Лекція 10

Типи даних

`bytes` і `bytearray`

Навіщо потрібні типи даних `bytes` і `bytearray`

Тип **`str`** використовується для зберігання текстової інформації.

Але що робити, якщо необхідно обробляти зображення?

Адже зображення не має кодування, а отже, воно не може бути перетворене в Unicode-рядок.

Для розв'язування цієї проблеми в Python 3 були введені типи `bytes` і `bytearray`, які дозволяють зберігати послідовність цілих чисел у діапазоні від 0 до 255.

Кожне таке число позначає код символу.

Тип даних `bytes` є незмінюваним типом, як і рядки,

тип даних `bytearray` – змінюваний, як і списки.

Способи створення об'єкта типу bytes

Спосіб 1.

За допомогою функції

```
bytes([<Рядок>,<Кодування>[,<Обробка помилок>]])
```

```
Наприклад: bytes("рядок", "cp1251")
```

1. Якщо параметри не зазначені, то повертається порожній об'єкт.

2. Щоб перетворити рядок в об'єкт типу `bytes`, необхідно передати мінімум два перші параметри.

3. Перші два параметри є обов'язковими. Якщо рядок зазначений тільки в першому параметрі, то виконується виключення `Typeerror`.

Приклад 1

```
>>> bytes()
b''

>>> bytes("рядок", "cp1251")
b'\xf1\xf2\xf0\xee\xea\xe0'

>>> bytes("рядок")
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Nameerror: name 'bytes' is not defined
```

Третій параметр: <Обробка помилок>

"strict" (при помилці виконується виключення Unicodeencodeerror – значення за замовчуванням),

"replace" (невідомий символ заміняється символом питання)

"ignore" (невідомі символи ігноруються).

Приклад 2. Приклади обробки помилок

```
>>> bytes("string\ufffd","cp1251","strict")
Traceback (most recent call last):
  File "<input>", line 1, in <module>
  File "C:\Program Files (x86)\Python35-
32\lib\encodings\cp1251.py", line 12, in encode
    return
codecs.charmap_encode(input,errors,encoding_tabl
e)
Unicodeencodeerror: 'charmap' codec can't encode
character '\ufffd' in position 6: character maps
to <undefined>

>>> bytes("string\ufffd", "cp1251", "replace")
b'string?'

>>> bytes ( "string\ufffd", "cp1251", "ignore")
b'string'
```

Спосіб 2
За допомогою методу рядків

```
encode([encoding="utf-8"][, errors="strict"]).
```

```
Наприклад:"рядок".encode(encoding="cp1251")
```

1. Якщо кодування не зазначене, то рядок перетворюється в послідовність байтів у кодуванні UTF-8.

2. У параметрі `errors` можуть бути зазначені значення `"strict"` (значення за замовчуванням),

`"replace"`,`"ignore"`,`"xmlcharrefreplace"` або `"backslashreplace"`.

Приклад 3. Приклади застосування методу `encode`

```
>>> "рядок".encode() #utf-8
b'\xd1\x81\xd1\x82\xd1\x80\xd0\xbe\xd0\xba\xd0\xb0'

>>> "рядок".encode(encoding="cp1251")
b'\xf1\xf2\xf0\xee\xea\xe0'

>>>"рядок\ufffd".encode(encoding="cp1251",errors="xmlcharrefreplace")
b'\xf1\xf2\xf0\xee\xea\xe0&#65533;'

>>> "рядок\ufffd".encode(encoding="cp1251",errors="backslashreplace")
b'\xf1\xf2\xf0\xee\xea\xe0\\ufffd'
```

Спосіб 3 Додаванням префікса b

1. Указавши букву b або B перед рядком в апострофах, лапках, потрійних апострофах або потрійних лапках.
2. У рядку можуть бути тільки символи з кодами, що входять у кодування ASCII.
3. Усі інші символи повинні бути представлені спеціальними послідовностями:

Приклад 4

```
>>> b"string", b'string', b"""string""", b'''string'''
(b'string', b'string', b'string', b'string')

>>> b"рядок"
Syntaxerror: bytes can only contain ASCII
literal characters.

>>> b"\xf1\xf2\xf0\xee\xea\xe0"
b'\xf1\xf2\xf0\xee\xea\xe0'
```

Спосіб 4

За допомогою функції

bytes(<Послідовність>),

яка перетворює послідовність цілих чисел від 0 до 255 в об'єкт типу `bytes`.

