Introduction to Python for Tech Managers

Discover the many benefits of Python and learn when it's the right choice for your tech stack.

STX NEXT
python powerhouse

Table of Contents

Introduction

When you're looking to build something, you want to make sure you have the right tools. In many cases, the choice of tools will determine your entire experience.

Choose the wrong tools, and you'll pay the price down the line; your work may be slower, your competition may outpace you, or your end result may fall below expectations.

But choose the right tools, and you'll be thankful for it every step of the way.

If what you're building is software, your programming language is your prime tool.

The choice of language at the start of your development will have a profound impact on it in the future. It may very well mean the difference between smooth scaling and costly refactoring, or meeting your deadline and missing it.

Do you want your software development to be successful?

Choose Python.

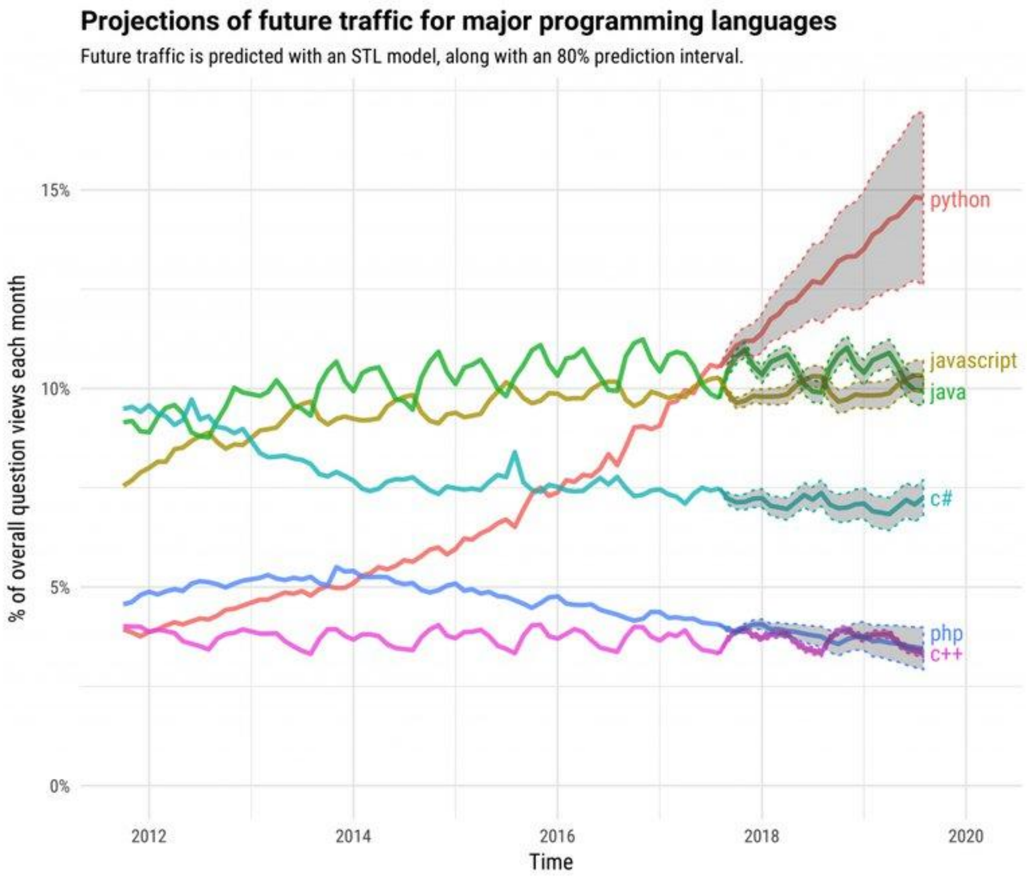
The main objective of this ebook is to explain why you should do that—especially if you're looking to manage software development projects. Read on, and you're also going to learn everything you need to get started on Python web frameworks and Python libraries.

Ready? Let's get to it.

Why Python?

Python is popular and widely supported

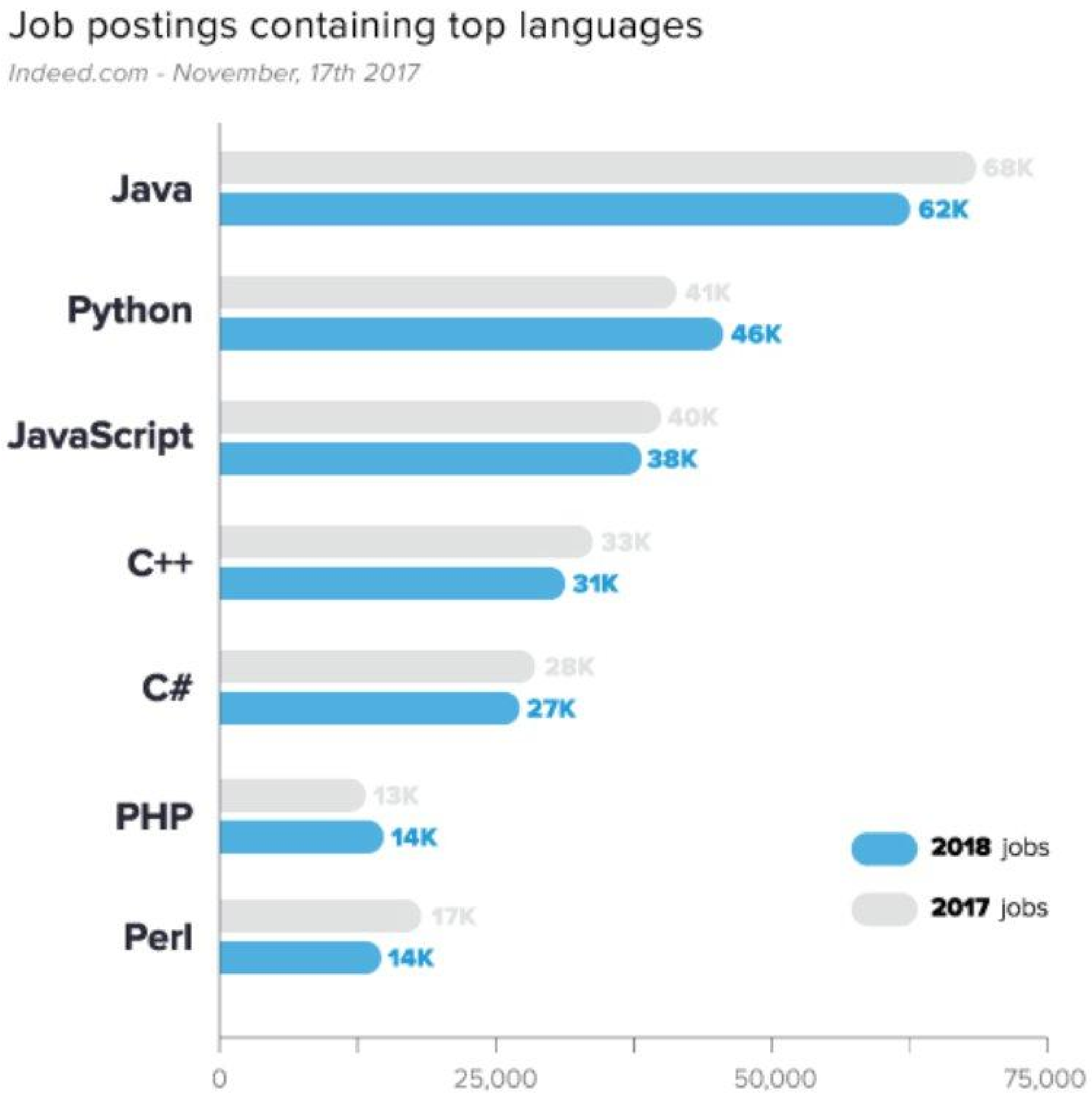
Python is quickly ascending to the forefront of the most popular programming languages in the world. StackOverflow very clearly shows the incredible growth of Python:



Graph source

Coding Dojo also identifies Python as one of the most in-demand programming languages of 2018.

