c:\Users\jocef\.vscode\extensions\ms-p
\launcher' '13066' '--' 'd:\INSET\programacion2\
a
PS D:\INSET\programacion2>
```

IMPORTANTE: las tuplas son un tipo de dato inmutable, esto significa que no es posible asignar directamente un valor a través del índice.

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

```
clase3.py
clase3.py > ...
1  t = (1, ('a', 3), 5.6)
2
3
```

Una de las peculiaridades de las tuplas es que es un objeto iterable; es decir, con un sencillo bucle for podemos recorrer fácilmente todos sus elementos:

Clase 3

```
clase3.py X
clase3.py > ...
1  t = (1, ('a', 3), 5.6)
2  for ele in t:
3      print(ele)
4
5
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL
PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python\launcher' '13264' '--' 'd:\INSET\programacion2\clase3.py'
1
('a', 3)
5.6
PS D:\INSET\programacion2> 
```

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.

Observemos el siguiente ejemplo y el resultado obtenido:

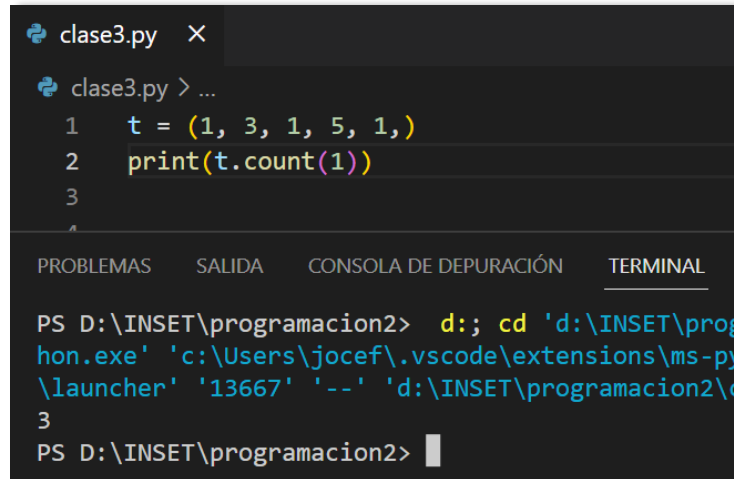
```
clase3.py X
clase3.py
1  print(('r', 2) * 3)
2
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL
PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python\launcher' '13393' '--' 'd:\INSET\programacion2\clase3.py'
('r', 2, 'r', 2, 'r', 2)
PS D:\INSET\programacion2> 
```

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 que ocupa en la tupla. Veamos el siguiente ejemplo:

```
clase3.py X
clase3.py > ...
1  t = (1, 3, 7)
2  print(t.index(3))
3
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL
PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python\launcher' '13561' '--' 'd:\INSET\programacion2\clase3.py'
1
PS D:\INSET\programacion2> 
```

Clase 3

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



```
clase3.py X
clase3.py > ...
1 t = (1, 3, 1, 5, 1,)
2 print(t.count(1))
3
4

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

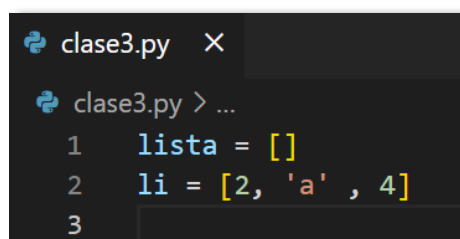
PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'
PS D:\INSET\programacion2> python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-py\launcher' '13667' '--' 'd:\INSET\programacion2\c
3
PS D:\INSET\programacion2>
```

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

8.- Listas

Básicamente, una lista es una colección ordenada de objetos, similar al array dinámico empleado en otros lenguajes de programación. Puede contener distintos tipos de objetos, es **mutable** y Python nos ofrece una serie de funciones y métodos integrados para realizar diferentes tipos de operaciones.

Para definir una lista se utilizan corchetes (`[]`) entre los cuales pueden aparecer diferentes valores separados por comas. Ejemplo donde se ven que ambas declaraciones son válidas:



```
clase3.py X
clase3.py > ...
1 lista = []
2 li = [2, 'a', 4]
3
```

Como las tuplas, las listas son también iterables, podemos recorrer sus elementos empleando un bucle:

Clase 3