Якщо число не попадає в діапазон, то виконується виключення `Valueerror`.

Приклад 5

```
>>> b = bytes([225, 226, 224, 174, 170, 160])
>>> b
b'\xe1\xe2\xe0\xae\xaa\xa0'

>>> str (b,"cp866")
'рядок'
```

Спосіб 5. (Задавання нульової послідовності)
За допомогою функції

bytes(<Число>),

яка задає кількість елементів у послідовності.

Кожний елемент буде містити нульовий символ:

Приклад 6

```
>>> bytes (10)
b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'


>>> bytes (5)
b'\x00\x00\x00\x00\x00'
```

Спосіб 6

За допомогою методу

```
bytes. fromhex (<Рядок>).
```

`<Рядок>` містить шістнадцяткові значення символів:

Приклад 6.

```
>>> b = bytes.fromhex('f0ffe4eeea')
>>> b
b'\xe1\xe2\xe0\xae\xaa\xa0'
>>> str(b, "cp1251")
'рядок'
```

1. Об'єкти типу `bytes` – це послідовності.
2. Елемент зберігає ціле число від 0 до 255, яке позначає код символу.
3. Об'єкти підтримують:
   - доступ до елемента по індексу,
   - одержання зрізу,
   - конкатенацію,
   - повторення й перевірку на входження:

Приклад 7. Приклади операцій з типом bytes

```
>>> b = bytes ( "string", "cp1251")
>>> b
b'string'
>>> b[0]                    # Доступ по індексу
115
>>> b[1:3]                  # Одержання зрізу
b'tr'

>>> b+b"123"                # Конкатенація
b'string123'

>>> b*3                     # Повторення
b'stringstringstring'

>>> 115 in b, b"tr" in b, b"as" in b
(True, True, False)
```

Приклад показав, що

- при виводі об'єкта цілком і при добуванні зрізу, проводиться спроба відображення символів.

```
>>> b[1:3]                    # Одержання зрізу
b'tr'
```

- доступ по індексу повертає ціле число, а не символ.

Якщо перетворити об'єкт у список, то ми одержимо послідовність цілих чисел:

Приклад 8

```
>>># Перетворення в список
>>> list(bytes("string","cp1251"))
[115, 116, 114, 105, 110, 103]

>>> # Доступ по індексу
>>> b[0]
115
```

Тип `bytes` є незмінюваним типом

Це означає, що можна одержати значення по індексу, але змінити його не можна:

Приклад 9

```
>>> b = bytes ("string", "cp1251")
>>> b[0] = 168
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Typeerror: 'bytes' object does not support item
assignment
```

1. Більшість рядкових методів з типу `str` підтримують об'єкти типу `bytes`
2. Деякі із цих методів можуть некоректно працювати з кирилицею.
3. У такому випадку використовують тип `str`, а не тип `bytes`.

Рядкові методи, не підтримувані об'єктами типу `bytes`

```
encode(),
isidentifier(), isprintable (),
isnumeric (),   isdecirnal (),
format()
```

При використанні методів слід враховувати, що в параметрах потрібно вказувати об'єкти типу `bytes`, а не рядка:

Приклад 10

```
>>> b= bytes( "string", "cp1251")
>>> c=b.replace(b"s", b"S")
>>> c
b'String'
>>> b
b'string' #сам об'єкт не змінився
>>> c is b
False
```

У виразах не можна змішувати рядки й об'єкти типу `bytes`

Попередньо необхідно явно перетворити об'єкти до одного типу, а лише потім виконувати операцію:

Приклад 11

```
>>># Помилка при додаванні об'єктів різних типів
>>> b"string" + "string"
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Typeerror: can't concat bytes to str

>>> # Рядок привели до типу bytes
>>> b"string" + "string".encode("ascii")
b'stringstring'
```

Однобайтні та багатобайтні дані в типі bytes

1. Об'єкт типу `bytes` може містити як однобайтні символи, так і багатобайтні.
2. При використанні багатобайтних символів деякі функції змінюють поведінку – наприклад, функція `len()` поверне кількість байтів, а не символів:

Приклад 12

```
>>> #Робота з типом str
>>> len("рядок")
6
>>> #Робота з типом bytes у кодуванні cp1251
>>> len(bytes("рядок","cp1251"))
6
>>> #Робота з типом bytes у кодуванні utf-8
>>> len(bytes("рядок", "utf-8"))
12
```

Метод decode() для об'єктів типу bytes

Перетворити об'єкт типу bytes у рядок дозволяє метод decode(). Метод має наступний формат:

```
decode([encoding="utf-8"][,errors="strict"])
```

1. Параметр encoding задає кодування символів (за замовчуванням UTF-8) в об'єкті bytes.
2. Параметр errors – задає обробку помилок при перетворенні за допомогою параметрів:

"strict" (значення за замовчуванням), "replace" "ignore".

Приклад 13

```
>>> b = bytes("рядок", "cp1251")
>>> b.decode(encoding="cp1251"), b.decode("cp1251")
('рядок', 'рядок')
```

Перетворення за допомогою функції str()

Приклад 14

```
>>> b = bytes("рядок", "cp1251")
>>> str(b, "cp1251")
'рядок'
```

Зміна кодування

1. Щоб змінити кодування даних, слід спочатку перетворити тип `bytes` у рядок.
2. Потім зробити зворотне перетворення, указавши потрібне кодування.

Перетворимо дані з кодування Windows-1251 у кодування KOI8-R, а потім назад

Приклад 15

```
>>> w = bytes("Рядок", "cp1251")
#w-"cp1251" k-"koi8-r"
>>> k = w.decode("cp1251") .encode("koi8-r")
>>> k, str(k, "koi8-r")#Данные в кодуванні KOI8-R
(b'\xf3\xd4\xd2\xcf\xcb\xc1', 'Рядок')

#k-"koi8-r" w- "cp1251"
>>> w = k.decode("koi8-r").encode("cp1251")
>>> w, str (w, "cp1251")
(b'\xd1\xf2\xf0\xee\xea\xe0', 'Рядок')
```

Підсумок по типу bytes

Перетворення в bytes:

1.bytes("рядок"), **bytes**([<Рядок>,<Кодування>[,<Обробка помилок>]])

2."рядок".encode(),**encode**([encoding="utf-8"][, errors="strict"])

3.b'string', додавання b: ~~b"рядок"~~ .

4.bytes([225, 226]), Перетворення list⇒bytes

5.bytes (10), Пустий bytes

6.bytes.fromhex('f0ffe4eeea') 16-ве ⇒ bytes

Операції з bytes:

b[0], b[1:3], b"wdf"+b"123", b*3, "w" in b

Перетворення в str:

b.decode(encoding="cp1251"),

str(b, "cp1251")

Тип даних `bytearray`

1. Тип даних `bytearray` є різновидом типу `bytes` і підтримує ті ж самі методи й операції.

2. На відміну від типу `bytes`, тип `bytearray` допускає можливість безпосередньої зміни об'єкта й містить додаткові методи, що дозволяють виконувати ці зміни.

Створити об'єкт типу `bytearray` можна такими способами:

Спосіб 1
За допомогою функції

```
bytearray([<Рядок>,<Кодування>[,<Обробка помилок>]])
```

```
Наприклад: bytearray("рядок", "cp1251")
```

1. Якщо параметри не зазначені, то повертається порожній об'єкт.

2. Щоб перетворити рядок в об'єкт типу `bytearray`, необхідно передати мінімум два перші параметри.

