ПРОГРАМУВАННЯ (PYTHON)

РЯДКИ ТА ФАЙЛИ

Літерали рядків

Послідовність будь-яких символів, записана у одинарних або подвійних лапках, називається рядком (тип str)

```
'це рядок'                    # літерал рядка з 8 символів
"і це теж рядок"              # літерал рядка з 14 символів
'1000' # і це рядок (не число!) з 4-х символів

'' # а це порожній рядок
"" # і ще один порожній рядок
```

Подвійні лапки зазвичай використовують, якщо серед символів, що складають рядок, є апостроф:

```
s1 = "П'ять – ціле число"
```

І навпаки, одинарні лапки використовують, якщо рядок містить подвійні лапки:

```
s2 = 'ТзОВ "Пролісок"'
```

Як аргумент функції print() можна використовувати рядок у потрійних лапках (''' або """), записаний у кілька рядків — тоді він буде виведений на екран теж у кілька рядків:

```
print('''Речення, яке ви бачите, записане
і буде надруковане
у трьох рядках''')
```

Навіть якщо символи рядка утворюють число ('100'), дату ('27.05.2019'), IP-адресу ('192.168.0.1') тощо, для інтерпретатора все це звичайний текст (послідовність символів)без жодного змісту.

Символи

Мова Python, на відміну від багатьох інших мов програмування, не має окремого символьного типу — для неї кожен поодинокий символ належить до типу **str**, тобто є рядком, який складається з одного символу.

Коди символів. Функції ord() і chr()

Кожному символу відповідає унікальне ціле число (код).

Для перетворення символу в код і навпаки призначені вбудовані функції ord() і chr():

```
code_A = ord('A')
code_a = ord("a")
char1041 = chr(1041)
print(code_A, code_a, char1041)          # 1040 1072 Б
```

1) великі і малі літери — це різні символи;
2) сусіднім літерам відповідають сусідні коди (не завжди!)

'\n' - символ кінця рядка

```
print('Весна\nЛіто\nОсінь\nЗима')
```

```
Весна
Літо
Осінь
Зима
```

'\t' - символ табуляції

```
print('1\t2\t3\n10\t100\t1000')
```

```
1               2               3
10              100             1000
```

Керівні символи. Екранування

Керівні символи починаються з оберненої косої риски \, за якою слідують один чи кілька інших символів (уся послідовність символів трактується інтерпретатором як один символ).

Обернена коса риска вказує на те, що наступний за нею символ (символи) мають трактуватися по особливому. Її часто використовують для екранування спеціальних символів (\\, \", \' та ін.):

```
print('П\'ять – ціле число')      # П'ять – ціле число
print('one\\two\\three')          # one\two\three
```

```
print('П\'ять – ціле число')
print('one\\two\\three')
```

Що буде надруковано, якщо символи не екранувати?

```
print('П'ять – ціле число')
print('one\two\three')
```

Неформатовані рядки

Щоб уникнути екранування символів, в Python використовуються неформатовані рядки, які починаються символом r:

```
print(r'one\two\three')            # one\two\three
```

Конкатенація (об’єднання) рядків

```
name = 'Роман'
s = 'Мене звати ' + name + '.'
print(s) # Мене звати Роман.
```

Щоб об’єднати рядок з об’єктом іншого типу, цей об’єкт треба попередньо перетворити в рядок за допомогою функції `str()`:

```
name, age = 'Роман', 44
s = 'Мене звати ' + name + '.'
s += ' Мені ' + str(age) + ' роки.'
print(s) # Мене звати Роман. Мені 44 роки.
```

Повторення рядків

```
s = '-'*5 + 'abc'*2 + '-'*5
print(s*3)
```

```
-----abcabc----------abcabc----------abcabc-----
```

```
# Програма друкує таблицю множення від 1 до 9

s = 'Таблиця множення:\n'

for factor_1 in range(1,10):
    for factor_2 in range(1,10):
        s += str(factor_1 * factor_2) + '\t'
    s += '\n'

print(s)
```

```
Таблиця множення:
1	2	3	4	5	6	7	8	9
2	4	6	8	10	12	14	16	18
3	6	9	12	15	18	21	24	27
4	8	12	16	20	24	28	32	36
5	10	15	20	25	30	35	40	45
6	12	18	24	30	36	42	48	54
7	14	21	28	35	42	49	56	63
8	16	24	32	40	48	56	64	72
9	18	27	36	45	54	63	72	81
```

