**Модуль 5: Просунута робота з рядками**

Ми вже поверхнево знайомилися з рядками у Python. Ви вмієте створювати рядки, об'єднувати рядки, міняти регістр рядків, перевіряти, що рядок закінчується або починається деякою послідовністю. Але можливості Python по роботі з рядками не вичерпуються цим. На цьому зайнятті ми дізнаємося більше про те, як працювати з рядками у Python, про роботу з регулярними виразами та перетвореннями рядків.

**Ще варіанти створення рядків**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson05/lesson-05/#%D1%89%D0%B5-%D0%B2%D0%B0%D1%80%D1%96%D0%B0%D0%BD%D1%82%D0%B8-%D1%81%D1%82%D0%B2%D0%BE%D1%80%D0%B5%D0%BD%D0%BD%D1%8F-%D1%80%D1%8F%D0%B4%D0%BA%D1%96%D0%B2)

Для створення рядків можна скористатися одинарними або подвійними лапками:

**this\_is\_string = "Hi there!"  
  
the\_same\_string = 'Hi there!'  
  
this\_is\_string == the\_same\_string *# True***

Але що робити, якщо нам потрібен текст із перенесенням рядків (коли в тексті більше одного рядка)? Для цього можна скористатися потрійним повторенням лапок:

**text = """This is first line  
And second line  
Last third line"""  
  
song = '''Jingle bells, jingle bells  
Jingle all the way  
Oh, what fun it is to ride  
In a one horse open sleigh'''**

В цьому прикладі змінна **text**містить три рядки, а **song** — чотири рядки.

Коли інтерпретатор виявляє лапки, повторені тричі, він сприймає усі символи до наступних трьох закриваючих лапок, як символи рядка.

Зворотна ситуація, у вас є довгий рядок, який не повинен містити перенесень, але в коді його незручно відобразити одним рядком.

**one\_line\_text = "Textual data in Python is handled with str objects, or strings. Strings are immutable sequences of Unicode code points. String literals are written in a variety of ways: single quotes, double quotes, triple quoted.**

Щоб структурувати код і не додавати зайвих перенесень, ви можете розбити одну рядкову змінну на декілька частин:

**one\_line\_text = "Textual data in Python is handled with str objects, or strings. "\  
 "Strings are immutable sequences of Unicode code points. "\  
 "String literals are written in a variety of ways: single quotes, double quotes, triple quoted."**

Зверніть увагу на символ \ в кінці першого та другого рядка коду, він вказує інтерпретатору ігнорувати закінчення рядка і продовжити відразу з наступного.

**one\_line\_text** в обох прикладах буде містити один і той самий текст без перенесень.

Python сприймає рядкові змінні, між якими немає нічого, крім пробілів та/або символів нового рядка, як один рядок:

**("spam " "eggs") == "spam eggs" *# True***

Вираз ліворуч та вираз праворуч — це два рівнозначні записи одного і того самого тексту 'spam eggs' і, з погляду Python, — вони нерозрізнені.

**Спеціальні символи**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson05/lesson-05/#%D1%81%D0%BF%D0%B5%D1%86%D1%96%D0%B0%D0%BB%D1%8C%D0%BD%D1%96-%D1%81%D0%B8%D0%BC%D0%B2%D0%BE%D0%BB%D0%B8)

Ми неодноразово згадували символ перенесення рядка. Це один із "спеціальних" або "керуючих" символів. Керуючі символи — це символи перенесення рядка, табуляції, повернення каретки та інші символи, які не можна або незручно ввести з клавіатури. Для того щоб можна було вводити символи, які незручно або не можна ввести з клавіатури , придумали додавати **екрануючий символ**'\' який означає, що наступний за ним знак потрібно сприймати як спеціальний символ, а не буквально.

Ось деякі керуючі символи:

| **Позначення у коді** | **Опис** |
| --- | --- |
| \n | Перенесення рядка |
| \f | Перенесення сторінки |
| \r | Повернення каретки |
| \t | Горизонтальна табуляція |
| \v | Вертикальна табуляція |

Наприклад, текст з явним розбиттям на рядки:

**jingle\_bells = "Jingle bells, jingle bells\nJingle all the way\nOh, what fun it is to ride\nIn a one horse open sleigh"**

Вибирайте ту форму запису, яка більше підходить під конкретну ситуацію, і керуйтеся зручністю для запису і читання коду, який містить рядкову змінну.

