Tema 1:
Python 4

Índice

1. Interpolación de cadenas

La interpolación de cadenas en el proceso se sustituir marcadores de posición en una cadena por valores lo que permite crear cadenas dinámicas ya que los valores se sustituyen en tiempo de ejecución. Estos valores pueden estar contenidos en variables, literales, expresiones, …

Veremos tres formas de interpolar cadenas en Python:

1.1. Interpolación de cadenas usando %.

Se puede usar el módulo para formatear cadenas. Cada marcador de posición será sustituido por el valor especificado después de %.

Veamos un ejemplo:

```
nombre="Lorenzo"
"Hola soy %s" % nombre
```
```
Hola soy Lorenzo
```

Donde tenemos los siguientes elementos junto al operador %:

- A la izquierda: cadena con los marcadores de posición. En el ejemplo es %s.
- A la derecha los datos, literales, expresiones, … que se usan para asignar el valor al marcador de posición.

Donde tenemos que tener en cuenta:

- Tiene que haber tantos valores a la derecha como marcadores de posición a la izquierda y ser del mismo tipo.
- Las sustitución de valores es posicional: el primer marcador de posición se substituye por el primer valor, el segundo por el segundo, …
- Si tenemos más de un marcador de posición los valores se han de especificar en un tupla.

Por ejemplo:

```
nombre="Lorenzo"
apellido="Freire"
print("Hola soy %s %s" % (nombre, apellido))
```
```
Hola soy Lorenzo Freire
```

Se puede usar un diccionario sustituyendo donde junto a marcador de posición se especifica la clave:

```
print("Hola soy %(nombre)s %(apellido)s" % {"nombre":"Lorenzo", "apellido":"Freire"})
```
```
Hola soy Lorenzo Freire
```

Los tipos de marcadores que se pueden usar son:

Marcador	Tipo dato	Descripción
%d o %i	Integer	Entero decimal con signo
%o	Octal	Octal con signo.
%x	Hexadecimal	Hexadecimal con signo (minúsculas
%X	Hexadecimal	Hexadecimal con signo (mayúsculas)
%f	Float	Número con decimales
%s	String	
%c	Integer o cad de 1 car	Cadena especificada por cad o carácter indicado por Integer.

Por ejemplo:

```
print("%d %f %X" % (10, 10, 10))
```
```
10 10.000000 A
```

Se pueden usar formatear la salida con usando un punto y hasta dos enteros:

- En el caso de los float si el entero si sitúa después del punto indica su número de decimales.

```
print("%.3f"% 123.1234)
   123.123
```

```
print("%.3f"% 123)
   123.000
```

- Si el numero va antes indica hasta que caracteres se rellena antes. Si el número comienza con 0 se rellena con ceros.

```
print("%7.d"% 5)  # La salida tiene 7 caracteres
      5
print("%05.f"% 6) # La salida tiene 6 caracteres, rellenando con ceros
   00006
```

- Se pueden combinar ambos:

```
 print("%08.4f"% 6) # La salida 8 carac., rellenando con ceros y 4 decimales
   006.0000
```

1.2. Método Format

Su funcionamiento es precedido al anterior pero usando llaves {}, en vez de %, para los marcadores de posición.

```
nombre="Lorenzo"
apellido="Freire"
print("Hola soy {} {}".format(nombre, apellido))
  Hola soy Lorenzo Freire
```

Donde la asignación es posicional. El prime marcador de posición se asignan el primer valor del format, el segundo con el segundo, …

Los valores a usar también se puede especificar con:

- Su posición. El valor entre las llaves indica su posición en la lista de valores:

```
print("Hola soy {1} {0}".format(nombre, apellido))
   Hola soy Freire Lorenzo
```

- Su nombre. El nombre entre las lleves indica el nombre del valor en la lista de valores:

```
print("Hola soy {nome} {apelido}".format(nome="Lorenzo", apelido="Freire"))
   Hola soy Lorenzo Freire
```

Si queremos directamente diccionarios tendremos que desempaquetarlos:

```
persona={"nombre":"Lorenzo", "apellido":"Freire"}
print("Hola soy {nombre} {apellido}".format(**persona))
```

Al igual que en el caso anterior se pueden especificar el formato de salida. En este caso mediante dos puntos:

```
print("{:07.2f}".format(5))
   0005.00
```

1.3. F-Strings

Desde la versión 3.6 de Python se pueden usar f-String. En ellas se pueden introducir directamente expresiones y variables. Es la opción recomendada actualmente.