Форматування рядків

Синтаксис: `рядок.format(аргументи)`

```
name, age = 'Roman', 43
text = 'My name is {0}. I\'m {1}.'.format(name, age)
print(text)           # My name is Roman. I'm 43.


pi = 3.14
text = 'pi = {0:7}'.format(pi)          # pi =    3.14
text = 'pi = {0:<7}'.format(pi)         # pi = 3.14
text = 'pi = {0:.3f}'.format(pi)        # pi = 3.140
```

f-рядки

```
name = 'Роман'
born = 1976
now = 2020

print(f"Мене звати {name}. Мені {now - born} роки.")
```

Індекси

Кожен елемент рядка (символ) має **індекс** — унікальне ціле число, яке однозначно вказує на порядковий номер символа у рядку. Індекси можна відраховувати з початку рядка (тоді нумерація починається з нуля: 0, 1, 2, ...), а можна з кінця (тоді індекси від'ємні: -1, -2, -3 і т. д.):

```
s = 'рядок'
print(s[0], s[2], s[-3])   # р д д
```

рядок – **НЕЗМІНЮВАНИЙ** тип!!!

Намагання змінити, вставити або вилучити елемент рядка викличе помилку:

```
s = 'рядок'
s[0] = 'Р'
TypeError: 'str' object does not support item assignment
```

Крім того, помилка виникне і у випадку, коли елемента з вказаним індексом не існує:

```
print(s[5])
IndexError: string index out of range
```

Зрізи

Операція зрізу [i:j] повертає частину рядка, починаючи з символу з індексом i та закінчуючи символом з індексом j-1:

```
s = 'рядок'
print(s[1:3])                # яд
```

При цьому наявність індексів не обов'язкова:

```
print(s[:3])
# ряд (відсутність першого індекса означає "від початку")
print(s[1:])
# ядок (відсутність другого індекса означає "до кінця")
print(s[:])
# рядок ("від початку до кінця" – копія рядка)
```

Зрізи: повний синтаксис

Повний синтаксис зрізу містить три індекси: **[i:j:k]**. Третім індексом **k** задається крок, який за його відсутності приймається рівним 1:

```
print(s[1::2])              # яо
```

У зрізах допускається використання від'ємних індексів, при цьому від'ємний крок вказує ва вибір елементів справа наліво:

```
print(s[1:-2])              # яд
print(s[3:0:-1])            # одя
print(s[-3:])               # док
```

рядок – **НЕЗМІНЮВАНИЙ** тип!!!

```
s = 'рядок'
s[::-1]
print(s)                                    # рядок


s = 'рядок'
s = s[::-1]
print(s)                                    # кодяр
```

Функції та методи рядків

```
s = 'Це рядок'
len(s)                  # повертає кількість символів у рядку, включаючи пробіли
                        # та керівні символи (8)
s.upper()               # переводить символи у верхній регістр ('ЦЕ РЯДОК')
s.lower()               # переводить символи у нижній регістр ('це рядок')
s.capitalize()          # перший символ у верхній регістр ('Це рядок')
s.replace('ок','ки')    # заміна усіх входжень 'ок' на 'ки' ('це рядки')
s.find('ряд')           # повертає індекс першого входження 'ряд' (3)
s.count('ряд')          # повертає кількість входжень 'ряд' (1)
s.isalpha()             # перевіряє, чи рядок містить тільки літери (False)
s.isdigit()             # перевіряє, чи рядок містить тільки цифри (False)
s.startswith('Це')      # перевіряє, чи рядок починається з 'Це' (True)
s.endswith('.')         # перевіряє, чи рядок закінчується крапкою (False)
s.strip()               # вилучає пробіли на початку та в кінці рядка
```

рядок – **НЕЗМІНЮВАНИЙ** тип!!!

