

[КАК СТАТЬ АВТОРОМ](#)[Поехали в гик-трип по Календарю IT-ивентов?](#)**Grigoriy_Alekseev**

4 мая 2019 в 06:07

Введение в Python

🕒 12 мин 👁 85K

Python*, Машинное обучение*

В данной статье мы затронем основы Python. Мы все ближе и ближе к цели, в общем, скоро приступим к работе с основными библиотеками для Data Science и будем использовать TensorFlow (для написания и развертывания нейросетей, тобишь Deep Learning).

Установка

Python можно скачать с python.org. Однако если он еще не установлен, то вместо него рекомендую дистрибутивный пакет Anaconda, который уже включает в себя большинство библиотек, необходимых для работы в области науки о данных.

Если вы не используете дистрибутив Anaconda, то не забудьте установить менеджер пакетов `pip`, позволяющий легко устанавливать сторонние пакеты, поскольку некоторые из них нам понадобятся. Стоит также установить намного более удобную для работы интерактивную оболочку IPython. Следует учитывать, что дистрибутив Anaconda идет вместе с `pip` и IPython.

Пробельные символы

Во многих языках программирования для разграничения блоков кода используются фигурные скобки. В Python используются отступы:

```
# пример отступов во вложенных циклах for
for i in [ 1, 2, 3, 4, 5 ] :
    print (i) # первая строка в блоке for i
    for j in (1, 2, 3, 4, 5 ] :
        print ( j ) # первая строка в блоке for j
        print (i + j) # последняя строка в блоке for j
    print (i) # последняя строка в блоке for i
    print ( "циклы закончились ")
```



форматированием. Пробел внутри круглых и квадратных скобок игнорируется, что облегчает написание многословных выражений:

```
# пример многословного выражения
long_winded_computation = (1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 +
11 + 12 + 13 + 14 + 15 + 16 + 17 + 18 + 19 + 20)
```

и легко читаемого кода:

```
# список списков
list_of_lists = [ [ 1 , 2, 3 ) , [4, 5, 6 ] , [ 7 , 8, 9 ] ]
# такой список списков легче читается
easy_to_read_list_of_lists = [1, 2, 3 ) ,
                             [4, 5, 6 ) ,
                             [7, 8, 9 ) ]
```

Для продолжения оператора на следующей строке используется обратная косая черта, впрочем, такая запись будет применяться редко:

```
two_plus_three = 2 + \
3
```

В следствие форматирования кода пробельными символами возникают трудности при копировании и вставке кода в оболочку Python. Например, попытка скопировать следующий код:

```
for i in [ 1, 2, 3, 4, 5] :
# обратите внимание на пустую строку
print (1)
```

в стандартную оболочку Python вызовет ошибку:

```
# Ошибка нарушения отступа : ожидается блок с отступом
IndentationError : expected an indented block
```

потому что для интерпретатора пустая строка свидетельствует об окончании блока кода с циклом `for`.

Оболочка IPython располагает «волшебной» функцией `%paste`, которая правильно вставляет все то, что находится в буфере обмена, включая пробельные символы.

Модули (Импортирование библиотек)

Некоторые библиотеки среды программирования на основе Python не загружаются по умолчанию. Для того чтобы эти инструменты можно было использовать, необходимо импортировать модули, которые их содержат.

Один из подходов заключается в том, чтобы просто импортировать сам модуль:

```
import re
my_regex = re.compile("[0-9]+",re.I)
```

Здесь *re* – это название модуля, содержащего функции и константы для работы с регулярными выражениями. Импортировав таким способом весь модуль, можно обращаться к функциям, предваряя их префиксом *re*.

Если в коде переменная с именем *re* уже есть, то можно воспользоваться псевдонимом модуля:

```
import re as regex
my_regex = regex.compile("[0-9]+",regex.I)
```

Псевдоним используют также в тех случаях, когда импортируемый модуль имеет громоздкое имя или когда в коде происходит частое обращение к модулю.