Se usan anteponiendo una f a la cadena a formatear. Su uso el similar a formar. Entre llaves se especifica la variable o cualquier expresión válida en Python a interpolar en la cadena.

Por ejemplo:

```
nombre="Lorenzo"
apellido="Freire"
print(f"Hola soy {nombre} {apellido}")
```

```
Hola soy Lorenzo Freire
```

```
print(f"5 * 4.2 es {5*4.2}")
```

```
5 * 4.2 es 21
```

Se puede especificar el formato de salida con el mismo formato que visto en el punto anterior:

```
print(f"5 * 4 es {5*4:08.2f}")
```

```
5 * 4 es 00020.00
```

2. Secuencias.

Las secuencias son tipos de datos en Python que permiten almacenar varios valores en una sola variable pudiendo se acceder a cada valor de forma individual (normalmente mediante un índice).

Sus características comunes son:

- No hay un límite a la cantidad de elementos que pueden contener.
- Pueden contener cualquier tipo de objeto, incluyendo otras secuencias.
- No es necesario saber el tamaño (cantidad de elementos) que tendrá la secuencia en el momento de su creación.

Los tipos de secuencias que podemos encontrar en Python son:

Tipo	Nombre	Descripción	Definición	Mutable
list	Listas	Conjunto de datos que se puede repetir y que pueden ser de distintos tipos.	[v1, v2, ...]	Si
tuple	tuplas	Iguales que las listas pero inmutables	(v1, v2, ...)	No
range	rangos	Secuencias de números que se suelen usar para bucles.	range()	No
str	cadenas de caracteres	Permiten guardar secuencias de caracteres.	" " o ' '	No
byte	secuencias de bytes	Permite guardar valores binarios representados por caracteres ASCII.	bytes (secuencia int)	No
bytearray	secuencias de bytes	Iguales que los byte pero mutable.	bytesarray (secuencia int)	Si
set	conjuntos	Permiten guardar conjuntos de datos, en los que no se existen repeticiones.	{v1, v2, ...}	Si
frozenset	conjuntos	Iguales que los set pero inmutables	frozenset(sec)	No

Las secuencias comparten los siguientes métodos:

- len(secuencia): devuelve la cantidad de elementos de la lista, tupla o cadena.
- max(secuencia): devuelve el elemento con un valor más alto.
- min(secuencia): devuelve el elemento con un valor más bajo.

Y los siguientes métodos:

- secuencia.index(valor): devuelve el índice de la primera aparición de valor en la secuencia.
- secuencia.count(valor): devuelve el número de veces que aparece valor en la secuencia.

Como ya comentamos para acceder a los elementos de una secuencia se utiliza su índice. Sus valores van desde el índice 0 hasta la longitud de la secuencia – 1.

Los índices también pueden ser negativos, con lo que comenzaría a contar por el final.

Todas las secuencias son iterables mediante un bucle for:

```
frase = "Hola mundo"
for letra in frase:
 print(letra, end=" - ")
```

```
H - o - l - a -   - m - u - n - d - o -
```

2.1. Operadores en secuencias

Con las secuencias tenemos, como ya vimos con las listas, los operadores:

De pertenencia: in y not in

Donde:

- in: devuelve verdadero si el elemento pasado **pertenece** a la secuencia.
- not in: devuelve falso si el elemento pasado **no pertenece** a la secuencia.

```
frase = "Hola mundo"
print("Hola" in frase)
```

```
True
```

```
print("hola" in frase)
```

```
False
```

Operador +

Una las dos secuencias en una nueva secuencia. Donde primero van los elementos de la parte izquierda del + en el mismo orden, y luego los de la derecha

Las secuencias deben ser del mismo tipo.

```
# cadena1, cadena2 = "Hola", "Mundo"
cadena1 = "Hola"
cadena2 = "Mundo"
cadena = cadena1 + " " + cadena2
print(cadena)
```

```
Hola Mundo
```

```
print(cadena[5])
```

```
M
```

Operador *

Este operador, como sucedía con las listas, nos permite repetir una secuencia el número de veces que le indiquemos:

```
cadena = "Hola Mundo"
print(cadena*3)
```

```
Hola MundoHola MundoHola Mundo
```