```
clase3.py X
clase3.py > ...
1  li = [2, 'a' , 4]
2  for ele in li:
3      print(ele)
4

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python\launcher' '13891' '--' 'd:\INSET\programacion2\clase3.py'
2
a
4
PS D:\INSET\programacion2> 
```

A diferencia de las tuplas, los elementos de las listas se pueden modificar y ser reemplazados accediendo directamente a través del índice que ocupan en la lista.

Por ejemplo, para cambiar el segundo elemento de nuestra lista li, bastaría como ejecutar la siguiente sentencia:

```
clase3.py X
clase3.py > ...
1  li = [2, 'a' , 4]
2  li[1] = 'b'
3  print(li)
4

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python\launcher' '14028' '--' 'd:\INSET\programacion2\clase3.py'
[2, 'b', 4]
PS D:\INSET\programacion2> 
```

También, los valores de las listas pueden ser accedidos utilizando el valor del índice que ocupan en la misma:

```
clase3.py X
clase3.py > ...
1  li = [2, 'a' , 4]
2
3  print(li[2])
4

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python\launcher' '14161' '--' 'd:\INSET\programacion2\clase3.py'
4
PS D:\INSET\programacion2> 
```

Clase 3

Comprobar si un determinado valor existe en una lista a través del operador in. Devuelve True en caso afirmativo y False en caso contrario.

```
clase3.py  X
clase3.py > ...
1  li = [2, 'a' , 4]
2
3  print('a' in li)
4
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python.python\python\launcher' '14294' '--' 'd:\INSET\programacion2\clase3.py'
True
PS D:\INSET\programacion2> 
```

Existen dos funciones integradas que relacionan las listas con las tuplas: list() y tuple(). La primera toma como argumento una tupla y devuelve una lista. En cambio, tuple() devuelve una tupla al recibir como argumento una lista. Por ejemplo, la siguiente sentencia nos devolverá una tupla:

```
clase3.py  X
clase3.py > ...
1  li = [2, 'a' , 4]
2
3  li_a_tupla = tuple(li)
4
5  print(li_a_tupla)
6  print(type(li_a_tupla))
7  print(type(li))
8
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python.python\python\launcher' '14511' '--' 'd:\INSET\programacion2\clase3.py'
(2, 'a', 4)
<class 'tuple'>
<class 'list'>
PS D:\INSET\programacion2> 
```

Operaciones como la suma (+) y la multiplicación (*) también pueden ser aplicadas sobre listas. Su funcionamiento es exactamente igual que en las tuplas.

Inserciones y borrados

Para agregar un nuevo elemento a una lista contamos con el método append().

Clase 3

Como parámetro hemos de pasar el valor que deseamos añadir y este será insertado automáticamente al final de la lista. Ejemplo:

```
clase3.py X
clase3.py > ...
1  li = [2, 'a' , 4]
2
3  li.append('nuevo')
4
5  print(li)
6
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python.python\python-launcher' '14652' '--' 'd:\INSET\programacion2\clase3.py'
[2, 'a', 4, 'nuevo']
PS D:\INSET\programacion2>
```

Si agregamos, no es posible utilizar un índice superior al número de elementos que contenga la lista. La siguiente sentencia lanza un error:

```
clase3.py X
clase3.py > ...
1  li = [2, 'a' , 4]
2
3  li[4] = 23
4
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

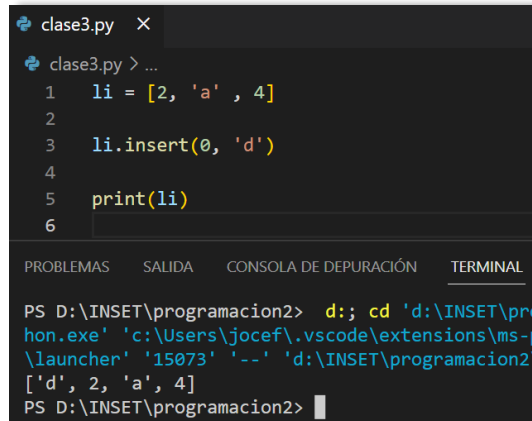
PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python.python\python-launcher' '14807' '--' 'd:\INSET\programacion2\clase3.py'
Traceback (most recent call last):
  File "d:\INSET\programacion2\clase3.py", line 3, in <module>
    li[4] = 23
IndexError: list assignment index out of range
PS D:\INSET\programacion2>
```

Pero si usamos el método insert(), el cual sirve para agregar un nuevo elemento especificando el índice, y si pasamos como índice un valor superior lo agregará igual pero al final de la lista. Ejemplo, estas dos instrucciones producirán el mismo resultado:

```
clase3.py ●
clase3.py > ...
1  li = [2, 'a' , 4]
2
3  li.insert(3, 'c')
4  li.insert(12, 'c')
5
```

Clase 3

Por ejemplo, para agregar un nuevo elemento en la primera posición de nuestra lista li, bastaría con ejecutar la siguiente sentencia:

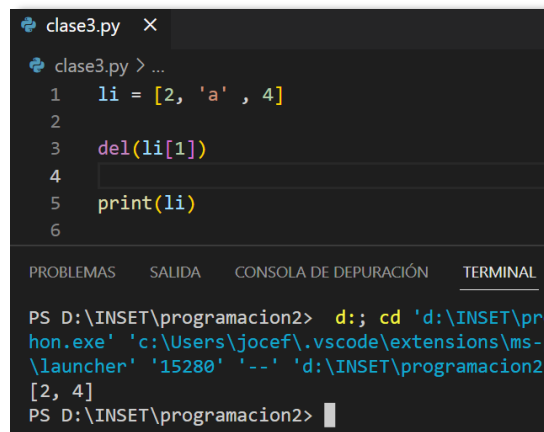


```
clase3.py x
clase3.py > ...
1  li = [2, 'a', 4]
2
3  li.insert(0, 'd')
4
5  print(li)
6

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python.python\python\launcher' '15073' '--' 'd:\INSET\programacion2\clase3.py'
['d', 2, 'a', 4]
PS D:\INSET\programacion2>
```

Si lo que necesitamos es borrar un elemento de una lista, podemos hacerlo gracias a la función `del()`, que recibe como argumento la lista junto al índice que referencia al elemento que deseamos eliminar. La siguiente sentencia ejemplo borra el valor 2 de nuestra lista li:



```
clase3.py x
clase3.py > ...
1  li = [2, 'a', 4]
2
3  del(li[1])
4
5  print(li)
6

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python.python\python\launcher' '15280' '--' 'd:\INSET\programacion2\clase3.py'
[2, 4]
PS D:\INSET\programacion2>
```

Como consecuencia de la sentencia anterior, la lista queda reducida en un elemento. Para comprobarlo contamos con la función `len()`, que nos devuelve el número de elementos de la lista:

Clase 3

```
clase3.py X
clase3.py > ...
1  li = [2, 'a' , 4]
2
3  del(li[1])
4
5  print(li)
6  print(len(li))
7

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python.python\python\launcher' '15432' '--' 'd:\INSET\programacion2\clase3.py'
[2, 4]
2
PS D:\INSET\programacion2>
```

También es posible borrar un elemento de una lista a través de su valor. Para ello contamos con el método `remove()`:

```
clase3.py X
clase3.py > ...
1  li = [2, 'a' , 4]
2
3  li.remove(2)
4
5  print(li)
6

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python.python\python\launcher' '15629' '--' 'd:\INSET\programacion2\clase3.py'
['a', 4]
PS D:\INSET\programacion2>
```

Si el elemento a remover se encuentra repetido, solo borrara la primera ocurrencia que encuentre en la misma.

Ordenación

Los elementos de una lista pueden ordenarse a través del método `sort()` o utilizando la función `sorted()`. Por ejemplo, ordenaremos una lista de enteros:

Clase 3

```
clase3.py X
clase3.py > ...
1 lista = [3, 1, 9, 8, 7]
2 print(sorted(lista))
3 print(sorted(lista, reverse=True))
4 print(lista)
5 |

