

# Рядки

## Розділ 6

Python для всіх  
[www.py4e.com](http://www.py4e.com)



# Рядковий тип даних

- Рядок – це послідовність символів
- Рядковий літерал використовує лапки 'Hello' або "Hello"
- Для рядків + означає «об'єднати»
- Коли рядок містить цифри, він залишається рядком
- Ми можемо перетворити цифри в рядку на число за допомогою int()

```
>>> str1 = "Hello"
>>> str2 = 'there'
>>> bob = str1 + str2
>>> print(bob)
Hellothere
>>> str3 = '123'
>>> str3 = str3 + 1
Traceback (most recent call
last):  File "<stdin>", line 1,
in <module>
TypeError: cannot concatenate
'str' and 'int' objects
>>> x = int(str3) + 1
>>> print(x)
124
>>>
```

# Читання та конвертація

- Ми вважаємо, що краще читати дані за допомогою **рядків**, а потім розбирати і конвертувати їх так, як нам потрібно
- Це дає нам більше контролю над ситуаціями із помилками та / або неправильним введенням даних користувачем
- Вхідні числа потрібно **конвертувати** з рядків

```
>>> name = input('Enter: ')
Enter: Chuck
>>> print(name)
Chuck
>>> apple = input('Enter: ')
Enter: 100
>>> x = apple - 10
Traceback (most recent call
last):  File "<stdin>", line 1,
in <module>
TypeError: unsupported operand
type(s) for -: 'str' and 'int'
>>> x = int(apple) - 10
>>> print(x)
90
```



# Всередині рядків

- Ми можемо дістатися до будь-якого окремого символу в рядку за допомогою індексу, вказаного в **квадратних дужках**
- Значення індексу має бути цілим числом і починатися з нуля
- Значенням індексу може бути вираз, який обчислюється

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> fruit = 'banana'
>>> letter = fruit[1]
>>> print(letter)
a
>>> x = 3
>>> w = fruit[x - 1]
>>> print(w)
n
```

# Символ за межами

- Ви отримаєте **помилку python**, якщо спробуєте взяти індекс далі кінця рядка
- Тому будьте обережні при побудові значень індексів та зрізів

```
>>> zot = 'abc'
>>> print(zot[5])
Traceback (most recent call
last):  File "<stdin>", line
1, in <module>
IndexError: string index out
of range
>>>
```

# Рядки мають довжину

Вбудована функція `len` дає  
нам довжину рядка

b	a	n	a	n	a
0	1	2	3	4	5

```
>>> fruit = 'banana'
>>> print(len(fruit))
6
```

# Функція len()

```
>>> fruit = 'banana'
>>> x = len(fruit)
>>> print(x)
6
```

Функція – це певний збережений код, який ми використовуємо. Функція отримує **вхідні дані** і видає **вихідні дані** (результат)

'banana'  
(рядок)



функція  
len()



6  
(число)

# Функція len()

```
>>> fruit = 'banana'
>>> x = len(fruit)
>>> print(x)
6
```

Функція – це певний збережений код, який ми використовуємо. Функція отримує **вхідні дані** і видає **вихідні дані** (результат)

'banana'  
(рядок)



```
def len(inp):
    blah
    blah
    for x in y:
        blah
        blah
```



6  
(число)



# Циклічне проходження рядків

Використовуючи інструкцію **while**, ітераційну змінну та функцію **len**, ми можемо побудувати цикл, щоб переглянути кожну літеру в рядку окремо

```
fruit = 'banana'
index = 0
while index < len(fruit):
    letter = fruit[index]
    print(index, letter)
    index = index + 1
```

```
0 b
1 a
2 n
3 a
4 n
5 a
```

# Циклічне проходження рядків

- Визначений цикл з використанням інструкції **for** більш СТИЛЬНИЙ
- Цикл **for** повністю бере на себе ітераційну змінну

```
fruit = 'banana'  
for letter in fruit:  
    print(letter)
```

b  
a  
n  
a  
n  
a

# Циклічне проходження рядків

- Визначений цикл з використанням інструкції **for** більш стильний
- Цикл **for** повністю бере на себе ітераційну змінну