2.2. Desempaquetado (unpacking)

Es posible asignar los elementos de una secuencia a diferentes variables de forma automática. Del lado izquierdo se escribe una lista separada por comas y del derecho, la secuencia.

```
a, b, c, d = "Hola"
print(f"{a} {b}")
```

```
H o
```

```
lista = [1, 2, 3, 4, 5]
a, b, c, d, e = lista
print(f"{b} {e}")
```
```
2 5
```

Es necesario que haya tantas variables como elementos en la secuencia.

```
a, b, c = lista
```
```
ValueError: too many values to unpack (expected 3)
```

3. Cadenas

Las cadenas son un tipo de secuencias de Python por lo que todo lo visto hasta ahora de secuencias también es aplicable a las cadenas.

Se definen con comillas dobles " o simples ' y son inmutables. Por ello, al igual que con las tuplas, no se puede modificar su contenido:

```
cadena = "Hola mundo"
cadena[1] = 'a'
```
```
TypeError: 'str' object does not support item assignment
```

Todos los métodos que se apliquen sobre una cadena devolverán otra cadena con el resultado pero no modificarán la cadena original.

3.1. Métodos de cadenas

Entre otros disponemos de los siguientes métodos teniendo en cuanta que para los ejemplos partimos de la siguiente cadena:

```
cadena = "hoLA mUNdo"
```

Para ver una lista completa de métodos de string podemos consultar el siguiente enlace:

https://www.w3schools.com/python/python_strings_methods.asp

3.1.1. Conversión a mayúsculas/minúsculas

Método	Que hace	Ejemplos con: "hoLA mUNdo"
lower()	Convierte una cadena a letras minúsculas.	cadena.lower() hola mundo
upper()	Convierte una cadena a letras mayúsculas.	cadena.upper() HOLA MUNDO
capitalize()	Pone la primera letra de la cadena en mayúsculas y el resto en minúsculas.	cadena.capitalize() Hola mundo
title()	Pone la primera letra de cada palabra en mayúsculas y el resto en minúsculas.	cadena.title() Hola Mundo
swapcase()	Invierte el caso de todas las letras de la cadena	cadena.swapcase() HOla MunDO

3.1.2. Insertar y eliminar espacios y/o caracteres

Método	Que hace	Ejemplos
center(ancho [, relleno])	Centra una cadena en otra de un ancho determinado en el parámetro ancho, añadiendo espacios antes y/o después. El parámetro relleno es opcional e indica el carácter de relleno, por defecto es un espacio.	print(("Estoy centrado").center(35,"-")) ----------Estoy centrado---------- filas = 5 ancho = filas * 2 + 3 for i in range(filas): ast = i * 2 + 1 print(("*" * ast).center(ancho,"-")) ------*------ -----***----- ----*****---- ---*******--- --*********--
strip([caracteres])	Elimina, por ambos lados, todos los caracteres de espacio de una cadena: espacios, tabuladores, saltos de línea. Si se pasa un segundo parámetro elimina todas las ocurrencias de esos caracteres por ambos lados.	print("#"," Hello \t ".strip(),"#", sep="") #Hello# "ababHelloabab".strip("ba") Hello " \t ababHelloabab".strip("ba") ababHello
rstrip([caracteres])	Hace lo mismo que strip pero solo por la derecha.	print("#"," Hello \t ".rstrip(),"#", sep="") # Hello# print("ababHelloabab".rstrip("ba")) ababHello

lstrip([carac,])	Hace lo mismo que strip pero solo por la izquierda.	print("#"," Hello \t ".lstrip(),"#", sep="")
		#Hello #
		print("ababHelloabab".rstrip("ba"))
		Helloabab

zfill(ancho)	Rellena con 0s a la izquierda hasta un ancho una cadena con números.	"14".zfill(3) 00014

3.1.3. Verificación de contenido: cadenas

Método	Que comprueba	Ejemplos con: "hoLA mUNdo"
isupper()	Verdadero si la cadena contiene solo caracteres en mayúsculas	cadena.isupper() False "HOLA MUNDO".isupper() True
islower()	Verdadero si la cadena contiene solo caracteres en minúsculas.	cadena.islower() False "hola mundo".islower() True
isspace()	Verdadero si la cadena contiene solo caracteres de espacio: espacios, tabuladores, saltos de línea, ...	cadena.isspace() False " \n \t ".isspace() True

