§5 СИМВОЛИ ТА РЯДКИ

Символи та таблиці кодування

Символ – це умовний знак, що позначає певну сутність. Походить від древньо-грецького слова σύμβολον – «умовний знак» , «сигнал».

Комп'ютерний символ – це елемент множини, яка використовується для зображення та організації (комп'ютерних) даних.

До такої множини у комп'ютерних науках відносять букви різноманітних алфавітів, знаки пунктуації, цифри та різноманітні допоміжні і технічні символи.

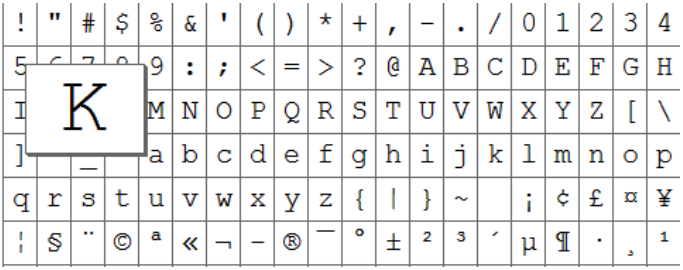

Рис. 5.1. Комп’ютерні символи

У подальшому у цьому посібнику під терміном **символ** будемо розуміти комп'ютерний символ, а вищезазначену множину символів, без обмеження загальності, будемо називати **алфавітом**.

Для збереження у пам’яті комп’ютера кожному символу співставляється деяке число, яке називають **кодом символа**, а бієктивне відображення обмеженої множини натуральних чисел у множину символів називається **таблицею кодування** символів (*eng.* **character set**).

Таблиць кодування існує багато. Нижче наведено приклад однієї з найпростіших та найпоширеніших таблиць кодування – таблиця ASCII.

Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char
32	[space]	48	0	64	@	80	P	96	`	112	p
33	!	49	1	65	A	81	Q	97	a	113	q
34	"	50	2	66	B	82	R	98	b	114	r
35	#	51	3	67	C	83	S	99	c	115	s
36	$	52	4	68	D	84	T	100	d	116	t
37	%	53	5	69	E	85	U	101	e	117	u
38	&	54	6	70	F	86	V	102	f	118	v
39	'	55	7	71	G	87	W	103	g	119	w
40	(	56	8	72	H	88	X	104	h	120	x
41	)	57	9	73	I	89	Y	105	i	121	y
42	*	58	:	74	J	90	Z	106	j	122	z
43	+	59	;	75	K	91	[	107	k	123	{
44	,	60	<	76	L	92	\	108	l	124	|
45	-	61	=	77	M	93	]	109	m	125	}
46	.	62	>	78	N	94	^	110	n	126	~
47	/	63	?	79	O	95	_	111	o	127	[backspace]

Рис. 5.2. Фрагмент таблиці кодування ASCII

Зверніть увагу, що цифри мають коди 48..57 (а не 0,...,9), великі та маленькі літери мають різні коди, бо це різні символи.

Символи у Python

У Python символи представлені у кодуванні UTF-8.

Під терміном **символьний літерал** (або символьна стала) будемо розуміти конкретний символ із таблиці кодування. Символьні літерали беруть у апострофи або подвійні лапки, щоб відрізняти їх від ідентифікаторів, цифр та інших символьних знаків.

```
a = "a" # Змінна a набуває значення символьного літералу "a"
A = "A" # Змінна A набуває значення символьного літералу "A"
A = ";" # Змінна A набуває значення символьного літералу ";"
```

Для задавання символу апострофа у Python використовують літерал `"'"`, а для задавання символу подвійних лапок – літерал `'"'`.

