



Стандартные функции

методы строк
дата и время


Работа с текстом

954



ΠΛ. 11, ΚΒ. 1656






Работа со строками — неотъемлемая часть создания практически любого приложения, где используется текст, и язык программирования Python предоставляет широкие возможности работы с такими данными.

Форматирование строковых значений

I Способ основан на модели printf языка C

```
>>> name = "Alex"
>>> 'Hello, %s' % name
'Привет, Alex'
```

1. '%d', '%i', '%u' — десятичное число;
2. '%c' — символ, точнее строка из одного символа или число — код символа;
3. '%r' — строка (литерал Python);
4. '%s' — строка.




В подстановки используется несколько аргументов, в правой части будет кортеж со строками:

```
>>> '%d %s, %d %s' % (6, 'bananas', 10, 'lemons')  
'6 bananas, 10 lemons'
```


Префикс `r` обозначает неформатируемые (или необрабатываемые) строки, в которых подавляется действие символов экранирования. Такие строки очень удобны, для хранения путей к файлам в Windows, например:

```
str = r'D:\мои документы\книги\Лутц.pdf'  
>>> 'Путь к файлу: %r' % str
```



Позиционирование аргументов по имени. В правой части будет словарь с ключами:

```
>>> '%(count_)d %(fruit)s' % {'fruit':'bananas',  
    'count_':100}
```




II Способ совпадает с выражениями форматирования строк
`str.format()`

```
>>> name = "Alex"
>>> 'Hello, {}'.format(name)
'Привет, Alex'
```

Позиция подстановки может быть изменена

```
>>> name = "Alex"
>>> 'Hello, {0} {1} {2}'.format(name, "Bob", "Lulu")
'Привет, Alex'
```



III способ форматирования строк появился в Python 3.6
f-строки

```
>>> name = "Alex"  
>>> f'Hello, {name}'  
'Привет, Alex'
```

Также можно указать тип при подстановке. Литералы букв аналогичны способу 1.

```
>>> name = "Alex"  
>>> age = 16  
>>> f'Hello, {name:#s age:16 }'
```

Способ допускает возможность встраивать выражения

```
x = y = 5  
F'Сумма чисел x и y:, {x + y:#d}'
```

```
x = y = 5.4  
F'Сумма чисел x и y:, {x + y:#f}'
```




Функции для работы со строками

`str(n)` — преобразование числового
или другого типа к строке;

`len(s)` — длина строки;

`chr(s)` — получение символа по его
коду ASCII;

`ord(s)` — получение кода ASCII по символу;



Методы для работы со строками

find(s, start, end) — возвращает индекс первого вхождения подстроки в s или -1 при отсутствии. Поиск идет в границах от start до end;

rfind(s, start, end) — аналогично, но возвращает индекс последнего вхождения;


replace(s, new) — меняет последовательность символов s на новую подстроку new;

split(x) — разбивает строку на подстроки при помощи выбранного разделителя x;

join(x) — соединяет строки в одну при помощи выбранного разделителя x;

strip(s) — убирает пробелы с обеих сторон;

lstrip(s), **rstrip**(s) — убирает пробелы только слева или справа;



lower() — перевод всех символов в нижний регистр;

upper() — перевод всех символов в верхний регистр;

capitalize() — перевод первой буквы в верхний регистр, остальных — в нижний.

isdigit() — состоит ли строка из цифр

isalpha() — состоит ли строка из букв

isalnum() — состоит ли строка из цифр или букв



`find(s, start, end)`

Метод `str.find()` возвращает индекс первого совпадения подстроки `sub` в строке `str`, где подстрока или символ `sub` находится в пределах среза `str[start:end]`.

```
>>> x = 'раз два три раз два три раз'
```

```
>>> x.find('раз')
```

```
# 0
```

```
>>> x.find('раз', 10, 23)
```

```
# 12
```

```
>>> x.find('раз', -12)
```

```
# 24
```

```
>>> x = 'раз два три раз два три раз'
```

```
>>> x.find('четыре')
```

```
# -1
```



`rfind(s, start, end)`

```
>>> txt = "Mi casa, su casa."
```

```
>>> x = txt.rfind("casa")
```

```
>>> print(x)
```

```
12
```

```
txt = "Hello, welcome to my world."
```

```
x = txt.rfind("e", 5, 10)
```

```
print(x)
```

```
8
```



`index(s, start, end)`

Метод выдает индекс первого вхождения.

```
txt = "Hello, welcome to my world."
```

```
x = txt.index("welcome")
```

```
print(x)
```

```
# В отличии от find выдаст ошибку
```

```
txt = "Hello, welcome to my world."
```

```
x = txt.index("goodbay")
```

```
print(x)
```

ValueError: substring not found



`replace(oldvalue, newvalue, count)`

Параметры:

`oldvalue` – строка для поиска

`newvalue` – строка замены

`count` – сколько вхождений заменить, по умолчанию `all()`

```
txt = "I like bananas"
```

```
x = txt.replace("bananas", "apples")
```

```
print(x)
```