isalnum()	Verdadero si la cadena contiene solo caracteres alfanuméricos: letras y/o números	cadena.isalnum() False "hoLAmUNdo123".isalnum() True
isalpha()	Verdadero si la cadena contiene solo caracteres alfabéticos.	cadena.isalnum() False "hoLAmUNdo123".isalnum() False "hoLAmUNdo".isalnum() True

startsswith (prefijo)	Verdadero si la cadena comienza por prefijo.	if cadena.startswith("hoLA"):
		print("Comienzo con hoLA")
		Comienzo con hoLA
		if not cadena.startswith("hola"):
		print("No comienzo con hola")
		No comienzo con hola

endswith (sufijo)	Verdadero si la cadena termina por sufijo por prefijo.	if cadena.endswith("mUNdo"):
		print("Termino con mUNdo")
		Termino con mUNdo
		if not cadena.startswith("mundo"):
		print("No termino con mundo")
		No termino con mundo

3.1.4. Verificación de contenido: números

En este apartado tenemos 3 métodos. Para una compresión más profunda de sus diferencias se puede consultar el siguiente enlace:

https://miguendes.me/python-isdigit-isnumeric-isdecimal

Método	Que devuelve	Ejemplos
isdigit()	Verdadero si **todos** los caracteres de la cadena son **dígitos** y la cadena no está vacía. Los espacios, puntos y el signo (+, -) **no son** considerados dígitos. Los caracteres Unicode que representan números **son** considerados dígitos.	"12345".isdigit() True "123.45".isdigit() False "-12345".isdigit() False "a12345".isdigit() False " 12345".strip().isdigit() True "\u2460".isdigit() # ① True "2⁷".isdigit() # '2'+'\u2077' True
isdecimal()	Verdadero si **todos** los caracteres de la cadena son **decimales** (números que se usan en base 10) y la cadena no está vacía. Los espacios, puntos y el signo (+, -) **no son** considerados decimales. Los caracteres Unicode que representan números **no son** considerados decimales.	"12345".isdecimal() True "123.45".isdecimal() False "\u2460".isdecimal() # ① False "2⁷".isdecimal() #'2'+'\u2077' False

isnumeric()	Verdadero si **todos** los caracteres de la cadena son **caracteres numéricos** y la cadena no está vacía. La diferencia entre un carácter numérico y un dígito es que un dígito es un único valor Unicode, mientras que un carácter numérico es cualquier símbolo Unicode que represente un valor numérico incluidas las fracciones, los números romanos, …	"12345".isnumeric() True "123.45".isnumeric() False " 12345".strip().isnumeric() True "\u2460".isnumeric() # ① True "⅕".isnumeric() # '\u2155' True "Ⅺ".isnumeric() # '\u216A' True

3.1.5. Verificar si una cadena contiene un float o un número negativo.

La mejor forma de verificar si una cadena contiene un float o es un número negativo es hacer un cast a float. Donde puede pasar dos cosas:

- Si no se lanzar una excepción: la cadena representa un float.
- Si se lanza la excepción: la cadena no representa un float.

Python sigue la filosofía EAFP: Easier to ask for forgiveness than permission (es más fácil pedir perdón que pedir permiso).

Este estilo de codificación prefiere suponer que el código es correcto y captura excepciones en caso contrario. Se caracteriza por tener muchas sentencias try … except.

Frente a este estilo se sitúa LBYL: Look before you leap (mira antes de saltar) usada un otros lenguajes como java, c, … En este estilo antes de ejecutar un código se verifican que se cumplen una serie de condiciones. Se caracteriza por muchas sentencias if.

3.1.6. Búsqueda y sustitución

Método	Que devuelve	Ejemplos con: "hoLA mUNdo"
count(subcadena [, inicio[, fin]]):	Número de veces que aparece la subcadena en la cadena. Si se especifica inicio empezara a contar desde esa posición y si se especifica fin terminara de contar en esa posición. Presente en todas la secuencias	cadena.count("o") 2 cadena.count("o", 3) 1 cadena.count("o", 3, -1) 0
index(subcadena [, inicio[, fin]])	Posición de la primera ocurrencia de subcadena en la cadena. Si se especifica inicio empezara a buscar desde esa posición y si se especifica fin terminara la búsqueda en esa posición. Si no encuentra la subcadena lanza una excepción. Presente en todas la secuencias	cadena.index("o") 1 cadena.index("o", 3) 9 cadena.index("o", 3, -1) ValueError: substring not found cadena.index("no") ValueError: substring not found
find(subcadena, [inicio], [fin])	Es igual a index excepto que si no encuentra la cadena devuelve -1	cadena.find("o") 1 cadena.find("no") -1

