Конспект лекції (3)

```
Все, що, як вам здється, ви знаєте про текст — неправда.
(c) Dive into Python
```

Коли ви кажете "текст", ви напевне маєте на увазі "букви та інші символи на екрані мого комп’ютера". Але комп’ютери не працюють з символами, вони працюють з бітами і байтами. Будь-який текст який ви бачите насправді зберігається в певному кодуванні. Говорячи дуже грубо, кодування символів - це бінарне відношення між зображенням символів які ви бачите на екрані, і даними які комп’ютер насправді зберігає в пам’яті та на диску. Існує багато різноманітних кодувань символів, деякі з них оптимізовані для конкретних мов, наприклад англійської, китайської або української, а інші можуть використовуватись в багатьох мовах.

Любий текст потребує декодуваня, прямо як шифр. Через купу кодувань на різних комп’ютерах виникали проблеми з відображенням багатомовного тексту або тексту на мовах, відмінних від англійської.

На привелике щастя зараз людство узгодило єдине кодуваня, що підтримується майже усіми і усюди: UTF-8 - система кодування Юнікоду змінної довжини.

▼ Строка — це більше ніж строка

Тип `str` у Python реалізує текстові рядки Unicode з операторами, вбудованими функціями, методами та спеціальними модулями. Дещо схожий тип `bytes` представляє довільні двійкові дані як послідовність байтів, також відому як байтовий рядок або `bytearray`. Над об'єктами обох типів можна виконувати багато текстових операцій: оскільки ці типи є незмінними, методи здебільшого створюють і повертають новий рядок, за винятком випадків, коли повертають предметний рядок незмінним.

Ми спочатку розглянемо методи, доступні для цих трьох типів, потім обговорено модуль string і форматування рядків (зокрема, форматовані рядкові літерали).

Строки є незмінними, що означає, що після створення рядка ви не можете змінити його вміст без створення нового рядка (або перезапису вмісту

змнної).

Рядки (строки) розглядаються як послідовності символів.

Це означає, що ви можете обробляти рядки, якщо це колекція окремих елементів (символів). Важливі риси рядків як послідовностей включають:

Індексація:

Символи в рядках можна отримати за допомогою індексації. Перший символ має індекс 0, другий - 1, і так далі

```
рядок = "Привіт"
перший_символ = рядок[0]  # Повертає "П"
другий_символ = рядок[1]  # Повертає "р"
```

Зрізи (Slices):

Ви можете отримувати підстрічки (зрізи) з рядка, вказуючи початковий та кінцевий індекси.

```
рядок = "Привіт"
підстрічка = рядок[1:4]  # Повертає "рив"
```

Довжина Рядка:

Ви можете використовувати функцію `len()` для отримання довжини рядка (кількість символів).

```
рядок = "Привіт"
довжина = len(рядок)  # Повертає 6
```

Ітерація:

Рядки можна ітерувати за допомогою циклу `for`, отримуючи кожен символ послідовно.

```
рядок = "Привіт"
for символ in рядок:
    print(символ)
```

Конкатенація:

Строки можна об'єднувати (конкатенувати) за допомогою оператора `+`.

```
рядок1 = "Привіт"
рядок2 = "Світ"
обєднаний_рядок = рядок1 + " " + рядок2  # Повертає "Привіт Світ"
```

▼ Розділення строки на частини – split()

Функція `split()` використовується для розділення рядка на частини на основі певного роздільника та повертає список отриманих частин. Основні варіанти використання `split()`:

1. **Розділення за пробілом:**
   - Якщо не вказати конкретний роздільник, `split()` використовує пробіли (і інші пробільні символи) як роздільники.

```
рядок = "Привіт світ"
частини = рядок.split()
print(частини)
# Результат: ['Привіт', 'світ']
```

- **Розділення за певним роздільником:**
  - Можна вказати власний роздільник для розділення рядка.

```
рядок = "apple,orange,banana"
частини = рядок.split(',')
print(частини)
# Результат: ['apple', 'orange', 'banana']
```

- **Обмеження кількості розділень:**
  - Можна вказати, скільки разів роздільник повинен бути використаний для розділення.

```
рядок = "apple,orange,banana,grape"
частини = рядок.split(',', 2)
print(частини)
# Результат: ['apple', 'orange', 'banana,grape']
```

- **Розділення за регулярним виразом:**
  - Можна використовувати регулярні вирази як роздільники.

```
import re
рядок = "apple orange,banana"
частини = re.split(r'\s+', рядок)
print(частини)
# Результат: ['apple', 'orange,banana']
```

▼ Перевірка початку .startswith та закінчення .endswith

Методи .startswith() та .endswith() використовуються для перевірки того, чи рядок починається або закінчується певною підстрічкою.