У наступному коді рядок s не зміниться!!! Інтерпретатор просто створить новий рядок з рядка s, але не прив'яже його до змінної s:

```
s = 'Це рядок'
s.replace('ок','ки')
print(s) # Це рядок
```

```
s = 'Це рядок'
s = s.replace('ок','ки')
print(s) # Це рядки
```

```
s = 'Це рядок'
print(s.replace('ок','ки')) # Це рядки
print(s) # Це рядок
```

Метод split()

Метод split() повертає об'єкт іншого типу — список підрядків. За замовчуванням рядок розбивається на підрядки по символу пробілу, але аргументом методу можна задати будь-який інший розділювач, який не обов'язково складається з одного символу:

```
sentence = 'Я прочитав про програмування мовою Python'
words = sentence.split()
print(words)
#['Я', 'прочитав', 'про', 'програмування', 'мовою', 'Python']

print(sentence.split('про'))
# ['Я ', 'читав ', ' ', 'грамування мовою Python']
```

Використовуючи метод split() та індекси, можна писати компактний код.

```
filename = 'фото_1.jpg'
print(filename.split('.')[-1])    # jpg
```

Метод join()

Метод join() утворює новий рядок шляхом конкатенації елементів списку, який передається аргументом, при цьому рядок, до якого застосовується метод, виступає розділювачем:

```
c = '--'.join(words)
print(c) # Я--прочитав--про--програмування--мовою--Python

c = ''.join(words)
print(c) # ЯпрочитавпропрограмуваннямовоюPython
```

Перевірка на входження (оператор in)

```
s = 'Це рядок'
s1 = 'ряд'
print(s1 in s)                  # True
```

Посимвольний обхід рядка

Обхід елементів рядка (символів) здійснюється з використанням інструкції for:

```
s = 'Це рядок'
for symb in s:
    print(symb.upper(), end = '')
# буде надруковано ЦЕ РЯДОК
```

У наведеному вище коді змінна symb почергово набуває значень символів рядка ('Ц', 'е', '', 'р' і т. д.), а у тілі циклу ці значення друкуються у верхньому регістрі.

Посимвольний обхід рядка може здійснюватися і з використанням індексів:

```
for i in range(len(s)):
    print(s[i].upper(), end = '')
```

Якщо одночасно потрібні символ та його індекс:

```
for index, symb in enumerate(s):
    print(index, symb)
```

Програма може приймати вхідні дані не тільки з клавіатури за допомогою функції **input()**, але й з файлів.

Основними перевагами цього методу є можливість отримання великих обсягів вхідних даних, які можуть використовуватися одночасно великою кількістю програм.

Альтернативою виводу результатів виконання програми на екран за допомогою функції **print()** є запис результатів у текстовий файл.

Дані, які зчитуються з **текстового** файлу і записуються у файл, завжди є рядками (тип **str**).

Для роботи з файлами в Python призначена вбудована функція **open()**, яка створює об’єкт файлу на основі переданих їй **імені файлу** та **режиму** роботи з ним (читання, запис тощо).

--- Запис у файл ---

Щоб створити об'єкт файлу для подальшого запису в нього, потрібно передати функції open() першим аргументом ім'я цього файлу, а другим — рядок 'w':

```
f = open('result.txt', 'w')
```

Інтерпретатор шукатиме файл result.txt у тій же папці, де знаходиться програма.

Якщо файлу з таким іменем немає, то він буде створений.

Якщо файл з таким іменем є, то (увага!) усі дані в ньому будуть стерті без попередження. Тому, щоб відкрити існуючий файл в режимі "додавання в кінець", замість 'w' другим аргументом функції open() потрібно передати 'a'.

Для запису у файл використовується метод write():

```
f.write('Квадрати перших десяти натуральних чисел:\n')
for i in range(1, 11):
    f.write(str(i**2) + '\t')
```

Кожна порція даних додається в кінець файлу, але не з нового рядка.

```
Квадрати перших десяти натуральних чисел:
1 4 9 16 25 36 49 64 81 100
```

По завершенні запису у файл його потрібно закрити:

```
f.close()
```

Незакриття файлу може призвести до втрати даних, метод `close()` гарантує, що дані будуть записані у файл.

--- Зчитування з файлу ---

Для відкриття файлу з метою подальшого зчитування даних з нього потрібно передати функції open() другим аргументом 'r':

```
f = open('result.txt', 'r', encoding='utf-8')
```

Інтерпретатор шукатиме файл data.txt у тій же папці, де знаходиться програма. Якщо файлу з таким іменем немає, то буде ініційована помилка.