rfind(subcadena, [inicio], [fin])	rfind es igual a find pero comenzando a buscar desde el final. El índice que devuelve es desde el principio.	cadena.rfind("o") 9 # La primera o que encuentra # es la del final con índice 9
replace(origen, destino [, cuantas])	En la cadena original sustituye todas las apariciones de origen por destino. Si se especifica el parámetro cuantas indica el número máximo de ocurrencias a reemplazar	cadena.replace("o"," - ") h - LA mUNd – cadena.replace("o"," - ", 1) h - LA mUNdo

3.1.7. Conversión entre listas y cadenas

Método	Que devuelve	Ejemplos con: "hoLA mUNdo"
split([,sep] [,maxSplit])	Devuelve una lista con las subcadenas resultante de dividir la cadena original por caracteres de espacio. Si se especifica sep se usara esta cadena como divisor de las cadenas. MaxSplit indica el número máximo de divisiones, siendo -1 sin límite.	cadena.split() ['hoLA', 'mUNdo'] cadena.split('o') ['h', 'LA mUNd', ''] cadena.split('o', 1) ['h', 'LA mUNdo']
"cad".join (secuencia)	Un todos los elementos de una secuencia en una única cadena. Los Elementos de la secuencia tienen que ser cadenas. Usa cad como separador de la unión.	"-".join(['hoLA', 'mUNdo',"nuevo"]) hoLA-mUNdo-nuevo "".join(['hoLA', 'mUNdo',"nuevo"]) hoLAmUNdonuevo "-".join(['hoLA', 'mUNdo', 5]) sequence item 2: expected str instance, int found
list(secuencia)	Convierte cualquier secuencia en una lista.	list(cadena) ['h', 'o', 'L', 'A', ' ', 'm', 'U', 'N', 'd', 'o']

3.1.8. Códigos Unicode

Para trabajar con códigos Unicode tenemos las dos siguientes funciones:

Método	Que devuelve	Ejemplos
ord(char)	Dado un carácter no devuelve su código Unicode.	ord("a") 97 ord("①") 9312 ord("\u2460") 9312
chr(cod_unicode)	Dado un código Unicode devuelve el carácter que tiene asociado.	print(chr(97)) a print(chr(9312)) ①

3.1.9. Comparación de cadenas

Podemos usar los operadores de comparación (<,<=, ==, i=, >, >=) para comparar cadenas. La comparación se realiza comparando un código Unicode carácter a carácter y de izquierda a derecha.

```
print("abc">"cde")
  False
```

```
print("abc"<"abe")
  True
```

No se deben comparar números como cadenas ya que puede dar resultados inesperados:

```
print("2">"1000")
  True
```

Se deben comparar como números:

```
print(int("2")>int("1000"))
  False
```

Para más información se puede consultar el siguiente enlace:

https://miguendes.me/python-compare-strings

4. Conjuntos

Los conjuntos son objetos inmutables que contienen elementos, heterogéneos, no ordenados donde no hay repeticiones: cada elemento solo puede aparecer una vez. Como no están ordenador no se puede acceder a ellos en función de un índice sino que hay que iterar sobre su contenido.

Se definan como las listas y la tuplas pero usado llaves { y } para definir sus elementos.

```
conjunto={"uno", "dos", "tres", "uno"}
print(conjunto)
  {'tres', 'uno', 'dos'}
```

Para saber su tamaño tenemos, al igual que en otras secuencias, la función len:

```
print(len(conjunto))
  3
```

O las funciones: max y min:

```
print(max(conjunto), min(conjunto))
  uno dos
```

O los operadores in y not in con el mismo significado que ya hemos visto.