The demand for Python is growing with no signs of stopping, second only to Java, whereas the number of Java job postings is projected to decrease. It is likely that Python will overtake Java in the coming years, reaching the #1 spot.



Job postings containing top languages

Indeed.com - November, 17th 2017

Graph source

How can you benefit from Python's popularity?

The popular option isn't always the best one. But when it comes to programming languages, popularity does pay off.

By using a popular language, you have a much higher chance of finding a solution to any problem you may encounter. In fact, if your issue is common enough, Python probably has a ready-made solution waiting for you already.

Python enjoys a healthy community of enthusiasts that strives to make the language better every day, by fixing bugs and opening new possibilities.

At the same time, Python has some strong corporate sponsors who will push the adoption of this language even further—Google is one of them. Because Google uses Python, they are actively working on guides, tutorials, and other resources to get the most out of it.

In fact, Guido van Rossum, the creator of Python, is currently employed at Google and devotes half of his time to developing our favorite programming language.

Writing Python code is easy, speeding up your development

Python is accessible by design, making it one of the fastest languages in terms of development speed. Just look at what Guido van Rossum had to say at the conception of the language:

> ***"The second stream of material that is going to come out of this project is a programming environment and a set of programming tools where we really want to focus again on the needs of the newbie. This environment is going to have to be extremely user-friendly."***
> **Guido van Rossum,** Former Benevolent Dictator for Life of the Python Language

(How cool is that title, by the way?)

A user-friendly environment in the hands of your development team means less time wrestling with your building tool and more time spent on the actual building.

In fact, Python's simplicity was the reason STX Next came to be in the first place. As we've mentioned in the Introduction to this ebook already, before our CEO Maciej Dziergwa started the company, he was a Java developer. Once he realized that he could recreate a 2-month Java project in mere 2 weeks using Python, there was no going back.

Get a headstart with frameworks and libraries

Python is known for its rich selection of libraries, saving you the hassle of coding some features by hand and accelerating time-to-market. If you're looking for scientific libraries, there's Pandas, NumPy, and SciPy, for example. Python even has specialized libraries for machine learning (scikit-learn) and natural language processing (nltk)—but more on that later in the ebook.

Aside from libraries, Python also offers a wide range of web development frameworks you can use to jump-start your project and avoid unnecessary coding. Some frameworks, such as Django, give you everything you need to build a web application, from low- to high-end. Other frameworks are more specialized, such as the Flask microframework. The next section of this ebook gives you an extensive overview of this.

Whether you decide to use just a few libraries or entire frameworks, Python will help you get results faster.

What about performance?

The way Python is structured and Python code executes is not without its drawbacks. One oft-cited disadvantage of Python is its runtime, which is often slower compared to other languages.

And yet, Python is still the language of choice for tech giants like Google. Why?

Because CPU time is rarely the limiting factor.

The limiting factor is your employees' time.

You want to optimize your most expensive resources—and developer hours don't come cheap. Therefore, you need all the help you can get to shorten time-to-market, even if it leads to slower runtime execution.

Mix it up with Cython

But that's not all! For situations where performance really matters, Python offers tried and tested solutions to incorporate other, faster languages into the code.

One such solution is using Cython. Usually, your performance bottleneck lies in only one of the methods in your code. You can use Cython to rewrite the method in C, optimizing your speed without the need to rewrite the whole code base.

"Using Cython you get the perfect mix of optimizing only at the bottleneck, and the beauty of Python everywhere else."

Nick Humrich at Hacker Noon

And so, once again, whatever your woes—Python has got you covered.

Reading Python code is intuitive, making maintenance a breeze

Python's syntax is clear and concise. The language is designed to be readable and close to actual English, making it easy to decipher. Python also requires fewer lines of code to achieve results compared to languages such as C or Java.

Python's approach is so deeply embedded that you can even learn about it from Python itself. Here's what you'll see when you execute `import this` in the Python interpreter:

>>> import this
The Zen of Python, by Tim Peters
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one—and preferably only one—obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
*Although never is often better than *right* now.*
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea—let's do more of those!

(We learned this little trick from this article over at Full Stack Python.)

How do you benefit from readable code?

The simplicity of Python helps greatly when you have to read the code you've written, or code from another developer. Code review goes much faster when you have fewer lines of code to actually review, and the code reads like English. There's also less catching up involved when the code changes hands; you can figure out pretty quickly what each function is supposed to do.

With code that's easier to understand and navigate, you can reduce the amount of work needed to maintain and expand your code base. And reducing work is one of the best things you can do, since we've already established that optimizing developer productivity should be your priority.

Python gives you tried and tested scalability

No one can really predict when your user numbers will start surging and scalability will become a priority. Which is why it's a good idea to use a language that scales great and, as we've mentioned above, is easy to maintain.

Some of the most ambitious projects around the web—YouTube, Reddit, EVE Online—all use Python to serve their user base reliably.

To be precise, Reddit was initially written in Lisp, and then rewritten in Python—*over a single weekend*.

Here's how the switch from Lisp to Python was justified on the Reddit blog:

> ***"We were already familiar with Python.***
> ***It's fast, development in Python is fast, and the code is clear."***
> **/u/shuffman56 on choosing Python for Reddit**

In short: if you dream big, go with Python.

Python Web Frameworks

If you've read this far, it's safe to assume you're interested in taking up Python—or maybe you've already started learning this awesome language.

It doesn't seem too daunting, right? You can code, after all, so it's just a matter of grasping the differences in syntax. Perhaps you're already watching tutorials and reading guides.

So let's take it up a notch and talk about collecting proper experience in Python.

Let's discuss creating your first Python project.

Where do you start?

With an idea, obviously, but that won't be a problem. Surely you already have several great concepts safely locked away in the vault of your mind, just waiting for some of that precious spare time and attention.

What's next, then? The choice of a framework.

And that's where the real conundrum starts, because the ecosystem of Python frameworks is quite extensive and varied.

In this section of the ebook, we're going to describe the best and most popular Python frameworks. It should be more than enough for you to pick the right one and get started.

Be warned, though, that this list is rather subjective. It came together mainly as a result of our collective experience of using the following frameworks in commercial projects.

What are web frameworks?

Before you decide on a particular framework, let's make sure we're on the same page when it comes to definitions.

What exactly do we have in mind when we talk about a web application framework?

In short, a web framework is **a package of generic functionalities that makes creating web apps easier for the developer.** It serves as a shortcut that removes the need to write unnecessary code by reusing existing solutions. As a result, it reduces the time your developers need to spend on writing code and makes their work more effective.

Web frameworks can be divided into two categories: frontend and backend. The former, also known as CSS frameworks, is all about the parts of the web app the users see and interact with. The latter relates to the behind-the-scenes aspects of creating a web app.

The crucial benefit of using Python frameworks is that you can mix and match frontend and backend elements within each framework to achieve the desired result. You can either focus on one or merge several of them, depending on the scope of your project.