**Методи рядків**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson05/lesson-05/#%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B8-%D1%80%D1%8F%D0%B4%D0%BA%D1%96%D0%B2)

Ми вже познайомилися з деякими методами рядків, зараз розберемо методи, пов'язані з пошуком в рядках і створенням нових рядків.

**Пошук у рядку**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson05/lesson-05/#%D0%BF%D0%BE%D1%88%D1%83%D0%BA-%D1%83-%D1%80%D1%8F%D0%B4%D0%BA%D1%83)

Для пошуку деякого символа або підрядка у рядку можна скористатися методом find:

**s = "Hi there!"  
  
start = 0  
end = 7  
  
print(s.find("er", start, end)) *# 5*  
print(s.find("q")) *# -1***

Цей метод виводить індекс початку першого збігу в рядку **s**, починаючи з **start** до **end**. Якщо **start** та **end** не вказані, то з початку і до кінця рядку. Повертає -1, якщо послідовність не знайдена.

Ще один метод пошуку підрядка у рядку дуже схожий на **find** — це **index**. Основна відмінність полягає в тому, що якщо **index** не знайде підрядок, то викличе виняток **ValueError**.

Якщо вам потрібно здійснити пошук підрядка у рядку справа, а не зліва як у **find**, то для цього є метод **rfind**:

**s = 'Some words'  
s.rfind('o') *# 6***

І "правий" аналог **index — rindex:**

**s = 'Some words'  
s.rindex('o') *# 6***

**Розбиття рядка на декілька підрядків**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson05/lesson-05/#%D1%80%D0%BE%D0%B7%D0%B1%D0%B8%D1%82%D1%82%D1%8F-%D1%80%D1%8F%D0%B4%D0%BA%D0%B0-%D0%BD%D0%B0-%D0%B4%D0%B5%D0%BA%D1%96%D0%BB%D1%8C%D0%BA%D0%B0-%D0%BF%D1%96%D0%B4%D1%80%D1%8F%D0%B4%D0%BA%D1%96%D0%B2)

Часта ситуація, коли необхідно розбити рядок на підрядки за деяким символом, наприклад, розбити текст на речення за символом крапки та пробілу після крапки, або речення за словами.

**s = "I am learning strings in Python. Some new methods got now."  
sentences = s.split(". ")  
  
print(sentences[0]) *# I am learning strings in Python*  
print(sentences[1]) *# Some new methods got now.***

Для такого завдання можна скористатися методом **split**, який приймає як аргумент підрядок-маркер, який буде межею, по якій потрібно розбити рядок на частини. В результаті виклику повертається список рядків.

**Нові рядки на основі рядків**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson05/lesson-05/#%D0%BD%D0%BE%D0%B2%D1%96-%D1%80%D1%8F%D0%B4%D0%BA%D0%B8-%D0%BD%D0%B0-%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D1%96-%D1%80%D1%8F%D0%B4%D0%BA%D1%96%D0%B2)

Усі рядки незмінні, якщо ми хочемо модифікувати рядок, то є тільки один спосіб — створити новий рядок на основі вихідного.

Усі методи, які якось "модифікують" рядки, повертають нові рядки, ніяк не змінюючи оригінальний.

Для об'єднання декількох рядків в один через деякий роздільник використовується метод **join:**

**sentences = ["I am learning strings in Python", "Some new methods got now."]  
text = ". ".join(sentences)  
print(text) *# I am learning strings in Python. Some new methods got now.***По суті, це зворотна операція **split**

Якщо потрібно видалити зайві пробіли на початку і в кінці рядка, є спеціальний метод **strip:**

**clean = ' spacious '.strip()  
print(clean) *# spacious***

У цього метода є два "брати":

* "лівий", **lstrip**, видаляє тільки пробіли на початку рядка;
* та "правий", **rstrip**, видаляє тільки пробіли в кінці рядка.