Могу ли я сделать так ?

```
txt[0] = "Y"
```



`replace(oldvalue, newvalue, count)`

Что произойдет ?

```
txt = "I like bananas"  
x = txt.replace("anas", "apples")  
print(x)
```

```
'I like banapples'
```




`split(separator, maxsplit)`

Параметры:

`separator` – разделитель используемый для разбивки

`maxsplit` – определяет кол-во операций разделений.
По умолчанию все вхождения.

```
txt = "hello, my name is Peter, I am 26 years old"
```

```
x = txt.split(", ")
```

```
print(x)
```

```
['hello', 'my name is Peter', 'I am 26 years old']
```



split(separator, maxsplit)

```
txt = "apple#banana#cherry#orange"
```

```
x = txt.split("#", 1)
```

```
print(x)
```

```
['apple', 'banana#cherry#orange']
```



join(iterable)

```
myTuple = ("John", "Peter", "Vicky")
```

```
x = "#".join(myTuple)
```

```
print(x)
```

```
John#Peter#Vicky
```

```
myDict = {"name": "John", "country": "Norway"}
```

```
mySeparator = "_"
```

```
x = mySeparator.join(myDict)
```

```
print(x)
```

```
'name_country'
```



`strip`(characters)

Параметры:

Characters – опциональный, устанавливает символы для удаления из текста

удаление пробелов

```
>>> text = "  test  "
```

```
>>> text.strip()
```

```
'test'
```

```
txt = ",,,,rrttgg.....banana.....rrr"
```

```
x = txt.strip(",.grt")
```

```
print(x)
```

```
banana
```



`lstrip(characters)` `rstrip(characters)`

Параметры:

`characters` – опциональный, устанавливает символы для удаления из текста

удаление пробелов слева

```
>>> text = "...test..."
```

```
>>> text.lstrip(".")
```

```
'test...'
```

```
>>> text = "...test..."
```

```
>>> text.rstrip(".")
```

```
'...test'
```



lower() , upper()

Параметры:

```
txt = "Hello my FRIENDS"
```

```
x = txt.lower()
```

```
print(x)
```

```
hello my friends
```

```
txt = "Hello my friends"
```

```
x = txt.upper()
```

```
print(x)
```

```
HELLO MY FRIENDS
```



capitalize()

Параметры:

```
txt = "python is FUN!"  
x = txt.capitalize()  
print (x)  
Python is fun!
```



`isdigit()`, `isalpha()`, `isalnum()`

Параметры:

```
block = "1024"
```

```
block.isdigit()
```

True

```
era = "XXI Centure"
```

```
era.isalpha()
```

False

```
era = "2022Centure"
```

```
era.isalnum()
```

True



Преобразование строки в дату.

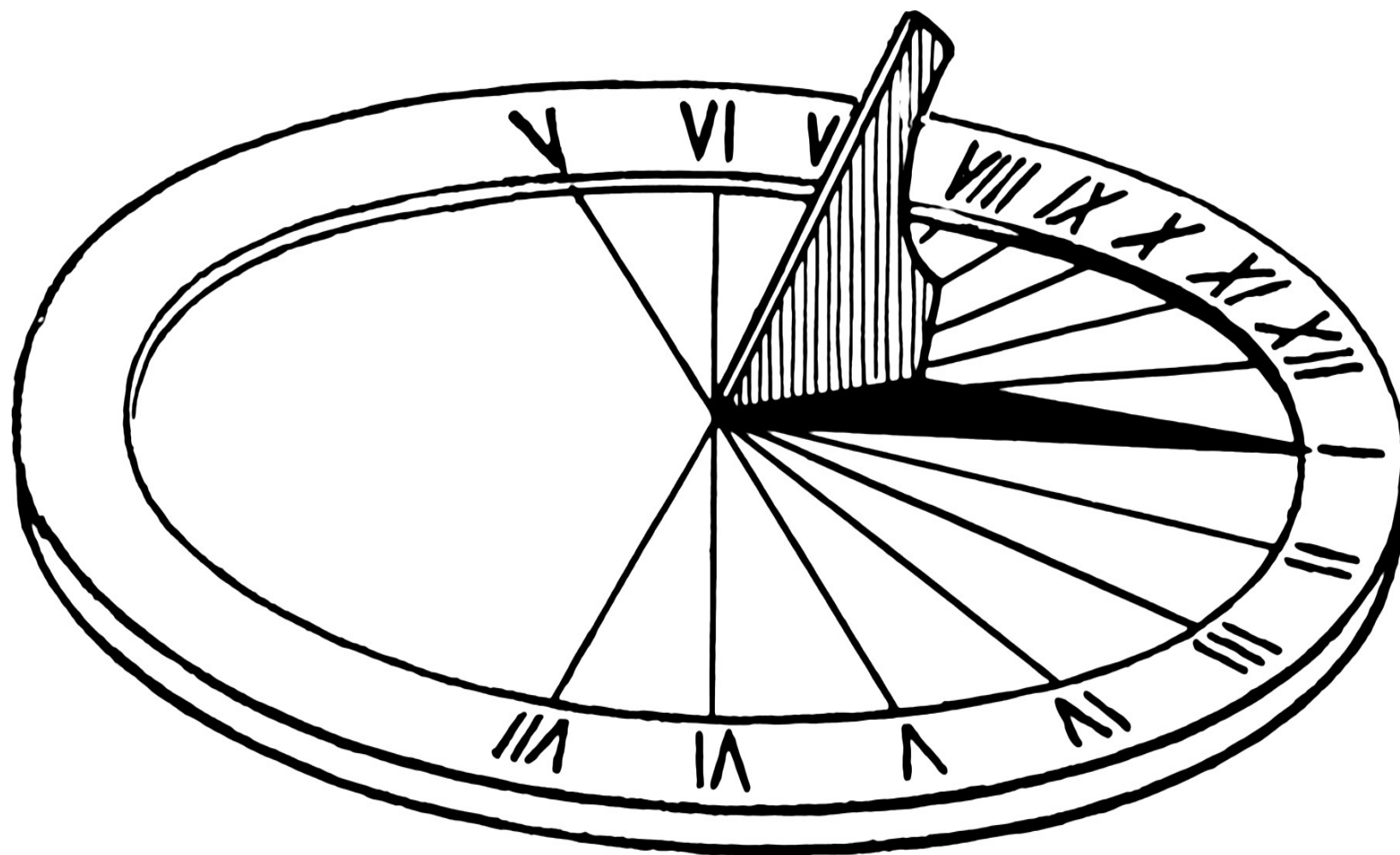
```
# Воспользуемся библиотекой datetime методом strptime
```

```
>>> from datetime import datetime
```

```
>>> print(datetime.strptime('22 04 2020 19:33', '%d  
%m %Y %H:%M'))
```

```
2020-04-22 19:33:00
```