Спеціальні (екрановані) символи у Python

Спеціальні або екрановані послідовності (escape-послідовності) дозволяють задавати символи, які неможливо ввести з клавіатури. Крім цього екрановані послідовності використовуються для задавання символів, що мають спеціальне призначення (наприклад, для символів `"\"`, подвійних лапок, апострофа, тощо). Екрановані послідовності починаються символом `"\"` (обернена коса риска)

Таблиця 5.1. Екрановані послідовності

Символ	Значення
\<новий рядок>	Ігнорується (продовження рядка на наступний)
\\	Обернена коса риска(зберігає \)
\'	Апостроф (зберігає ')
\"	Подвійні лапки (зберігає ")
\a	Дзвінок
\b	Крок назад
\f	Завершення форми
\n	Кінець рядка
\r	Повернення каретки
\t	Табуляція
\v	Вертикальна табуляція
\0	Символ Null

```
>>> S = "Кав\'ярня \"Львівська казка\".\nТут найкраща кава!"
>>> print(S)
Кав'ярня "Львівська казка".
Тут найкраща кава!
```

Режим екранації спеціальних символів може бути вимкнений, якщо перед рядковим літералом, що містить екрановані символи поставити символ `r`.

```
>>> S = r"Кав\'ярня \"Львівська казка\".\nТут найкраща кава!"
>>> print(S)
Кав\'ярня \"Львівська казка\".\nТут найкраща кава!
```

Як бачимо, у цьому випадку рядок містить всі символи у тому вигляді у якому його задали.

Крім стандартного способу визначення символьних літералів, описаного вище, символьні літерали можна задавати через їхні коди. Для цього використовують екрановані послідовності наведені у таблиці

Таблиця 5.2. Екрановані послідовності

Символ	Значення
\N{id}	Ідентифікатор ID бази даних Юникоду
\uhhhh	16-бітний символ Юнікоду (в 16-ковому зображенні)
\Uhhhhhhhh	32-бітний символ Юнікоду (в 16-ковому зображенні)
\xhh	16-ве значення символа (в 16-ковому зображенні)
\ooo	8-ве значення символа

```
>>> "\u0063"
'c'
```

Операції з символами

Таблиця 5.3. Найпростіші операції з символами

Операція	Значення
`ord(c)`	Код символа `c` з таблиці кодування
`chr(n)`	Символ що відповідає у таблиці кодування коду `n`
`int(c)`	Перетворення символа `c` у цифру, що йому відповідає
`str(n)`	Перетворення цифри `n` у символ

```
>>> ord("A")
65
>>> chr(65)
'A'
>>> int("5")
5
```

Символів визначені 6 стандартних відношень

```
==, !=, >, <, >=, <=
```

які фактично проводяться над кодами відповідних символів

```
>>> "A" >= "C" # код "A" менший за код "C"
False
>>> "1" < "4"  # код "1" менший за код "4"
True
```

Приклад 5.1. Вивести на екран всі великі літери латинського алфавіту.

Розв'язок. Скористаємося тим фактом, що латинські літери в таблиці кодування розташовані у алфавітному порядку. Скористаємося циклом по проміжку від коду символа "A" до коду символа "Z":

```
for i in range(ord("A"), ord("Z") + 1):
    print(chr(i), end = '')
```

Нагадаємо, що параметр `end` в інструкції `print` вказує яким символом завершувати інструкцію виведення на екран. Типовим значенням є символ `'\n'` – символ нового рядка. У цій програмі ми використали порожній символ, для того, щоб символи алфавіту були виведені в одному рядку.

Результат виконання програми

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Приклад 5.2. Визначити, чи є даний символ латинською літерою (великою або маленькою), цифрою або ні тим ні іншим.

Розв'язок. Зауважимо, що цю задачу легко розв'язати, використовуючи стандартні функції Python. Проте для глибшого розуміння роботи з символами напишемо програму без їхнього використання. Скористаємося тим фактом, що літери у таблиці кодування розташовані в алфавітному порядку, а цифри – в порядку зростання.

```
Ch = input("Задайте символ ")
if Ch >= 'a' and Ch <= 'z':
    print("%c – маленька літера" % Ch)
elif Ch >= 'A' and Ch <= 'Z':
    print("%c – велика літера" % Ch)
elif Ch >= '0' and Ch <= '9':
    print("%c – цифра" % Ch)
else:
    print("%c – ні літера, ні цифра" % Ch)
```

Приклад 5.3. Написати програму, що переводить маленькі латинські літери у відповідні великі літери (у верхній регістр).