Коли ж нам потрібно замінити деякий підрядок в рядку, ми можемо скористатися методом **replace:**

**dogs\_text = "All dogs bark like dogs."  
cats\_text = dogs\_text.replace("dogs", "cats")  
print(cats\_text) *# All cats bark like cats.***

Для видалення фіксованої послідовності на початку рядка є метод **removeprefix:**

**print('TestHook'.removeprefix('Test')) *# Hook*  
print('TestHook'.removeprefix('Hook')) *# TestHook***

Є парний метод для видалення послідовності в кінці рядка, **removesuffix:**

**print('TestHook'.removesuffix('Hook')) *# Test*  
print('TestHook'.removesuffix('Test')) *# TestHook***

**Translate**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson05/lesson-05/#translate)

В рядках є метод **translate**, цей метод дозволяє замінити символ в рядку на інший з мапи відповідності, яку можна задати.

**map = {ord('з'): 'z', ord('ю'): 'u')}  
translated = 'зю'.translate(map)  
print(translated) *# zu***

В цьому прикладі мапою відповідності виступає словник **map** і в цій мапі ми встановлюємо відповідність між символами '**з**' та '**z**', та '**ю**', та '**u**'.

Функція **ord** (від order) дозволяє дізнатися, яким числом кодується символ. Кодування символів ми розглянемо на іншому зайнятті, а поки що достатньо знати, що усі символи кодуються у пам'яті комп'ютера числами, і функція **ord** дозволяє дізнатися, яким саме числом кодується символ, щоб однозначно визначити, що це за символ.

***Неочевидний факт — багато складних символів, емодзі, наприклад, можна закодувати декількома способами, а відображатися вони будуть однаково.***

Наприклад, символ '**a**' кодується числом 97:

**ord('a') *# 97***

Найчастіше застосування методу **translate** пов'язане з транслітерацією рядків.

**Форматування рядків (метамова форматування)**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson05/lesson-05/#%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F-%D1%80%D1%8F%D0%B4%D0%BA%D1%96%D0%B2-%D0%BC%D0%B5%D1%82%D0%B0%D0%BC%D0%BE%D0%B2%D0%B0-%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)

Будь-яке число можна записати кількома варіантами запису:

* десятковий запис
* двійкове представлення
* шістнадцяткове представлення
* наукова нотація
* з фіксованою точністю (кількістю знаків після коми)

та інші.

Наприклад, вивести числа від 1 до 15 в десятковому, шістнадцятковому, вісімковому і двійковому представленні:

**for i in range(16):  
 s = "int: {0:d}; hex: {0:#x}; oct: {0:#o}; bin: {0:#b}".format(i)  
 print(s)**

Крім того, при створенні рядків буває корисним відформатувати рядок так, щоб знаки на різних рядках були один під одним (додати пробілів), додати заповнення в рядки для того, щоб результат був завжди однієї і тієї самої довжини.

Для таких і подібних завдань в Python вбудована [міні-мова форматування рядків](https://docs.python.org/3/library/string.html#format-specification-mini-language).

Або вивести квадрати та куби чисел до 12 у вигляді таблиці, відцентрувавши значення у стовпцях по 10 символів шириною:

**width = 5  
for num in range(12):  
 print('{:^10} {:^10} {:^10}'.format(num, num\*\*2, num\*\*3))**

Мета роботи з метамовою форматування полягає у зазначенні у фігурних дужках, яким чином варто перетворити значення перед підстановкою. Форматування працює і з f-рядками, але для більшої читабельності краще скористатися методом **format**, який приймає як аргументи значення для підстановки в рядок, замість виразів у фігурних дужках.

Самі вирази у фігурних дужках можуть складатися з опису, в якому вигляді варто вивести число (формат запису, кількість знаків після коми та ін.), і зазначення, чим доповнити рядок (додати пробілів на початку або в кінці, щоб рядок був N знаків завдовжки тощо).