Например, при визуализации данных на основе модуля *matplotlib* для него обычно используют следующий стандартный псевдоним:

```
import matplotlib.pyplot as plt
```

Если из модуля нужно получить несколько конкретных значений, то их можно импортировать в явном виде и использовать без ограничений:

```
from collections import defaultdict, Counter
lookup = defaultdict(int)
my_counter = Counter()
```

Функции

Функция – это правило, принимающее ноль или несколько входящих аргументов и возвращающее соответствующий результат. В Python функции обычно определяются при помощи оператора `def`:

```
def double(x) :
    """здесь, когда нужно, размещают
    многострочный документирующий комментарий docstring,
    который поясняет, что именно функция вычисляет.
    Например, данная функция умножает входящее значение на 2"""
    return x * 2
```

Функции в Python рассматриваются как объекты первого класса. Это означает, что их можно присваивать переменным и передавать в другие функции так же, как любые другие аргументы:

```
# применить функцию f к единице
def apply_to_one(f):
    """вызывает функцию f с единицей в качестве аргумента """
    return f(1)
my_double = double # ссылка на ранее определенную функцию
x = apply_to_one(my_double) # = 2
```

Кроме того, можно легко создавать короткие анонимные функции или лямбда выражения:

```
y = apply_to_one(lambda x: x + 4) # = 5
```

Лямбда-выражения можно присваивать переменным. Однако рекомендуют пользоваться оператором `def`:

```
another_double = lambda x: 2 * x # так не делать
```

```
def another_double (x) : return 2 * x # лучше так
```

Параметрам функции, помимо этого, можно передавать аргументы по умолчанию, которые следует указывать только тогда, когда ожидается значение, отличающееся от значения по умолчанию:

```
def my_print (message="мое сообщение по умолчанию" ):  
print (message )  
my_print ( "привет") # напечатает 'привет'  
my_print () # напечатает 'мое сообщение по умолчанию'
```

Иногда целесообразно указывать аргументы по имени:

```
# функция вычитания  
def subtract ( a=0, b=0 ) :  
return a - b  
subtract (10, 5)# возвращает 5  
subtract (0, 5)# возвращает -5  
subtract (b=5 )# то же, что и в предыдущем примере
```

В дальнейшем функции будут использоваться очень часто.

Строки

Символьные строки (или последовательности символов) с обеих сторон ограничиваются одинарными или двойными кавычками (они должны совпадать):

```
single_quoted_string = ' наука о данных ' # одинарные  
double_quoted_string = "наука о данных" # двойные
```

Обратная косая черта используется для кодирования специальных символов. Например:

```
tab_string = "\t" # обозначает символ табуляции  
len (tab_string)# = 1
```

Если требуется непосредственно сама обратная косая черта, которая встречается в именах каталогов в операционной системе Windows, то при помощи г '""' можно создать

неформатированную строку:

```
not_tab_string = r"\t" # обозначает символы ' \ ' и ' t '  
len(not_tab_string) # = 2
```

Многострочные блоки текста создаются при помощи тройных одинарных (или двойных) кавычек:

```
multi_line_string = """Это первая строка .  
это вторая строка  
а это третья строка """
```

Исключения

Когда что-то идет не так, Python вызывает исключение. Необработанные исключения приводят к непредвиденной остановке программы. Исключения обрабатываются при помощи операторов *try* и *except*:

```
try:  
    print(0 / 0)  
except ZeroDivisionError :  
    print("нельзя делить на ноль")
```

Хотя во многих языках программирования использование исключений считается плохим стилем программирования, в Python нет ничего страшного, если он используется с целью сделать код чище, и мы будем иногда поступать именно так.