Розв'язок. Аналогічно до попередньої задачі, розв'яжемо поточну задачу, не використовуючи готові функції Python. Аналогічно до попереднього прикладу скористаємося властивостями таблиці кодування. Помітимо, що різниця кодів між символами маленької і великої літери однакова і становить

```
ord('A') – ord('a')
```

Тоді програма має такий вигляд:

```
Ch = input("Задайте символ ")
if Ch >= 'a' and Ch <= 'z':
    UpperCh = chr(ord(Ch) + ord('A') – ord('a'))
    print("%c -> %c" % (Ch, UpperCh))
else:
    print("Введений символ не є латинською літерою")
```

Рядки

Рядок – це послідовність символів

Рядки у Python належать до незмінюваних (immutable) типів.

Аналогічно до символів, щоб відрізняти літерали рядків від ідентифікаторів, цифр та інших символьних знаків рядкові літерали (тобто послідовність символів) оточують одним з таких способів:

- апострофами або подвійними лапками для рядків, що розташовуються у одному фізичному рядку;
- потрійними апострофами або потрійними подвійними лапками для рядків, що розташовуються у кількох фізичних рядках.

```
'Рядок-літерал'
"Рядок-літерал"

'''Це рядок-літерал, який
   розташовується у кількох рядках'''

"""Це також рядок-літерал, який
   розташовується у кількох рядках"""
```

Створення рядків

Існує кілька способів створення списків. Розглянемо їх

1. Створення рядка за допомогою літерала.

```
empty_string = ""     # Порожній рядок

symbol = "a"       # Рядок, що складається з одного елементу.

few_lines = '''Рядки дуже потужний механізм
обробки текстів.'''
```

2. Створення рядка за допомогою перетворення у рядок використовуючи функцію `str`.

```
>>> s = str(12345)
>>> print(s)
'12345'
```

Робота з рядком

Фактично рядок – це кортеж символів, тому робота з рядком дуже подібна до роботи з кортежом. Отже,

- до символів рядка можна звертатися за їхніми індексами;

- всі символи рядка можна послідовно перебрати використовуючи цикл по колекції `for`;
- для рядка можна брати зріз;
- інші операції, що визначені для кортежів.

Підсумуємо у таблиці операції для рядків успадковані від кортежів

Таблиця 5.4. Операції для рядків успадковані від кортежів

Операція	Опис
`s[i]`	`i`-й символ рядка `s` Аналогічно до кортежів, `i` може бути від'ємним, тоді така операція повертає `(-i)` символ з рахуючи з кінця рядка.
`s[i:j]`	Зріз з `s` від `i` до `j` (підрядок, що починається з `i`–го символа та закінчується `j-1`-м символом)
`s[i:j:k]`	Зріз з `s` від `i` до `j` з кроком `k`
`len(s)`	довжина `s`
`min(s)`	Найменший символ рядка `s`
`max(s)`	Найбільший символ рядка `s`

```
>>> s = "Python"
>>> s[2]
't'
>>> for c in s:
        print(c)
P
y
t
h
o
n
>>> s[2:]
'thon'
>>> s[::-1]
'nohtyP'
>>> len(s)
6
>>> max(s)
'y'
```

Проте, на відміну від кортежів, для рядків визначено інструкцію `input` введення рядка з клавіатури, відомої нам з попередніх тем

```
s = input(prompt)
```

де `s` – змінна-рядок, у яку буде записано результат введення з клавіатури, `prompt` – рядок підказка, яку буде виведено на екран під час очікування введення даних з клавіатури.

Розглянемо кілька прикладів роботи з рядками.