Работа с датой и временем





Получение текущей даты и времени.

Одним из классов, определенных в модуле `datetime`, является класс `datetime`. После импортирования класса мы использовали метод `now()` для создания объекта `datetime`, содержащего текущие локальные дату и время.

```
from datetime import datetime  
  
datetime_object = datetime.now()  
  
print(datetime_object)  
  
2021-03-22 12:04:13.536031
```



Создание даты и времени

```
from datetime import datetime  
a = datetime(2017, 11, 28, 23, 55, 59, 342380)  
print("year =", a.year)  
print("month =", a.month)  
print("hour =", a.hour)  
print("minute =", a.minute)  
print("timestamp =", a.timestamp())
```



Получение текущей даты.

В этой программе мы использовали метод `today()`, определенный в классе `date`, чтобы получить объект `date`, содержащий текущую локальную дату.

```
from datetime import date  
  
today = date.today()  
  
print("Current date =", today)  
  
2022-04-22
```



Конструирование даты

```
import datetime

dt = datetime.date(2020, 6, 29)

print(dt)

2020-06-29

# получение значений

print("Current year:", dt.year)
print("Current month:", dt.month)
print("Current day:", dt.day)
```



Получение даты из метки времени (timestamp).

Термин `timestamp` употребляется для обозначения POSIX-времени — количества секунд, прошедшего с 00:00:00 UTC 1 января, 1970 года. Вы можете преобразовать метку времени в дату при помощи метода `fromtimestamp()`

Создание метки:

```
import time  
  
from datetime import date  
  
now = time.time()
```

Конструирование объекта времени `date`

```
timestamp = date.fromtimestamp(now)
```

```
print("Date =", timestamp)
```

```
2022, 4, 22
```



Форматирование даты

```
from datetime import datetime
```

```
now = datetime.now()
```

```
t = now.strftime ("%H:%M:%S")
```

```
print("time:", t)
```

```
time: 15:00:24
```

```
s1 = now.strftime ("%m/%d/%Y, %H:%M:%S")
```


```
print("s1:", s1)
```

```
s1: 04/22/2022, 15:00:24
```

```
s2 = now.strftime ("%d/%m/%Y, %H:%M:%S")
```

```
# dd/mm/YY H:M:S format
```

```
print("s2:", s2)
```

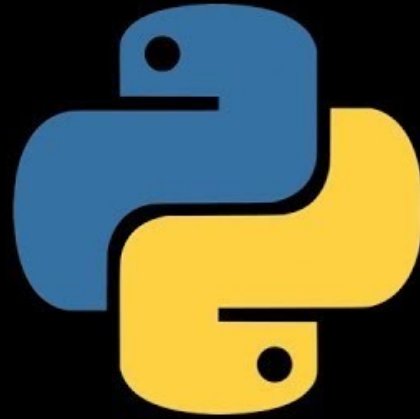



Основные коды для определения формата:

- %Y — год [0001, ..., 2018, 2019, ..., 9999]
- %m — месяц [01, 02, ..., 11, 12]
- %d — день [01, 02, ..., 30, 31]
- %H — час [00, 01, ..., 22, 23]
- %M — минута [00, 01, ..., 58, 59]
- %S — секунда [00, 01, ..., 58, 59]



Спасибо за внимание!



PYTHON

PROGRAMMING