Списки

Наверное, наиважнейшей структурой данных в Python является список. Это просто упорядоченная совокупность (или коллекция), похожая на массив в других языках программирования, но с дополнительными функциональными возможностями.

```
integer_list = [1, 2, 3] # список целых чисел  
heterogeneous_list = ["строка", 0.1, True] # разнородный список  
list_of_lists = [integer_list, heterogeneous_list, []] # список списков
```

```
list_length = len(integer_list) #длина списка = 3
list_sum = sum(integer_list)#сумма значений в списке = 6
```

Устанавливать значение и получать доступ к n-му элементу списка можно при помощи квадратных скобок:

```
x = list(range (10)) # задает список {0, 1 , . . . , 9]
zero = x [0] # = 0 , списки нуль-индексные, т. е . индекс 1-го элемента = 0
one = x [1] # = 1
nine = x [-1] # = 9, по-питоновски взять последний элемент
eight = x [-2] # = 8, по-питоновски взять предпоследний элемент
x [0] = -1 # теперь x = { - 1 , 1 , 2, 3, . . . , 9]
```

Помимо этого, квадратные скобки применяются для «нарезки» списков:

```
first_three = x[:3] # первые три = [-1 , 1, 2]
three_to_end = x[3:] #с третьего до конца = {3, 4, ... , 9]
one_to_four = x[1:5] # с первого по четвертый = {1 , 2, 3, 4]
last_three = x[-3:] # последние три = { 7, 8, 9]
without_first_and_last = x[1:-1] # без первого и последнего = {1 , 2, ... , 8]
copy_ of _x = x[:] # копия списка x= [ -1, 1, 2, ... , 9]
```

В Python имеется оператор `in`, который проверяет принадлежность элемента списку:

```
1 in [1, 2, 3] #True
0 in [1, 2, 3] #False
```

Проверка заключается в поочередном просмотре всех элементов, поэтому пользоваться им стоит только тогда, когда точно известно, что список небольшой или неважно, сколько времени уйдет на проверку.

Списки легко сцеплять друг с другом:

```
x = [1, 2, 3]
x. extend ( [ 4, 5, 6] ) # теперь x = {1, 2, 3, 4, 5, 6}
```

Если нужно оставить список `x` без изменений, то можно воспользоваться сложением

СПИСКОВ:

```
x = [1, 2, 3]
y = x + [4, 5, 6] #y= (1, 2, 3, 4, 5, 6) ; x не изменился
```

Обычно к спискам добавляют по одному элементу за одну операцию:

```
x = [1, 2, 3]
x.append(0) # теперь x = [1,2,3,0]
y= x [-1] # = 0
z = len(x) # = 4
```

Нередко бывает удобно распаковать список, если известно, сколько элементов в нем содержится:

```
x, y = [1, 2] # теперь x = 1, y = 2
```

Если с обеих сторон выражения число элементов не одинаково, то будет выдано сообщение об ошибке `ValueError`.

Для отбрасываемого значения обычно используется символ подчеркивания:

```
_, y = [1, 2] # теперь y == 2, первый элемент не нужен
```

Кортежи

Кортежи – это неизменяемые (или иммутабельные) двоюродные братья списков.

Практически все, что можно делать со списком, не внося в него изменения, можно делать и с кортежем. Вместо квадратных скобок кортеж оформляют круглыми скобками, или вообще обходятся без них:

```
my_list = [1, 2] # задать список
my_tuple = (1, 2) # задать кортеж
other_tuple = 3, 4 # еще один кортеж
my_list [1] = 3 # теперь my_list = [1 , 3]
```



```
try:
my_tuple [1] = 3
except TypeError :
print ( "кортеж изменять нель зя" )
```

Кортежи обеспечивают удобный способ для возвращения из функций нескольких значений:

```
# функция возвращает сумму и произведение двух параметров
def sum_and_product (x, y ) :
return (x + y) , (x * y)
sp = sum_and_product (2, 3) # = (5, 6)
s, p = sum_and_product (5, 10) # s = 15, p = 50
```

Кортежи (и списки) также используются во множественном присваивании:

```
x, y = 1, 2 # теперь x = 1, y = 2
x, y = y, x # обмен переменными по-питоновски; теперь x = 2, y = 1
```

Словари

Словарь или ассоциативный список – это еще одна основная структура данных.

