Cadena de caracteres, Tuplas y Conjuntos congelados

07

Curso Básico python

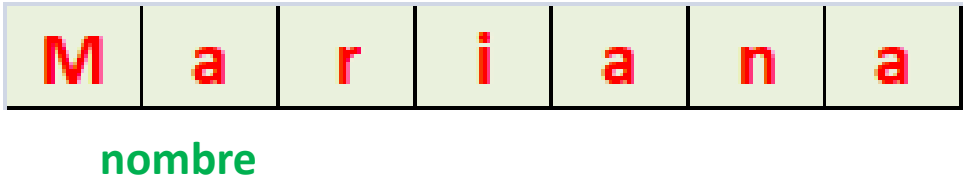
Tipos de Datos

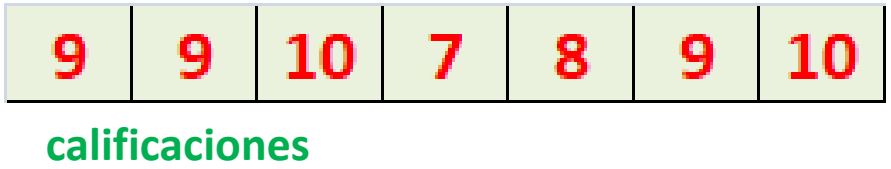
Conceptos generales

Tanto los datos simples como compuestos en Python son tratados como un objeto

- Simples. Tiene asociado un único valor: un entero, un real o un booleano. Son objetos escalares por ser indivisibles, es decir no tienen una estructura interna accesible.

- Compuestos. como los textos (string), las secuencias en forma de listas. Se pueden dividir en elementos y acceder a ellos, son **datos estructurados**.





Datos estructurados o compuestos

- Formados por datos simples u otros datos compuestos.
- Datos estructurados homogéneos: Todos los elementos son del mismo tipo: string
- Datos estructurados heterogéneos: Los elementos pueden ser de distinto tipo: listas, diccionarios…

Datos estructurados o compuestos

- Clasificación de acuerdo a la característica de si sus elementos pueden o no ser cambiados, reducidos o ampliados

- Inmutables (estáticos) valores/tamaño fijos, sus elementos no pueden ser cambiados ni eliminados, tampoco se pueden añadir elementos nuevos. Si se quiere modificar este tipo de datos se utiliza el recurso de crear un nuevo valor.

- Mutables (dinámicos): sus elementos pueden cambiar de valor y se puede añadir o eliminar elementos

Datos estructurados o compuestos

- Inmutables (estáticos):
  - Cadenas de caracteres (string)
  - Tuplas (tuple)
  - Conjuntos congelados (frozenset)
- Mutables (dinámicos):
  - Listas
  - Conjuntos
  - Diccionarios

Cadena de caracteres (string)

Cadena de caracteres (string)

- Es una secuencia de caracteres (letras, números, caracteres especiales; cualquier carácter Unicode)

•Acceso: s[0]

•Tamaño: len(s)

•Los strings son inmutables

```
>>> s = 'casa de madera'
>>> letra_1 = s[0]
>>> long = len(s)
>>> letra_ultima = s[long-1]     # alternativa: s[-1]
>>> print(letra_1, letra_ultima, long)
c a 14
```

c	a	s	a		d	e		m	a	d	e	r	a
0	1	2	3	4	5	6	7	8	9	10	11	12	13
-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Recorte o rebanado (slicing)

- Para extraer un subconjunto de elementos (o segmento) de un string o de cualquier secuencia se usa el operador de corte (slice) **[n:m]**, donde **n** es el primer elemento a extraer y **m-1** el último.

```
>>> s = 'casa de madera'
>>> segm1 = s[0:3]        # segm1 <- 'cas'
>>> segm1 = s[:3]         # segm1 <- 'cas' , equivale al slice anterior
>>> segm2 = s[8:len(s)]   # segm2 <- 'madera'
>>> segm2 = s[8:]         # segm2 <- 'madera' , equivale al slice anterior
>>> segm3 = s[0:14:2]   # segm3 <- 'cs emdr', slice 0:14 en pasos de 2 en 2
>>> letra_u = s[-1]     # letra_u <- 'a', equivale a acceso último elemento
>>> letra_penu = s[-2] # letra_penu <- 'r', equivale acceso penúltimo elem
```