Приклад 5.4. Перевірити, чи є рядок симетричним

Розв'язок. Рядок є симетричним якщо його запис справа на ліво збігається з записом з ліва на право. Для розв'язання цієї задачі використаємо те, що до рядка можна будувати зріз (з від'ємним кроком). Отже, отримаємо таку просту програму.

```
S = input("Введіть рядок ")

if S == S[::-1]:
    print ("Заданий рядок є симетричним")
else:
    print ("Заданий рядок НЕ є симетричним")
```

Приклад 5.5. Обчислити кількість входжень символа `a` у рядок `S`.

Розв'язок. Для розв'язання цієї задачі проітеруємо рядок по-символьно за допомогою циклу по колекції. Використаємо лічильник, який будемо збільшувати на один кожного разу коли будемо зустрічати символ `a`. Отже програма матиме вигляд

```
S = input("Введіть рядок ")
a = input("Введіть символ ")

n = 0              # Змінна лічильник
for ch in S:       # Змінна ch пробігає всі символи рядка S
    if ch == a:    # Якщо ch == a
        n += 1     # збільшуємо лічильник на одиницю

print("\"%s\" входить у рядок \"%s\" %d разів" % (a, S, n))
```

Приклад 5.6. Перевірити, чи правильно в заданому виразі розставлені круглі дужки (тобто, чи стоїть справа від кожної відкритої дужки відповідна до неї закрита дужка, а зліва від кожної закритої – відповідна до неї відкрита).

Розв'язок. Використаємо змінну `k` для підрахунку різниці кількості відкритих і закритих дужок. Додатково використаємо змінну `r`, яка буде набувати значення **`False`**, якщо закритій дужці не відповідає жодна із відкритих дужок, що їй передують. Наприклад, якщо текст має вигляд

```
"a * ( b + c ) ) * ( d"
```

то, `r = False`, хоча при цьому `k = 0`, бо кількість відкритих і закритих дужок однакова.

```
expresion = input("Введіть вираз ")
k = 0    # різниця кількості відкритих і закритих дужок
r = True # закритій дужці відповідає відкрита
for c in expresion:
    if c == '(':
        k += 1
    if c == ')':
        k -= 1
    if k < 0: # закритій дужці не відповідає відкрита
        r = False
        break

if (k == 0 and r):
    print("Дужки розставлені правильно")
else:
    print("Дужки розставлені неправильно")
```

Приклад 5.7. У заданий текст входять тільки цифри та літери. Визначити, чи правда, що сума числових значень цифр, які входять у текст, дорівнює довжині тексту.

Розв'язок. Використаємо змінну `m` для підрахунку суми цифр, що містяться у заданому тексті.

```
S = input("Введіть вираз ")
m = 0
for c in S:
    if c >= '0' and c <= '9':
        m = m + int(c) # тут використовується операція int(c)
                       # перетворення символа c у число

if len(S) == m:
    print("Так")
else:
    print("Ні")
```

Операції над рядками

Таблиця 5.5. Конкатенація і зчеплення

Операція	Опис
`s1 + s2`	конкатенація рядків `s1` та `s2`, тобто створення нового рядка, що утворений дописуванням до кінця першого рядка символів другого.
`s * n` `n * s`	`n` зчеплених копій рядка `s`

```
>>> s1 = "інформатика"
>>> s2 = " та програмування"
>>> print(s1 + s2)
інформатика та програмування
>>> print(s1 * 2)
інформатикаінформатика
```

Приклад 5.8. Надрукувати заданий текст, виключивши з нього всі знаки арифметичних операцій.