By offering ready-made solutions, web app frameworks help developers add complex and dynamic elements that would otherwise be very difficult or time-consuming to develop from scratch.

12+ best Python web frameworks

1) Django

Django is one of the most popular Python frameworks. Offering all the tools you need to build a web application within a single package, from low- to high-end, is its trademark.

Django applications are based on a design pattern similar to MVC, the so-called MVT (Model-View-Template) pattern. Models are defined using the Django ORM, while SQL databases are mainly used as storage.

Django has a built-in admin panel, allowing for easy management of the database content. With minimal configuration, this panel is generated automatically based on the defined models.

Views can include both functions and classes, and the assignment of URLs to views is done in one location (the urls.py file), so that after reviewing that single file you can learn which URLs are supported. Templates are created using a fairly simple Django Templates system.

Django is praised for strong community support and detailed documentation describing the functionality of the framework. This documentation coupled with getting a comprehensive environment after the installation makes the entry threshold rather low. Once you go through the official tutorial, you'll be able to do most of the things required to build an application.

Unfortunately, Django's monolithism also has its drawbacks. It is difficult, though not impossible, to replace one of the built-in elements with another implementation. For example, using some other ORM (like SQLAlchemy) requires abandoning or completely rebuilding such items as the admin panel, authorization, session handling, or generating forms.

Because Django is complete but inflexible, it is suitable for standard applications (i.e. the vast majority of software projects). However, if you need to implement some unconventional design, it leads to struggling with the framework, rather than pleasant programming.

Sample model in Django

```
class Company(models.Model):
    name = models.CharField(max_length=255)
    email = models.EmailField(max_length=75, null=True, blank=True)
    website_url = models.URLField(blank=True, null=True)
    city = models.CharField(max_length=100, null=True, blank=True)
    street = models.CharField(max_length=100, null=True, blank=True)
    size = models.IntegerField(null=True, blank=True)
    date_founded = models.CharField(
        help_text='MM/YYYY', null=True, blank=True, max_length=7,
    )
    @property
    def urls(self):
        return {
            'view': reverse('view-company', args=(self.pk,)),
            'edit': reverse('edit-company', args=(self.pk,)),
        }
    def __unicode__(self):
        return self.name
```

Flask

Flask is considered a microframework. It comes with basic functionality, while also allowing for easy expansion. Therefore, **Flask works more as the glue that allows you to join libraries with each other.**

For example, “pure Flask” does not provide support for any storage, yet there are many different implementations that you can install and use interchangeably for that purpose (such as Flask-SQLAlchemy, Flask-MongoAlchemy, and Flask-Redis). Similarly, the basic template system is Jinja2, but you can use a replacement (like Mako).

The motto of this framework is “one drop at a time,” and this is reflected in its comprehensive documentation. The knowledge of how to build an application is acquired in portions here; after reading a few paragraphs, you will be able to perform basic tasks.

You don’t have to know the more advanced stuff right away—you’ll learn it once you actually need it. Thanks to this, students of Flask can gather knowledge smoothly and avoid boredom, making Flask suitable for learning.

A large number of Flask extensions, unfortunately, are not supported as well as the framework itself. It happens quite often that the plug-ins are no longer being developed or their documentation is outdated. In cases like these, you need to spend some time googling a replacement that offers similar functionality and is still actively supported.

When building your application with packages from different authors, you might have to put quite a bit of sweat into integrating them with each other. You will rarely find ready-made instructions on how to do this in the plug-ins’ documentation, but in such situations the Flask community and websites such as Stack Overflow should be of help.

Sample view in Flask

```python
@image_view.route(
    '/api/<string:version>/products/<int:prod_id>/images',
    methods=['GET'],
)
@auth_required()
@documented(prod_id="ID of a product")
@output(ProductImagesSeq)
@errors(MissingProduct)
@jsonify
def images_get(version, prod_id):
    """Retrieves a list of product images."""
    return [i.serialize() for i in find_product(prod_id).images]
```

Pyramid

Pyramid, the third noteworthy Python web framework, is rooted in two other products that are no longer developed: Pylons and repoze.bfg. **The legacy left by its predecessors caused Pyramid to evolve into a very mature and stable project.**

The philosophies of Pyramid and Django differ substantially, even though both were released in the same year (2005). Unlike Django, Pyramid is trivial to customize, allowing you to create features in ways that the authors of the framework themselves hadn't foreseen. It does not force the programmer to use framework's idioms; it's meant to be a solid scaffolding for complex or highly non-standard projects.

Pyramid strives to be persistence-agnostic. While there is no bundled database access module, a common practice is to combine Pyramid with the powerful, mature SQLAlchemy ORM. Of course, that's only the most popular way to go. Programmers are free to choose whatever practices suit them best, such as using the peewee ORM, writing raw SQL queries, or integrating with a NoSQL database, just to name a few.

All options are open, though this approach requires a bit of experience to smoothly add the desired persistence mechanisms to the project. The same goes for other components, such as templating.

Openness and freedom are what Pyramid is all about. Modules bundled with it relate to the web layer only and users are encouraged to freely pick third-party packages that will support other aspects of their projects.

However, this model causes a noticeable overhead at the beginning of any new project, because you have to spend some time choosing and integrating the tools your team is comfortable with. Still, once you put the effort into making additional decisions during the early stages of the work, you are rewarded with a setup that makes it easy and comfortable to start a new project and develop it further.

Pyramid is a self-proclaimed "start small, finish big, stay finished framework." This makes it an appropriate tool for experienced developers who are not afraid of playing the long game and working extra hard in the beginning, without shipping a single feature within the first few days. Less experienced programmers may feel a bit intimidated.

Sample "Hello world" app in Pyramid

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response


def hello_world(request):
   return Response('Hello, world!')


if __name__ == '__main__':
   with Configurator() as config:
       config.add_route('hello', '/')
       config.add_view(hello_world, route_name='hello')
       app = config.make_wsgi_app()

   server = make_server('0.0.0.0', 6543, app)
   server.serve_forever()
```

web2py

WEB2PY

Created in 2007, web2py is a framework originally designed as a teaching tool for students, so the main concern for its authors was ease of development and deployment.

Web2py is strongly inspired by Django and Ruby on Rails, sharing the idea of **convention over configuration. In other words, web2py provides many *sensible defaults* that allow developers to get off the ground quickly.**

This approach also means there are a lot of goodies bundled with web2py. You will find everything you'd expect from a web framework in it, including a built-in server, HTML-generating helpers, forms, validators, and many more—nothing unusual thus far, one could argue. Support for multiple database engines is neat, though it's a pretty common asset among current web frameworks.

However, some other bundled features may surprise you, since they are not present in other frameworks:

- helpers for creating JavaScript-enabled sites with jQuery and Ajax;
- scheduler and cron;
- 2-factor authentication helpers;
- text message sender;
- an event-ticketing system, allowing for automatic assignment of problems that have occurred in the production environment to developers.

The framework proudly claims to be a full-stack solution, providing everything you could ever need.

Web2py has extensive documentation available online. It guides newcomers step by step, starting with a short introduction to the Python language. The introduction is seamlessly linked with the rest of the manual, demonstrating different aspects of web2py in a friendly manner, with lots of code snippets and screenshots.

Despite all its competitive advantages, web2py's community is significantly smaller than Django's, or even Pyramid's. Fewer developers using it means your chances of getting help and support are lower. The official mailing list is mostly inactive.

Additionally—and unfortunately—web2py is not compatible with Python 3 at the moment. This state of things puts the framework's prospects into question, as support for Python 2 ends in 2020. This issue is being addressed on the project's github. Here is where you can track the progress.

Sample model in web2py