Operadores: concatenar (+) y repetir (*)

```
>>> s1='casa'
>>> s2 = s1 + ' grande'
>>> s2
'casa grande'
>>> s3 = 3*s1 + '!'
>>> s3
'casacasacasa!'
```

Operador **in**

- El operador **in** se considera un operador booleano sobre dos strings y devuelve **True si el string de la izquierda es un segmento (o substring) del de la derecha**. Si no lo es, devuelve False.
- El operador **not in** devuelve el resultado lógico opuesto.

```
>>> s = 'casa de madera'
>>> 'asa' in s
True
>>> 'casade' in s
False
>>> 'casade' not in s
True
```

Métodos de los strings

- upper toma un string y devuelve otro string pero con las letras en mayúsculas. El método se aplica sobre sus propios valores: cad.upper()

```
>>> s = 'casa de madera'
>>> sM = s.upper()  # convierte las letras en mayúsculas
>>> sM
'CASA DE MADERA'
>>> sM.lower()      # convierte las letras en minúsculas
'casa de madera'
>>> s.capitalize() # primera letra del string en mayúscula
'Casa de madera'
>>> s.title()       # primera letra de cada palabra del string en mayúscula
'Casa De Madera'
>>> i = s.find('e') # busca el índice (posición) del primer string 'e'
>>> i               # si no encontrara el string devuelve -1
6
```

Métodos de los strings

```
>>> s = 'casa de madera'
>>> s.count('a')    # cuenta las veces que aparece el elemento o string
4
>>> s.count('de')
2
>>> s.replace('a','e')    # remplaza el primer string por el segundo
'cese de medere'
>>> s.split(' ')    # parte s usando el string ' ' produciendo lista
['casa', 'de', 'madera']
>>> s1 = 'Hola'
>>> s1.isupper()    # True si todos los caracteres en S son mayúsculas
False               # Falso en caso contrario
>>> s1[0].isupper()
True
>>> s1.islower()    # True si todos los caracteres en S son minúsculas
False               # Falso en caso contrario
>>> s1[1].islower()
True
```

Comillas dentro de comillas

Se pueden escribir comillas simples en cadenas delimitadas con comillas dobles y viceversa:

```
>>> print("Las comillas simples ' delimitan cadenas.")
Las comillas simples ' delimitan cadenas.
>>> print('Las comillas dobles " delimitan cadenas.')
Las comillas dobles " delimitan cadenas.
```

Pero no se pueden escribir en el interior de una cadena comillas del mismo tipo que las comillas delimitadoras:

```
>>> print("Las comillas dobles " delimitan cadenas")
SyntaxError: invalid syntax
>>>
```

Caracteres especiales

Los caracteres especiales empiezan por una contrabarra (\).

- Comilla doble: \"

```
>>> print("Las comillas dobles \" delimitan cadenas.")
Las comillas dobles " delimitan cadenas.
```

- Comilla simple: \'

```
>>> print('Las comillas simples \' delimitan cadenas.')
Las comillas simples ' delimitan cadenas.
```

- Salto de línea: \n

```
>>> print("Una línea\nOtra línea")
Una línea
Otra línea
```

- Tabulador: \t

```
>>> print("1\t2\t3")
1       2       3
```

Cadenas "f"

- En Python 3.6 se añadió una nueva notación para cadenas llamada cadenas "f", que simplifica la inserción de variables y expresiones en las cadenas. Una cadena "f" contiene variables y expresiones entre llaves ({}) que se sustituyen directamente por su valor. Las cadenas "f" se reconocen porque comienzan por una letra f antes de las comillas de apertura.

```
semanas = 4
print(f"En {semanas} semanas hay {7 * semanas} días.")
```

```
nombre = "Pepe"
edad = 25
print(f"Si escribe {{nombre}} se escribirá el valor de la variable
nombre, "
      f"en este caso {nombre}.")
```

Tuplas
(tuple)

