Лекція 7

Створення рядків і операції над рядками

Алгоритм вибору варіанту контрольної роботи №1

1. Взяти свій номер у списку групи.
2. Розділити свій номер на 5.
3. Одержаний залишок від ділення буде номером вашого варіанту.
4. Навести обчислення в контрольній роботі.

Наприклад.

Номер у списку: 5. Тоді 5:5 = 1 залишок: 0

Номер у списку: 1. Тоді 1:5 = 0 залишок: 1

Номер у списку: 12. Тоді 12:5 = 2 залишок: 2

Номер у списку: 28. Тоді 28:5 = 5 залишок: 3

Номер у списку: 19. Тоді 19:5 =3 залишок: 4

Увага!! Всі приклади виконати дома. Одержані програми налаштувати на інтерпретаторі та компіляторі

Контрольна робота №1

Питання 1. Який буде вивід інтерпретатора?

```
0.>>> False - True

1.>>> True * (- True)

2.>>> a = 2 * (- True); a

3.>>> a = 2 * (- True); a += 6; a

4.>>> a=6; a-= True; a
```

Питання 2.

0.Задайте список, який складається з п’яти елементів цілочисельного типу. Створіть кортеж з 1-го,3-го та 5-го елементів даного списку.

1.Задайте кортеж, який складається з п’яти елементів дійсного типу. Створіть список з двох довільних елементів даного кортежу.

2.Задайте рядок з своїм ім’ям і виведіть на друк його довжину та останню літеру.

3.Задайте рядок з своїм ім’ям. Створіть новий рядок, що містить символи попереднього рядка, за виключенням першого і останнього.

4. Задайте рядок з своїм ім’ям і виведіть на друк останній елемент рядка, вважаючи, що Вам невідомий вміст рядка

Питання 3.

З використанням **діапазонів** створіть фрагмент програми для:

0. Обчислення суми подвоєних парних чисел від 0 до 20.

1. Обчислення суми других степенів парних чисел від 0 до -20

2. З рядка з Вашим ім'ям створіть рядок з 1-го, 3-го, 5-го … символів Вашого імені

3. З рядка з Вашим ім'ям створіть рядок з 2-го, 4-го і 6-го … символів Вашого імені

4. Обчисліть значення 5!

Перевірка на ввід числа

```
s=input('Write the number')
a=0
for i in s:
    for j in (range(10)):
        if i==str(j):a+=1
if a == len(s): print("Number")
else: print("Not number")
```

Загальна характеристика рядків в Python

Рядки – упорядковані послідовності символів.

1. Довжина рядка обмежена лише обсягом оперативної пам'яті комп'ютера.
2. Рядки підтримують:
   - доступ до елемента по індексу,
   - одержання зрізу,
   - конкатенацію (оператор +),
   - повторення (оператор *),
   - перевірку на входження (оператори `in` та `not in`).

Крім того, рядки є незмінюваними типами даних. Тому практично всі строкові методи як значення повертають новий рядок. При використанні невеликих рядків це не приводить до проблем, але при роботі з більшими рядками можна зіткнутися з проблемою нестачі пам'яті.

Приклад 1. Спроба змінити символ по індексу

Можна одержати символ по індексу, але змінити не можна.

```
>> s="Python"
>>> s[0]
'P'
>>> s[0]="J"
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Typeerror: 'str' object does not support
item assignment
```

У деяких мовах програмування кінцем рядка є нульовий символ. У мові Python нульовий символ може бути розташований всередині рядка:

```
>>> "string\x00string" # Нульовий символ -
це НЕ кінець рядка
'string\x00string'
```

Строкові типи, підтримувані в Python 3

`str` - Unicode-рядок. Конкретне кодування: UTF-8, UTF-16 або UTF-32 не вказується. Розглядайте такі рядки, як рядки в якомусь абстрактному кодуванні, що дозволяє зберігати символи Unicode і виконувати маніпуляції з ними. При виводі Unicode-рядок необхідно перетворити в послідовність байтів у якому-небудь кодуванні:

Приклад 2

```
>>> type ("рядок")
<class 'str'>
>>> "рядок".encode(encoding="cp1251")
b'\xf0\xff\xe4\xee\xea'

>>> "рядок".encode(encoding="utf-8")
b'\xd1\x80\xd1\x8f\xd0\xb4\xd0\xbe\xd0\xba'
```

`bytes` – незмінювана послідовність байтів.