```
class Ads(BaseModel):
    tablename = "ads"

    def set_properties(self):
        T = self.db.T
        self.fields = [
            # main
            Field("title", "string"),
            Field("description", "text"),
            Field("picture", "upload"),
            Field("thumbnail", "upload"),
            Field("link", "string"),
            Field("place", "string")
        ]

        self.computations = {
            "thumbnail": lambda r: THUMB2(r['picture'],
gae=self.db.request.env.web2py_runtime_gae)
    }

        self.validators = {
            "title": IS_NOT_EMPTY(),
            "description": IS_LENGTH(255, 10),
            "picture": IS_IMAGE(),
            "place": IS_IN_SET(["top_slider", "top_banner",
"bottom_banner", "left_sidebar", "right_sidebar", "inside_article",
"user_profile"], zero=None)
```

```
            }
```

Sanic

Sanic differs considerably from the aforementioned frameworks because unlike them, it is based on asyncio—Python's toolbox for asynchronous programming, bundled with the standard library starting from version 3.4.

In order to develop projects based on Sanic, you have to grasp the ideas behind asyncio first. This involves a lot of theoretical knowledge about coroutines, concurrent programming caveats, and careful reasoning about the data flow in the application.

Once you get your head around Sanic/asyncio and applies the framework to an appropriate problem, the effort pays off. Sanic is especially useful when it comes to handling long-living connections, such as websockets. If your project requires support for websockets or making a lot of long-lasting external API calls, Sanic is a great choice.

Another use case of Sanic is writing a "glue-web application" that can serve as a mediator between two subsystems with incompatible APIs. Note that it requires at least Python 3.5, though.

The framework is meant to be very fast. One of its dependencies is uvloop—an alternative, drop-in replacement for asyncio's not-so-good built-in event loop. Uvloop is a wrapper around libuv, the same engine that powers Node.js. According to the uvloop documentation, this makes asyncio work 2–4 times faster.

In terms of “what’s in the box,” Sanic doesn’t offer as much as other frameworks. It is a microframework, just like Flask. Apart from routing and other basic web-related goodies like utilities for handling cookies and streaming responses, there’s not much inside. Sanic imitates Flask, for instance by sharing the concept of Blueprints—tiny sub-applications that allow developers to split and organize their code in bigger applications.

Sanic also won’t be a good choice for simple CRUD applications that only perform basic database operations. It would just make them more complicated with no visible benefit.

Handling websockets in Sanic

```
@app.websocket('/websocket')
async def time(websocket, path):
   while True:
      now = datetime.datetime.utcnow().isoformat() + 'Z'
      await websocket.send(now)
      await asyncio.sleep(random.random() * 3)
```

Japronto

Have you ever imagined handling 1,000,000 requests per second with Python?

It seems unreal, since Python isn’t the fastest programming language out there. But when a brilliant move was made to add asyncio to the standard library, it opened up countless possibilities.

Japronto is a microframework that leverages some of them. As a result, **this Python framework was able to cross the magical barrier of 1 million requests handled per second.**

You may still be at a loss as to how that is possible, exactly.

It all comes down to two aces up Japronto’s sleeve: uvloop and PicoHTTPParser. Uvloop is an asyncio backend based on libuv, while PicoHTTPParser is a lightweight HTTP headers parser written in C. All core components of the framework are also implemented in C. A wide variety of low-level optimizations and tricks are used to tweak performance.

Japronto is designed for special tasks that could not be accomplished with bloated mainstream frameworks. It is a perfect fit for problems where every nanosecond counts.

Knowledgeable developers, obsessed with optimization, will reap all of its possible benefits.

Additionally, Japronto is meant to provide a solid foundation for microservices using REST APIs with minimal overhead. In other words, there's not much in the box. Developers only need to set up routing and decide which routes should use synchronous or asynchronous handlers.

It might seem counterintuitive, but if a request can be handled in a synchronous way, you shouldn't try to do it asynchronously, as the overhead of switching between coroutines will limit performance.

What is quite unfortunate is that Japronto is not being actively developed. On the other hand, the project is licensed under MIT, and the author claims he is willing to accept any contributions. Like Sanic, the framework is meant to work with Python 3.5+ versions.

Sample "Hello world" app in Japronto

```python
from japronto import Application

def hello(request):
    return request.Response(text='Hello world!')

app = Application()
app.router.add_route('/', hello)
app.run(debug=True)
```

aiohttp

Aiohttp is another library based on asyncio, the modern Python toolkit for writing asynchronous code. **Not meant to be a framework in a strict sense, aiohttp is more of a toolbox, supplementing the async arsenal with everything related to HTTP.**

This means aiohttp is helpful not only for writing server applications, but also to clients. Both will benefit from asyncio's goodies, most of all the ability to handle thousands of connections at the same time, provided the majority of operations involves I/O calls.

Such powerful clients are great when you have to issue many API calls at once, for example for scraping web pages. Without asyncio, you would have to use threading or multiprocessing, which are harder to get right and require much more memory.

Apart from building standalone applications, aiohttp's clients are a great supplement to any asyncio-based application that needs to issue non-blocking HTTP calls. The same is true for websockets. Since they are part of the HTTP specification, you can connect to websocket servers and easily exchange messages with them.

When it comes to servers, aiohttp gives you everything you can expect from a microframework. The features available out-of-the-box include routing, middleware, and signals. It may seem like it's very little, but it will suffice for a web server.

"What about the remaining functionalities?" you may ask.

As far as those are concerned, you can build the rest of the functionalities using one or many asyncio-compatible libraries. You will find plenty of them using sources like this one.

Aiohttp is built with testing in mind. Developers who want to test an aiohttp-based application will find it extremely easy, especially with the aid of pytest.

Even though aiohttp offers satisfactory performance by default, there are a few low-hanging fruits you can pick. For example, you can install additional libraries: cchardet and aiodns. Aiohttp will detect them automatically. You can also utilize the same uvloop that powers Sanic.

Last but not least: **one definite advantage of aiohttp is that it is being actively maintained and developed.** Choosing aiohttp when you build your next application will certainly be a good call.

Websocket client using aiohttp

```
async with session.ws_connect('http://example.org/ws') as ws:
    async for msg in ws:
        if msg.type == aiohttp.WSMsgType.TEXT:
            if msg.data == 'close cmd':
                await ws.close()
                break
            else:
                await ws.send_str(msg.data + '/answer')
        elif msg.type == aiohttp.WSMsgType.ERROR:
            break
```

Twisted

With Twisted, Python developers were able to do async programming long before it was cool. Twisted is one of the oldest and most mature Python projects around.

Originally released in 2002, Twisted predates even PEP8, so the code of the project does not follow the famous code style guide recommendations. Admittedly, this may somewhat discourage people from using it these days.

Twisted's heart is an event-driven networking engine called `reactor`. It is used for scheduling and calling user-defined callbacks.

In the beginning, developers had to use explicit callbacks by defining functions and passing them around separately for cases when an operation succeeded and when it failed.

Although this technique was compelling, it could also lead to what we know from early JavaScript: callback hell. In other words, the resultant code was tough to read and analyze.

At some point, Twisted introduced inlineCallbacks—the notation for writing asynchronous code that was as simple to read as regular, synchronous code. This solution played very well with Python's syntax and greatly influenced modern async toolkit from the standard library, asyncio.

The greatest advantage of this framework is that although Twisted itself is just an engine with few bundled extensions, there are many additional extensions available to expand its functionality. They allow for both low-level network programming (TCP/USP) and high, application-level work (HTTP, IMAP, SHH, etc).

This makes Twisted a perfect choice for writing specialized services; however, it is *not* a good candidate for regular web applications. Developers would have to write a lot of things on their own to get the functionality they take for granted with Django.

Twisted is being actively maintained. There is an undergoing effort to migrate all of its code to be compatible with Python 3. The core functionality was rewritten some time ago, but many third-party modules are still incompatible with newer versions of the interpreter.

This may raise some concerns whether Twisted is the best choice for new projects. On the other hand, though, it is more mature than some asyncio-based solutions. Also, Twisted has been around for quite some time now, which means it will undoubtedly be maintained at least for a good while.

inlineCallbacks code in Twisted