В нем значения связаны с ключами, что позволяет быстро извлекать значение, соответствующее конкретному ключу:

```
empty_dict = {} # задать словарь по-питоновски
empty_dict2 = dict () # не совсем по-питоновски
grades = { "Grigoriy" : 80, "Tim" : 95 } # литерал словаря (оценки за экзамены)
```

Доступ к значению по ключу можно получить при помощи квадратных скобок:

```
rigory_aleksee = grades[ "Grigoriy" ] # = 80
```

При попытке запросить значение, которое в словаре отсутствует, будет выдано сообщение об ошибке `KeyError`:

```
try:
    kates_grade = grades [ "Kate " ]
except KeyError:
    print ( "оценки для Кэйт отсутствуют ! " )
```

Проверить наличие ключа можно при помощи оператора `in`:

```
grigoriy_has_grade = "Grigoriy" in grades #true
kate_has_grade = "Kate" in grades #false
```

Словари имеют метод `get()`, который при поиске отсутствующего ключа вместо вызова исключения возвращает значение по умолчанию:

```
grigoriy_grade = grades. get ( "Grigoriy ", 0) # =80
kates_grade = grades.get ("Kate" , 0) # = 0
no_ones_grade = grades.get ( "No One" ) # значение по умолчанию = None
```

Присваивание значения по ключу выполняется при помощи тех же квадратных скобок:

```
grades [ "Tim" ] = 99 # заменяет старое значение
grades [ "Kate" ] = 100 # добавляет третью запись
num_students = len(grades) # = 3
```

Словари часто используются в качестве простого способа представить структурные данные:

```
tweet = {
    "user" : " grinaleks",
    "text" : "Наука о данных - потрясающая тема",
    " retweet_count" : 100,
    "hashtags " : [ "# data", " #science", " #datascience " , " #awesome", "#yolo" ]
```



Помимо поиска отдельных ключей можно обратиться ко всем сразу:

```
tweet_keys = tweet.keys() # список ключей
tweet_values = tweet.values() # список значений
tweet_items = tweet.items() # список кортежей (ключ, значение)

"user" in tweet_keys # True, но использует медленное in списка
"user" in tweet # по-питоновски, использует быстрое in словаря
"grinaleks" in tweet_values # True
```

Ключи должны быть неизменяемыми; в частности, в качестве ключей нельзя использовать списки. Если нужен составной ключ, то лучше воспользоваться кортежем или же найти способ, как преобразовать ключ в строку.

Словарь defaultdict

Пусть в документе необходимо подсчитать слова. Очевидным решением задачи является создание словаря, в котором ключи – это слова, а значения – частотности слов (или количества вхождений слов в текст). Во время проверки слов в случае, если текущее слово уже есть в словаре, то его частотность увеличивается, а если отсутствует, то оно добавляется в словарь:

```
# частотности слов
word_counts = { }
document = { } # некий документ; здесь он пустой
for word in document :
    if word in word_counts:
        word_counts[word] += 1
    else :
        word_counts[word] = 1
```

Кроме этого, можно воспользоваться приемом под названием «лучше просить прощения, чем разрешения» и перехватывать ошибку при попытке обратиться к отсутствующему ключу:

```
word_counts = { }
for word in document :
    try:
        word_counts[word] += 1
    except KeyError :
        word_counts[word] = 1
```

Третий прием – использовать метод `get()`, который изящно выходит из ситуации с отсутствующими ключами:

```
word_counts = { }  
for word in document :  
    previous_count = word_counts.get (word, 0)  
    word_counts [word] = previous_count + 1
```

Все перечисленные приемы немного громоздки, и по этой причине целесообразно использовать словарь `defaultdict` (который еще называют словарем со: значением по умолчанию). Он похож на обычный словарь за исключением одной особенности – при попытке обратиться к ключу, которого в нем нет, он сперва добавляет для него значение, используя функцию без аргументов, которая предоставляется при его создании. Чтобы воспользоваться словарями `defaultdict`, их необходимо импортировать из модуля `collections`:

```
from collections import defaultdict  
  
word_counts = defaultdict(int) # int () возвращает 0  
for word in document :  
    word_counts[word] += 1
```