Кожний елемент послідовності може зберігати ціле число від 0 до 255, яке позначає код символу.

Об'єкт типу `bytes` підтримує більшість строкових методів і, якщо це можливо, виводиться як послідовність символів. Однак доступ по індексу повертає ціле число, а не символ.

Приклад 3

```
>>> s = bytes ("стр str", "cp1251")
>>> s[0], s[5], s[0:3], s[4:7]
(241, 116, b'\xf1\xf2\xf0', b'str')
>>> s[0], hex(s[0])
(241, '0xf1')
>>> s[5],hex(s[5])
(116, '0x74')
>>> s
b'\xf1\xf2\xf0 str'
```

Об'єкт типу `bytes` може містити як однобайтні, так і багатобайтні символи.
Зверніть увагу на те, що функції й методи рядків некоректно працюють із багатобайтними кодуваннями, – наприклад, функція `len ()` поверне кількість байтів, а не символів:

Приклад 4

```
>>> len("рядок")
5

>>> len (bytes("рядок", "cp1251"))
5

>>> len (bytes ("рядок", "utf-8"))
10
```

`bytearray - змінювана послідовність байтів.`

Тип `bytearray` аналогічний типу `bytes`, але дозволяє змінювати елементи по індексу й містить додаткові методи, що дають можливість додавати й видаляти елементи.

Приклад 5

```
>>> s = bytearray("str", "cp1251")
>>> s[0] = 49; s # Можна змінити символ
bytearray(b'1tr')
>>> s.append(55); s
bytearray(b'1tr7')# Можна додати символ
```

1. У всіх випадках, коли йдеться про текстові дані, слід використовувати тип `str`. Саме цей тип ми будемо називати словом «рядок».
2. Типи `bytes` і `bytearray` слід задіяти для запису бінарних даних – наприклад, зображень, а також для проміжного зберігання текстових даних.

Створення рядка

Способи створення рядка:

- за допомогою функції

```
str ([<Об'єкт>[,<Кодування>[,<Обробка помилок>]]])
```

1. Якщо зазначений тільки перший параметр, то функція повертає строкове представлення будь-якого об'єкта.
2. Якщо параметри не зазначені взагалі, то повертається порожній рядок.

Приклад 6.

```
>>> str(), str([1, 2]), str((3, 4)), str({"x": 1})
('', '[1, 2]', '(3, 4)', "{'x': 1}")

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

Перетворення об'єктів типу `bytes`

Щоб одержати з об'єктів типу `bytes` і `bytearray` саме рядок, слід указати кодування в другому параметрі:

Приклад 7

```
>>> str(b"\xf0\xff\xe4\xee\xea", "cp1251")
'рядок'
```

У третьому параметрі можуть бути зазначені значення `"strict"` (при помилці виконується виключення `Unicodedecodeerror` – значення за замовчуванням), `"replace"` (невідомий символ замінюється символом, що має код `\ufffd`) або `"ignore"` (невідомі символи ігноруються):

Приклад 8

```
 >>> obj1 = bytes("рядок1", "utf-8")
>>> obj2 = bytearray("рядок2", "utf-8")
>>> str(obj1, "utf-8"), str(obj2, "utf-8")
('рядок1', 'рядок2')

>>> str(obj1, "ascii", "strict")
Traceback (most recent call last):
  File "<input>", line 1, in <module>
UnicodeDecodeError: 'ascii' codec can't
decode byte 0xd1 in position 0: ordinal not
in range(128)
>>> str (obj1, "ascii", "ignore")
'1'
```

Апострофи й подвійні лапки:

Приклад 9

```
>>> 'рядок', "рядок", '"x":5', "'x':5"
('рядок', 'рядок', '"x":5', "'x':5")

>>> print ('рядок1\nрядок2')
рядок1
рядок2
```

У деяких мовах програмування (наприклад, у PHP) рядок в апострофах відрізняється від рядка в лапках тим, що всередині апострофів спеціальні символи виводяться як є, а всередині лапок вони інтерпретуються.

У мові Python жодної відмінності між рядком в апострофах і рядком у лапках немає.

Правила роботи з лапками

1. Якщо рядок містить лапки, то його краще «взяти» в апострофи, і навпаки.

2. Усі спеціальні символи в таких рядках інтерпретуються.

3. Послідовність символів \n перетвориться в символ нового рядка.

4. Щоб спеціальний символ виводився як є, його необхідно екранувати за допомогою слеша.

Приклад 10. Варіанти екранування слешем.

