avito.tech Москва 2024

Переход с python на go: год спустя

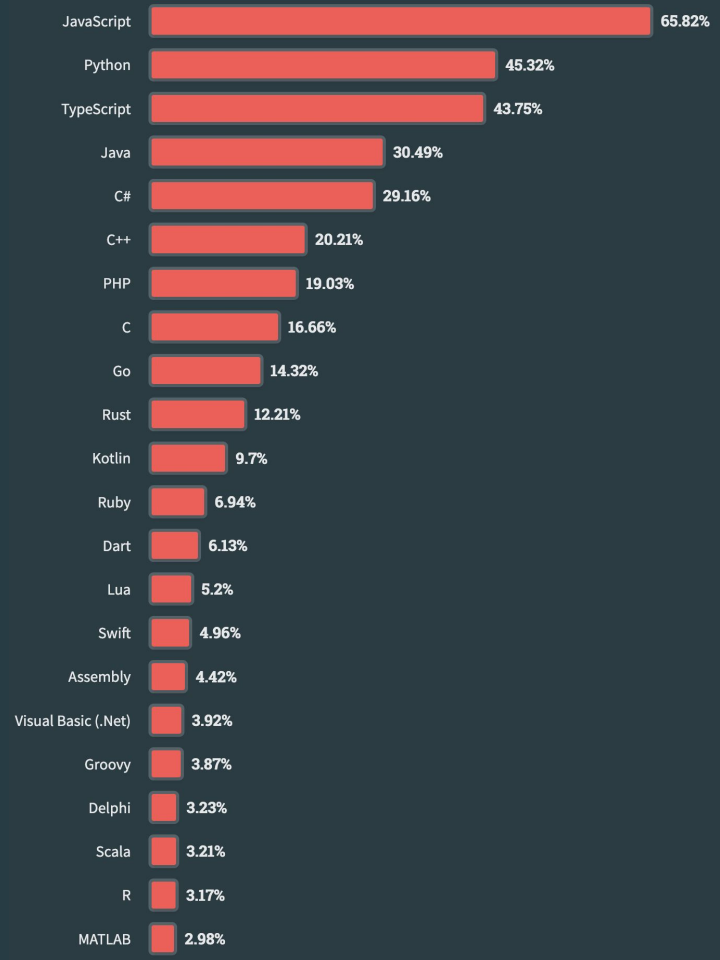
Дмитрий Королев

Дмитрий Королев

Backend-разработчик в Growth

- Software Engineer
- Профессионально занимаюсь разработкой с 2020 года.
- Рад ответить на вопросы tg: @jimiliani.