3. Якщо рядок зазначений тільки в першому параметрі, то виконується виключення `Typeerror`.

Приклад 16

```
>>> bytearray()
bytearray(b'')      #повертає порожній об'єкт
>>> bytearray("рядок", "cp1251")
bytearray(b'\xf1\xf2\xf0\xee\xea\xe0')
>>> bytearray ("рядок ")
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Typeerror: string argument without an encoding
```

Третій параметр:

"strict" (при помилці виконується виключення Unicodeencodeerror – значення за замовчуванням),

"replace" (невідомий символ заміняється символом питання) або

"ignore" (невідомі символи ігноруються).

Приклад 17. Третій параметр

```
>>> bytearray("string\ufffd", "cp1251","strict")
Traceback (most recent call last):
  File "<input>", line 1, in <module>
  File "C:\Program Files(x86)\Python35-32\lib\
encodings\cp1251.py",line 12,in encode return
codecs.charmap_encode(input,errors,encoding_table)
Unicodeencodeerror: 'charmap' codec can't encode
character '\ufffd' in position 6: character maps
to <undefined>
```

```
>>> bytearray("string\ufffd","cp1251", "replace")
bytearray(b'string?')
```

```
>>> bytearray ("string\ufffd", " cp1251", "ignore")
bytearray(b'string')
```

Спосіб 2

За допомогою функції `bytearray(<Послідовність>)`

1. Перетворює послідовність цілих чисел від 0 до 255 в об'єкт типу `bytearray`.
2. Якщо число не потрапляє в діапазон, то виконується виключення `Valueerror`.

Приклад 18

```
>>> b = bytearray([224, 239, 164, 174, 170])

>>> b
bytearray(b'\xe0\xef\xa4\xae\xaa')

>>> str(b, "cp866")
'рядок'
```

Спосіб 3

За допомогою функції `bytearray(<Число>)`

1. Задає кількість елементів у послідовності.
2. Кожний елемент буде містити нульовий символ:

Приклад 19

```
>>> bytearray(5)
bytearray(b'\x00\x00\x00\x00\x00')

>>> bytearray(10)
bytearray(b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00')
```

Спосіб 4

За допомогою методу `bytearray.fromhex(<Рядок>)`. Рядок у цьому випадку повинен містити шістнадцяткове значення символів:

Приклад 20

```
>>> c = bytearray.fromhex('f0ffe4eeea')
>>> c
bytearray(b'\xf0\xff\xe4\xee\xea')
>>> str(c, "cp1251")
'рядок'
```

1. Тип `bytearray` є змінюваним типом.

2. Можна не тільки одержати значення по індексу, але й змінити його (що не притаманно рядкам):

Приклад 21

```
>>> c = bytearray ("Python", "ascii")
>>> c[0]
80 # Можемо одержати значення
>>> c[0]=b"J"[0] # Можемо змінити значення
>>> c
bytearray(b'jython')
```

1. Значення, що присвоюється, повинно бути цілим числом у діапазоні від про до 255.
2. Щоб одержати число в попередньому прикладі, ми
   - створили об'єкт типу `bytearray`,
   - присвоїли значення, розташоване по індексу `0 (c[0]= b"J"[0])`.

Можна, звичайно, відразу вказати код символу, але ж тримати всі коди символів у пам'яті властиво комп'ютеру, а не людині.

Для зміни об'єкта можна також використовувати наступні методи:

(Числовий код додаємо в кінець)`append (<Число>)` – додає один елемент у кінець об'єкта. Метод змінює поточний об'єкт і нічого не повертає.

Приклад 22

```
>>> c = bytearray("string", "ascii")
>>> c.append(b"1"[0]); c
bytearray(b'string1')
```

(Послідавність додаємо в кінець)`extend(<Послідовність>)` – додає елементи послідовності в кінець об'єкта. Метод змінює поточний об'єкт і нічого не повертає.