```
@inlineCallbacks
def getUsers(self):
    try:
        responseBody = yield makeRequest("GET", "/users")
    except ConnectionError:
        log.failure("makeRequest failed due to connection error")
        returnValue([])

    returnValue(json.loads(responseBody))
```

Falcon

Falcon is another microframework on our list. The goal of the Falcon project is to create a minimalist foundation for building web apps where the slightest overhead matters.

Authors of the framework claim it is a **bare-metal, bloat-free toolkit for building very fast backend code and microservices**. Plus, it is compatible with both Python 2 and 3.

A big advantage of Falcon is that it is indeed very fast. Benchmarks published on its website show an incredible advantage over mainstream solutions like Django or Flask.

The downside, though, is that **Falcon offers very little to start with**. There's routing, middlewares, hooks—and that's basically everything. There are no extras: no validation, no authentication, etc. It is up to the developer to extend functionality as needed.

Falcon assumes it will be used for building REST APIs that talk JSON. If that is the case, you really need literally zero configuration. You can just sit down and code.

This microframework might be an exciting proposition for implementing highly-customized services that demand the highest performance possible. Falcon is an excellent choice when you don't want or can't invest in asyncio-based solutions.

If you're thinking, "Sometimes the simplest solution is the best one," you should definitely consider Falcon.

Sample “Hello world” app in Falcon

```
import falcon

class QuoteResource:
def on_get(self, req, resp):
    quote = {
        'quote': (
            "I've always been more interested in "
            "the future than in the past."
        ),
        'author': 'Grace Hopper'
    }
    resp.media = quote

api = falcon.API()
api.add_route('/quote', QuoteResource())
```

API Star

API Star is the new kid on the block. It is yet another microframework, but this one is compatible with Python 3 only. Which is not surprising, because it leverages type hints introduced in Python 3.5.

API Star uses type hints as a notation for building validation schemata in a concise, declarative way. Such a schema (called a “Type” in the framework’s terminology) can then be bound to request a handling function.

Additionally, API Star features automatically generated API docs. They are compatible with OpenAPI 3. Such docs can facilitate communication between API authors and its consumers, i.e. frontend developers. If you use the Types we've mentioned, they are included in the API docs.

Another outstanding feature is the dependency injection mechanism. It appears to be an alternative to middlewares, but smarter and much more powerful.

For example, you can write a so-called Component that will provide our views with a currently authenticated User. On the view level, you have to explicitly state that it will require a User instance.

The rest happens behind the scenes. API Star resolves which Components have to be executed to finally run our view with all the required information.

The advantage that automatic dependency injection has over regular middlewares is that Components do not cause any overhead for the views where they are not used.

Last but not least, API Star can also be run atop asyncio in a more traditional, synchronous, WSGI-compliant way. **This makes it probably the only popular framework in the Python world capable of doing that.**

The rest of the goodies bundled with API Star are pretty standard: optional support for templating with jinja2, routing, and event hooks.

All in all, API Star looks extremely promising. At the time of writing, it has over 4,500 stars in its GitHub repository. The repository already has a few dozen contributors, and pull requests are merged daily. Many of us at STX Next are keeping our fingers crossed for this project!

Schema validation on views in API Star

```python
from apistar import types, validators

class Product(types.Type):
     name = validators.String(max_length=100)
     rating = validators.Integer(minimum=1, maximum=5)
     in_stock = validators.Boolean(default=False)
     size = validators.String(enum=['small', 'medium', 'large'])
```

```
def create_product(product: Product):
    # Save a new product record in the database.
    ...