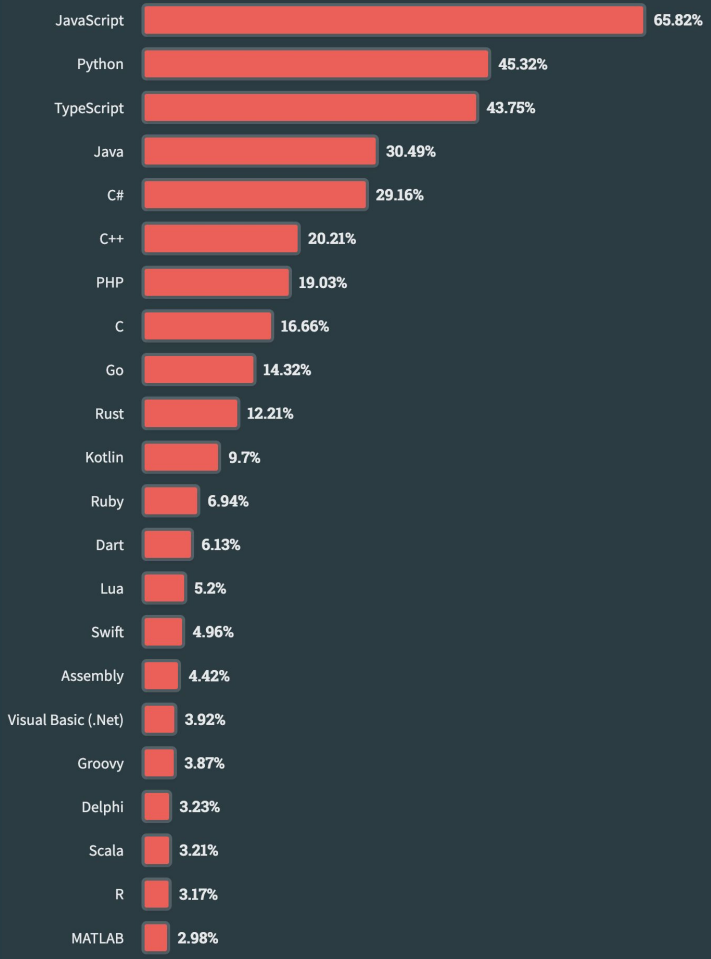
Статистика от StackOverflow



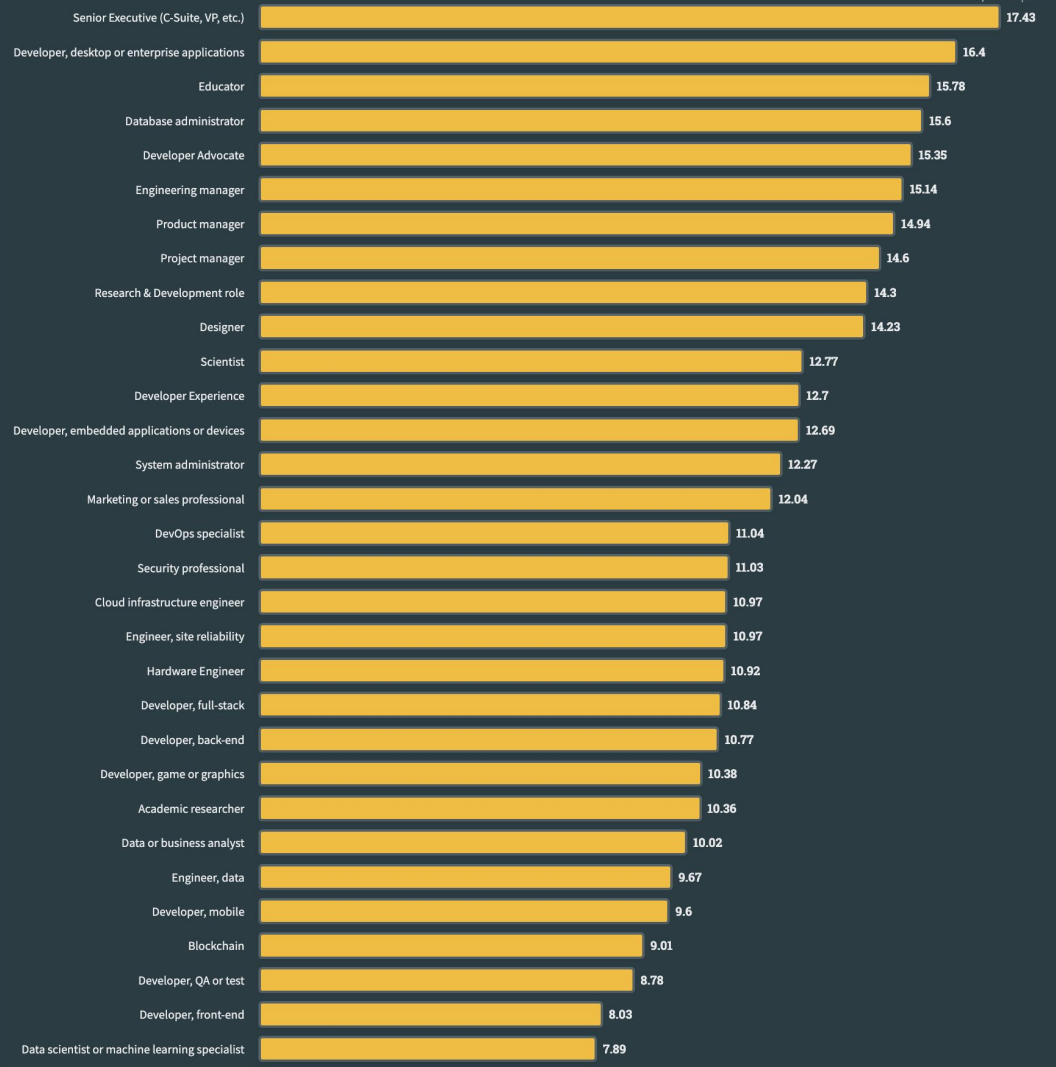
*из статистики исключены языки разметки и SQL

avito.tech

Москва | 2024



66,136 responses



Многие языки похожи

```python
def factorial(n: int):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)


if __name__ == '__main__':
    result = factorial(5)
    print(f"The factorial of 5 is: {result}")
```

```go
func factorial(n int) int {
    if n == 0 || n == 1 {
        return 1
    } else {
        return n * factorial(n-1)
    }
}

func main() {
    result := factorial(5)
    fmt.Printf("The factorial of 5 is: %d", result)
}
```