Кроме того, использование словарей `defaultdict` имеет практическую пользу во время работы со списками, словарями и даже с пользовательскими функциями:

```
dd_list = defaultdict (list)# list () возвращает пустой список  
dd_list [2].append (1) # теперь dd_list содержит (2: {1} )  
dd_dict = defaultdict (dict ) # dict () возвращает пустой словарь dict  
dd_dict ["Grigoriy"] [ "City" ] = "Seattle" # { "Grigoriy" : { "City" : Seattle"  
dd_pair = defaultdict (lambda: [0,0] )  
dd_pair [2][1] = 1 # теперь dd_pair содержит (2 : {0,1} )
```

Эти возможности понадобятся, когда словари будут использоваться для «сбора» результатов по некоторому ключу и когда необходимо избежать повторяющихся проверок на присутствие ключа в словаре.

Словарь Counter

Подкласс словарей `counter` трансформирует последовательность значений в похожий на словарь `defaultdict(int)` объект, где ключам поставлены в соответствие частотности или, выражаясь более точно, ключи отображаются (map) в частотности.

Он в основном будет применяться при создании гистограмм:

```
from collections import Counter
с = Counter([0,1,2,0]) # в результате с = { 0 : 2, 1 : 1, 2 : 1 }
```

Его функционал позволяет достаточно легко решить задачу подсчета частотностей слов:

```
# лучший вариант подсчета частотностей слов
word_counts = Counter (document)
```

Словарь `counter` располагает методом `most_common()`, который нередко бывает полезен:

```
# напечатать 10 наиболее встречаемых слов и их частотность (встречаемость)
for word, count in word_counts.most_common(10) :
    print (word, count )
```

Множества

Структура данных `set` или множество представляет собой совокупность неупорядоченных элементов без повторов:

```
s = set ()# задать пустое множество
s.add (1) # теперь s = { 1 }
s.add (2) # теперь s = { 1, 2 }
s.add (2) # s как и прежде = { 1, 2 }
x = len (s) # = 2
y = 2 in s # = True
z = 3 in s # = False
```

Множества будут использоваться по двум причинам. Во-первых, операция `in` на множествах очень быстрая. Если необходимо проверить большую совокупность элементов на принадлежность некоторой последовательности, то структура данных `set` подходит для

этого лучше, чем список:

```
# список стоп-слов
stopwords_list = [ "a", "an" , "at " ] + hundreds_of_other_words + [ "yet ", " you
" zip" in stopwords_list # False, но проверяется каждый элемент
# множество стоп-слов
stopwords_set = set(stopwords_list)
" zip" in stopwords_set # очень быстрая проверка
```

Вторая причина – получение уникальных элементов в наборе данных:

```
item_list = [1, 2, 3, 1, 2, 3] # список
num_items = len( item_list) # количество = 6
item_set = set(item_list) # вернет множество (1, 2, 3}
num_distinct_items = len(item_set) # число недублирующихся = 3
distinct_item_list = list(item_set) # назад в список = [1,2,3]
```

Множества будут применяться намного реже словарей и списков.

Управляющие конструкции

Как и в большинстве других языков программирования, действия можно выполнять по условию, применяя оператор `if`:

```
if 1 > 2:
message "если 1 была бы больше 2 . . . "
elif 1 > 3:
message "elif означает 'else if '"
else:
message = " когда все предыдущие условия не выполняются, используется else "
```

Кроме того, можно воспользоваться однострочным трехместным оператором `if-then-else`, который будет иногда использоваться в дальнейшем:

```
parity = "четное" if x % 2 == 0 else "нечетное "
```

В Python имеется цикл `while`:

```
x = 0
while x < 10:
    print (x, "меньше 10")
    x += 1
```

Однако чаще будет использоваться цикл for совместно с оператором in:

```
for x in range (10) :
    print (x, "меньше 10" )
51
```

Если требуется более сложная логика управления циклом, то можно воспользоваться операторами

```
continue и break:
for x in range (10) :
    if x == 3:
        continue # перейти сразу к следующей итерации
    if x == 5:
        break
    print (x)
# выйти из цикла
```

В результате будет напечатано 0, 1, 2 и 4.