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python.exe 'c:\Users\jocef\.vscode\extensions\ms-python.python\python\launcher' '15784' '--' 'd:\INSET\programacion2\clase3.py'
[1, 3, 7, 8, 9]
[9, 8, 7, 3, 1]
[3, 1, 9, 8, 7]
PS D:\INSET\programacion2> |
```

Fíjense que, si bien aplicamos la función, la lista quedo sin ordenarse, en el final. Para que quede ordenada y modificada usaremos el método `sort()`. Ejemplo:

```
clase3.py X
clase3.py > ...
1 lista = [3, 1, 9, 8, 7]
2 lista.sort()
3 print(lista)
4 |

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python.exe 'c:\Users\jocef\.vscode\extensions\ms-python.python\python\launcher' '15938' '--' 'd:\INSET\programacion2\clase3.py'
[1, 3, 7, 8, 9]
PS D:\INSET\programacion2> |
```

Otro método que contienen las listas relacionado con la ordenación de valores es `reverse()`, que automáticamente ordena una lista en orden inverso al que se encuentran sus elementos originales.

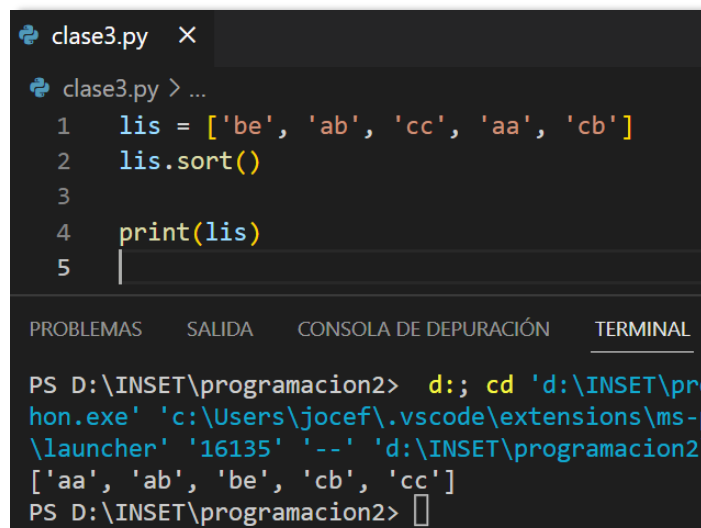
```
clase3.py X
clase3.py > ...
1 lista = [3, 1, 9, 8, 7]
2 lista.reverse()
3 print(lista)
4 |

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python.exe 'c:\Users\jocef\.vscode\extensions\ms-python.python\python\launcher' '16025' '--' 'd:\INSET\programacion2\clase3.py'
[7, 8, 9, 1, 3]
PS D:\INSET\programacion2> |
```

Clase 3

Los métodos y funciones de ordenación no solo funcionan con números, sino también con caracteres y con cadenas de texto:

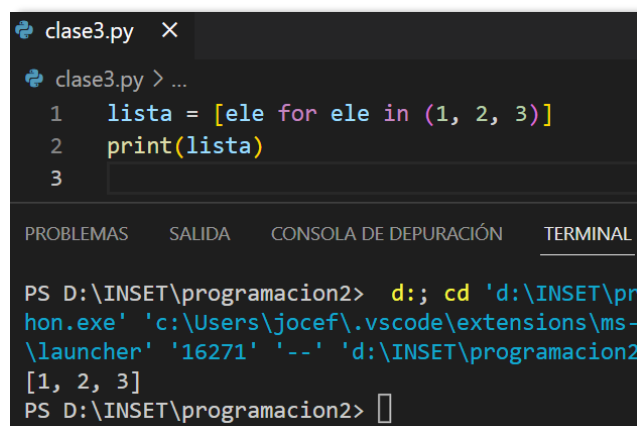


```
clase3.py X
clase3.py > ...
1  lis = ['be', 'ab', 'cc', 'aa', 'cb']
2  lis.sort()
3
4  print(lis)
5
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL
PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python\launcher' '16135' '--' 'd:\INSET\programacion2\'
['aa', 'ab', 'be', 'cb', 'cc']
PS D:\INSET\programacion2> 
```

Comprensión

La comprensión de listas es una construcción sintáctica de Python que nos permite declarar una lista a través de la creación de otra. Esta construcción está basada en el principio matemático de la teoría de comprensión de conjuntos.

Ejemplo:



```
clase3.py X
clase3.py > ...
1  lista = [ele for ele in (1, 2, 3)]
2  print(lista)
3
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL
PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python\launcher' '16271' '--' 'd:\INSET\programacion2\'
[1, 2, 3]
PS D:\INSET\programacion2> 
```

Matrices

Anidando listas podemos construir matrices de elementos. Estas estructuras de datos son muy útiles para operaciones matemáticas. Debemos tener en cuenta que complejos problemas matemáticos son resueltos empleando matrices.

Ejemplo, una matriz matemática de dos dimensiones puede definirse como:

Clase 3

```
clase3.py
clase3.py > ...
1  matriz = [[1, 2, 3],[4, 5, 6]]
2
```

Para acceder al segundo elemento de la primera matriz, bastaría ejecutar la siguiente línea:

```
clase3.py X
clase3.py > ...
1  matriz = [[1, 2, 3],[4, 5, 6]]
2
3  print(matriz[0][1])
4

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python\launcher' '16494' '--' 'd:\INSET\programacion2\clase3.py'
PS D:\INSET\programacion2> 
```

Para modificar, podemos acceder de la misma manera que a listas.

Ejemplo, cambiemos un elemento directamente:

```
clase3.py X
clase3.py > ...
1  matriz = [[1, 2, 3],[4, 5, 6]]
2
3  matriz[0][1] = 33
4  print(matriz)
5
6

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python\launcher' '16624' '--' 'd:\INSET\programacion2\clase3.py'
[[1, 33, 3], [4, 5, 6]]
PS D:\INSET\programacion2> 
```

Clase 3

9.- Diccionarios

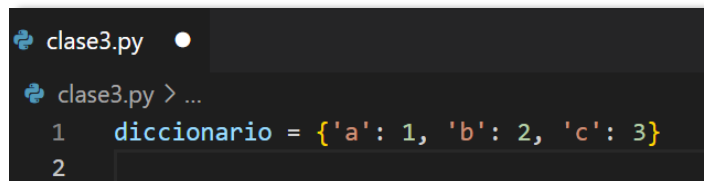
Los diccionarios nos permiten almacenar datos de diferente tipo, como las listas y las tuplas que ya hemos visto.

Sin embargo, introducen una novedad pues cada elemento se introduce asociado a una clave.

En las tuplas y listas el orden entre los elementos era parte de la información almacenada. Esto no cuenta para los diccionarios.

Las claves no pueden repetirse dentro del diccionario. Jugarán el mismo rol que en las listas y tuplas jugaba la posición y nos permitirán identificar y encontrar los elementos.

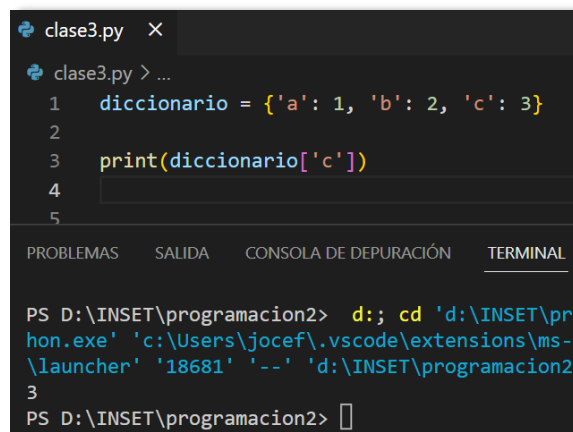
Para declarar un diccionario en Python se utilizan las llaves ({}), entre las que se encuentran los pares clave-valor separados por comas. La clave de cada elemento aparece separada del correspondiente valor por el carácter `:`. El siguiente ejemplo muestra la declaración de un diccionario con tres valores:



```
clase3.py •
clase3.py > ...
1  diccionario = {'a': 1, 'b': 2, 'c': 3}
2
```

Acceso

En los diccionarios necesitamos utilizar la clave para acceder al valor de cada elemento. Volviendo a nuestro ejemplo, para obtener el valor indexado por la clave 'c' bastará con ejecutar la siguiente sentencia:



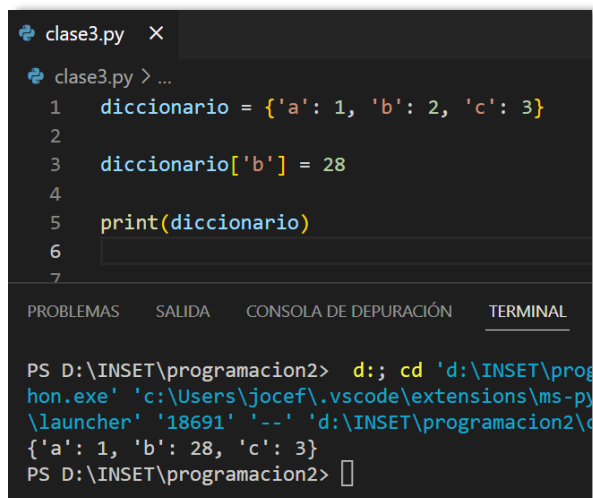
```
clase3.py x
clase3.py > ...
1  diccionario = {'a': 1, 'b': 2, 'c': 3}
2
3  print(diccionario['c'])
4
5

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'
PS D:\INSET\programacion2> python.exe 'c:\Users\jocef\.vscode\extensions\ms-p
\launcher' '18681' '--' 'd:\INSET\programacion2\'
3
PS D:\INSET\programacion2> 
```

Para modificar el valor de un diccionario, basta con acceder a través de su clave:

Clase 3



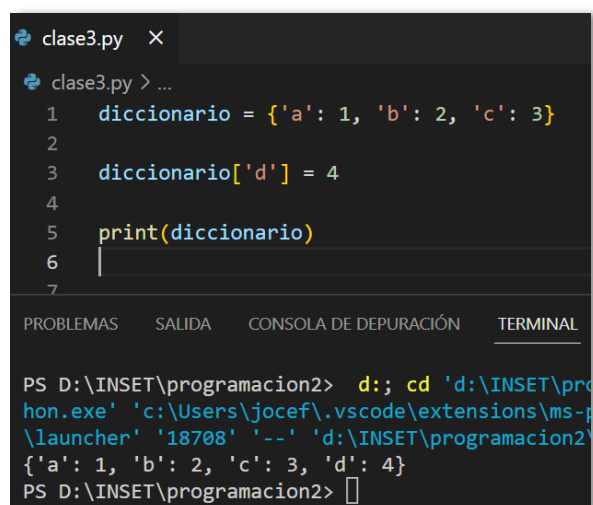
```
clase3.py X
clase3.py > ...
1  diccionario = {'a': 1, 'b': 2, 'c': 3}
2
3  diccionario['b'] = 28
4
5  print(diccionario)
6
7

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python.python\python\launcher' '18691' '--' 'd:\INSET\programacion2\clase3.py'
{'a': 1, 'b': 28, 'c': 3}
PS D:\INSET\programacion2> 
```

Inserción

Añadir un nuevo elemento es tan sencillo como modificar uno ya existente, ya que, si la clave no existe, automáticamente Python la añadirá con su correspondiente valor. Así pues, la siguiente sentencia insertará un nuevo valor en nuestro diccionario ejemplo:



```
clase3.py X
clase3.py > ...
1  diccionario = {'a': 1, 'b': 2, 'c': 3}
2
3  diccionario['d'] = 4
4
5  print(diccionario)
6
7

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-python.python\python\launcher' '18708' '--' 'd:\INSET\programacion2\clase3.py'
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
PS D:\INSET\programacion2> 
```

Borrado

La función integrada del() es la que nos ayudará a eliminar un valor de un diccionario. Para ello, necesitaremos pasar la clave que contiene el valor que deseamos eliminar. Por ejemplo, para eliminar el valor que contiene la clave 'c' de nuestro diccionario, basta con ejecutar:

Clase 3

```
clase3.py X
clase3.py > ...
1  diccionario = {'a': 1, 'b': 2, 'c': 3}
2
3  del(diccionario['b'])
4
5  print(diccionario)
6
7