Многие языки похожи

01 Мы инженеры, а язык это просто инструмент

02

03

Многие языки похожи

01 Мы инженеры, а язык это просто инструмент

02 Язык – это лишь набор концепций

03

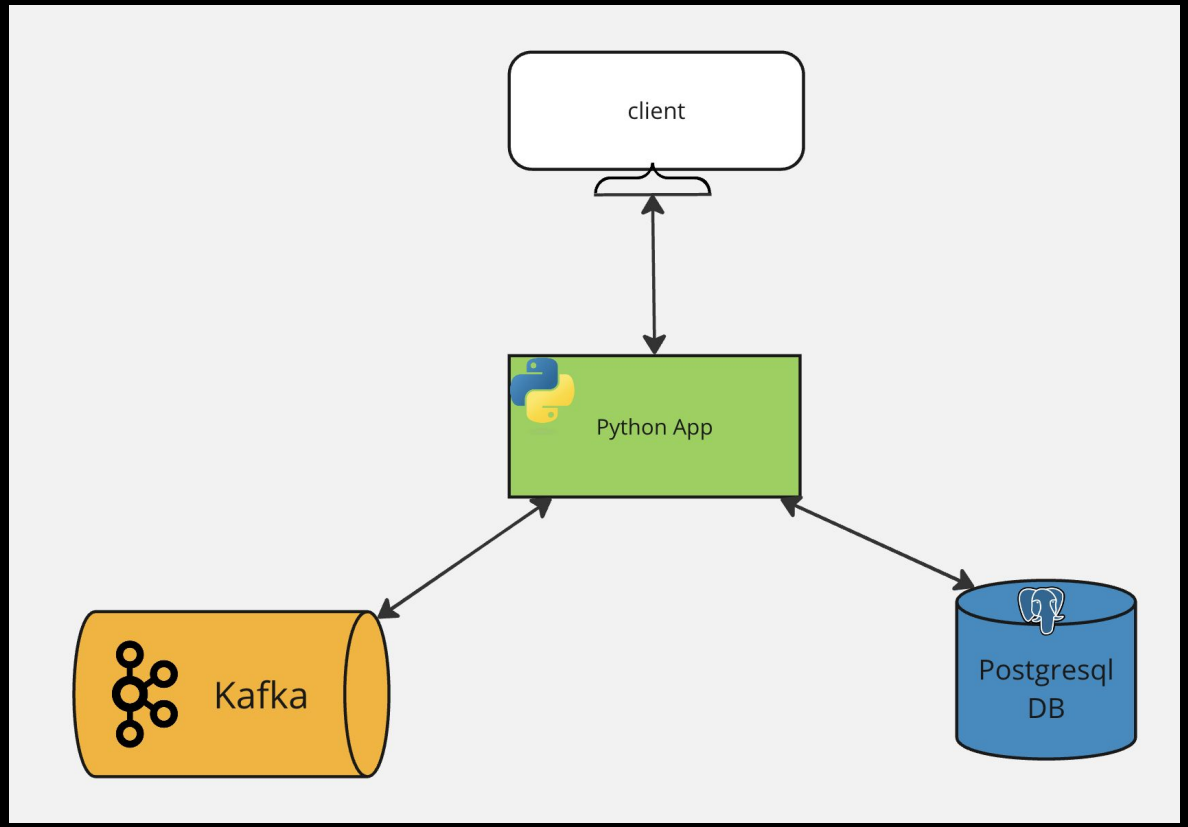
Многие языки похожи

01 Мы инженеры, а язык это просто инструмент

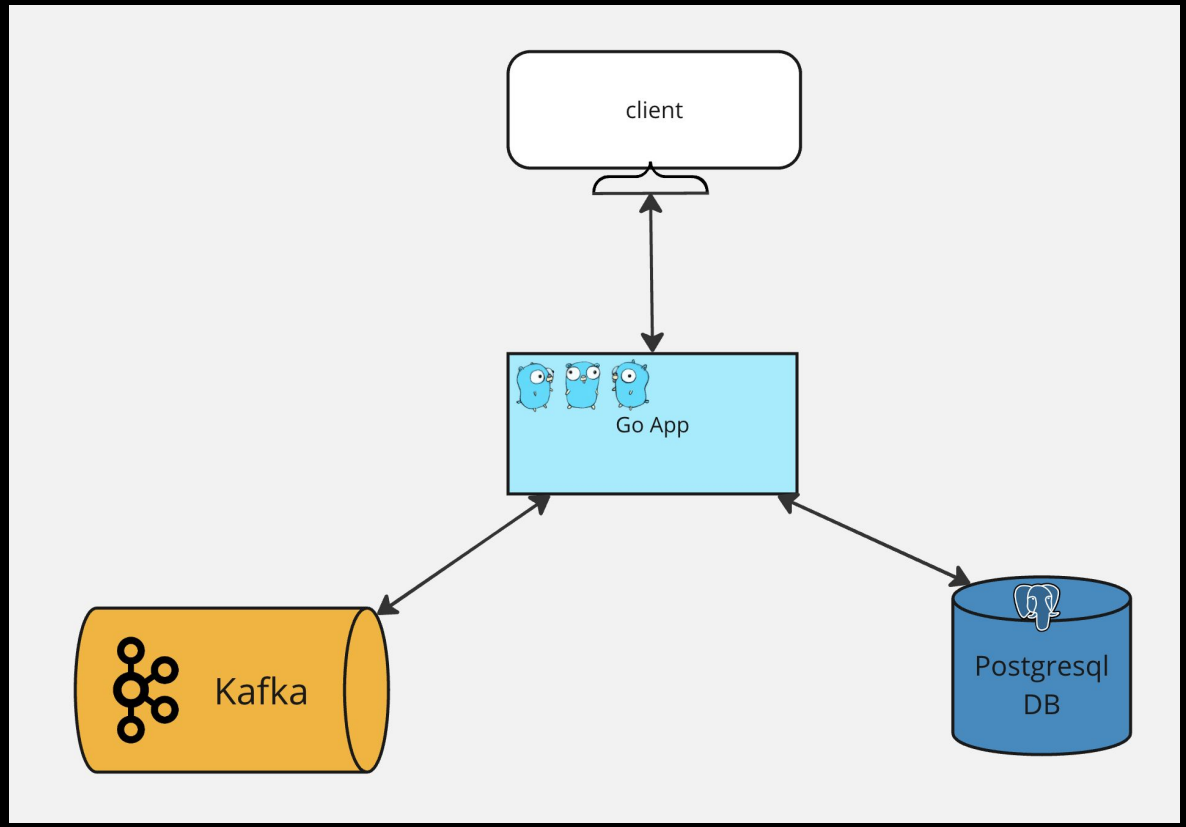
02 Язык – это лишь набор концепций

03 Изучение второго языка программирования – это часто более ускоренный и более гладкий процесс по сравнению с изучением первого

Смена языка ≠ смена технологий



Смена языка ≠ смена технологий



Смена языка ≠ смена инструментов

Новое ≠ плохое

```python
async def send_req(url):
    response = requests.post(url)
    return response.text

async def main():
    url = "https://example.com/"
    with multiprocessing.Pool(processes=multiprocessing.cpu_count()) as pool:
        results = await asyncio.gather(
            *[loop.run_in_executor(pool, send_req, url) for _ in range(1000)]
        )

    for result in results:
        print(result)

if __name__ == "__main__":
    loop = asyncio.get_event_loop()
    loop.run_until_complete(main())
```

```go
func sendRequest(url string, wg *sync.WaitGroup) {
  defer wg.Done()
  resp, err := http.Post(url, "", nil)
  if err != nil {
     fmt.Printf("Error: %v\n", err)
     return
  }
  fmt.Printf("Response from %s: %s\n", url,
resp.Status)
}

func main() {
  url := "https://example.com/"
  var wg sync.WaitGroup

  for range 10000 {
     wg.Add(1)
     go sendRequest(url, &wg)
  }

  wg.Wait()
}
```

Разница в структурах проекта

Organizing a Go module

Table of Contents

project-layout Public

Watch 574 Fork 4.8k Star 44.8k

master 1 Branch 0 Tags Go to file t Add file Code

kcq Update README.md ae58eb1 · last month 297 Commits

api Replace broken link in api folder examples. 4 years ago

About

Standard Go Project Layout

go golang project-template standards project-structure

Опциональные параметры

```go
func SomeFuncWithOptionalParams(ctx context.Context, param1 *int, param2 *string, param3 *bool) {
    // some logic
}

func main() {
    ctx := context.Background()
    SomeFuncWithOptionalParams(ctx, nil, nil, nil)
}
```

Опциональные параметры

Overview

Package context defines the Context type, which carries deadlines, cancellation signals, and other request-scoped values across API boundaries and between processes.

Опциональные параметры

```go
func SomeFuncWithOptionalParams(ctx context.Context) {
    param1 := ctx.Value("param1")
    param2 := ctx.Value("param2")
    param3 := ctx.Value("param3")
    fmt.Println(param1, param2, param3)
}

func main() {
    ctx := context.Background()
    ctx = context.WithValue(ctx, "param1", 1)
    ctx = context.WithValue(ctx, "param1", "val")
    ctx = context.WithValue(ctx, "param1", true)
    SomeFuncWithOptionalParams(ctx)
}
```

Опциональные параметры

Overview

Package context defines the Context type, which carries deadlines, cancellation signals, and other request-scoped values across API boundaries and between processes.

Use context Values only for request-scoped data that transits processes and APIs, not for passing optional parameters to functions.