Перевірка, чи рядок починається з певної підстрічки

```
рядок = "Hello, World!"

# Перевірка, чи рядок починається з певної підстрічки
if рядок.startswith("Hello"):
    print("Рядок починається з 'Hello'")
else:
    print("Рядок не починається з 'Hello'")
```

Перевірка, чи рядок закінчується певною підстрічкою

```
рядок = "Hello, World!"

# Перевірка, чи рядок закінчується певною підстрічкою
if рядок.endswith("World!"):
    print("Рядок закінчується на 'World!'")
else:
    print("Рядок не закінчується на 'World!'")
```

У вищенаведених прикладах `.startswith()` перевіряє, чи рядок починається з "Hello", а `.endswith()` перевіряє, чи рядок закінчується на "World!".

Обидва методи повертають булеве значення (`True` або `False`), що робить їх корисними для логічних перевірок та умов в коді.

▼ Регістр символів строки, перевірка і перетворення регістру

Перевірка Регістру

1. **Перевірка, чи рядок складається тільки з великих літер:**

```
рядок = "HELLO"
if рядок.isupper():
    print("Рядок складається тільки з великих літер.")
```

- **Перевірка, чи рядок складається тільки з малих літер:**

```
рядок = "hello"
if рядок.islower():
    print("Рядок складається тільки з малих літер.")
```

- **Перевірка, чи перший символ рядка є великою літерою:**

```
рядок = "Hello"
if рядок.istitle():
    print("Перший символ рядка є великою літерою.")
```

Перетворення Регістру

1. **Перетворення всіх літер рядка у великі:**

```
рядок = "hello"
великі_літери = рядок.upper()
print(великі_літери)
```

- **Перетворення всіх літер рядка у малі:**

```
рядок = "HELLO"
малі_літери = рядок.lower()
print(малі_літери)
```

- **Перетворення першої літери рядка у велику:**

```
рядок = "hello"
велика_перша_літера = рядок.capitalize()
print(велика_перша_літера)
```

- **Перетворення першої літери кожного слова у велику:**

```
рядок = "hello world"
великі_перші_літери = рядок.title()
print(великі_перші_літери)
```

- **Перетворення великих літер у малі і навпаки:**

```
рядок = "Hello World"
інверсія_регістру = рядок.swapcase()
print(інверсія_регістру)
```

Ці методи дозволяють вам ефективно перевіряти та змінювати регістр символів у рядках в залежності від ваших потреб.

▼ Пошук у строці: .find()

Метод `.find()` використовується для пошуку підстрічки в рядку і повертає індекс першого входження знайденої підстрічки. Якщо підстрічка не знайдена, повертається -1.

Ось приклад використання методу `.find()`:

```
string = "Це приклад для пошуку у рядку."

# Пошук індексу першого входження підстрічки
index = string.find("пошук")
if index != -1:
    print(f"Знайдено на позиції {index}.")
else:
    print("Підстрічка не знайдена.")
```

У цьому прикладі, якщо слово "пошук" знаходиться в рядку, програма виведе позицію першого входження. Якщо слово не знайдено, виведеться повідомлення про те, що підстрічка не знайдена.

Також слід зазначити, що метод `.find()` може приймати додаткові аргументи для обмеження діапазону пошуку:

```
індекс = рядок.find("пошук", початковий_індекс, кінцевий_індекс)
```

У цьому випадку, пошук буде виконуватися тільки у підстрічці рядка від `початковий_індекс` до `кінцевий_індекс`.