```
>>> print("Рядок1\\nрядок2")
Рядок1\nрядок2
>>> print('Рядок1\nрядок2')
Рядок1
рядок2
```

Лапки всередині рядка в лапках і апостроф всередині рядка в апострофах також необхідно екранувати за допомогою захисного слеша:

Приклад 11.

```
>>>"\"x\": 5", '\'x\': 5'
('"x": 5', "'x': 5")# Але апостроф в лапках
>>> "'x': 5", '"x": 5'#і лапки в апострофах
("'x': 5", '"x": 5')  # не екранують
```

Особливості переходу на новий рядок

1. «Взяти» об'єкт в одинарні лапки (або апострофи) на декількох рядках не можна. Перехід на новий рядок викличе синтаксичну помилку:

Приклад 12.

```
>>> "string
Syntaxerror: EOL while scanning string
literal
```

2. Щоб розташувати об'єкт на декількох рядках, слід

А) перед символом переводу рядка вказати символ \,

Б) помістити два рядки всередині дужок,

В) використовувати конкатенацію всередині дужок.

Приклад 13. Способи задавання багаторядкового тексту

```
>>> "string1\
string2" #після \ не повинно бути жодних символів
'string1string2'

>>> ("string1"
"string2")          # Неявна конкатенація рядків
'string1string2'

>>> ("string1" +
"string2")           # Явна конкатенація рядків
'string1string2'
```

Екранування слеша наприкінці рядка

Якщо наприкінці рядка розташований символ \, то його необхідно екранувати, інакше буде виведене повідомлення про помилку:

Приклад 14

```
>>> print("http:string\")
Syntaxerror: EOL while scanning string
literal

>>> print("string\\")
string\
```

Рядки з потроєними апострофами й лапками

1. Рядки між потроєними апострофами й потроєними лапками можна розмістити на декількох рядках.

2. У таких рядках допускається одночасно використовувати й лапки, і апострофи без необхідності їх екранувати.

3. В решт випадків такі об'єкти еквівалентні рядкам в апострофах і лапках. Усі спеціальні символи в таких рядках інтерпретуються.

Приклад 15

```
>>> print('''Рядок1
Рядок2''')
Рядок1
Рядок2

>>> print ("""Іван
Марія""")
Іван
Марія
```

Рядок документування

Якщо рядок не присвоюється змінній, то він вважається рядком документування. Такий рядок зберігається в атрибуті __doc__ того об'єкта, у якому розташований. Як приклад створимо функцію з рядком документування, а потім виведемо вміст рядка:

```
>>> def test():
"""Це опис функції"""
pass
>>> print(test.__doc__
Це опис функції
```

Оскільки вирази всередині таких рядків не виконуються, то потроєні лапки (або потроєні апострофи) дуже часто використовуються для коментування більших фрагментів коду на етапі налаштування програми.

Рядок з модифікатором r

Якщо перед рядком розмістити модифікатор `r`, то спеціальні символи усередині рядка виводяться як є. Наприклад, символ `\n` не буде перетворений у символ переводу рядка. Іншими словами, він буде вважатися послідовністю двох символів: `\` і `n`:

Приклад 16

```
>>> print("Рядок1\nрядок2")
Рядок1
Рядок2
>>> print(r"Рядок1\nрядок2")
Рядок1\nрядок2
>>> print(r"""Рядок1\nрядок2""")
Рядок1\nрядок2
```

Застосування неформатованих рядків з модифікатором r

Такі неформатовані рядки зручно використовувати в шаблонах регулярних виразів, а також при вказівці шляху до файлу або каталогу:

Приклад 17

```
>>>print(r"C:\Python35-2\lib\site-packages")
C:\Python35-2\lib\site-packages
```

Якщо модифікатор не вказати, то всі слеші при вказівці шляху необхідно екранувати:

Приклад 18