Además tenemos los siguientes métodos propios:

Método	Que devuelve	Ejemplos
add(elemento/s)	En un conjunto no se pueden modificar los elementos que contiene pero se puede añadir nuevos elementos.	conjunto.add('cinco') {'uno', 'dos', 'cinco', 'tres'}
update(sec)	Permite añadir a un conjunto los elementos de otra secuencia.	conjunto2={"diez", "once"} conjunto.update(conjunto2) {'uno', 'tres', 'diez', 'dos', 'once'} lista=["diez", "once"] conjunto.update(lista) {'uno', 'tres', 'diez', 'dos', 'once'}

remove(elemen)	En un conjunto no se pueden modificar los elementos que contiene pero se puede eliminar elementos. Si el elemento no existe lanza una excepción.	conjunto.remove('dos') {'uno', 'tres'} conjunto.remove('doce') KeyError: 'doce'
discard(elemen)	Es igual que remove pero sin lanzar una excepción si el elementos no existe.	conjunto.discard('dos') {'uno', 'tres'} conjunto. discard ('doce') {'uno', 'tres'}
pop()	Elimina un elemento aleatorio del conjunto y lo devuelve.	conjunto.pop() {'dos', 'tres'}
clear()	Vacía el conjunto	conjunto.clear() set()
del conjunto	Elimina el conjunto y ya no se puede usar	del conjunto

El los conjuntos existe la noción matemática de unión, intersección y diferencia de conjuntos. Tomamos un nuevo conjunto como:

```
conj2={"diez", "once", "dos"}
```

Método	Que devuelve	Ejemplos
unión(sec)	Un nuevo conjunto con los elementos de ambas secuencias.	nuevo=conjunto.union(conj2) print(nuevo) {'dos', 'uno', 'once', 'diez', 'tres'}
intersection(sec)	Un nuevo conjunto con los elementos comunes	nuevo=conjunto.intersection(conj2) print(nuevo) {'dos'}
differrence(sec)	Un nuevo conjunto con los elementos del conjunto que no están en la secuencia.	nuevo=conjunto.difference(conj2) print(nuevo) {'uno', 'tres'}
symmetric_difference(sec)	Un nuevo conjunto con todos los elementos menos los duplicados.	nuevo=conjunto.symmetric_difference(conj2) print(nuevo) {'tres', 'diez', 'uno', 'once'}

Podemos ver una lista completa en el siguiente enlace.

https://www.w3schools.com/python/python_sets_methods.asp

5. Diccionarios

Son una estructura de datos (no son secuencias) mutable que permite almacenar pares clave:valor. Donde a cada clave se le asigna un valor. Desde la versión 3.7 son ordenados.

Las claves son de un tipo de datos de Python (sin son cadenas son case sensitive) mientras que los valores pueden ser de cualquier tipo o estructura de datos de Python incluso otros diccionarios dando lugar a diccionarios anidados.

Se definen entre llaves y separando las claves y valores por dos putos.

El acceso no es indexado si no que es por su clave. Si esta no existe lanza un excepción

```
ruedas ={"moto":2, "coche":4, "camion":16, "barco":0}
ruedas ["camion"]
```
```
16
```

```
ruedas ["tren"]
```
```
KeyError: 'tren'
```

Se puede modificar sus valores:

```
ruedas ["barco"]=3
print(ruedas)
```
```
{'moto': 2, 'coche': 4, 'camion': 16, 'barco': 3}
```

Para saber su tamaño tenemos la función len:

```
print(len(ruedas))
```
```
4
```

Y los operadores in y not in que nos indica si una clave pertenece o no al diccionario.

```
print("moto" in ruedas)
```
```
True
```

```
print("tren" in ruedas)
```
```
False
```

Los métodos más interesantes que posee son:

Método	Que devuelve	Ejemplos
get(mi_clave [, defecto])	Devuelve el valor del elemento con clave miclave. Si la clave no existe devuelve None. Si la clave no existe y se ha definido un valor por defecto, segundo parámetro, devuelve este valor. Es similar a: ruedas ["camion"]	ruedas.get("camion") 16 ruedas.get("tren") None ruedas.get("tren", "No existe") No existe
update(diccio)	Permite añadir al diccionario pares clave:valor.	ruedas.update({"tren":20, "monociclo":1}) {'moto': 2, 'coche': 4, 'camion': 16, 'barco': 0, 'tren': 20, 'monociclo': 1}
setdefault(miclave, valor)	Devuelve el elemento con clave miclave. Si no existe inserta el par clave:valor.	ruedas.setdefault("coche",20) 4 ruedas.setdefault("tren",20) 20 print(ruedas) {'moto': 2, 'coche': 4, 'camion': 16, 'barco': 0, 'tren': 20}
keys()	Lista con las claves del diccionario.	ruedas.keys() dict_keys(['moto', 'coche', 'camion', 'barco'])
values()	Lista con las valores del diccionario.	ruedas.values() dict_values([2, 4, 16, 0])
items()	Lista de las claves: valor como tuplas	ruedas.items() dict_items([('moto', 2), ('coche', 4), ('camion', 16), ('barco', 0)])