**Модифікатори**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson05/lesson-05/#%D0%BC%D0%BE%D0%B4%D0%B8%D1%84%D1%96%D0%BA%D0%B0%D1%82%D0%BE%D1%80%D0%B8)

Міні-мова форматування рядків складається з таких модифікаторів, як:

* ім'я поля — необов'язковий елемент, можемо вказати, яку саме змінну сюди підставити за її ім'ям:

**s = "{name} {last\_name}".format(last\_name="Dilan", name="Bob")  
print(s) *# Bob Dilan***

* перетворення — необов'язковий аргумент, вказується після символу ! і може бути або **r**або **s**. Відповідає за те, чи потрібно спробувати перетворити елемент, або відобразити елемент "як є":

**s = "{name!r} {last\_name!s}".format(last\_name="Dilan", name="Bob")  
print(s) *# 'Bob' Dilan***

* специфікація вказується після : і відповідає за те, як відобазити значення.

**Специфікація**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson05/lesson-05/#%D1%81%D0%BF%D0%B5%D1%86%D0%B8%D1%84%D1%96%D0%BA%D0%B0%D1%86%D1%96%D1%8F)

Специфікація – набагато складніший модифікатор. З її допомогою можна:

* змінювати розрядність представлення цілих чисел (десяткові, вісімкові, шістнадцяткові та ін.);

**print('dec: {:d} hex: {:x} bin: {:b}'.format(15, 15, 15)) *# dec: 15 hex: f bin: 1111***

* змінювати точність представлення дробових чисел (округлювати до вказаної кількості знаків);

**print('pi: {:0.3}'.format(3.1415)) *# pi: 3.14***

* як відображати знак чисел:

**print('"{}" "{:+}" "{:-}" "{: }"'.format(1, 2, -3, 4)) *# "1" "+2" "-3" " 4"***

* як вирівняти положення елементу і чим (якими символами) доповнити;

**print("|{:<10}|{:\*^10}|{:>10}|".format('left', 'center', 'right')) *# |left |\*\*center\*\*| right|***

Знання про цей інструмент не є критично важливим, але використання форматування рядків в домашній роботі для конструювання "красивішого" результату виконання скрипта, буде плюсом.

**Регулярні вирази**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson05/lesson-05/#%D1%80%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D1%96-%D0%B2%D0%B8%D1%80%D0%B0%D0%B7%D0%B8)

У програмуванні регулярний вираз (від англ. Regular expression, скорочено regex або regexp) — це рядок, який описує деяку множину рядків відповідно до набору спеціальних синтаксичних правил [wiki](https://uk.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%B8%D0%B9_%D0%B2%D0%B8%D1%80%D0%B0%D0%B7).

Регулярні вирази — це окрема псевдомова програмування, яка широко використовується у багатьох мовах програмування і у Python також.

Детальне вивчення синтаксису регулярних виразів виходить за рамки цього курсу. Кому цікаво, можете продовжити знайомство з регулярними виразами за [посиланням](https://docs.python.org/3/library/re.html) або детальніший опис з прикладами [тут](https://docs.python.org/3/howto/regex.html#regular-expression-howto).

Основне завдання регулярних виразів — це пошук рядка, або підрядка, який відповідає опису в термінах регулярних виразів.

Використовуючи цей механізм можна:

* перевіряти, що рядок відповідає деяким вимогам (це номер телефону або email);
* розділяти рядки на підрядки за деяким виразом (розбити текст на речення, використовуючи усі розділові знаки, а не тільки якийсь один);
* замінювати підрядок в рядку (замінити усі слова, що починаються на деяку послідовність);
* знаходити підрядок в рядку, який відповідає виразу.

Регулярний вираз або коротко "регулярка" складається зі звичайних символів і спеціальних командних послідовностей. Наприклад, **\d** задає будь-яку цифру, а **\d+**— задає будь-яку послідовність з однієї або більше цифр.

[Ось](https://www.programiz.com/python-programming/regex) гарна стаття на тему регулярних виразів, там багато прикладів і корисних посилань.

Для роботи з регулярними виразами у Python є стандартний модуль **re.** Щоб скористатися цим модулем, його потрібно спочатку імпортувати:

**import re**

Загальним для усіх функцій модуля **re** є те, що першим аргументом йде регулярний вираз у вигляді рядка.