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'
PS D:\INSET\programacion2> python.exe 'c:\Users\jocef\.vscode\extensions\ms-p
\launcher' '18736' '--' 'd:\INSET\programacion2\'
{'a': 1, 'c': 3}
PS D:\INSET\programacion2> 
```

Iteración

Tres son los métodos principales que nos permiten iterar sobre un diccionario: `items()`, `values()` y `keys()`. El primero nos da acceso tanto a claves como a valores, el segundo se encarga de devolvernos los valores, y el tercero y último es el que nos devuelve las claves del diccionario. Veamos un ejemplo de los tres:

```
clase3.py X
clase3.py > ...
1  diccionario = {'a': 1, 'b': 2, 'c': 3}
2
3  print("\n")
4  print("Usando item():")
5  for k, v in diccionario.items():
6      print("clave={0}, valor={1}".format(k, v))
7  print("\n")
8
9  print("Usando keys():")
10 for k in diccionario.keys():
11     print("clave={0}".format(k))
12 print("\n")
13
14 print("Usando values():")
15 for v in diccionario.values():
16     print("valor={0}".format(v))
17
18
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'
PS D:\INSET\programacion2> python-2023.6.0\pythonFiles\lib\python\debugpy\adapter\../
python.exe 'c:\Users\jocef\.vscode\extensions\ms-p
\launcher' '18736' '--' 'd:\INSET\programacion2\'

Usando item():
clave=a, valor=1
clave=b, valor=2
clave=c, valor=3

Usando keys():
clave=a
clave=b
clave=c

Usando values():
valor=1
valor=2
valor=3
PS D:\INSET\programacion2> 
```

Clase 3

Por defecto, si iteramos sobre un diccionario con un bucle for, obtendremos las claves del mismo sin necesidad de llamar explícitamente al método keys():

```
clase3.py X
clase3.py > ...
1  diccionario = {'a': 1, 'b': 2, 'c': 3}
2
3  for k in diccionario:
4      print(k)
5

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'
a
b
c
PS D:\INSET\programacion2>
```

Comprensión

De forma similar a las listas, los diccionarios pueden también ser creados por comprensión. El siguiente ejemplo muestra cómo crear un diccionario utilizando la iteración sobre una lista:

```
clase3.py X
clase3.py > ...
1  dicc_x_comprnsion = {k: k+1 for k in (1, 2, 3)}
2
3  print(dicc_x_comprnsion)
4
5

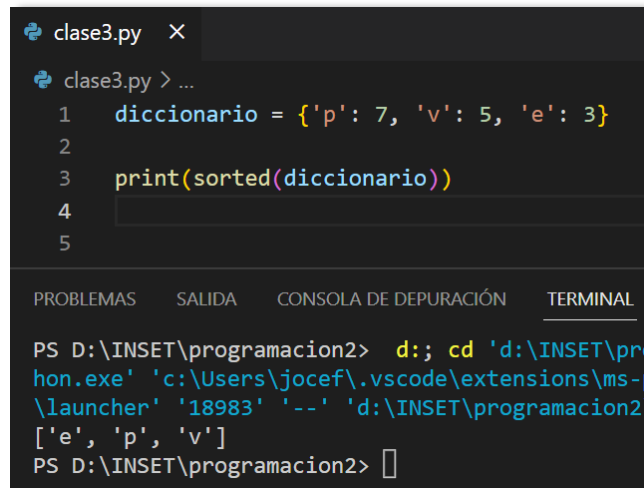
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'
{1: 2, 2: 3, 3: 4}
PS D:\INSET\programacion2>
```

Clase 3

Ordenación

A diferencia de las listas, los diccionarios no tienen el método `sort()`, pero sí que es posible utilizar la función integrada `sorted()` para obtener una lista ordenada de las claves contenidas.



```
clase3.py  X
clase3.py > ...
1  diccionario = {'p': 7, 'v': 5, 'e': 3}
2
3  print(sorted(diccionario))
4
5

PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

PS D:\INSET\programacion2> d:; cd 'd:\INSET\programacion2'; python hon.exe 'c:\Users\jocef\.vscode\extensions\ms-p\launcher' '18983' '--' 'd:\INSET\programacion2\'
['e', 'p', 'v']
PS D:\INSET\programacion2> 
```