```
fruit = 'banana'
for letter in fruit :
    print(letter)
```

```
index = 0
while index < len(fruit) :
    letter = fruit[index]
    print(letter)
    index = index + 1
```

b  
a  
n  
a  
n  
a

# Цикли та підрахунки

Це простий цикл, який перебирає кожну літеру в рядку і підраховує, скільки разів цикл зустріне символ «а»

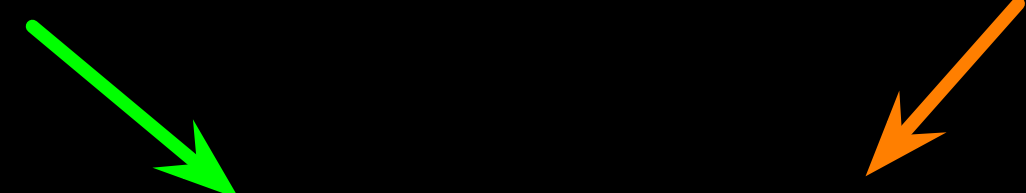
```
word = 'banana'
count = 0
for letter in word :
    if letter == 'a' :
        count = count + 1
print(count)
```

# Розглядаємо глибше in

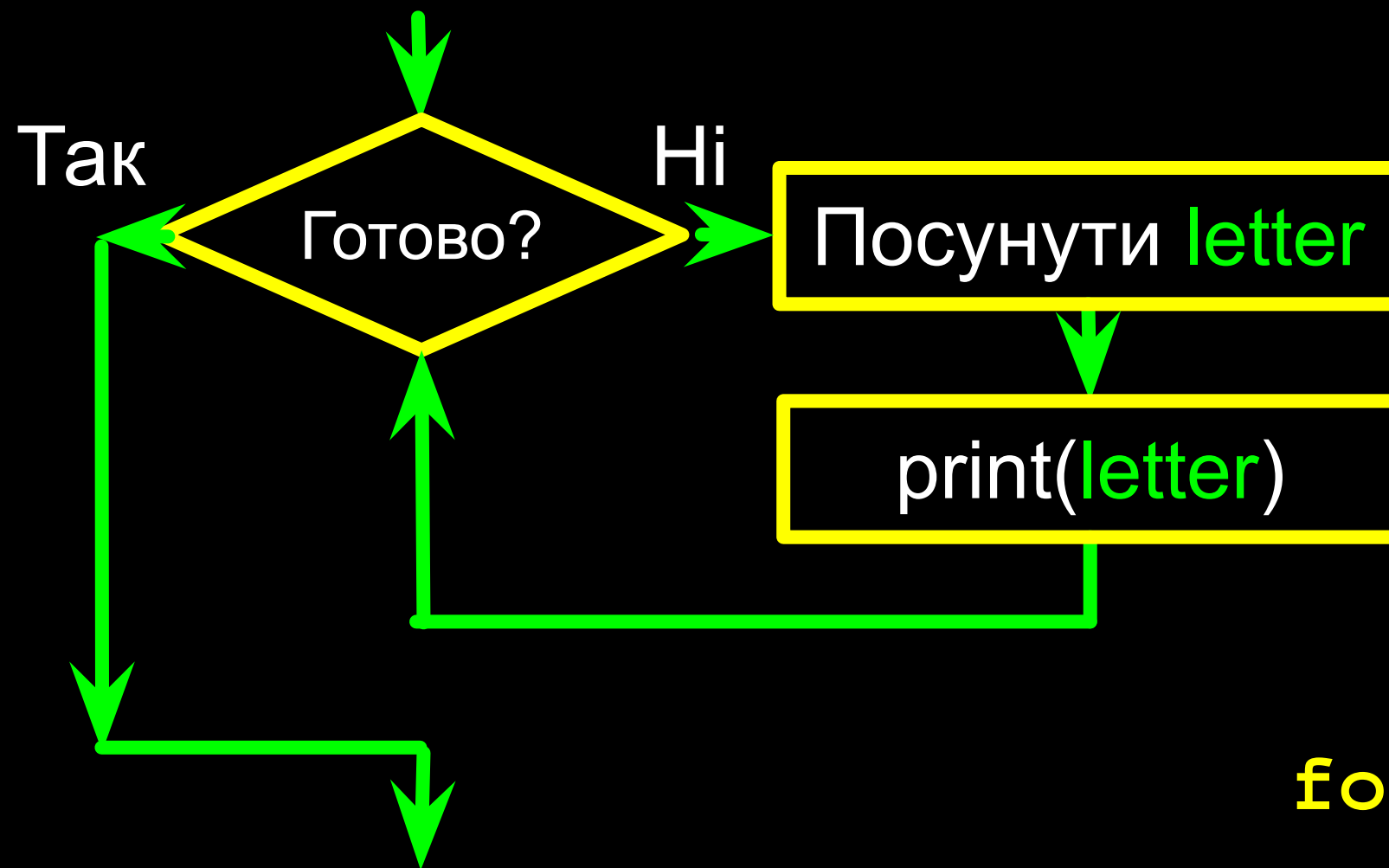
- Ітераційна змінна «перебирає» послідовність (впорядковану множину)
- Блок (тіло) коду виконується один раз для кожного значення в послідовності
- Ітераційна змінна перебирає всі значення в послідовності

Ітераційна  
змінна

Рядок з шести  
СИМВОЛІВ