Истинность

Булевы переменные в Python работают так же, как и в большинстве других языков программирования лишь с одним исключением – они пишутся с заглавной буквы:

```
one_is_less_than_two = 1 < 2 #True
true_equals_false = True == False #False
```

Для обозначения несуществующего значения применяется специальный объект None, который соответствует значению null в других языках:

```
x = None
print (x == None )# напечатает True, но это не по-питоновски
print ( x is None ) # напечатает True по-питоновски
```

В Python может использоваться любое значение там, где ожидается логический тип Boolean. Все следующие элементы имеют логическое значение False:

- False; .
- None;
- set() (множество):
- [] (пустой список);
- {} (пустой словарь);

Практически все остальное рассматривается как True. Это позволяет легко применять операторы if для проверок на наличие пустых списков, пустых строк, пустых словарей и т. д. Иногда, правда, это приводит к труднораспознаваемым ошибкам, если не учитывать следующее:

```
s = some_function_that_returns_a_string () # возвращает строковое значение
if s:
    first_char = s [0] # первый символ в строке
else:
    first char = ""
```

Вот более простой способ сделать то же самое:

```
first_char = s and s [0]
```

поскольку логический оператор and возвращает второе значение, в случае если первое истинное, и первое значение, в случае если оно ложное. Аналогичным образом, если x в следующем ниже выражении является либо числом, либо, возможно, None, то результат так или иначе будет числом:

```
safe x = x or 0 # безопасный способ
```


Встроенная функция `all` языка Python берет список и возвращает `True` только тогда, когда каждый элемент списка истинен, а встроенная функция `any` возвращает `true`, когда истинен хотя бы один элемент:

```
all ( [True, 1, { 3 }]) # True
all ( [True, 1, {}] ) # False, {} = ложное
any ( [ True, 1, {}]) # True, True = истинное
all ( [ ] ) # True, ложные элементы в списке отсутствуют
any ( [ ] ) # False, истинные элементы в списке отсутствуют
```

Теги: Python, machine learning, программирование для начинающих, введение в python

Хабы: Python, Машинное обучение

Редакторский дайджест



Присылаем лучшие статьи раз в месяц

**2**

Карма

0

Рейтинг

Алексеев Григорий @Grigoriy_Alekseev

iOS Developer

Реклама

 Комментарии 98

Публикации

ЛУЧШИЕ ЗА СУТКИ ПОХОЖИЕ

**OlgaPy**

15 часов назад

Капибара, Новый Старый Пикабу

 8 мин  14K +83 48 87

vvvphoenix

18 часов назад

Made at Intel. Окаянные дни – продолжение

 Простой 9 мин 9.9K +68 22 12

Abejorgo

20 часов назад

Примите участие в общественном обсуждении проекта о блокировке информации по обходу блокировок

 2 мин 3.8K +42 9 31

aavezel

23 часа назад

Собеседование в руках маньяков

 Простой 7 мин 11K[Мнение](#) +32 53 24

DAN_SEA

21 час назад

Разработка индивидуальных средств передвижения?