Приклад 23

```
>>> c = bytearray ( "string", "ascii")
>>> c.extend(b"123"); c
bytearray(b'string123')
```

(Послідавність додаємо в кінець)+ і += використовувати оператори для додавання декількох елементів:

Приклад 24

```
>>> c = bytearray("string", "ascii")
>>> c + b"123" # Повертає новий об'єкт
bytearray (b' string123')
>>> c += b"456"; c
bytearray(b'ptring456')# Змінює поточний об'єкт
```

(Послідавність додаємо в кінець) Присвоювання значення зрізу:

Приклад 25

```
>>> c = bytearray( "string", "ascii")
>>> c[len(c):] = b"123"
# Додаємо елементи в послідовність
>>> c
bytearray(b'string123')
```

(Вставка числового коду в позицію)

`insert(<Індекс>, <Число>)` – додає один елемент у зазначену позицію. Інші елементи зміщаються. Метод змінює поточний об'єкт і нічого не повертає. Додамо елемент у початок об'єкта:

Приклад 26

```
>>> c = bytearray("string", "ascii")
>>> c.insert(0, b"1"[0]); c
bytearray(b'1string')
```

Метод `insert()` дозволяє додати тільки один елемент. Щоб додати кілька елементів, можна скористатися операцією присвоювання значення зрізу.

Додамо кілька елементів у початок об'єкта по зрізу:

```
>>> c = bytearray("string", "ascii")
>>> c[:0] = b"123"; c
bytearray(b'123string')
```

(Вставка послідовності по зрізу)

(Видалення елемента по індексу)

`pop([<Індекс>])` – видаляє елемент, розташований по зазначеному індексу, і повертає його.

Якщо індекс не зазначений, то видаляє й повертає останній елемент.

Приклад 27

```
>>> c = bytearray ( "string", "ascii")
>>> c.pop() # Видаляємо останній елемент
103

>>> c
bytearray(b'strin')

>>> c.pop(0)   # Видаляємо перший елемент
115

>>> c
bytearray(b'trin')
```

Оператор `del`

(Видалення елемента, вибраного по індексу або зрізу)

Вилучити елемент послідовності дозволяє також оператор `del`:

Приклад 28

```
>>> c = bytearray("string", "ascii")

>>> del c[5] # Видаляємо останній елемент

>>> c
bytearray(b'strin')

>>> del c[:2] # Видаляємо перший і другий елементи

>>> c
bytearray(b'rin')
```

(Видалення елемента по числовому коду)

`remove(<Число>)` – видаляє перший елемент, що містить зазначене значення.

1. Якщо елемент не знайдений, виконується виключення `Valueerror`.
2. Метод змінює поточний об'єкт і нічого не повертає.

Приклад 29

```
>>> c = bytearray("strstr", "ascii")

>>> c.remove(b"s"[0])  # Видаляє тільки перший елемент

>>> c
bytearray(b'trstr')
```

(Зміна порядку елементів)

`reverse ()` – змінює порядок проходження елементів на протилежний.

1. Метод змінює поточний об'єкт і нічого не повертає.

Приклад 30

```
>>> c = bytearray("string", "ascii")
>>> c.reverse(); c
bytearray(b'gnirts')
>>> c = bytearray("123456789", "cp1251")
>>> c.reverse();c
bytearray(b'987654321')
>>> c = bytearray("абвгдеж", "utf-8")
>>> c
bytearray(b'\xd0\xb0\xd0\xb1\xd0\xb2\xd0\xb3\xd0\xb4\xd0\xb5\xd0\xb6')

>>> c.reverse(); c
bytearray(b'\xb6\xd0\xb5\xd0\xb4\xd0\xb3\xd0\xb2\xd0\xb1\xd0\xb0\xd0')
```

Перетворення bytearray→ str

Перетворити об'єкт типу `bytearray` у рядок дозволяє метод `decode()` .