Опциональные параметры

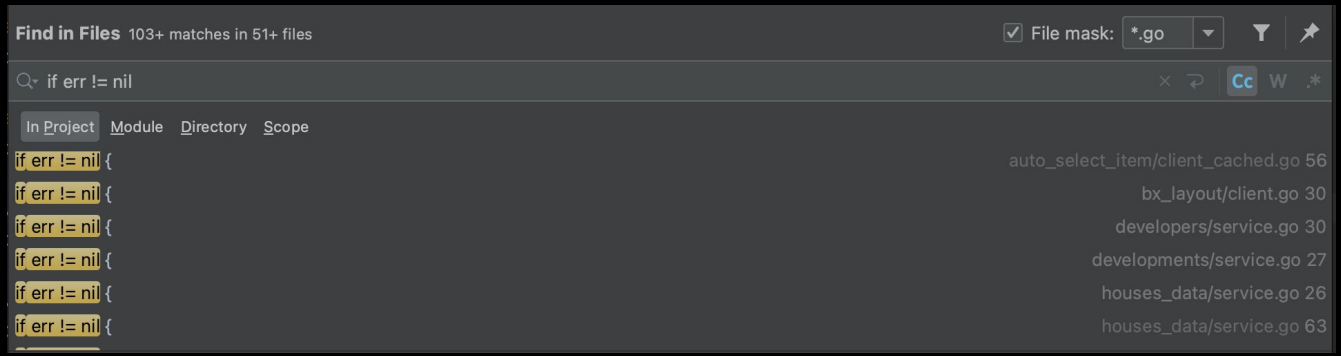
```
type OptionalParams struct {
    param1 *int
    param2 *string
    param3 *bool
}

func SomeFuncWithOptionalParams(ctx context.Context, params OptionalParams) {
    if params.param1 != nil {
        fmt.Println(params.param1)
    }
    if params.param2 != nil {
        fmt.Println(params.param2)
    }
    if params.param3 != nil {
        fmt.Println(params.param3)
    }
}

func main() {
    ctx := context.Background()
    param1 := 1
    param2 := "val"
    param3 := true
    SomeFuncWithOptionalParams(ctx, OptionalParams{})
    SomeFuncWithOptionalParams(ctx, OptionalParams{
        param1: &param1,
        param2: &param2,
        param3: &param3,
    })
}
```

Обработка ошибок

```
func sendRequest(url string, wg *sync.WaitGroup) {
  defer wg.Done()
  resp, err := http.Post(url, "", nil)
  if err != nil {
     fmt.Printf("Error: %v\n", err)
     return
  }
  fmt.Printf("Response from %s: %s\n", url, resp.Status)
}

func main() {
  url := "https://example.com/"
  var wg sync.WaitGroup

  for range 10000 {
     wg.Add(1)
     go sendRequest(url, &wg)
  }

  wg.Wait()
}
```