Розв'язок. Для спрощення перевірки чи є деякий символ арифметичною операцією, створимо кортеж, що містить символи арифметичних операцій. Далі, оскільки рядок це колекція, можемо пробігти його по-символьно за допомогою циклу **for**. Якщо поточний символ не є знаком арифметичної операції будемо дописувати його (за допомогою операції конкатенації) до рядка-результату.

```
expresion = input("Введіть вираз ")
operators = ('+', '-', '*', '/') # кортеж операцій
result = ""                      # рядок-результат
for c in expresion:          # пробігаємо рядок по-символьно
    if c not in operators:   # якщо поточний символ не є
                             # арифметичною операцією
        result = result + c # дописуємо його до результату

print(result)
```

Таблиця 5.6. Перевірка входження символів до рядка

Операція	Опис
`c in S`	Повертає значення **True**, якщо символ `c` міститься у рядку `S`
`c not in S`	Повертає значення **True**, якщо символ `c` не міститься у рядку `S`

```
>>> S = "інформатика"
>>> "р" in S
True
>>> "2" in S
False
>>> "2" not in S
True
```

Таблиця 5.7. Пошук за заміна

Операція	Опис
`s.count(p)`	Повертає кількість входжень рядка `p` до рядка `s`
`s.find(p,i,j)`	Індекс першого входження рядка `p` у рядок `s`. Пошук здійснюється у зрізі `s[i:j]`. Параметри `i` і `j` мають типові значення – 0 і кількість символів у рядку відповідно. Повертає значення -1 якщо `p` не знайдено.
`s.index(p,i,j)`	Індекс першого входження рядка `p` у рядок `s`. Пошук здійснюється у зрізі `s[i:j]`. Параметри `i` і `j` мають типові значення – 0 і кількість символів у рядку відповідно. Породжує помилку, якщо `p` не знайдено.
`s.replace(old, new, count)`	Повертає рядок s, у якому всі входження рядка `old` замінено рядком `new`. Параметр `count` можна опустити. Якщо задано `count`, то замінюється не більше `count` перших входжень.

```
>>> s = "Pentium 4"
>>> s.find("4")
8
>>> s.replace("4", "Pro")
'Pentium Pro'
```

Приклад 5.9. Створити програму перетворення рядка, замінивши в ньому кожну крапку трьома крапками.

Розв'язок. Для написання програми досить скористатися функцією `replace` з вищенаведеної таблиці.

```
S = input("Введіть рядок ")
S = S.replace('.', '...')
print(S)
```

Таблиця 5.8. Функції аналізу рядка

Функція	Опис
`s.isalnum()`	Повертає **`True`**, якщо всі символи рядка `s` є літерами або цифрами.
`s.isalpha()`	Повертає **`True`**, якщо всі символи рядка `s` є літерами.
`s.isdigit()`	Повертає **`True`**, якщо всі символи рядка `s` є цифрами.
`s.isidentifier()`	Повертає **`True`**, якщо рядок `s` є ідентифікатором.
`s.islower()`	Повертає **`True`**, якщо всі літери рядка `s` у нижньому регістрі
`s.isnumeric()`	Повертає **`True`**, якщо всі символи рядка `s` є числовими.
`s.isprintable()`	Повертає **`True`**, якщо всі символи рядка `s` є друкованими.
`s.isspace()`	Повертає **`True`**, якщо всі символи рядка `s` є пропусками.
`s.istitle()`	Повертає **`True`**, якщо рядок `s` є заголовком (усі слова починаються з великої літери).
`s.isupper()`	Повертає **`True`**, якщо всі літери рядка `s` у верхньому регістрі

```
>>> s = "інформатика"
>>> s.islower()
True
>>> S1 = "Pentium 4"
>>> S1.isalnum()
False
>>> S2 = "Pentium4"
>>> S2.isalnum()
True
```

Приклад 5.10. Надрукувати лише великі латинські літери, що входять до заданого рядка.