```
>>>print("C:\\Python35-2\\lib\\site-packages")
C:\Python35-2\lib\site-packages
```

Спеціальні символи

Спеціальні символи – це комбінації знаків, що позначають службові символи або символи, що не друкуються, які неможливо вставити у звичайний спосіб.

Перелік спеціальних символів, припустимих усередині рядка, перед яким немає модифікатора `r`:

`\n` - перевід рядка;
`\r` - повернення каретки;
`\t` - знак табуляції;
`\v` - вертикальна табуляція;
`\a` - дзвінок;
`\b` – вибій;

\f – перевід (змін) формату;
\0 – нульовий символ (не є кінцем рядка);
\" - лапки;
\' - апостроф;
\N - вісімкове значення N. Наприклад, \74 відповідає символу <;
\xn - шістнадцяткове значення N. Наприклад,
\x6a відповідає символу j;
\\ - зворотний слеш;
\uxxxx- 16-бітний символ Unicode. Наприклад,
\u04 3a відповідає російській букві к;
\uxxxxxxxx - 32-бітний символ Unicode

Слеш наприкінці рядка при використанні модифікатора r

Якщо наприкінці неформатованого рядка розташований слеш, то його необхідно екранувати.

Приклад 19:

```
>>> print(r"C:\Python35-2\lib\site-packages\")
  File "<input>", line 1
    print(r"C:\Python35-2\lib\site-packages\")
                                            ^
Syntaxerror: EOL while scanning string literal

>>> print("C:\Python35-2\lib\site-packages\\")
C:\Python35-2\lib\site-packages\
```

Інші способи забрати слеш наприкінці рядка

Щоб позбутися зайвого слеша, можна використовувати операцію конкатенації рядків, звичайні рядки або вилучити слеш явно. **Приклад 20.**

```
# Конкатенація(виділяємо в окремий рядок)
>>> print("C:\Python35-2\lib\site-packages"
+ "\\")
C:\Python35-2\lib\site-packages\
# Звичайний рядок
>>> print("C:\\Python33-2\\lib\\site-
packages\\")
C:\Python34\lib\site-packages\
# Видалення слеша
>>> print("C:\Python34\lib\site-
packages\\"[:-1])
C:\Python34\lib\site-packages\
```

Використання слеша без спецсимволів

Якщо після слеша немає символа, який разом зі слешем інтерпретується як спецсимвол, то слеш зберігається в складі рядка:

Приклад 21:

```
>>> print("Цей символ \не спеціальний:")
Цей символ \не спеціальний
```

Проте, краще екранувати слеш явно:

Приклад 22:

```
>>> print("Цей символ \\не спеціальний:")
Цей символ \не спеціальний
```

Операції з рядками

1. Рядки є послідовностями. Тому вони підтримують

   - доступ до елемента по індексу,
   - одержання зрізу,
   - конкатенацію,
   - повторення,
   - перевірку на входження.

Розглянемо ці операції докладно.

Доступ до елемента по індексу

До будь-якого символу рядка можна звернутися як до елемента списку – достатньо указати його індекс у квадратних дужках. Нумерація починається з нуля:

Приклад 23. Доступ по індексу

```
>>> s = "Python"
>>> s[0], s[1], s[2], s[3], s[4], s[5]
('P', 'y', 't', 'h', 'o', 'n')
```

Якщо символ, відповідний до зазначеного індексу, відсутній у рядку, то виконується виключення `Indexerror`:

Приклад 24. Індекс за межами рядка

```
>>> s = "Python"
>>> s[10]
Traceback (most recent call last):
  File "<input>", line 1, in <module>
Indexerror: string index out of range
```

Доступ по від’ємному індексу

Як індекс можна вказати від’ємне значення.

У цьому випадку зсув буде вестись від кінця рядка, а точніше – щоб одержати додатний індекс, віднімається від довжини рядка:

Приклад 25

```
>>> s = "Python"
>>> s[-1], s[len(s)-1]
('n', 'n')
```

Символ у рядку по індексу змінити не можна

Оскільки рядки відносяться до незмінних типів даних, змінити символ по індексу не можна:

Приклад 26

```
>>> s = "Python"
>>> s [0] = "J" # Змінити рядок не можна
Traceback (most recent call last):
File "<Input>", line 1, in <module>

>>> News ="J"+s[1]+s[2]+s[3]+s[4]+s[5]
>>> News
'Jython'
```

Операція добування зрізу по рядку

Щоб виконати зміну, можна скористатися операцією добування зрізу, яка повертає зазначений фрагмент рядка.
Формат операції: `[<Початок>:<Кінець>:<Крок>]`
Усі параметри тут не є обов'язковими.

1. Якщо параметр `<Початок>` не зазначений, то використовується значення 0.

2. Якщо параметр `<Кінець>` не зазначений, то повертається фрагмент до кінця рядка. Слід також помітити, що символ з індексом, зазначеним у цьому параметрі, не входить у фрагмент, що повертається.

3. Якщо параметр `<Крок>` не зазначений, то використовується значення 1. Як значення параметрів можна вказати від'ємні значення.

Використання параметрів при добуванні зрізу

1. Одержуємо копію рядка: **Приклад 27.**