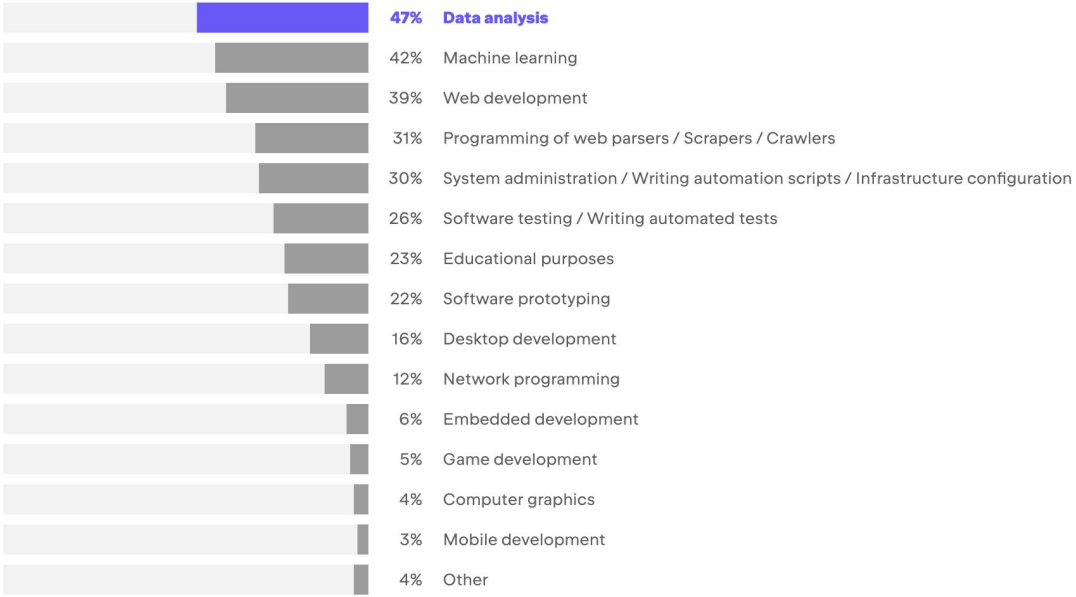
if err != nil


Bitbucket
"if err != nil"
Found matches in 10000 files

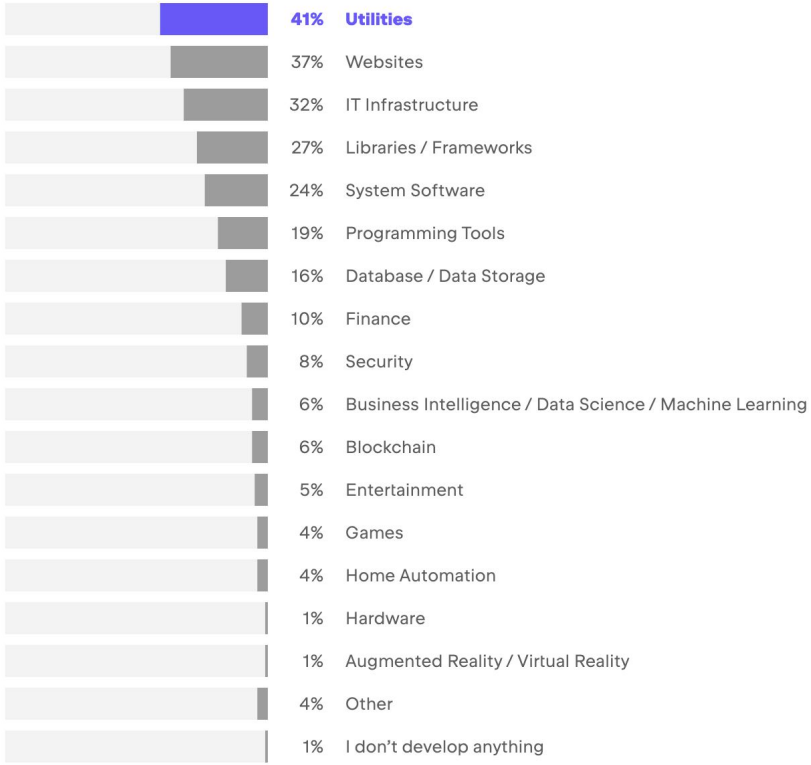

Find in Files 103+ matches in 51+ files
File mask: *.go
if err != nil
In Project Module Directory Scope
if err != nil { auto_select_item/client_cached.go 56
if err != nil { bx_layout/client.go 30
if err != nil { developers/service.go 30
if err != nil { developments/service.go 27
if err != nil { houses_data/service.go 26
if err != nil { houses_data/service.go 63

Разные проекты – разные языки

Python



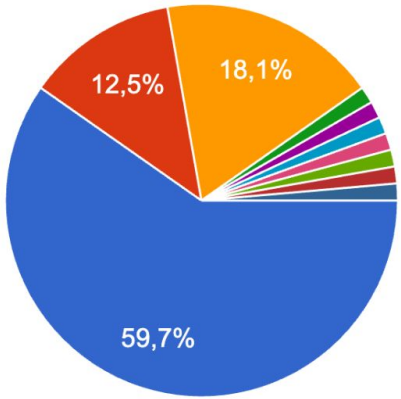
Golang



Avito Statistics

Когда вы впервые столкнулись с необходимостью профессионально писать на го?

72 ответа



- При смене места работы
- При смене проекта без смены мест…
- Язык начали использовать на текущ…
- Другое, но лень расписывать
- Искал работу с Go
- После того как пришел в Авито
- Появился язык
- первая работа, сразу на го
- тут во время тишейпинга (до этого мини pet-проекты и cli-утилиты)
- Целенаправленно сменил место работы, чтобы писать на го :)

Best Practice

```python
from typing import List

def should_use(annotations: List[str]) -> bool:
    print("It is awesome")
    return True
```

```go
import "fmt"

func main() {
    var types []string
    var strictTyping bool = true

    if strictTyping {
        fmt.Printf("Allowed types: %v", types)
    }
}
```

Best Practice

Go Proverbs

Simple, Poetic, Pithy

Don't communicate by sharing memory, share memory by communicating.

Concurrency is not parallelism.

Channels orchestrate; mutexes serialize.

The bigger the interface, the weaker the abstraction.

Make the zero value useful.

interface{} says nothing.

Gofmt's style is no one's favorite, yet gofmt is everyone's favorite.

A little copying is better than a little dependency.

Syscall must always be guarded with build tags.

Cgo must always be guarded with build tags.

Cgo is not Go.

With the unsafe package there are no guarantees.

Clear is better than clever.

Reflection is never clear.

Errors are values.

Don't just check errors, handle them gracefully.

Design the architecture, name the components, document the details.

Best Practice

PEP 8 – Style Guide for Python Code

Author: Guido van Rossum <guido at python.org>, Barry Warsaw <barry at python.org>, Alyssa Coghlan <ncoghlan at gmail.com>
Status: Active
Type: Process
Created: 05-Jul-2001
Post-History: 05-Jul-2001, 01-Aug-2013

▶ **Table of Contents**

Introduction

This document gives coding conventions for the Python code comprising the standard library in the main Python distribution. Please see the companion informational PEP describing style guidelines for the C code in the C implementation of Python.

This document and PEP 257 (Docstring Conventions) were adapted from Guido's original Python Style Guide essay, with some additions from Barry's style guide [2].

This style guide evolves over time as additional conventions are identified and past conventions are rendered obsolete by changes in the language itself.

Many projects have their own coding style guidelines. In the event of any conflicts, such project-specific guides take precedence for that project.

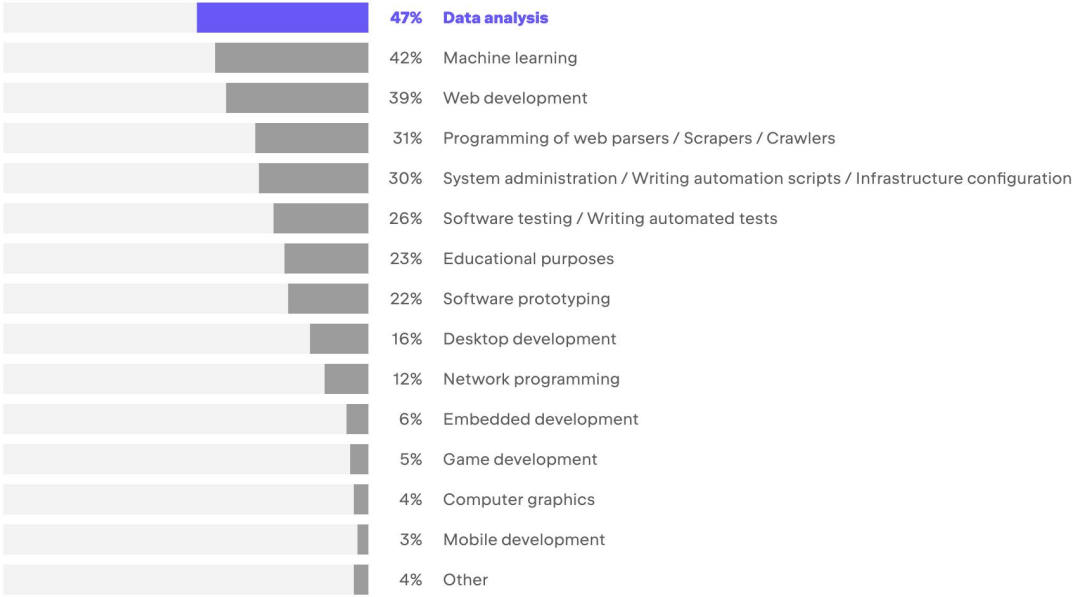
Почему я сменил язык?

```python
df = pd.DataFrame({"column1": [1.1]})
df["column1"] =
df["column1"].astype("float32")
print(df.dtypes)
df.to_csv("some.csv", index=False)
df_new = pd.read_csv("some.csv")
print(df_new.dtypes)
```