Розв'язок. Проітеруємо рядок по-символьно і скористаємося функцією `isupper()` з вищенаведеної таблиці, для того щоб вияснити чи є поточний символ великою латинською літерою.

```
S = input("Введіть рядок ")
```

```
for ch in S:
    if ch.isupper():
        print(ch)
```

Таблиця 5.9. Функції модифікації рядка

Функція	Опис
`s.capitalize()`	Повертає копію рядка `s`, у якій перший символ – велика літера, а інші – маленькі.
`s.lower()`	Повертає копію рядка `s`, у якій всі літери рядка s переведені до нижнього регістру
`s.swapcase()`	Повертає копію рядка `s`, в якій маленькі літери змінені на великі та навпаки.
`s.title()`	Повертає копію рядка `s` у форматі заголовку (усі слова починаються з великої літери).
`s.upper()`	Повертає копію рядка `s` у форматі з усіма великими літерами.

```
>>> s1 = "інформатика"
>>> s1.upper()
'ІНФОРМАТИКА'
```

Таблиця 5.10. Розбиття та склейка

Функція	Опис
`s.join(t)`	Будує та повертає рядок з усіма елементами `t` (`t` – колекція рядків – кортеж або список), між якими в якості розділювача стоїть рядок `s`
`s.split(sep)`	Повертає список, у якому рядок `s` розбито на підрядки рядками-розділювачами `sep`. Якщо `sep` не вказано, то мається на увазі рядок з довільної кількості пропусків
`s.splitlines()`	Повертає список рядків, який є розбиттям `s` на підрядки, обмежені символами кінця рядка (`\n`, `\r` або `\r\n`).

```
>>> l = ["інформатика", "та", "програмування"]
>>> separator = "____"
>>> separator.join(l)
'інформатика____та____програмування'
>>> separator1 = " "
>>> s = separator1.join(l)
>>> s
'інформатика та програмування'
>>> s.split()
['інформатика', 'та', 'програмування']
```

Приклад 5.11. Підрахувати кількість слів у введеному з клавіатури рядку.

Розв'язок. У цій задачі словом будемо називати послідовність символів, що відокремлена символами пропуску. Для розв'язання задачі достатньо застосувати до заданого рядка метод `split`, який фактично побудує список всіх слів рядка. Отже програма буде мати такий вигляд

```
S = input("Введіть рядок ")
# Наприклад, S = "  інформатика   та  програмування   "
l = S.split() # l = ['інформатика', 'та', 'програмування']
print("Кількість слів у рядку", len(l))
```

Приклад 5.12. Нехай у рядку міститься послідовність слів, що розділена одним чи декількома символами пропуску. Необхідно видалити зайві пропуски між словами, так, щоб слова розділялися лише одним символом пропуску. Також видалити всі пропуски на початку та вкінці рядка.

Розв'язок. Найпростіший спосіб розв'язання цієї задачі – розбили рядок на слова, а далі склеїти отриманий список за допомогою функції `join` застосувавши її до символа пропуска у якості розділювача.

Отже програма

```
S = input("Введіть рядок ")
                  # S = "  інформатика   та  програмування   "
l = S.split()     # l = ['інформатика', 'та', 'програмування']
result = " ".join(l)
                  # result = "інформатика та програмування"
print(result)
```

Приклад 5.13. У заданому рядку всі символи `'0'` замінити на `'1'`, а символи `'1'` на `'0'`.

Розв'язок. У цій задачі ми не можемо скористатися операцією `replace`, Якщо ми спочатку замінимо символи `'0'` на `'1'`, то рядок не буде містити нулів і відповідно, коли ми потім навпаки замінимо `'1'` на `'0'`, то рядок не буде містити одиниць. Отже, напишемо програму, що реалізує поставлену задачу здійснюючи заміну для кожного окремого символа рядка.

Нагадаємо, рядок належить до незмінюваних типів, тому ми не можемо безпосередньо змінювати символи у рядку. У цій задачі можна застосувати підхід, що використовувався у Прикладі 5.8. Проте застосуємо інший підхід, а саме перетворимо рядок у список символів, проведемо необхідну заміну у отриманому списку символів і склеїмо символи до купи використовуючи операцію `join`.