routes = [
    Route('/create_product/', method='POST', handler=create_product)
]
```

Others

There are many more Python web frameworks out there you might find interesting and useful. Each of them focuses on a different issue, was built for distinct tasks, or has a particular history.

The first that comes to mind is **Zope2,** one of the oldest frameworks, still used mainly as part of the Plone CMS. **Zope3** (later renamed BlueBream) was created as Zope2's successor. The framework was supposed to allow for easier creation of large applications, but hasn't won too much popularity, mainly because of the need to master fairly complex concepts (e.g. Zope Component Architecture) very early in the learning process.

Also noteworthy is the **Google App Engine,** which allows you to run applications written in Python, among others. This platform lets you create applications in any framework compatible with WSGI. The SDK for the App Engine includes a simple framework called webapp2, and this exact approach is often used in web applications adapted to this environment.

Another interesting example is **Tornado,** developed by FriendFeed and made available by Facebook. This framework includes libraries supporting asynchronicity, so you can build applications that support multiple simultaneous connections (like long polling or WebSocket).

Other libraries similar to Tornado include **Pulsar** (async) and **Gevent** (greenlet). These libraries allow you to build any network applications (multiplayer games and chat rooms, for example). They also perform well at handling HTTP requests.

Developing applications using these frameworks and libraries is more difficult and requires you to explore some harder-to-grasp concepts. We recommend getting to them later on, as you venture deeper into the wonderful world of Python.

Python Libraries

Python is many things.

Cross-platform. General-purpose. High-level.

As such, the programming language has numerous applications and has been widely adopted by all sorts of communities, from data science to business. These communities value Python for its precise and efficient syntax, relatively flat learning curve, and good integration with other languages (e.g. C/C++).

The language's popularity has resulted in a plethora of Python packages being produced for data visualization, machine learning, natural language processing, complex data analysis, and more.

Here is our list of the most popular Python libraries:

Astropy

Astropy is a collection of packages designed for use in astronomy.

The core Astropy package contains functionality aimed at professional astronomers and astrophysicists, but may be useful to anyone developing software for astronomy.

Biopython

Biopython is a collection of non-commercial Python tools for computational biology and bioinformatics.

It contains classes to represent biological sequences and sequence annotations. The library can also read and write to a variety of file formats.

Bokeh

Bokeh is a Python interactive visualization library that targets modern web browsers for presentation.

It can help anyone who wishes to quickly and easily create interactive plots, dashboards, and data applications.

The purpose of Bokeh is to provide elegant, concise construction of novel graphics in the style of D3.js, but also deliver this capability with high-performance interactivity over very large or streaming datasets.

Cubes

Cubes is a light-weight Python framework and set of tools for the development of reporting and analytical applications, Online Analytical Processing (OLAP), multidimensional analysis, and browsing of aggregated data.

Dask

Dask is a flexible parallel computing library for analytic computing, composed of two components:

1) dynamic task scheduling optimized for computation and interactive computational workloads;
2) Big Data collections like parallel arrays, dataframes, and lists that extend common interfaces such as NumPy, Pandas, or Python iterators to larger-than-memory or distributed environments.

DEAP

DEAP is an evolutionary computation framework for rapid prototyping and testing of ideas.

It incorporates the data structures and tools required to implement the most common evolutionary computation techniques, such as genetic algorithms, genetic programming, evolution strategies, particle swarm optimization, differential evolution, and estimation of distribution algorithms.

DMelt

DataMelt, or DMelt, is a software for numeric computation, statistics, analysis of large data volumes (Big Data), and scientific visualization.

It can be used with several scripting languages, including Python/Jython, BeanShell, Groovy, Ruby, and Java.

The library has numerous applications, such as natural sciences, engineering, modeling, and analysis of financial markets.

graph-tool

Graph-tool is a module for the manipulation and statistical analysis of graphs.

matplotlib

Matplotlib is a Python 2D plotting library that produces publication-quality figures in a variety of hard-copy formats and interactive cross-platform environments.

It allows you to generate plots, histograms, power spectra, bar charts, error charts, scatter plots, and more.

Mlpy

Mlpy is a machine learning library built on top of NumPy/SciPy, the GNU Scientific Libraries.

It provides a wide range of machine learning methods for supervised and unsupervised problems, and is aimed at finding a reasonable compromise between modularity, maintainability, reproducibility, usability, and efficiency.

NetworkX

NetworkX is a library for studying graphs which helps you create, manipulate, and study the structure, dynamics, and functions of complex networks.

Nilearn

Nilearn is a Python module for fast and easy statistical learning on neuroimaging data.