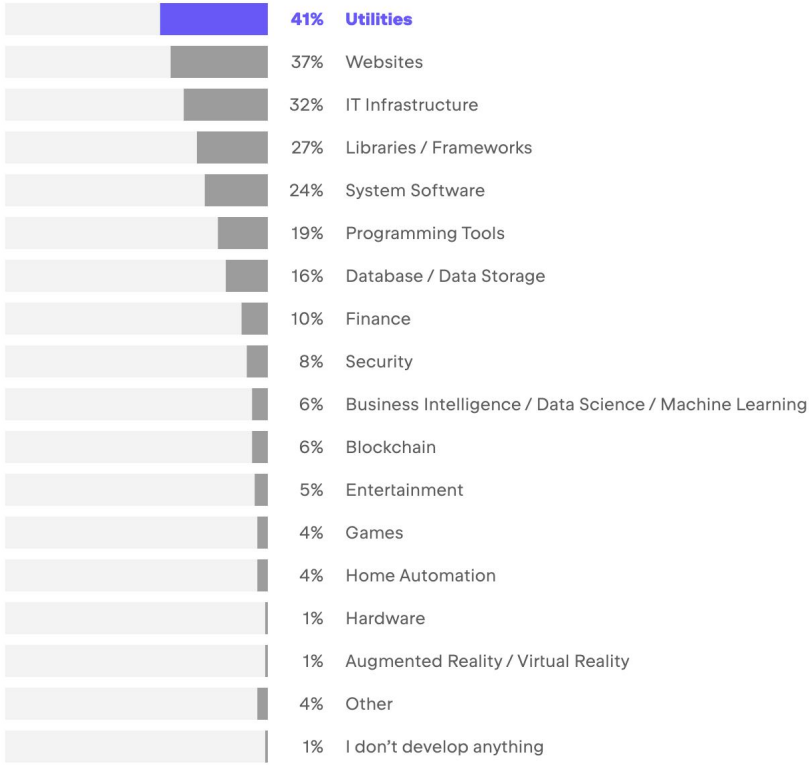
```
column1    float32
dtype: object

column1    float64
dtype: object
```