```
>>> s = "Python"
>>> s[:]#повертається фрагмент від позиції 0
до кінця рядка
'Python'
```

2. Виводимо символи у зворотному порядку:

Приклад 28

```
>>> s(::-1]# Від'ємне значення в параметрі
<Крок>
'nohtyp'
```

3. Замінимо перший символ у рядку: **Приклад 29**

```
>>> "J"+s[1:] #  фрагмент від символу 1 до
кінця рядка
'Jython'
```

4. Вилучимо останній символ: **Приклад 30**

```
>>> s[:-1] # повертається фрагмент від 0 до
len(s)-1
'Pytho'
```

5. Одержимо перший символ у рядку: **Приклад 31**

```
>>> s[0:1] # Символ з індексом 1 не входить
у діапазон
'P'
```

6. Одержимо останній символ: **Приклад 32**

```
>>> s[-1:] #отримуємо фрагмент від len(s)-1
до кінця рядка
'n'
```

7. Виведемо символи з індексами 2, 3 і 4: **Приклад 33**

```
>>> s[2:5] # повертаються символи з
індексами 2, 3 и 4
'tho'
```

Застосування функції len()

Визначити кількість символів у рядку дозволяє функція `len()`:

Приклад 34

```
>>> len("Python"),len("\r\n\t"),len(r"\r\n\t")
(6, 3, 6)
```

Тепер, коли ми знаємо кількість символів, можна перебрати всі символи за допомогою циклу `for`:

Приклад 35

```
>>> s = "Python"
>>> for i in range(len(s)): print(s[i],
end=" ")
```

Результат виконання:

```
P y t h o n
```

Безпосередній перебір елементів рядка

Оскільки рядки підтримують ітерації, ми можемо просто вказати рядок як параметр циклу:

Приклад 36

```
>>> s = "Python"
>>> for i in s: print(i, end=" ")
```

Результат виконання буде таким же:

```
P y t h o n
```

З'єднання рядків

1. З'єднати два рядки в один рядок дозволяє оператор **+**:

Приклад 37

```
>>> print("Рядок1" + "Рядок2")
Рядок1Рядок2
```

2. Крім того, можна виконати неявну конкатенацію рядків. У цьому випадку два рядки вказуються поруч без оператора між ними:

Приклад 38

```
>>> print("Рядок1" "Рядок2")
Рядок1Рядок2
>>> print("Рядок1"
...       "Рядок2")
...
Рядок1Рядок2
```

Одержання кортежу рядків

Якщо між рядками вказати кому, то ми одержимо кортеж, а не рядок:

Приклад 39

```
>>> s = "Рядок1", "Рядок2"
>>> type(s)  # Одержуємо кортеж, а не рядок
<class 'tuple'>
```

З'єднання змінної й рядка

Якщо з'єднуються змінна й рядок, то слід обов'язково вказувати символ конкатенації рядків, інакше буде виведене повідомлення про помилку:

Приклад 40

```
>>> s = "Рядок1"
>>> print(s + "Рядок2") # Нормально
Рядок1Рядок2
>>> print(s "Рядок2")   # Помилка
Syntaxerror: invalid syntax
```

З'єднання рядка з іншим типом даних

При необхідності з'єднати рядок з іншим типом даних (наприклад, з числом)

слід зробити явне перетворення типів за допомогою функції `str ()`:

Приклад 41

```
>>> "string" + str(10)
'string10'
```

Інші операції над рядками

Крім розглянутих операцій, рядка підтримують операцію повторення, перевірку на входження й невходження. Повторити рядок зазначена кількість раз можна за допомогою оператора *, виконати перевірку на входження фрагмента в рядок дозволяє оператор `in`, а перевірити на невходження – оператор `not in`:

Приклад 42

```
>>> "-" * 20
'--------------------'
>>> "yt" in "Python"
True
>>> "yt" in "Perl"
False
>>> "PHP" not in "Python"
True
```