pop(mi_clave [, defecto])	Elimina y devuelve el elemento con clave miclave. Si la clave no existe lanza un excepción. Si la clave no existe y se ha definido un valor por defecto, segundo parámetro, devuelve este valor. Es similar a: del ruedas["moto"]	ruedas.pop("moto") 2 print(ruedas) {'coche': 4, 'camion': 16, 'barco': 0}
popitem()	Elimina y devuelve el último elemento insertado en el diccionario.	print(ruedas.popitem()) ('barco', 0) print(ruedas) {'moto': 2, 'coche': 4, 'camion': 16}
clear()	Vacía el diccionario	ruedas.clear(); {}
copy()	Copia en diccionario en otro.	ruedas2=ruedas.copy() id(ruedas)==id(ruedas2) False
del diccionario	Elimina el diccionario y ya no se puede usar.	del ruedas

Podemos encontrar una lista completa de métodos en el siguiente enlace:

https://www.w3schools.com/python/python_dictionaries_methods.asp

Para recorrer los elementos de un diccionario podemos usar:

```
for x in ruedas:  # En x vamos obteniendo las claves
    print(f"{x} {ruedas[x]}")
```

Otra forma de obtener las claves.

```
for x in ruedas.keys():  # En x vamos obteniendo las claves
    print(x)
```

Para obtener los valores.

```
for x in ruedas.values():  # En x vamos obteniendo los valores
    print(x)
```

Para obtener la clave y el valor

```
for clave, valor in ruedas.items(): # Devuelve una tupla y desempaqueta
 print(clave, valor)
```

Para obtener las tuplas con la clave y el valor

```
for tupla in ruedas.items(): # Devuelve una tupla
 print(tupla)
```

6. Conversión entre secuencias.

Cada secuencia tiene un método que permite:

- Definir la secuencia como si fuese un constructor.
- Convertir una secuencia en otra.

Veamos los siguientes ejemplos fijándonos en los dobles paréntesis.

- Listas: método list.

```
lista=list(("uno", "dos", "tres"))
print(lista, type(lista))
```

```
['uno', 'dos', 'tres'] <class 'list'>
```

- Tuplas: método tuple.

```
tupla=tuple(("uno", "dos", "tres"))
print(tupla, type(tupla))
   ('uno', 'dos', 'tres') <class 'tuple'>
```

- Conjunto: método set.

```
conjunto=set(("uno", "dos", "tres"))
print(conjunto, type(conjunto))
   {'uno', 'tres', 'dos'} <class 'set'>
```

Para convertir tipos:

```
# Se convierte una tupla y un conjunto a dos listas
print(list(tupla), list(conjunto))
  ['uno', 'dos', 'tres'] ['tres', 'uno', 'dos']
```

```
# Se convierte una lista y un conjunto a dos tuplas
print(tuple(lista), tuple(conjunto))
  ('uno', 'dos', 'tres') ('tres', 'uno', 'dos')
```

```
# Se convierte una lista y una tupla a dos conjuntos
print(set(lista), set(tupla))
  {'tres', 'uno', 'dos'} {'tres', 'uno', 'dos'}
```

7. Map

La función map aplica una función a cada elemento de una secuencia y devuelve una lista con el resultado de aplicar la función a cada elemento.

Su formato es:

```
map(funcion, secuencia):
```

Por ejemplo

```
def cifrar(cad):
    cad=cad.replace("a","4").replace("e","3").replace("i","1")
    cad=cad.replace("o","0").replace("u","5")
    return cad

palabras=["leon", "aguila", "sepia", "murcielago"]
palabras_cifradas=map(cifrar, palabras)
print(list(palabras_cifradas))
  ['l30n', '4g51l4', 's3p14', 'm5rc13l4g0']
```