Заміну будемо здійснювати таким чином: будемо пробігати список символів і якщо поточний символ буде `'0'` або `'1'`, то будемо змінювати його на `'1'` або `'0'` відповідно.

Отже, програма буде мати вигляд

```
S = input("Введіть рядок ")
l = list(S) # l - список символів рядка S
for i in range(len(l)):
    if l[i] == '0':
        l[i] = '1'
    elif l[i] == '1':
        l[i] = '0'

S = "".join(l)

print(S)
```

Порівняння рядків

Рядки у Python можна порівнювати. Порівняння рядків проводиться лексикографічно.

Будемо казати, що послідовність $a = \{a_1 a_2 \dots a_n\}$ **лексикографічно менша** (або просто менша) за послідовність $b = \{b_1 b_2 \dots b_n\}$, якщо існує таке натуральне k, що для всіх $i \le k$ виконується рівність $a_i = b_i$, а для $i = k + 1$ справедлива нерівність $a_{k+1} < b_{k+1}$. Якщо для всіх $i \le n$ виконується рівність $a_i = b_i$, то будемо казати, що послідовності a і b **лексикографічно рівні** (або просто рівні).

Наприклад, послідовність $\{012\}$ менша за послідовність$\{021\}$. Також якщо згадати, що символи можна порівнювати, порівнюючи їхні номери в алфавіті, то послідовність символів $\{aaa\}$ менша за послідовність $\{aba\}$.

Вищенаведене означення можна узагальнити на випадок послідовностей, у які входить неоднакова кількість членів. Для цього будемо вважати, що порожній член послідовності (тобто його відсутність) менший за будь-який наявний. Отже, з цього можемо зробити висновок, що, наприклад, послідовність символів $\{aaa\}$ менша за послідовність символів $\{abcd\}$.

Лексикографічний порядок послідовностей – це порядок, при якому послідовності відсортовані за (лексикографічним) зростанням.

Прикладом лексикографічного порядку є послідовність слів у словнику.

Для порівняння рядків, у Python як і для символів визначені 6 стандартних відношень

```
==, !=, >, <, >=, <=
```

```
>>> "Pentium Pro" >= "Pentium"
True
>>> "True" <= "False"
False
```

Приклад 5.14. Перевірити чи слова, що записані у рядку знаходяться у словниковому порядку.

Розв'язок. Для розв'язання цієї задачі достатньо розбити рядок на слова і далі, скористатися операцією порівняння рядків для слів отриманого списку. Цю задачу можна віднести до задач пошуку. Дійсно, припустимо спочатку, що слова у рядку розташовані у порядку зростання. Далі будемо шукати послідовну пару слів, таку, що перше слово більше за друге. Якщо таку пару ми знайдемо, то слова у рядку не впорядковані. Отже програма буде мати вигляд

```
S = input("Введіть рядок ")
l = S.split()
ok = True
for i in range(1, len(l)):
    if l[i - 1] > l[i]:
        ok = False
        break

if ok:
    print("Слова в алфавітному порядку")
else:
    print("Слова НЕ в алфавітному порядку")
```

Форматування рядків

Досить часто виникає ситуація, коли потрібно створити рядок, у яких підставляються деякі дані, отримані у процесі виконання програми. Одним зі способів такої підстановки, а саме використання оператора %, ми вже неодноразово користувалися. Недоліком цієї операціє є те, що необхідно наперед знати типи даних, які очікуються для підстановки. Наприклад, у падку використання такої інструкції

```
    print("%c - велика літера" % Ch)
```

Python очікує для підстановки саме символьний літерал у змінній Ch. У випадку, якщо ж тип змінної наперед не відомий, то результат виконання інструкції може бути неочікуваним. Звичайно, можна перед підстановкою

робити зведення літералу до конкретного типу, наприклад, вищенаведену інструкцію змінити на таку

```
    print("%s – велика літера" % str(Ch))
```

проте, зручнішим способом подолання згаданої проблеми є використання методу `format()`.