 Простой 9 мин 2.8K[Тutorial](#) +31 13 16

Wladradchenko

21 час назад

Клонирование голоса, замена лица по фото, удаления объектов в видео и все в одном open-source проекте Wunjo AI

 Простой  4 мин  3.6K

Обзор

 +21

 69

 17



Nurked

20 часов назад

Развлекательное чтение на выходные

 Простой  4 мин  4.1K

Дайджест

 +20

 36

 29



OlegSivchenko

13 часов назад

Блеск и нищета гипотезы симуляции

 9 мин  2.6K

 +18

 23

 43



Ilya12c

23 часа назад

Влад Грозин о PhD в США, философии в Data Science, пузыре рекомендаций и голодающих геймерах

 Простой  7 мин  1.7K

Мнение

 +17

 17

 2



JustJeremy

23 часа назад

Опасный гибрид: разгон Super Game Boy

 6 мин  1.7K

Кейс

Перевод

 +17

 10

 0

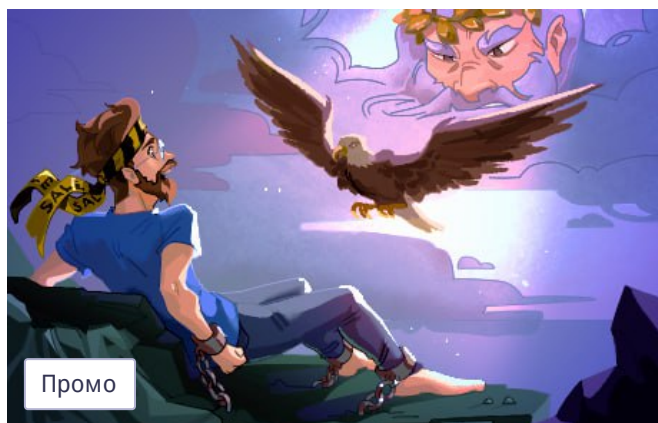
Никакой магии, мы просто берём выпускников курсов и растим из них тестировщиков

[Турбо](#)[Показать еще](#)

МИНУТОЧКУ ВНИМАНИЯ



Как небольшой компании расцвести на Хабре за полгода



Боги приковали Промокодуса за слишком щедрые скидки

ЗАКАЗЫ

Написать Макрос. Сравнение данных из таблиц и выявление ошибок

1000 руб./за проект • 1 отклик • 8 просмотров

Разработать комплексного Телеграмм Бота с JS webarr меню и БД

25000 руб./за проект • 5 откликов • 9 просмотров

Сверстать сайт для строительно-производственной компании

100000 руб./за проект • 32 отклика • 66 просмотров

Сделать анимацию для маркетплейс по примеру

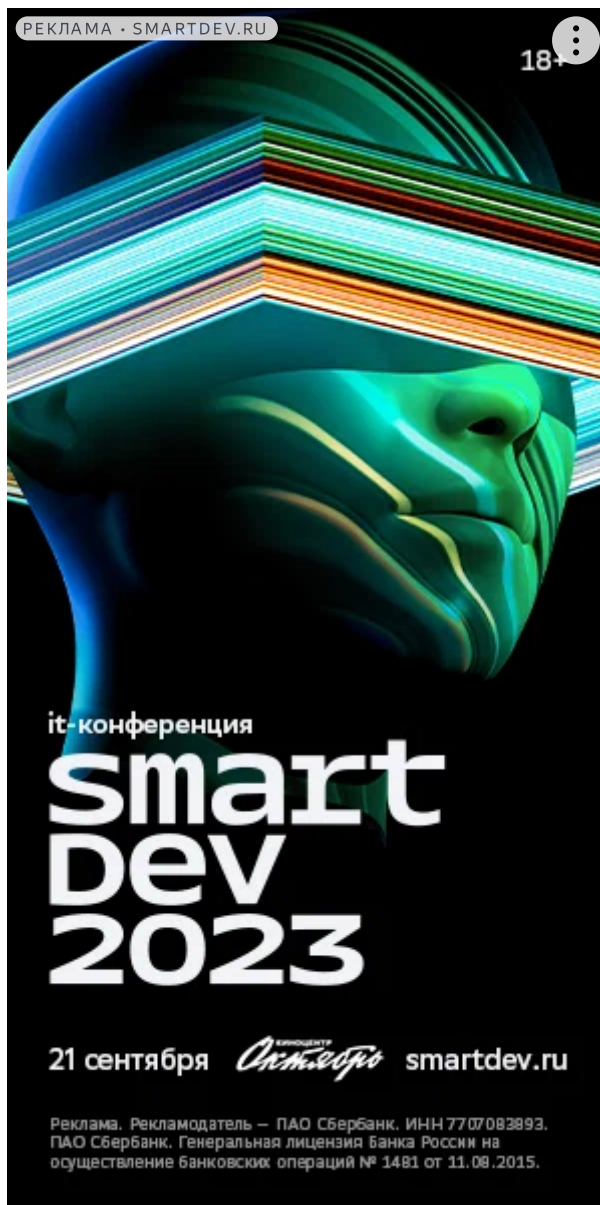
1200 руб./за проект • 5 откликов • 37 просмотров