Разные языки – разные проекты

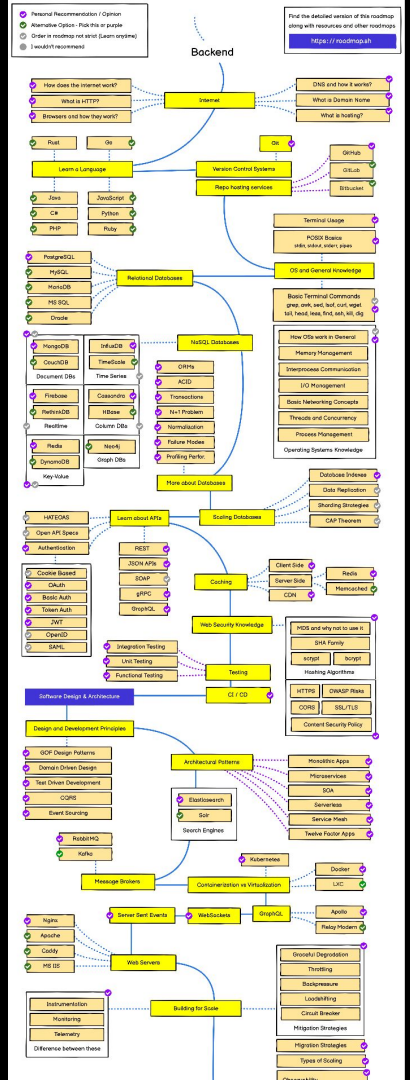
Python



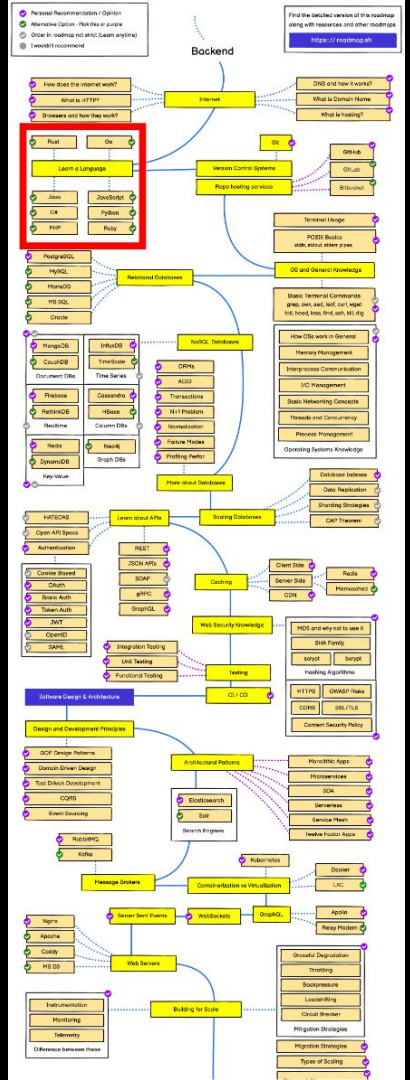
Golang



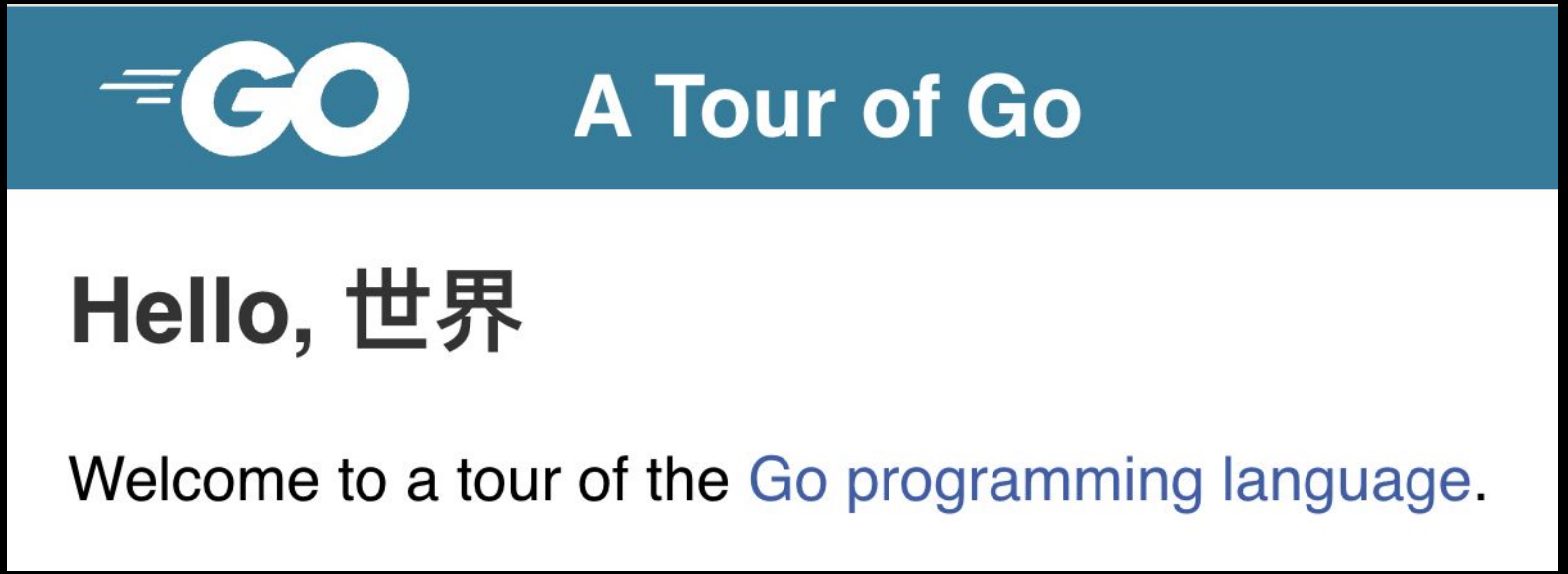
Новый язык – новое дыхание



Новый язык – новое дыхание



Как научиться?



Как научиться?

Basics

The starting point, learn all the basics of the language.

Declaring variables, calling functions, and all the things you need to know before moving to the next lessons.

Packages, variables, and functions.

Learn the basic components of any Go program.

Flow control statements: for, if, else, switch and defer

Learn how to control the flow of your code with conditionals, loops, switches and defers.

More types: structs, slices, and maps.

Learn how to define types based on existing ones: this lesson covers structs, arrays, slices, and maps.

Methods and interfaces

Learn how to define methods on types, how to declare interfaces, and how to put everything together.

Methods and interfaces

This lesson covers methods and interfaces, the constructs that define objects and their behavior.

Generics

Learn how to use type parameters in Go functions and structs.

Generics

Go supports generic programming using type parameters. This lesson shows some examples for employing generics in your code.

Concurrency

Go provides concurrency features as part of the core language.

This module goes over goroutines and channels, and how they are used to implement different concurrency patterns.

Concurrency

Go provides concurrency constructions as part of the core language. This lesson presents them and gives some examples on how they can be used.

Как научиться?

Практикуемся

```python
class BookRepository:
    def __init__(self):
        self.books = []

    def get_books(self):
        return self.books

    def add_book(self, book):
        self.books.append(book)


class BookService:
    def __init__(self, repo):
        self.repository = repo

    def get_books(self):
        books = []
        for book in self.repository.get_books():
            books.append(json.loads(book))
        return books

    def add_book(self, book_data):
        json_book_data = json.dumps(book_data)
        self.repository.add_book(json_book_data)
        return json_book_data


if __name__ == '__main__':
    book_repository = BookRepository()
    book_service = BookService(book_repository)
    book_service.add_book({"title": "Тошнота", "author": "Сартр"})
    print(book_service.get_books())
```

```python
class BookRepository:
    def __init__(self):
        self.books = []

    def get_books(self):
        return self.books

    def add_book(self, book):
        self.books.append(book)


class BookService:
    def __init__(self, repo):
        self.repository = repo

    def get_books(self):
        books = []
        for book in self.repository.get_books():
            books.append(json.loads(book))
        return books

    def add_book(self, book_data):
        json_book_data = json.dumps(book_data)
        self.repository.add_book(json_book_data)
        return json_book_data


if __name__ == '__main__':
    book_repository = BookRepository()
    book_service = BookService(book_repository)
    book_service.add_book(
        {"title": "Тошнота", "author": "Сартр"}
    )
    print(book_service.get_books())
```

```go
type BookRepository struct {
    books []string
}

type BookService struct {
    repository BookRepository
}

func main() {

}
```

```python
class BookRepository:
    def __init__(self):
        self.books = []

    def get_books(self):
        return self.books

    def add_book(self, book):
        self.books.append(book)


class BookService:
    def __init__(self, repo):
        self.repository = repo

    def get_books(self):
        books = []
        for book in self.repository.get_books():
            books.append(json.loads(book))
        return books

    def add_book(self, book_data):
        json_book_data = json.dumps(book_data)
        self.repository.add_book(json_book_data)
        return json_book_data


if __name__ == '__main__':
    book_repository = BookRepository()
    book_service = BookService(book_repository)
    book_service.add_book(
        {"title": "Тошнота", "author": "Сартр"}
    )
    print(book_service.get_books())
```

```go
type BookRepository struct {
    books []string
}

func (r *BookRepository) getBooks() []string {}

func (r *BookRepository) addBook(book string) {}


type BookService struct {
    repository BookRepository
}

func (s *BookService) getBooks() []map[string]string {}

func (s *BookService) addBook(bookData map[string]string) string {}


func main() {

}
```