Метод `format()` подібний до форматування рядка за допомогою оператора `%`, тобто він також задає спосіб підстановки відповідних значень у рядок у відповідні позиції. Проте він має інший синтаксис і не вимагає вказування типів даних, що підставляються.

Розглянемо його синтаксис у найпростішому випадку.

Нехай задано рядок `s`, що містить послідовності символів `{}` (послідовність символів, що складається з відкритої та закритої фігурної дужки), наприклад

```
>>> s = "Мій улюблений предмет {}"
```

Метод формат, застосований до рядка, підставляє аргументи у рядок замість позицій виділених фігурними дужками `{}`.

```
>>> s.format("програмування")
'Мій улюблений предмет програмування'
```

Кількість підстановок у рядок може бути довільна, головне, щоб кількість аргументів дорівнювала кількості підстановок виділених з допомогою `{}` у рядку до якого застосовується метод `format()`.

```
>>> '{}, {}, {}'.format('a', 'b', 'c')
'a, b, c'
```

При задаванні рядка, до якого буде застосовуватися метод `format()`, у фігурних дужках може вказуватися який за номером аргумент буде підставлено. У цьому випадку кількість аргументів методу `format()` може бути меншою за кількість підстановок рядка

```
>>> '{0}, {1}, {2}'.format('a', 'b', 'c')
'a, b, c'
>>> '{2}, {1}, {0}'.format('a', 'b', 'c')
'c, b, a'
>>> '{0}-{1}{0}'.format('abra', 'cad')
'abra-cadabra'
```

Можливості форматування рядків за допомогою оператора % або методу format не обмежуються можливостями описаними у цьому посібнику. Зокрема, під час їхнього використання, можна вказувати точність для дійсних чисел, кількість символів виділених для підстановки, вирівнювання рядка, тощо. Для детальнішого ознайомлення рекомендуємо звернутися до офіційної документації [1] або іншої додаткової літератури.

Задачі для самостійної роботи

5.1. Вивести на екран таку таблицю символів:

```
Aa Bb Cc Dd Ee Ff Gg Hh
Ii Jj Kk Ll Mm Nn Oo Pp
Qq Rr Ss Tt Uu Vv Ww Xx
```

5.2. Визначити, яка з двох заданих літер у даному тексті трапляється частіше.

5.3. Визначити, чи входить до даного тексту

a) кожна з літер слова "`key`";

b) кожна з літер заданого слова.

5.4. Скласти програму підрахунку загального числа входжень символів основних арифметичних операцій (`'+'`, `'-'`, `'*'`, `'/'`) у рядку.

5.5. Дано рядок, що містить принаймні одну цифру. Знайти

a) максимальну цифру, що міститься у рядку;

b) цифру, що входить до рядка найбільшу кількість разів;

c) суму цифр, що входять до рядка.

d) цифри, що трапляються у рядку рівно два рази;

e) цифри, що не міститься у рядку;

5.6. Визначити, кількість різних символів у рядку.

5.7. Знайти символ, який входить у текст найбільшу кількість разів.

5.8. Визначити, чи є заданий текст правильним записом цілого числа (можливо, зі знаком).

5.9. Перевірити, чи є даний рядок ідентифікатором, натуральним числом, чи ні тим ні іншим.

5.10. Задано рядок, що містить електронну адресу користувача. Переконатися, що ця адреса є коректною (наявність символа @ і крапки, а також наявність принаймні двох символів після останньої крапки)

5.11. У заданий текст входять тільки цифри та літери. Визначити, чи задовольняє він такі властивості:

a) текст є десятковим записом числа, кратного 9 $(6, 4)$;

b) текст починається з деякої ненульової цифри, за якою стоять тільки літери, і їхня кількість дорівнює числовому значенню цієї цифри;

c) текст містить (крім літер) тільки одну цифру, причому її числове значення дорівнює довжині тексту;

d) сума парних числових значень цифр, які входять у текст, дорівнює