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

x = map(une, ('uno', 'dos', 'tres'), ('cuatro', 'cinco', 'seis'))
print(list(x))
  ['unocuatro', 'doscinco', 'tresseis']
```

8. Glosario

https://docs.python.org/es/3/glossary.html

9. Ejercicios

1. Función que continúe pidiendo una cadena mientras solo se introduzcan caracteres de espacio.
2. Función a la que se le pasa una cadena y la devuelve al revés.
3. Función al que se le pase una cadena, una posición de inicio y una cantidad de caracteres y devuelve el fragmento indicado. Si se le pasan parámetros no válidos devuelve cadena vacía.
4. Función que indique si en una cadena todas las palabras comienzan con mayúsculas.
5. Escribe una función que compruebe si una palabra es un palíndromo. Un palíndromo es una palabra o frase que se lee igual al derecho y al revés.
6. Función que dada una cadena devuelve solo las letras mayúsculas que contenga.
7. Crea una función que reciba una frase y devuelva el número de palabras de la misma.
8. Crea un programa en Python que pida al usuario una cadena y le muestre solo la primera letra de cada palabra de la frase en mayúsculas.
9. Crea una función que reciba una cadena e invierta el orden de las palabras.
10. Crea una función que reciba el nombre y los dos apellidos de un usuario y devuelva su identificador, que se formará con las dos primeras letras de su nombre y sus apellidos en mayúsculas.

    Por ejemplo para "Xiana Fernández Rodríguez" su identificador será `XIFERO`.

11. Vamos a jugar a palabras encadenadas. Implementa un programa en Python que a partir de una cadena, indique si las palabras que la forman están o no encadenadas. Supón que solamente tenemos en cuenta las dos primeras y las dos últimas letras de cada palabra.

    mata tapa papa pato
    ```
    Palabras encadenadas
    ```

    seto taco coma matute
    ```
    Palabras NO encadenadas
    ```

    sien encima mapa patuco comida
    ```
    Palabras encadenadas
    ```

    cata tasama malote tejaba batama
    ```
    Palabras encadenadas
    ```

    kiosko comida
    ```
    Palabras NO encadenadas
    ```

12. Crea una matriz de tamaño NxM y que en cada posición almacena un carácter aleatorio entre 'A' y 'Z'. Muestra el contenido de la matríz.

13. A partir de dos cadenas, devuelve un conjunto con las letras que estén en ambas.
14. Escribe un programa que a partir de dos tuplas con el mismo número de elementos, una con nombres y otra con edades las convierta en un diccionario donde las claves sean los nombres y los valores sean las edades.
15. Con la intención de organizar mejor la FCT, crea varios conjuntos, uno por cada lenguaje de programación que conozcas, y añade a cada uno los alumnos que sepan ese lenguaje.

    Acaba de llegar una oferta de empleo que requiere que los candidatos sepan Java, C# y Python. ¿Qué alumnos o alumnas podrían optar a ella?
16. Crea una función de devuelva el número de veces que aparece cada carácter en una cadena de texto.
17. Crea una función de devuelva un diccionario con el número de veces que aparece cada palabra en una cadena de texto.
18. Escribe una función que compruebe si una frase (puede contener espacios y signos de puntuación) es un palíndromo.
19. Cifrado César.

    Uno de los métodos más antiguos para codificar mensajes es el conocido como cifrado Cesar. Su funcionamiento es simple: cada una de las letras del mensaje original es sustituida por otra letra que se encuentra un número fijo de posiciones más adelante en el alfabeto.

    Así, si utilizamos un desplazamiento de 2, las apariciones de la letra 'a' se sustituyen por la 'c', todas las apariciones de la 'b' por 'd', etc. El método tradicional comienza de nuevo al llegar al final del alfabeto, de forma que, con el desplazamiento de 2, la 'y' se sustituye por la 'a' y la 'z' se sustituye por la 'b'.

    Los desplazamientos también pueden ser negativos; si utilizamos un desplazamiento de -1, la 'E' se convertirá en 'D', mientras que la 'a' pasará a ser 'z'.

    Nuestro cifrado Cesar no codifica los caracteres que no sean letras anglosajonas. Así, por ejemplo, los espacios o los símbolos de puntuación no sufrirán cambio alguno.

    Crea una función que reciba un mensaje y el desplazamiento y devuelva el mensaje cifrado.

    Crea una función que reciba el mensaje cifrado y el desplazamiento que se usó para cifrarlo y devuelva el mensaje descifrado.

    Puedes usar esta aplicación para comprobar tus funciones:

    https://eduescaperoom.com/cifrado-cesar/
20. Realiza el juego del ahorcado. Las palabras inicialmente estarán en un vector de 20 posiciones. Trabaja sólo con mayúsculas. Debe aparecer el dibujo correspondiente a cada fallo.