Практикуемся

```python
class BookRepository:
    def __init__(self):
        self.books = []

    def get_books(self):
        return self.books

    def add_book(self, book):
        self.books.append(book)
```

```go
type BookRepository struct {
    books []string
}

func (r *BookRepository) getBooks() []string {}

func (r *BookRepository) addBook(book string) {}


func NewBookRepository() BookRepository {
    return BookRepository{books: []string{}}
}
```

Практикуемся

```python
class BookRepository:
    def __init__(self):
        self.books = []

    def get_books(self):
        return self.books

    def add_book(self, book):
        self.books.append(book)
```

```go
type BookRepository struct {
    books []string
}

func (r *BookRepository) getBooks() []string {
    return r.books
}

func (r *BookRepository) addBook(book string) {
    r.books = append(r.books, book)
}

func NewBookRepository() BookRepository {
    return BookRepository{books: []string{}}
}
```

Практикуемся

```python
class BookService:
    def __init__(self, repo):
        self.repository = repo

    def get_books(self):
        books = []
        for book in self.repository.get_books():
            books.append(json.loads(book))
        return books

    def add_book(self, book_data):
        json_book_data = json.dumps(book_data)
        self.repository.add_book(json_book_data)
        return json_book_data
```

```go
type BookService struct {
    repository BookRepository
}

func (s *BookService) getBooks() ([]map[string]string, error) {
    var books []map[string]string
    for _, book := range s.repository.getBooks() {
        var bookData map[string]string
        err := json.Unmarshal([]byte(book), &bookData)
        if err != nil {
            return nil, fmt.Errorf("failed to get books: %w", err)
        }
        books = append(books, bookData)
    }
    return books, nil
}

func (s *BookService) addBook(bookData map[string]string) (string,
error) {
    jsonBookData, err := json.Marshal(bookData)
    if err != nil {
        return "", fmt.Errorf("failed to marshall json: %w", err)
    }
    s.repository.addBook(string(jsonBookData))
    return string(jsonBookData), nil
}

func NewBookService(repo BookRepository) BookService {
    return BookService{repository: repo}
}
```

Практикуемся

```python
if __name__ == '__main__':
    book_repository = BookRepository()
    book_service = BookService(book_repository)
    book_service.add_book(
        {"title": "Тошнота", "author": "Сартр"}
    )
    print(book_service.get_books())
```

```go
func main() {
    bookRepository := NewBookRepository()
    bookService := NewBookService(bookRepository)
    _, err := bookService.addBook(
        map[string]string{"title": "Тошнота", "author": "Сартр"},
    )
    if err != nil {
        fmt.Println(fmt.Errorf("failed to add book: %w", err))
    }

    books, err := bookService.getBooks()
    if err != nil {
        fmt.Println(fmt.Errorf("failed to get booksL %w", err))
    }

    for _, book := range books {
        fmt.Println(book)
    }
}
```

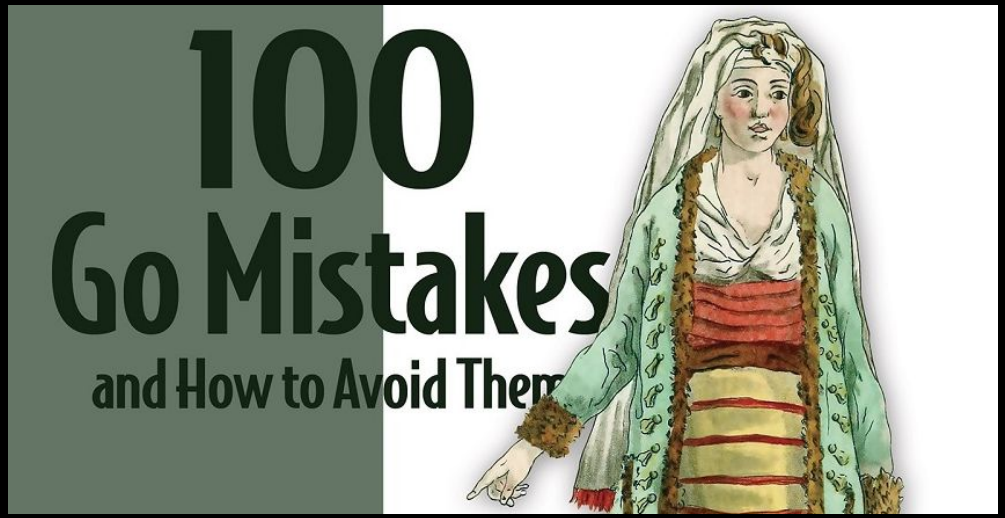
Куда развиваться дальше?

GO Why Go Learn Docs Packages Community

Documentation

The Go programming language is an open source project to make programmers more productive.

Go is expressive, concise, clean, and efficient. Its concurrency mechanisms make it easy to write programs that get the most out of multicore and networked machines, while its novel type system enables flexible and modular program construction. Go compiles quickly to machine code yet has the convenience of garbage collection and the power of run-time reflection. It's a fast, statically typed, compiled language that feels like a dynamically typed, interpreted language.



The Go Blog

Robust generic functions on slices, 22 February 2024
Valentin Deleplace
Avoiding memory leaks in the slices package.

Routing Enhancements for Go 1.22, 13 February 2024
Jonathan Amsterdam, on behalf of the Go team
Go 1.22's additions to patterns for HTTP routes.

Go 1.22 is released!, 6 February 2024
Eli Bendersky, on behalf of the Go team
Go 1.22 enhances for loops, brings new standard library functionality and improves performance.

Share your feedback about developing with Go, 23 January 2024
Alice Merrick, for the Go team
Help shape the future of Go by sharing your thoughts via the Go Developer Survey

Finding unreachable functions with deadcode, 12 December 2023
Alan Donovan
deadcode is a new command to help identify functions that cannot be called.

Go Developer Survey 2023 H2 Results, 5 December 2023
Todd Kulesza
What we learned from our 2023 H2 developer survey

Fourteen Years of Go, 10 November 2023
Russ Cox, for the Go team
Happy Birthday, Go!

Занимайтесь тем, что любите и не переставайте развиваться

Дмитрий Королев

Изучить новый ЯП проще, чем кажется

А процесс изучения может подарить те же положительные эмоции, которые когда-то привели вас в профессиональную разработку

t.me/jimiliani

avito.tech Москва 2024

Переход с python на go: год спустя

Дмитрий Королев