```
for letter in 'banana' :  
    print(letter)
```



```
for letter in 'banana' :  
    print(letter)
```

Ітераційна змінна «перебирає» рядок, а блок (тіло) коду виконується один раз для кожного значення в послідовності

Більше операцій з рядками

# Зрізи рядків

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

- Ми також можемо переглянути будь-яку неперервну ділянку рядка, використовуючи **оператор двокрапки**
- Друге число – позиція одразу за кінцем зрізу – «до, але не включаючи»
- Якщо друге число більше довжини рядка, зріз зупиняється на кінці рядка

```
>>> s = 'Monty Python'
>>> print(s[0:4])
Mont
>>> print(s[6:7])
P
>>> print(s[6:20])
Python
```



# Зрізи рядків

M	o	n	t	y		P	y	t	h	o	n
0	1	2	3	4	5	6	7	8	9	10	11

Якщо ми опускаємо перше  
або останнє число зріза, то  
вважаємо, що це початок  
або кінець рядка відповідно

```
>>> s = 'Monty Python'
>>> print(s[:2])
Mo
>>> print(s[8:])
thon
>>> print(s[:])
Monty Python
```

# Об'єднання рядків

Коли оператор `+`  
застосовується до рядків,  
це означає  
«конкатенацію» або об'  
єднання

```
>>> a = 'Hello'
>>> b = a + 'There'
>>> print(b)
HelloThere
>>> c = a + ' ' + 'There'
>>> print(c)
Hello There
>>>
```

# Використання `in` як логічного оператора

- Ключове слово `in` також можна використовувати для перевірки того, чи знаходиться один рядок «всередині» іншого рядка
- Вираз `in` — це логічний вираз, який повертає значення `True` або `False` і може використовуватися в інструкції `if`

```
>>> fruit = 'banana'
>>> 'n' in fruit
True
>>> 'm' in fruit
False
>>> 'nan' in fruit
True
>>> if 'a' in fruit :
...     print('Found it!')
...
Found it!
>>>
```

# Порівняння рядків

```
if word == 'banana':  
    print('All right, bananas.')  
if word < 'banana':  
    print('Your word, ' + word + ', comes before banana.')elif word > 'banana':  
    print('Your word, ' + word + ', comes after banana.')else:  
    print('All right, bananas.')
```

# Бібліотека рядків

- У Python є набір **рядкових функцій**, які знаходяться в бібліотеці рядків
- Ці **функції** вже вбудовано в кожен рядок — ми викликаємо їх, додаючи функцію до **рядкової змінної**
- Ці **функції** не змінюють вихідний рядок, натомість повертають новий рядок, який було змінено