**Функція search**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson05/lesson-05/#%D1%84%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F-search)

Пошук на відповідність регулярному виразу виконує функція **search** модуля **re.** В результаті її виконання повертається спеціальний об'єкт Match або None, якщо нічого не знайшлося.

**s = "I am 25 years old"  
age = re.search('\d+', s)  
print(age) *# <re.Match object; span=(5, 7), match='25'>***

Щоб витягнути власне знайдене значення із age, можна скористатися його методом **group**

**s = "I am 25 years old."  
age = re.search('\d+', s)  
print(age.group()) *# 25***

Функція **search** — "ледача" і знаходить тільки першу відповідність заданій умові.

**Findall**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson05/lesson-05/#findall)

Коли потрібно знайти всі відповідні шаблону значення, можна скористатися функцією **findall.**

**s = "I bought 7 nuts for 6$ and 10 bolts for 3$."  
numbers = re.findall('\d+', s)  
print(numbers) *# ['7', '6', '10', '3']***

**findall** повертає список усіх відповідностей шаблону.

**Підстановка, sub**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson05/lesson-05/#%D0%BF%D1%96%D0%B4%D1%81%D1%82%D0%B0%D0%BD%D0%BE%D0%B2%D0%BA%D0%B0-sub)

Щоб замінити всі підрядки, що відповідають регулярному виразу, можна скористатися функцією **sub.** Замінимо кольори blue, red, white на слово colour:

**p = re.sub(r'(blue|white|red)', 'colour', 'blue socks and red shoes')  
print(p) *# colour socks and colour shoes***

**Складання регулярних виразів**[**​**](https://textbook.edu.goit.global/python/core-pz9qu8/v1/uk/docs/lesson05/lesson-05/#%D1%81%D0%BA%D0%BB%D0%B0%D0%B4%D0%B0%D0%BD%D0%BD%D1%8F-%D1%80%D0%B5%D0%B3%D1%83%D0%BB%D1%8F%D1%80%D0%BD%D0%B8%D1%85-%D0%B2%D0%B8%D1%80%D0%B0%D0%B7%D1%96%D0%B2)

Складання регулярних виразів — це окрема велика тема варта окремого вивчення. Регулярні вирази складаються з блоків та модифікаторів.

Прикладом блоку може бути:

* **[a,b,c,z]** у квадратні дужки беруть набір символів, з яких повинен складатися рядок
* **.** будь-який символ
* **\d** число або **[0-9]**
* **\D** не число, або **[^0-9]**
* **\s** будь-який символ, позначає пробіл або табуляцію, перенесення рядка та ін.
* **\w** будь-яке число або літера, включаючи нижнє підкреслення, або **[a-zA-Z0-9\_]**
* **\W** не літера, не число і не нижнє підкреслення

Модифікатори можуть вказувати на кількість повторень блоку у виразі, наприклад:

* **?**0 або 1 раз
* **+** 1 або більше разів
* **\*** 0 або більше разів
* **{n}** суворо **n** разів (**n** ціле число)
* **{n, m}** від **n** до **m**разів

Комбінуючи блоки та вирази, можна скласти вирази для ваших потреб:

| **Регулярка** | **Її призначення** |
| --- | --- |
| simple text | В точності текст «simple text» |
| **\d{5}** | Послідовність із 5 цифр **\d** означає будь-яку цифру **{5}** — рівно 5 разів |
| **\d\d/\d\d/\d{4}** | Дати у форматі ДД/ММ/РРРР (та інші шматки схожі на них, наприклад, 98/76/5432) |
| **\b\w{3}\b** | Слова в точності з трьох літер **\b** означає межу слова (з однієї сторони літера, а з другої — ні) **\w** — будь-яка літера, **{3}**— рівно три рази |
| **[-+]?\d+** | Ціле число, наприклад, 7, +17, -42, 0013 (можливі провідні нулі) **[-+]**? — або -, або **+**, або порожньо **\d+**— послідовність з 1 або більше цифр |
| `[-+]?(?:\d+(?:.\d\*)? | .\d+)(?:[eE][-+]?\d+)?` |