▼ Заміна у строці: .replace()

Метод `.replace()` використовується для заміни входжень певної підстрічки іншою підстрічкою в рядку. Ось приклад використання:

```
рядок = "Це приклад для заміни у рядку."

# Заміна слова "заміни" на "підміни"
новий_рядок = рядок.replace("заміни", "підміни")

print("Оригінальний рядок:", рядок)
print("Результат заміни:", новий_рядок)
```

У цьому прикладі всі входження слова "заміни" у рядку будуть замінені на слово "підміни". Важливо зазначити, що `.replace()` повертає новий рядок, і оригінальний рядок залишається незмінним.

Також метод `.replace()` може приймати третій аргумент, який вказує кількість входжень, які потрібно замінити. Наприклад:

```
рядок = "Це приклад для заміни у рядку, заміни лише перше входження."

# Заміна першого входження слова "заміни" на "підміни"
новий_рядок = рядок.replace("заміни", "підміни", 1)

print("Оригінальний рядок:", рядок)
print("Результат заміни:", новий_рядок)
```

У цьому випадку буде замінено лише перше входження слова "заміни".

▼ Обрізання зайвих символів строки: strip(), lstrip() та rstrip()

Методи `strip()`, `lstrip()`, та `rstrip()` використовуються для обрізання зайвих (пробільних або інших) символів з початку та/або кінця рядка відповідно. Ось їх приклади:

1. **`strip()`: Обрізка з обох кінців**

```
рядок = "    Привіт, світ!    "
очищений_рядок = рядок.strip()

print("Оригінальний рядок:", рядок)
print("Очищений рядок:", очищений_рядок)  # > Привіт, світ!
```

2. `lstrip()`: Обрізка з початку рядка

```
рядок = "    Привіт, світ!    "
очищений_рядок = рядок.rstrip()

print("Оригінальний рядок:", рядок)
print("Очищений рядок:", очищений_рядок)  # > Привіт, світ!
```

3. `rstrip()`: Обрізка з кінця рядка.

```
рядок = "    Привіт, світ!    "
очищений_рядок = рядок.strip()

print("Оригінальний рядок:", рядок)
print("Очищений рядок:", очищений_рядок)  # >     Привіт, світ!
```

▼ Комбінування строкових змінних ','.join(str)

Метод `','.join(str)` використовується для об'єднання (комбінування) рядка зі списку або ітерабельного об'єкту за допомогою певного роздільника. Ось приклад використання:

```
# Список рядків
список_рядків = ["apple", "orange", "banana"]

# Об'єднання рядків зі списку за допомогою роздільника ','
об'єднані_рядки = ','.join(список_рядків)

print("Список рядків:", список_рядків)
print("Об'єднані рядки:", об'єднані_рядки)
```

У цьому прикладі метод `','.join()` об'єднує рядки зі списку `["apple", "orange", "banana"]` за допомогою коми як роздільника, і результат виглядає так:

```
Список рядків: ['apple', 'orange', 'banana']
Об'єднані рядки: apple,orange,banana
```

Цей метод особливо корисний, коли вам потрібно об'єднати елементи списку в рядок з певним роздільником.

▼ Перетворення строкових даних в інший тип даних

В Python існують різні функції для перетворення строкових даних в інші типи даних. Ось кілька основних прикладів:

У ціле число (int):

```
рядок = "123"
ціле_число = int(рядок)
```

У дійсне число (float):

```
рядок = "123.45"
дійсне_число = float(рядок)
```

У список (list) елементів, розділених певним роздільником:

```
рядок = "1,2,3,4,5"
список = рядок.split(',')
```

У кортеж (tuple) елементів, розділених певним роздільником:

```
рядок = "1,2,3,4,5"
кортеж = tuple(рядок.split(','))
```

У булеве значення (bool):

```
рядок = "True"
булеве_значення = bool(рядок)
```

💡 **НЕБЕЗПЕЧНИЙ МОМЕНТ:**

```
string = "False"
bool_var = bool(string)
```

`bool_var` буде мати значення `True`, оскільки усе, що не пуста строка - `True`

Важливо враховувати, що перетворення може викликати виникнення виключень, якщо рядок не може бути коректно перетворений у вказаний тип. Тому рекомендується обробляти винятки або використовувати функції, які надають можливість безпечно перевірити можливість перетворення перед його виконанням (наприклад, `isdigit()` для перевірки, чи рядок містить тільки цифри перед спробою перетворення в ціле число).