This library makes it easy to use many advanced machine learning, pattern recognition, and multivariate statistical techniques on neuroimaging data for applications such as MVPA (Multi-Voxel Pattern Analysis), decoding, predictive modelling, functional connectivity, brain parcellations, or connectomes.

NumPy

NumPy is the fundamental package for scientific computing with Python, adding support for large, multidimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

Pandas

Pandas is a library for data manipulation and analysis, providing data structures and operations for manipulating numerical tables and time series.

Pipenv

Pipenv is a tool designed to bring the best of all packaging worlds to the Python world.

It automatically creates and manages a virtualenv for your projects, along with adding or removing packages from your Pipfile as you install or uninstall packages.

Pipenv is primarily meant to provide users and developers of applications with an easy method to set up a working environment.

PsychoPy

PsychoPy is a package for the generation of experiments for neuroscience and experimental psychology.

It is designed to allow the presentation of stimuli and collection of data for a wide range of neuroscience, psychology, and psychophysical experiments.

PySpark

PySpark is the Python API for Apache Spark.

Spark is a distributed computing framework for big data processing. It serves as a unified analytics engine, built with speed, ease of use, and generality in mind.

Spark offers modules for streaming, machine learning, and graph processing. It's also completely open-source.

python-weka-wrapper

Weka is a suite of machine learning software written in Java, developed at the University of Waikato, New Zealand.

It contains a collection of visualization tools and algorithms for data analysis and predictive modeling, together with graphical user interfaces for easy access to these functions.

The python-weka-wrapper package makes it easy to run Weka algorithms and filters from within Python.

PyTorch

PyTorch is a deep learning framework for fast, flexible experimentation.

This package provides two high-level features: Tensor computation with strong GPU acceleration and deep neural networks built on a tape-based autodiff system.

It can be used either as a replacement for numpy to use the power of GPUs, or a deep learning research platform that provides maximum flexibility and speed.

SQLAlchemy

SQLAlchemy is an open-source SQL toolkit and Object-Relational Mapper that gives application developers the full power and flexibility of SQL.

It provides a full suite of well-known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language.

The main goal of the library is to change the way we approach databases and SQL.

SageMath

SageMath is a mathematical software system with features covering multiple aspects of mathematics, including algebra, combinatorics, numerical mathematics, number theory, and calculus.

It uses Python to support procedural, functional, and object-oriented constructs.

ScientificPython

ScientificPython is a collection of modules for scientific computing.

It contains support for geometry, mathematical functions, statistics, physical units, IO, visualization, and parallelization.

scikit-image

Scikit-image is an image processing library.

It includes algorithms for segmentation, geometric transformations, color space manipulation, analysis, filtering, morphology, feature detection, and more.

scikit-learn

Scikit-learn is a machine learning library.

It features various classification, regression, and clustering algorithms, including support vector machines, random forests, gradient boosting, k-means, and DBSCAN.

The library is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

SciPy

SciPy is a library used by scientists, analysts, and engineers doing scientific computing and technical computing.

It contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers, and other tasks common in science and engineering.

SCOOP

SCOOP is a Python module for distributing concurrent parallel tasks on various environments, from heterogeneous grids of workstations to supercomputers.

SunPy

SunPy is a data-analysis environment specializing in providing the software necessary to analyze solar and heliospheric data in Python.

SymPy

SymPy is a library for symbolic computation, offering features ranging from basic symbolic arithmetic to calculus, algebra, discrete mathematics, and quantum physics.

It provides computer algebra capabilities either as a standalone application, a library to other applications, or live on the web.

TensorFlow

TensorFlow is an open-source software library for machine learning across a range of tasks, developed by Google to meet their needs for systems capable of building and training neural networks to detect and decipher patterns and correlations, analogous to the learning and reasoning employed by humans.

It is currently used for both research and production at Google products, often replacing the role of its closed-source predecessor, DistBelief.

Theano

Theano is a numerical computation Python library, allowing you to define, optimize, and evaluate mathematical expressions involving multidimensional arrays efficiently.

TomoPy

TomoPy is an open-source Python toolbox for performing tomographic data processing and image reconstruction tasks.

It offers a collaborative framework for the analysis of synchrotron tomographic data, with the goal to unify the efforts of different facilities and beamlines performing similar tasks.

Veusz

Veusz is a scientific plotting and graphing package designed to produce publication-quality plots in popular vector formats, including PDF, PostScript, and SVG.

Final Thoughts

Just to make sure we've covered all the bases, let's recap why you should choose Python as your programming language:

- you will write your code faster, optimizing developer resources;
- you will have access to tons of dedicated Python libraries and frameworks, so you won't have to build everything from scratch;