Метод має наступний формат:

```
decode ([encoding="utf-8"] [, errors="strict"])
```

```
Наприклад: c.decode(encoding="cp1251")
```

1. Параметр `encoding` задає кодування символів (за замовчуванням UTF-8) в об'єкті `bytearray`,
2. Параметр `errors` – спосіб обробки помилок шляхом задавання значень:

`"strict"` (значення за замовчуванням), вивід виключення

`"replace"`- знаки питання в місцях помилок

`"ignore"` – ігнорування помилок

Приклад 31. Приклад перетворення bytearray в str:

```
>>> c = bytearray("рядок", "cp1251")
>>> c.decode(encoding="cp1251"), c.decode("cp1251")
('рядок','рядок')

>>> c = bytearray("123\n456", "cp1251")
>>> b.decode(encoding="cp1251", errors="replace")
'123\n456'
```

Для перетворення можна також скористатися функцією str():

```
>>> c = bytearray("рядок", "cp1251")

>>> str (c, "cp1251")

'рядок''рядок'
```

Підсумок по модифікації bytearray

Додавання елементів

```
c.append(b"1"[0])  append (<Число>)
c.extend(b"123")   extend(<Послідовність>)
c += b"456"        + і +=
c.insert(0, b"1"[0])insert(<Індекс>, <Число>)
```

Видалення елементів

```
c.pop(0)   pop([<Індекс>])
del c[:2], del c[5] по елементу або зрізу
c.remove(b"s"[0])  remove(<Число>)
```

Інші методи

```
c.reverse()          reverse ()
c.decode("cp1251") b.decode(encoding="cp1251", errors="replace")
```

Перетворення об'єкта в послідовність байтів
Модуль pickle

Модуль pickle реалізує потужний алгоритм серіалізації і десеріалізації об'єктів Python.

"Pickling"(серіалізація) - процес перетворення об'єкта Python в потік байтів.

"Unpickling" (десеріалізація)- зворотна операція, в результаті якої потік байтів перетворюється назад в Python-об'єкт.

Так як потік байтів легко можна записати в файл, модуль pickle широко застосовується для збереження і завантаження складних об'єктів в Python.

3. Перш ніж використовувати функції із цього модуля, необхідно підключити модуль за допомогою інструкції:

```
import pickle
```

Для перетворення призначено дві функції:

`dumps` і `loads`.

Функція dumps

1. Виконує *серіалізацію* об'єкта й повертає послідовність байтів спеціального формату.

```
dumps(<Об'єкт>[,<Протокол>])
```

2. Формат залежить від зазначеного протоколу: число від 0 до 4 (підтримка протоколу 4 з'явилася в Python 3.4).

3. Якщо другий параметр не зазначений, буде використаний протокол 4 для Python 3.4 або 3 – для попередніх версій Python 3.

Приклад 32. Приклад перетворення списку й кортежу:

```
>>> import pickle
>>> obj1 = [1, 2, 3, 4, 5]  # Список
>>> obj2 = (6, 7, 8, 9, 10) # Кортеж
>>> pickle.dumps(obj1)
b'\x80\x03]q\x00(K\x01K\x02K\x03K\x04K\x05e.'
>>> pickle. dumps(obj2)
b'\x80\x03(K\x06K\x07K\x08K\tk\ntq\x00.'
```

Функція loads

```
loads(<Послідов.байтів>[,encoding="ASCII"] [,
errors=" strict" ])
```

1. Перетворює послідовність байтів спеціального формату назад в об'єкт, виконуючи його *десеріалізацію*.

Приклад відновлення списку й кортежу:

Приклад 33.

```
>>>pickle.loads(b'\x80\x03]q\x00(K\x01K\x02K\x03K\x04K\x05e.')
[1, 2, 3, 4, 5]
>>> pickle.loads(b'\x80\x03(K\x06K\x07K\x08K\tK\ntq\x00.')
(6, 7, 8, 9, 10)
```

Шифрування рядків

Для шифрування рядків призначений модуль `hashlib`.

1. Перш ніж використовувати функції із цього модуля, необхідно підключити модуль за допомогою інструкції:

```
import hashlib
```

2. Модуль надає наступні функції:
`md5()`, `sha1()`, `sha224()`, `sha256()`, `sha384()` і `sha512()`.