Tuplas

- Secuencia de elementos ordenados
- pueden contener elementos de cualquier tipo, incluso elementos de diferente tipo.
- Los elementos se indexan, a través de un número entero.
- La sintaxis de las tuplas es una secuencia de valores separados por comas. Aunque no son necesarios, se suelen encerrar entre paréntesis,

Tuplas (ejemplos)

```
# Ejemplo de tuplas
>>> a = 1, 2, 3
>>> a
(1, 2, 3)
>>> b = (3, 4, 5, 'a')
>>> b
(3, 4, 5, 'a')
>>> type(a)
<class 'tuple'>
>>> type(b)
<class 'tuple'>
```

Lo importante es incluir las comas entre los elementos. Por ejemplo

```
>>> t = 'k',
>>> t
('k',)
>>> type(t)
<class 'tuple'>
>>> t2 = 'k'
>>> t2
'k'
>>> type(t2)
<class 'str'>
```

Tuplas

- Se puede crear una tupla vacía usando paréntesis sin que incluya nada: ().
- tuple(), convierte una secuencia iterable, como un string o una lista, a tupla, también puede crear una tupla vacía

```
>>> tuple('Hola')
('H', 'o', 'l', 'a')
>>> tuple([1, 2])
(1, 2)
>>> tuple()
()
```

Indexación, recorte y otras operaciones de tuplas

- Tamaño: len()
- Acceso: tupla[indice]
- Slicing: tupla[n:m]
- Operadores *, +
- Las tuplas son inmutables
- Son Iterables (se puede usar for)
- Se pueden incluir tuplas dentro de tuplas

Ejemplos

```
>>> b = (3, 4, 5, 'a')
>>> b[0]
3
>>> b[-1]
'a'
>>> b[0:3]
(3, 4, 5)
>>> t = ('las','tuplas','son','inmutables')
>>> t[0]
'las'
>>> t[1] = 'listas'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

```
>>> b = (3, 4, 5, 'a')
>>> c = (b, 2)
>>> b + c
(3, 4, 5, 'a', (3, 4, 5, 'a'), 2)
>>> 3*b
(3, 4, 5, 'a', 3, 4, 5, 'a', 3, 4, 5, 'a')
```

Asignaciones múltiples

- Python permite asignaciones múltiples mediante asignaciones con tuplas. Estas acciones permiten que a una tupla de variables a la izquierda de una asignación le sea asignada una tupla de valores a la derecha de ésta. La condición a cumplir es que el número de variables de la tupla de variables sea igual al número de elementos de la tupla de valores.

```
>>> a,b,c = (1,2,3)
>>> a
1
>>> type(a)
<class 'int'>
>>> d,e,f = 'xyz'
>>> d
'x'
>>> type(d)
<class 'str'>
```

```
>>> x = 5
>>> y = 7
>>> x, y = y, x
>>> print(x, y)
7 5
```

Métodos de las tuplas

- count()
- index()

```
>>> valores = ("Python", True, "Zope", 5)
>>> print "True ->", valores.count(True)
True -> 1
>>> print "'Zope' ->", valores.count('Zope')
'Zope' -> 1
>>> print "5 ->", valores.count(5)
5 -> 1
```

```
>>> valores = ("Python", True, "Zope", 5)
>>> print valores.index(True)
1
>>> print valores.index(5)
3
```

Conjuntos congelados (frozenset)

Conjuntos congelados (Frozenset)

- Grupo de datos estructurados heterogéneos que tratan de guardar cierta relación con la teoría de conjuntos.
- Es una colección de elementos no ordenados que sean únicos (no estén repetidos) e inmutables.
- Puede contener números, string, tuplas, pero no listas.
- Los conjuntos congelados son inmutables porque no se pueden cambiar, ni quitar o añadir elementos

Conjuntos congelados (Frozenset)

```
>>> FS1 = frozenset({25, 4, 'a',2, 25, 'casa', 'a'})
>>> FS1
frozenset({2, 'a', 4, 'casa', 25})
>>> type(FS1)
<class 'frozenset'>
>>> len(FS1)
5
```

Cadena de caracteres, Tuplas y Conjuntos congelados

07

Curso Básico python