```
>>> greet = 'Hello Bob'
>>> zap = greet.lower()
>>> print(zap)
hello bob
>>> print(greet)
Hello Bob
>>> print('Hi There'.lower())
hi there
>>>
```

```
>>> stuff = 'Hello world'
>>> type(stuff)
<class 'str'>
>>> dir(stuff)
[...'capitalize', 'casefold', 'center', 'count', 'encode',
'endswith', 'expandtabs', 'find', 'format', 'format_map',
'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace',
'istitle', 'isupper', 'join', 'ljust', 'lower', 'lstrip',
'maketrans', 'partition', 'replace', 'rfind', 'rindex', 'rjust',
'rstrip', 'rsplit', 'rstrip', 'split', 'splitlines',
'startswith', 'strip', 'swapcase', 'title', 'translate', 'upper',
'zfill']
```

<https://docs.python.org/3/library/stdtypes.html#string-methods>

**str.replace(*old*, *new*[, *count*])**

Return a copy of the string with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

**str.rfind(*sub*[, *start*[, *end*]])**

Return the highest index in the string where substring *sub* is found, such that *sub* is contained within *s*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation. Return `-1` on failure.

**str.rindex(*sub*[, *start*[, *end*]])**

Like `rfind()` but raises `ValueError` when the substring *sub* is not found.

**str.rjust(*width*[, *fillchar*])**

Return the string right justified in a string of length *width*. Padding is done using the specified *fillchar* (default is an ASCII space). The original string is returned if *width* is less than or equal to `len(s)`.

**str.rpartition(*sep*)**

Split the string at the last occurrence of *sep*, and return a 3-tuple containing the part before the separator, the separator itself, and the part after the separator. If the separator is not found, return a 3-tuple containing two empty strings, followed by the string itself.

**str.rsplit(*sep*=None, *maxsplit*=-1)**

Return a list of the words in the string, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done, the *rightmost* ones. If *sep* is not specified or `None`, any whitespace string is a separator. Except for splitting from the right, `rsplit()` behaves like `split()` which is described in detail below.



# Бібліотека рядків

`str.capitalize()`

`str.center(width[, fillchar])`

`str.endswith(suffix[, start[, end]])`

`str.find(sub[, start[, end]])`

`str.lstrip([chars])`

`str.replace(old, new[, count])`

`str.lower()`

`str.rstrip([chars])`

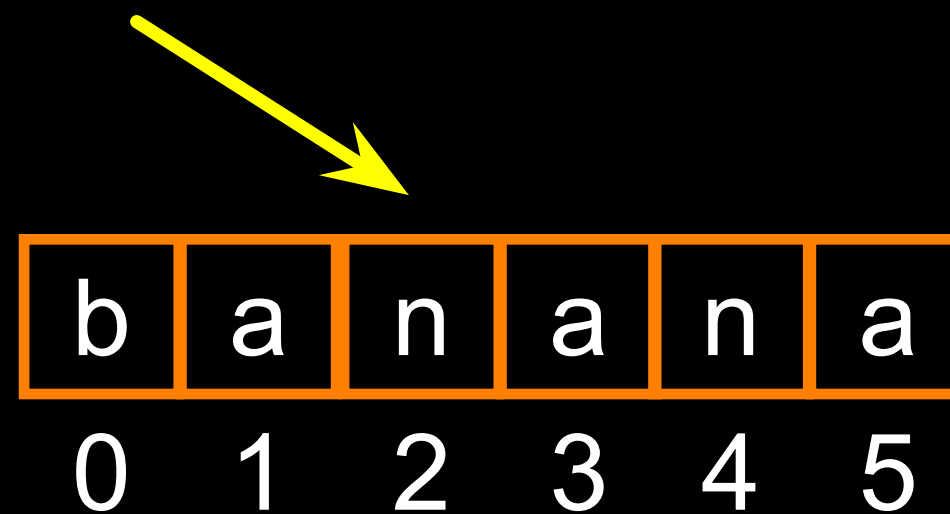
`str.strip([chars])`

`str.upper()`



# Пошук у рядку

- Для пошуку підрядка в іншому рядку використовуємо функцію `find()`
- `find()` знаходить перше входження підрядка
- Якщо підрядок не знайдено, `find()` повертає `-1`
- Пам'ятайте, що перший індекс рядка це нуль