3. Як необов'язковий параметр функціям можна передати послідовність байтів, яка має бути зашифрована.

Приклад 34.

```
>>> import hashlib
>>> h = hashlib.sha1(b"password")
```

`update()`. Передати послідовність байтів можна також за допомогою методу `update()` . У цьому випадку об'єкт приєднується до попереднього значення:

```
>>> h = hashlib.sha1()
>>> h.update(b"password")
```

Методи: `digest()` і `hexdigest( )`

Одержати зашифровану послідовність байтів і рядок дозволяють два методи: `digest()` і `hexdigest()`.

1. Перший метод повертає значення типу `bytes`,
2. Другий метод повертає рядок, що містить шістнадцяткові цифри.

Приклад 35.

```
>>> h = hashlib.sha1(b"password")
>>> h.digest()
b'[\xaaa\xe4\xc9\xb9??\x06\x82%\x0bl\xf83\x1b~\x
e6\x8f\xd8'

>>> h.hexdigest()
'5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8'
```

Функція `md5()`

Найбільш часто застосовуваною є функція `md5()`, яка шифрує рядок за допомогою алгоритму MD5.

Ця функція використовується для шифрування паролів.

Для порівняння введеного користувачем пароля зі збереженим у базі необхідно:

зашифрувати введений пароль,

потім виконати порівняння.

Приклад 36. Перевірка правильності введення пароля

```
>>> import hashlib

>>> h = hashlib.md5(b"password")

>>> p = h.hexdigest()

>>> p # Пароль, збережений у базі
'5f4dcc3b5aa765d61d8327deb882cf99'

>>>h2 = hashlib.md5(b"password")
# Пароль, введений користувачем

>>> if p == h2.hexdigest():print("Пароль
правильний")
```

Властивість `digest_size` зберігає розмір значення, згенерованого описаними раніше функціями шифрування, у байтах:

```
>>> h = hashlib.sha512(b"password")
>>> h.digest_size
64
```

Python 3.4 підтримує новий спосіб стійкого до зламу шифрування паролів за допомогою функції `pbkdf2_hmac()`:

```
pbkdf2_hmac(<Основний алгоритм шифрування>,
<Пароль, що зашифровують>, <"Сіль">, <Кількість
проходів шифрування>, dklen=None)
```

Як `<основний алгоритм шифрування>` слід указати рядок з найменуванням цього алгоритму: `"md5"`, `"sha1"`, `"sha224"`, `"sha256"`, `"sha384"` і `"sha512"`.

<Пароль, що зашифровують> вказується у вигляді значення типу bytes.
"Сіль" – це особлива величина типу bytes, що виступає як ключ шифрування, – її довжина не повинна бути меншою за 16 символів. Кількість проходів шифрування слід указати достатньо великою (так, при використанні алгоритму SHA512 вона повинна становити 100000).

ПРИМІТКА

Кодування даних із застосуванням функції pbkdf2_hmac() вимагає багато системних ресурсів і може забрати значний час, особливо на малопотужних комп'ютерах.
Параметр dklen функції pbkdf2_hmac() вказує довжину результуючого закодованого значення в байтах – якщо вона не задана або дорівнює None, буде створено значення стандартної для вибраного алгоритма довжини (64 байта для алгоритма SHA512).

Закодований пароль повертається у вигляді величини типу `bytes`.

Приклад 37.

```
>>> import hashlib
>>> dk = hashlib.pbkdf2_hmac('sha512',
b'1234567', b'saltsaltsaltsalt', 100000)
>>> dk
b"Sb\x85tc-\xcb@\xc5\x97\x19\x90\x94@\x9f\xde\x07\xa4p-
\x83\x94\xf4\x94\x99\x07\xec\xfa\xf3\xcd\xc3\x88jv\xd1\xe5\x9a\x11
9\x15/\xa4\xc2\xd3N\xaba\x02\xc0s\xc1\xd1\x0b\x86xj(\x8c>Mr'@\xb
b"
```