Зчитування файлу в один рядок: метод read()

```
allText = f.read()
```

Якщо намагатися ще раз виконати попередню інструкцію, то отримаємо порожній рядок, тому для повторного зчитування даних з файлу його треба наново відкрити методом open()

```
f = open('result.txt', 'r')
```

або перевстановити вказівник на початок методом seek():

```
f.seek(0)
```

Зчитування файлу порядково: цикл for

Синтаксис:

```
for line in f:
    ...
```

Змінна line почергово прийматиме значення рядків файлу.

Наприклад, якщо у файлі data.txt містяться числа (по одному в кожному рядку), то їхню суму можна порахувати так:

```
f = open('data.txt'); s = 0
for line in f:
    s += float(line.strip())
f.close()
```

Зчитування файлу порядково: метод readline()

```
f = open('data.txt')
s = 0
while True:
    line = f.readline()
    if line == '':
        break
    s += float(line.strip())
f.close()
```

А що, якщо порожній рядок всередині файлу?

Зчитування файлу порядково: метод readlines()

Метод readlines() повертає список рядків файлу

```
f = open('result.txt')
lines = f.readlines()
print(lines)
f.close()
```

```
['Квадрати перших десяти натуральних чисел:\n',

      '1\t4\t9\t16\t25\t36\t49\t64\t81\t100\t']
```

Режими відкриття файлів

Крім розглянутих вище трьох режимів відкриття файлів (для запису, для додавання в кінець і для зчитування), які задаються значеннями аргументів **'w'**, **'a'** і **'r'** відповідно, існує ще режим створення нового файлу для запису (якщо файл з вказаним іменем вже існує, то буде виведено повідомлення про помилку). Йому відповідає значення **'x'**.

Під час роботи з нетекстовими файлами (зображеннями, аудіозаписами та ін.), потрібно вказувати, що вони повинні відкриватися у двійковому режимі. Це здійснюється додаванням символу **'b'** перед потрібним режимом: **'bw'**, **'ba'**, **'br'**, **'bx'**.

Інструкція with

Інструкція with є альтернативою використання функції open() разом з методом close(). Вона теж створює дескриптор файлу і автоматично закриває файл по завершенню виконання блоку інструкцій.

```
with open('result.txt', 'w') as f:
    f.write('Квадрати перших десяти натуральних чисел:\n')
    for i in range(1, 11):
        f.write(str(i**2) + '\t')
```

Імена файлів 'data.txt' і 'result.txt', які використовувалися вище, є короткими іменами файлів. Приймаючи їх, функція open() шукає файли з такими іменами у поточній папці, тобто в тій же папці, у якій збережена програма.

Для встановлення зв'язку з файлами, збереженими в інших місцях файлової системи, використовуються повні імена файлів.

Повне ім'я файлу містить шлях (ланцюжок папок) до вказаного файлу. Елементи шляху відокремлюються символом '/' (для операційних систем Linux і macOS) або '\' (Windows). Щоб уникнути керівних символів у шляху, потрібно використовувати екранування або неформатовані рядки.

```
# приклади абсолютних шляхів:
'/home/user/code/numbers.txt'
'C:\\user\\code\\numbers.txt'       # використано екранування
r'C:\user\code\numbers.txt'         # використано неформатований рядок

# приклади відносних шляхів:
'../user/code/numbers.txt'
r'..\code\numbers.txt'
```

Абсолютний імпорт (шлях від робочої директорії)

```
└── project
    ├── package1
    │   ├── module1.py
    │   └── module2.py
    └── package2
        ├── __init__.py
        ├── module3.py
        ├── module4.py
        └── subpackage1
            └── module5.py
```

```python
from package1 import module1
from package1.module2 import function1
from package2 import class1
from package2.subpackage1.module5 import function2
```

```
└── project
    ├── package1
    │   ├── module1.py
    │   └── module2.py
    └── package2
        ├── __init__.py
        ├── module3.py
        ├── module4.py
        └── subpackage1
            └── module5.py
```

Відносний імпорт (шлях від папки, у якій файл)

```python
# package1/module1.py

from .module2 import function1
```

```python
# package2/module3.py

from . import class1
from .subpackage1.module5 import function2
```

Самостійна робота

os.path - підмодуль модуля os для роботи з шляхами

shutil - модуль для роботи з файлами та директоріями