Tipo de datos binarios: bytes, bytearray

Bytes

El tipo `bytes` es una secuencia inmutable de bytes. Solo admiten caracteres ASCII. También se pueden representar los bytes mediante números enteros cuyo valores deben cumplir `0 <= x < 256`.

Definición de bytes. Constructor bytes

Podemos definir un tipo `bytes` de distintas formas:

```
>>> byte1 = b"Hola"
>>> byte2 = b'¿Qué tal?'
>>> byte3 = b'''Hola,
    que tal?'''
```

También podemos crear cadenas con el constructor bytes a partir de otros tipos de datos.

```
>>> byte1=bytes(10)
>>> byte1
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
>>> byte2=bytes(range(10))
>>> byte2
b'\x00\x01\x02\x03\x04\x05\x06\x07\x08\t'
>>> byte3=bytes.fromhex('2Ef0 F1f2')
>>> byte3
b'.\xf0\xf1\xf2'
```

Bytearray

El tipo `bytearray` es un tipo mutable de bytes.

Definición de bytearray. Constructor bytearray

```
>>> ba1=bytearray()
>>> ba1
bytearray(b'')
>>> ba2=bytearray(10)
>>> ba2
bytearray(b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00')
>>> ba3=bytearray(range(10))
>>> ba3
bytearray(b'\x00\x01\x02\x03\x04\x05\x06\x07\x08\t')
>>> ba4=bytearray(b"hola")
>>> ba4
bytearray(b'hola')
>>> ba5=bytearray.fromhex('2Ef0 F1f2')
>>> ba5
bytearray(b'.\xf0\xf1\xf2')
```

Operaciones básicas con bytes y bytearray

Como veíamos en el apartado "Tipo de datos secuencia" podemos realizar las siguientes operaciones:

- Recorrido
- Operadores de pertenencia: `in` y `not in`.
- Concatenación: `+`
- Repetición: `*`
- Indexación
- Slice

Entre las funciones definidas podemos usar: `len`, `max`, `min`, `sum`, `sorted`.

Los bytes son inmutables, los bytearray son mutables

```
>>> byte=b"hola"
>>> byte[2]=b'g'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'bytes' object does not support item assignment

>>> ba1=bytearray(b'hola')
>>> ba1[2]=123
>>> ba1
bytearray(b'ho{a')
>>> del ba1[3]
>>> ba1
bytearray(b'ho{')
```

Métodos de bytes y bytearray

```
byte1.capitalize  byte1.index       byte1.join        byte1.rindex      byte1.strip
byte1.center      byte1.isalnum     byte1.ljust       byte1.rjust       byte1.swapcase
byte1.count       byte1.isalpha     byte1.lower       byte1.rpartition  byte1.title
byte1.decode      byte1.isdigit     byte1.lstrip      byte1.rsplit      byte1.translate
byte1.endswith    byte1.islower     byte1.maketrans   byte1.rstrip      byte1.upper
byte1.expandtabs  byte1.isspace     byte1.partition   byte1.split       byte1.zfill
byte1.find        byte1.istitle     byte1.replace     byte1.splitlines
byte1.fromhex     byte1.isupper     byte1.rfind       byte1.startswith

bytearray1.append      bytearray1.index       bytearray1.lstrip      bytearray1.rstrip
bytearray1.capitalize  bytearray1.insert      bytearray1.maketrans   bytearray1.split
bytearray1.center      bytearray1.isalnum     bytearray1.partition   bytearray1.splitlines
bytearray1.clear       bytearray1.isalpha     bytearray1.pop         bytearray1.startswith
bytearray1.copy        bytearray1.isdigit     bytearray1.remove      bytearray1.strip
bytearray1.count       bytearray1.islower     bytearray1.replace     bytearray1.swapcase
bytearray1.decode      bytearray1.isspace     bytearray1.reverse     bytearray1.title
bytearray1.endswith    bytearray1.istitle     bytearray1.rfind       bytearray1.translate
bytearray1.expandtabs  bytearray1.isupper     bytearray1.rindex      bytearray1.upper
bytearray1.extend      bytearray1.join        bytearray1.rjust       bytearray1.zfill
bytearray1.find        bytearray1.ljust       bytearray1.rpartition
bytearray1.fromhex     bytearray1.lower       bytearray1.rsplit
```

Si nos fijamos la mayoría de los métodos en el caso de los `bytes` son los de las cadenas de caracteres, y en los `bytearray` encontramos también métodos propios de las listas.

Métodos encode y decode

Los caracteres cuyo código es mayor que 256 no se pueden usar para representar los bytes, sin embargo si podemos indicar una codificación de caracteres determinada para que ese carácter se convierte en un conjunto de bytes.

```
>>> byte1=b'piña'
  File "<stdin>", line 1
SyntaxError: bytes can only contain ASCII literal characters.
>>> byte1=bytes('piña',"utf-8")
>>> byte1
b'pi\xc3\xb1a'
>>> len(byte1)
5
>>> byte1=bytes('piña',"latin1")
>>> byte1
b'pi\xf1a'
```

Podemos también convertir una cadena unicode a bytes utilizando el método `encode` :

```
>>> cad="piña"
>>> byte1=cad.encode("utf-8")
>>> byte1
b'pi\xc3\xb1a'
```

Para hacer la función inversa, convertir de bytes a unicode utilizamos el método `decode`:

```
>>> byte1.decode("utf-8")
'piña'
```

El problema lo tenemos si hemos codificado utilizando un código e intentamos decodificar usando otro.

```
>>> byte1=bytes('piña',"latin1")
>>> byte1.decode("utf-8")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
UnicodeDecodeError: 'utf-8' codec can't decode byte 0xf1 in position 2: invalid continuation byte
>>> byte1.decode("utf-8","ignore")
'pia'
>>> byte1.decode("utf-8","replace")
'pi�a'
```