- you will review your code more easily with a simple and clear language;
- you will enjoy support from guides and tutorials by Google and the Python community;
- you will use the same language that Reddit, YouTube, EVE Online, and thousands of global projects have trusted.

Python also offers an extensive selection of web frameworks, all of which have their own strengths and weaknesses. At STX Next, we use whatever framework fits a given project best, even learning new ones on the go if need be. What you should do is choose the one you like the most and delve right into it.

In addition, if it's the many applications of Python in data science or business you're interested in, our favorite programming language has got you covered on that front, too, with its wide range of libraries. We couldn't possibly tell you which libraries to use, though; it depends entirely on your individual needs.

So if you're looking for the one language to rule them all, the choice is clear.

But don't take our word for it! Take a look at our Portfolio and learn the many examples of applying the power of Python to a variety of projects across industries such as fintech, marketing and advertising, or blockchain and crypto.

And if you have any questions about Python, feel free to contact us and we'll answer them for you. We're here to help.

Hire an exclusive Python development team

Accelerate your software project with Europe’s largest Python software house.
For companies with big projects and fast deadlines.

Team Extension

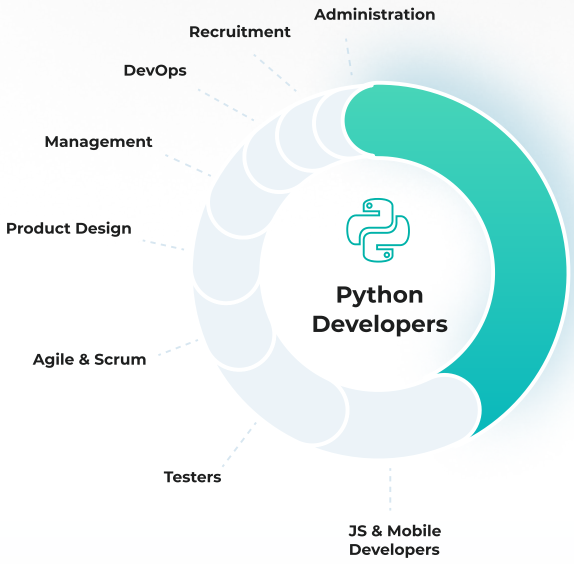
Additional developers or experts supporting your development efforts within 14 days

End-to-End Development

Full development team taking your project all the way from discovery to deployment

Consulting & Expertise

Solving your problems or improving your product with the help of subject matter experts



Over 400 developers

Ready to empower any project with well-reviewed code and a results-driven Agile process

17+ years
market experience

750+
projects delivered

3.5+ years
average partnership

300+
clients served

550+
professionals on board

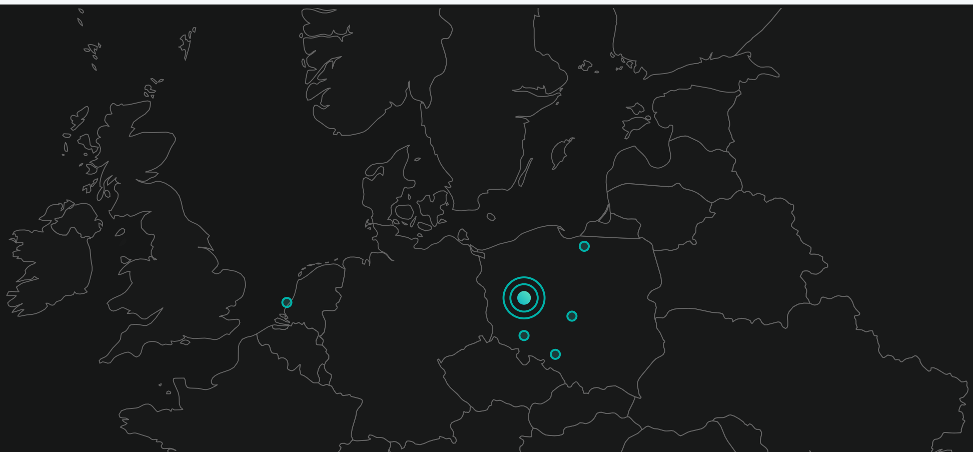
6.5+ years
average experience of our developers

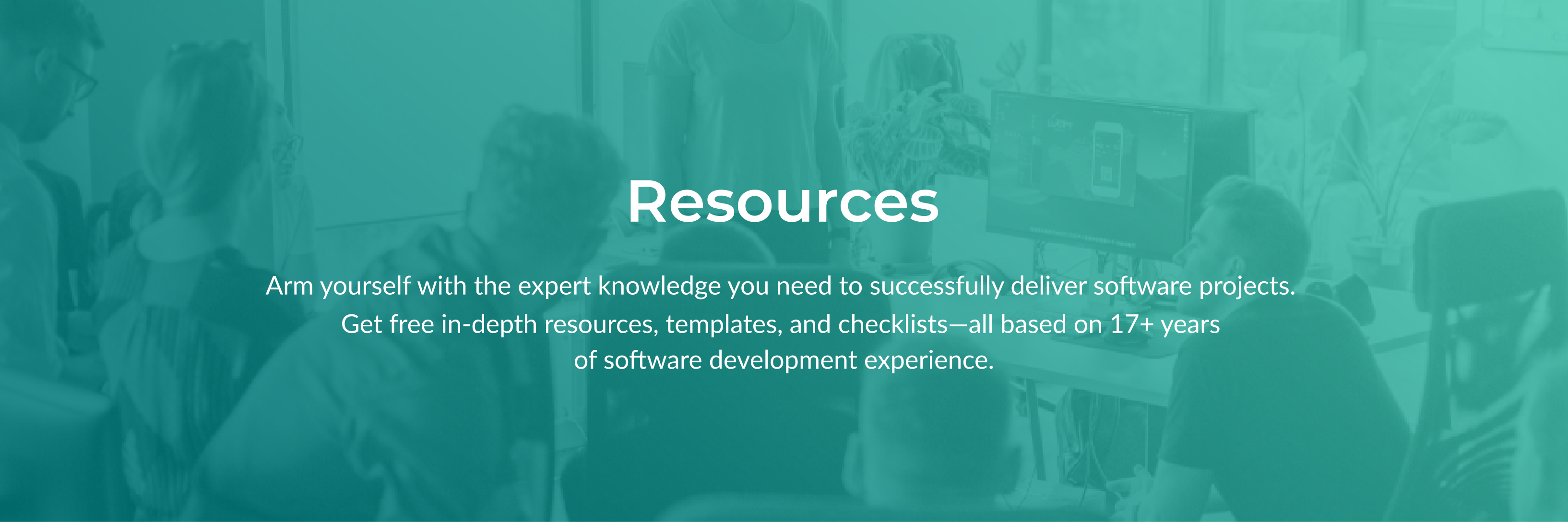
Locations

Poznań (HQ) ◉

Mostowa 38
61-854 Poznań, Poland
+48 61 610 01 92

Wrocław ●
Olsztyn ●
Katowice ●
Łódź ●
Hague (Netherlands) ●





Resources

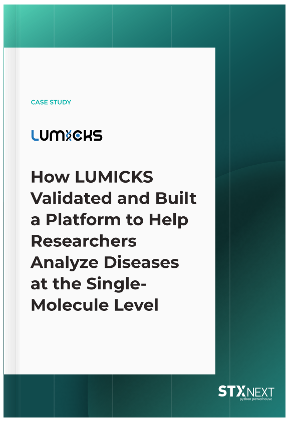
Arm yourself with the expert knowledge you need to successfully deliver software projects. Get free in-depth resources, templates, and checklists—all based on 17+ years of software development experience.

Reports and case studies

TheGlobalCTOSurvey × STXNEXT

SURVEY RESULTS & INSIGHTS

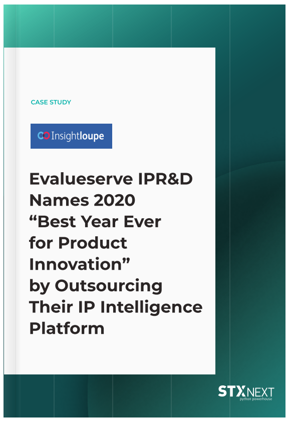
The Global CTO Survey 2021 Report



CASE STUDY

LUMICKS

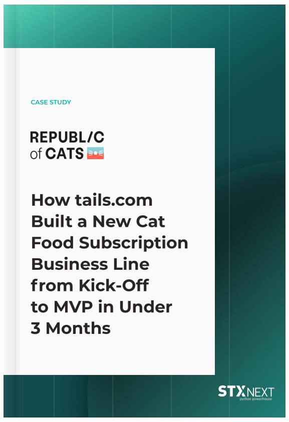
How LUMICKS Validated and Built a Platform to Help Researchers Analyze Diseases at the Single-Molecule Level



CASE STUDY

Insightloupe

Evalueserve IPR&D Names 2020 “Best Year Ever for Product Innovation” by Outsourcing Their IP Intelligence Platform



CASE STUDY

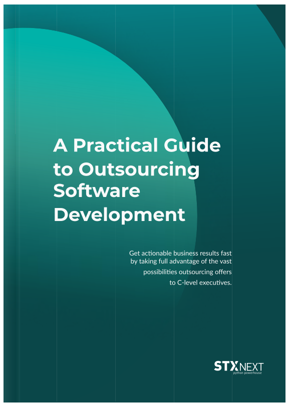
REPUBL/C of CATS

How tails.com Built a New Cat Food Subscription Business Line from Kick-Off to MVP in Under 3 Months

Guides

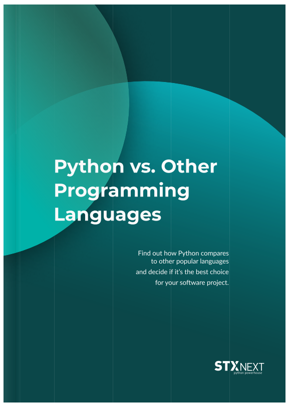
What Is Python Used for?

Discover why Python is such a great fit for so many industries and what areas of technology or business it benefits the most.



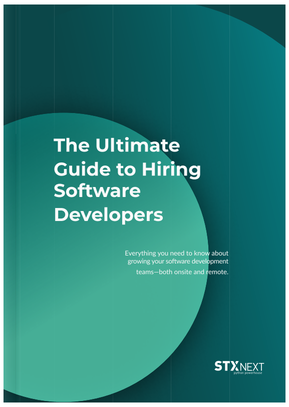
A Practical Guide to Outsourcing Software Development

Get actionable business results fast by taking full advantage of the vast possibilities outsourcing offers to C-level executives.



Python vs. Other Programming Languages

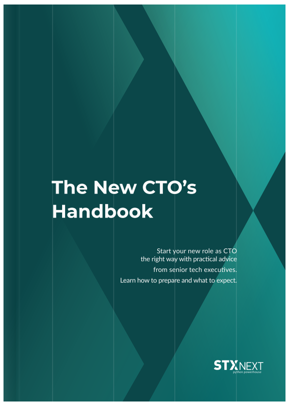
Find out how Python compares to other popular languages and decide if it's the best choice for your software project.



The Ultimate Guide to Hiring Software Developers

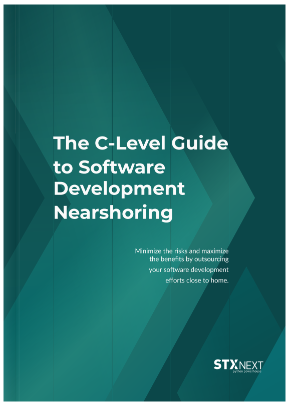
Everything you need to know about growing your software development teams—both onsite and remote.

Ebooks



The New CTO’s Handbook

Start your new role as CTO the right way with practical advice from senior tech executives. Learn how to prepare and what to expect.



The C-Level Guide to Software Development Nearshoring

Minimize the risks and maximize the benefits by outsourcing your software development efforts close to home.

TECH LEADERS HUB

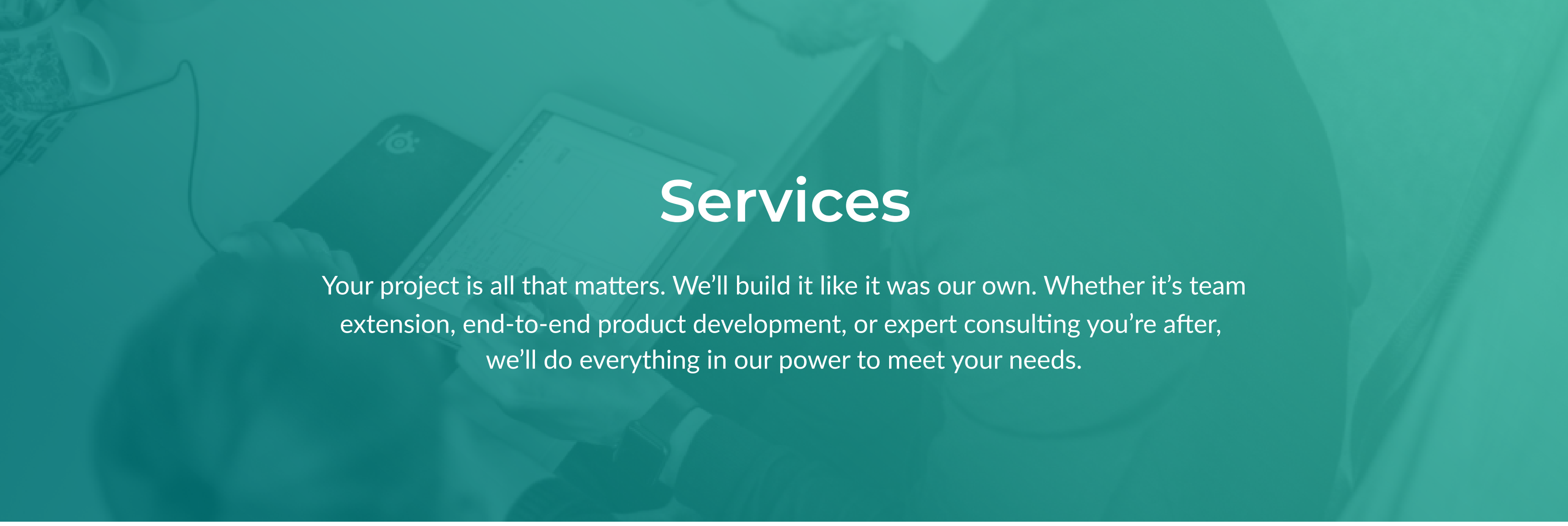
Tech Leaders Hub: Management & Growth

Become a great leader by leveraging the huge experience of tech leadership experts who have spent years managing and growing teams.

The True Cost of Hiring In-House Developers

From training through benefits to paid leave, the cost of adding new members to your team is never just the salary.

DISCOVER MORE →



Services

Your project is all that matters. We'll build it like it was our own. Whether it's team extension, end-to-end product development, or expert consulting you're after, we'll do everything in our power to meet your needs.

- Python Development
- JavaScript Development
- .NET Development
- Web Development
- Software Testing & QA
- Mobile Development
- Django Development
- Node.js Development
- React Native Development
- Fintech Development
- Machine Learning
- Data Engineering
- DevOps
- Product Design
- Discovery Workshops

REVIEWED ON Clutch ★★★★★ 82 REVIEWS

python SOFTWARE FOUNDATION

TOP PYTHON & DJANGO Clutch DEVELOPERS 2021

TOP DEVELOPERS Clutch POLAND 2021

TOP CUSTOM SOFTWARE DEVELOPMENT Clutch COMPANIES 2021

Tell us about your project

Speed up work on your software projects and outpace the competition.

HIRE US →

Marta Błażejewska
DIRECTOR OF SALES
marta@stxnext.com
+48 506 154 343

Sebastian Resz
HEAD OF SALES
sebastian@stxnext.com
+48 690 433 578

Follow us