# Строки

## Ввод и вывод

### хелау ворлд

Возможно использовать одинарные, двойные и тройные кавычки. В последнем примере двойные кавычки находятся внутри одинарных, и поэтому воспринимаются как часть строки.

In [ ]:
```
print('Hello world!')
print("Hello world!")
print('''Hello world!''')
print("""Hello world!""")
print('"Hello world!"')
```

```
Hello world!
Hello world!
Hello world!
Hello world!
"Hello world!"
```

Ввод данных с клавиатуры осуществляется функцией **input()**, которая на выходе даёт строку (тип string).

In [ ]:
```
# input: 123
a = input()
print(a)
print(type(a))
```

```
123
<class 'str'>
```

Зададим три строки: **x**, **y** и **z**

In [ ]:
```
x = '123'
y = '456'
z = '789'
```

Jupyter Notebook умеет выводить значения переменных без фукнции **print()**, однако при этом выведется лишь значение последней переменной

In [ ]:
```
x
y
z
```

Out[ ]:
```
'789'
```

Сами скрипты на python (.py) без команды **print()** не выведут ничего, поэтому необходимо указать команду

In [ ]:
```
print(x)
print(y)
print(z)
```

```
123
456
789
```

---

# Действия со строками

Конкатенация (сложение)

In [ ]:
```
numbers = x+y+z

print(numbers)
```

```
123456789
```

Доступ по индексу. Нумерация "букв" производится с нуля.

In [ ]:
```
print(numbers[0])
print(numbers[1])
print(numbers[2])
print(numbers[3])
```

```
1
2
3
4
```

Дублирование строки

In [ ]:
```
b = x*3
b
```

Out[ ]:
```
'123123123'
```

Извлечение среза.

[ с какого начать : перед каким остановиться : с каким шагом идти ]

In [ ]:
```
numbers[2:4]
```

Out[ ]:
```
'34'
```

Индекс "-1" указывает на последний элемент

In [ ]:
```
numbers[2:-1]
```

Out[ ]:
```
'345678'
```

Шаг "-1" указывает, что движение по строке происходит в обратном порядке

In [ ]:
```
numbers[::-1]
```

Out[ ]:
```
'987654321'
```

Извлекаем срез со второго (на самом деле третьего) элемента до четвёртого с конца элемента включительно, двигаемся с шагом 2

In [ ]:
```
numbers[2:-3:2]
```

Out[ ]:
```
'35'
```

---

## Функции и методы строк

Зададим строку **a**

In [ ]:
```
a = '   4 bbd 00 abb p   '
```

Длина строки получается при использовании фунции len()

In [ ]:
```
len(a)
```

Out[ ]:
```
20
```

Удалить пробелы в начале и в конце строки можно с помощью метода **strip()**.

Выведем саму строку, после чего обновим значение переменной **a**, убрав лишние пробелы и выведем значение обновлённой строки

In [ ]:
```
print(a)
a = a.strip()
print(a)
```

```
   4 bbd 00 abb p
4 bbd 00 abb p
```

Часто в алгоритмических задачках вводятся несколько значений, разделенных пробелами, в этом случае для работы подходит метод **split()**, который возвращает список подстрок

In [ ]:
```
b = a.split(' ')
b
```

Out[ ]:
```
['4', 'bbd', '00', 'abb', 'p']
```

Вообще говоря разделитель может быть произвольным

In [ ]:
```
a.split('00')
```

Out[ ]:
```
['4 bbd ', ' abb p']
```

Существует обратный метод, который принимает список **b** и возвращает строку, с указанными разделителями

In [ ]:
```
print('_'.join(b))
```

```
4_bbd_00_abb_p
```

Опять-таки разделитель может быть произвольным

In [ ]:
```
print('_avadakedavra_'.join(b))
```

```
4_avadakedavra_bbd_avadakedavra_00_avadakedavra_abb_avadakedavra_p
```

Существует группа методов для работы со строками, как с текстом. Например, **title()** меняет регистр у первых букв "слов" (подстрок, разделенных пробелами).

**upper()** и **lowwer()** меняют регистр у всех символов на высокий и низкий соответственно

In [ ]:
```
print(a.title())
print(a.upper())
```

```
4 Bbd 00 Abb P
4 BBD 00 ABB P
```

Часто в задачах требуется работа со строками фиксированной длины. Тут на помощь придут методы, позволяющие дополнить строку до нужной длины нулями или иными символами до нужной длины.

**zfill()** - дополнить нулями

**ljust()** - дополнить символами слева

**rjust()** - дополнить симполами справа

In [ ]:
```
print(a)
print(a.zfill(20))
print(a.ljust(20, '8'))
print(a.rjust(20, '7'))
```

```
4 bbd 00 abb p
0000004 bbd 00 abb p
4 bbd 00 abb p888888
7777774 bbd 00 abb p
```

Поиск индекса подстроки.

**find()** - выводит индекс первой подходящей подстроки, при движении слева направо

**rfind()** - выводит индекс первой подходящей подстроки, при движении справа налево

In [ ]:
```
print(a.find('bb'))
print(a.rfind('bb'))
```

```
2
10
```

Существуют методы **index()** и **rindex()**, примерно того же функционала

In [ ]:
```
print(a.index('bb'))
print(a.rindex('bb'))
```

```
2
10
```

Разница между ними видна при попытке найти несуществующую подсроку.

**find()** выводит "-1"

**index()** выводит ошибку

In [ ]:
```
print(a.find('123'))
print(a.index('123'))
```

```
-1
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[24], line 2
      1 print(a.find('123'))
----> 2 print(a.index('123'))

ValueError: substring not found
```

Метод **replace()** позволяет заменить подстроку на другую

In [ ]:
```
a.replace('00', 'zerozero')
```

Out[ ]:
```
'4 bbd zerozero abb p'
```

---

# Задание

На примере текста "The Zen of Python, by Tim Peters".

Написать программу, которая для данного текста выводила:

1. Количество предложений.
2. Число слов в каждом предложении.
3. Предложения в обратном порядке.
4. Текст, в котором "is better then" заменено на "luchshe".
5. Число букв в тексте.