Парсинг на php сайт БК теннис с сигналом в телеграмм

10000 руб./за проект • 2 отклика • 28 просмотров

Больше заказов на Хабр Фрилансе

Реклама



ЧИТАЮТ СЕЙЧАС

Капибара, Новый Старый Пикабу

👁 14K 💬 87

Разработчики Starfield на просьбу оптимизировать игру посоветовали обновить компьютеры

👁 3.5K 💬 25

Anki — настолько круто, что я даже представить не мог

👁 2.7K 💬 1

Что бесит рекрутеров и соискателей на интервью. Собрала топ-5 триггеров

👁 32K 💬 115

Собеседование в руках маньяков

 11K

 24

Никакой магии, мы просто берём выпускников курсов и растим из них тестировщиков

Турбо

ИСТОРИИ




В ОДНО ОБЫЧНОЕ УТРО СТАНИСЛАВ, ИНЖЕНЕР-РАЗРАБОТЧИК C++, ДУМАЛ НАД ПРОБЛЕМАМИ СВОЕГО ПРОЕКТА

У НАС ВЫСОКОНАГРУЖЕННЫЙ ПРОЕКТ, А ОНИ ПРОШУТ БЫК-ФАЙЛСОВЫЙ, НИ ДОКУМЕНТАЦИИ, НИ АВТОМАТИЗАЦИИ ТЕСТИРОВАНИЯ, НИ АВТОМАТИЗАЦИИ ДЕЛОВ. ПРОШЕ ТАКИЕ ЖЕ МИЛЫ С++ УКАЗАТЕЛИ ДА ПАМЯТЬ ВЕРТЕТЬ ЧАЩЕ! А В КОНЦЕ ЗАКРЫТЬ ПОТОК


IA

Золотая рыбка, хочу, чтоб у меня все было



В аэропорту Внуково открыли метро

6 сентября на Солнцевской линии московского метро открыли две новые станции, одна из которых — «Аэропорт Внуково» — расположена в одноимённом аэропорту. Теперь добраться до Внуково из столицы можно будет прямо на метро.



В аэропорту «Внуково» открыли метро

Хабр Карьера




Где работать в IT: Softline

Поговорили с компанией Softline — ведущим поставщиком решений и сервисов в цифровой трансформации и ИБ




Где работать в IT: Softline



Что почитать

Топ-7 годных статей из блогов компаний



Топ-7 годных статей из блогов компаний

Хабр Карьера • Дайджест

События для эйчаров и рекрутеров в IT в сентябре 2023





Дайджест HR-событий в сентябре

РАБОТА

- Data Scientist

115 вакансий
- Python разработчик

155 вакансий
- Django разработчик

51 вакансия

Все вакансии

Реклама

Войти	Статьи	Устройство сайта	Корпоративный блог
Регистрация	Новости	Для авторов	Медийная реклама
	Хабы	Для компаний	Нативные проекты
	Компании	Документы	Образовательные
	Авторы	Соглашение	программы
	Песочница	Конфиденциальность	Стартапам
			Спецпроекты



Настройка языка

Техническая поддержка

Вернуться на старую версию

© 2006–2023, Habr