```
>>> fruit = 'banana'
>>> pos = fruit.find('na')
>>> print(pos)
2
>>> aa = fruit.find('z')
>>> print(aa)
-1
```

# Переведення у ВЕРХНІЙ РЕГІСТР

- Ви можете зробити копію рядка в **нижньому** або **верхньому** регістрі
- Часто, коли ми шукаємо рядок за допомогою **find()**, ми спочатку перетворюємо рядок на нижній регістр, щоб можна було шукати рядок незалежно від регістру

```
>>> greet = 'Hello Bob'
>>> nnn = greet.upper()
>>> print(nnn)
HELLO BOB
>>> www = greet.lower()
>>> print(www)
hello bob
>>>
```

# Пошук та заміна

- Функція `replace()`  
схожа на операцію  
«знайти і замінити» у  
текстовому редакторі

- Вона замінює **всі**  
**входження**  
**відшукуваного рядка**  
на **рядок заміни**

```
>>> greet = 'Hello Bob'
>>> nstr = greet.replace('Bob', 'Jane')
>>> print(nstr)
Hello Jane
>>> nstr = greet.replace('o', 'x')
>>> print(nstr)
Hel1x Bxb
>>>
```

# Видалення пробілів

- Іноді ми хочемо взяти рядок і видалити пробіли на початку та / або в кінці
- `lstrip()` та `rstrip()` видаляють пробіли ліворуч або праворуч
- `strip()` видаляє пробіли як на початку, так і в кінці рядка

```
>>> greet = '    Hello Bob    '  
>>> greet.lstrip()  
'Hello Bob '  
>>> greet.rstrip()  
'    Hello Bob'  
>>> greet.strip()  
'Hello Bob'  
>>>
```

# Префікси

```
>>> line = 'Please have a nice day'
```

```
>>> line.startswith('Please')
```

```
True
```

```
>>> line.startswith('p')
```

```
False
```

# Парсинг та виокремлення

21



31



From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008

```
>>> data = 'From stephen.marquard@uct.ac.za Sat Jan 5 09:14:16 2008'
```

```
>>> atpos = data.find('@')
```

```
>>> print(atpos)
```

```
21
```

```
>>> sppos = data.find(' ', atpos)
```

```
>>> print(sppos)
```

```
31
```

```
>>> host = data[atpos+1 : sppos]
```

```
>>> print(host)
```

```
uct.ac.za
```



# Рядки та набори символів

Python 2.7.10

```
>>> x = '이광춘'
>>> type(x)
<type 'str'>
>>> x = u'이광춘'
>>> type(x)
<type 'unicode'>
>>>
```

Python 3.5.1

```
>>> x = '이광춘'
>>> type(x)
<class 'str'>
>>> x = u'이광춘'
>>> type(x)
<class 'str'>
>>>
```

У Python 3 всі рядки в Unicode

# Підсумки

- Типи рядків
- Читання / конвертація
- Індесування рядків []
- Зрізи рядків [2:4]
- Проходження циклами **for** та **while**
- Конкатенація рядків +
- Операції з рядками
- Бібліотека рядків
- Порівняння рядків
- Пошук у рядках
- Заміна тексту
- Прибирання пробілів



# Права власності / застереження



Авторські права на ці слайди з 2010 року належать Чарльзу Северенсу ([www.dr-chuck.com](http://www.dr-chuck.com)) зі Школи інформації Мічиганського університету та застережені ліцензією Creative Commons Attribution 4.0. Будь ласка, збережіть цей фінальний слайд у всіх копіях документа, щоб відповідати вимогам ліцензії щодо посилань на джерела. При повторній публікації матеріалів, якщо щось зміните, додайте ім'я та організацію до переліку співавторів нижче.

Першоджерело: Чарльз Северенс, Школа інформації Мічиганського університету

Переклад: Платформа Prometheus