The Productive Advantage of the Python Programming Language

How Python can add value to your business process

SPHEREINC.COM

How Python can add value to your Business Process

Python ranks among the most popular and fastest growing programming languages.[1] It has risen to the top language for coding education and technical interviews.[2] Subsequently, a growing portion of developers now enter the industry with Python among the main development tools at their disposal.

So, should your organization use Python? It depends on how your company operates. The language uniquely balances agility and power, supporting a rich package library and web ecosystem developed by a vibrant community, lending itself to many different business processes. This wide array of specialized features, along with its easy setup and natural, succinct syntax, empower speedy delivery and maintenance for your company's continued convenience.[3] This makes Python ideal for experimentation, rapid prototyping, scripting, iteration and internal tools in a variety of problem spaces. Python boasts particularly powerful packages for data science, game development and embedded systems for the Internet of Things as well as supporting web frameworks in widespread use throughout a variety of different kinds of businesses.

Python's general purpose and ease of use does not preclude its performance and scalability. In fact, huge development teams employ Python to build products, services and experiences that serve millions of users every day. In fact, this program exemplifies built-in assets that can enhance how a business functions, even when its verified weaknesses are taken into account. As a rule, a programming language's optimal performance is usually evaluated in terms of compute-bound versus I/O bound regimes. Due to certain language design choices, Python does at first appear less performant for certain compute-bound applications at scale. However, today's applications are usually I/O bound anyway and a well-informed developer can manageably optimize around any of Python's performance drawbacks.

The Pythonic Approach

Python is primarily object-oriented (OO), representing all data as objects such that one can pass around structures, functions or modules all in the same context. However it is multiparadigm, supporting procedural and functional styles as well. Developers coming from chiefly OO languages can maintain OO's modular design patterns, switching to the procedural form in cases such as executing simple print statements or functional for MapReduce applications.

```java
/* Java */
public class HelloWorld
{
  public static void main (String[] args)
  {
     System.out.println("Hello world!");
  }
}
```

```python
print "Hello, world!"  # Python version 2
print("Hello, world!") # Python 3
```

Example 1: OO vs. procedural style for printing "Hello, world!"

```
>>> some_list = [1, 2, 3, 4, 5]
>>> reduce(lambda x, y: x*y, filter(lambda x: x>0, map(lambda x: x - 1, some_list)))[4]
24
```

Example 2: Compute the product of all the numbers 1 less than the elements in a list, excluding any numbers equal to 0

Python's standard library includes built-in tools for "database functionality, a variety of data persistence features, routines for interfacing with the operating system, website interfacing, email and networking tools, data compression support, cryptography, xml support, regular expressions, unit testing, multithreading, and much more".[5] With these integrated resources made available for use, your business can take advantage of centralized mechanisms that are fundamental to so many operations. To be sure, Python's flexibility allows developers to achieve different programming tasks within the same language while providing necessary organizational functions. Its concise syntax closely resembles natural English by expressing routines through brief, highly readable idioms, referred to as a "pythonic" style. Compare readability of Python versus C for looping through elements of a list.

```c
/* C */
for (i=0; i < mylist_length; i++)
{
   do_something(mylist[i]);
}
```

```python
#Python
for element in mylist:
   do_something(element)[6]
```

Example 3: Python syntax strives to read like natural English

In natural English, we learn predicates like “add” to describe actions on numbers, but readily apply the same sense of meaning when we “add” groceries to our shopping cart. By analogy, Python’s natural grammar derives from predicates, which don’t depend on the type of subject. From its dynamic typing system, Python derives language polymorphism: methods adapt to the type of data they process. Statically typed languages require explicit type declaration for all variables. In Python, everything has an implicit type assigned by the object it binds to as a rule. Variables simply point to addresses in memory. We can reassign variables to objects of a different type at any time and Python simply replaces the value referenced by the variable’s address with the new object. Objects can have any type and Python containers, such as lists or dictionaries, may store objects of any kind. No type declarations (and simple typecasting methods) make Python faster to write or easier to read.

So why does the natural English syntax matter for your business? It’s important because your business will enjoy more effective functions that are streamlined and easily adaptable. The integration of universally recognizable English makes Python more intuitive, and accessible to its users.

Furthermore, since Python methods check types implicitly and not based on the variable itself, the same function can take arguments of different types. Objects perform operations themselves so methods can fulfill the same operations, regardless of input type. Programs also avoid compilation errors due to type incompatibility without having to cover every possible type case separately. The following secret number game does not compile in Rust due to a comparison between strings and integers. But the Python program executes unhindered, reads easily and can handle type errors at runtime. Thus, background processes will not slow you down.

```
>>> a = 9
>>> b = «9»
>>> str(a) + b
‘99’
>>> a + int(b)
18
>>>
```

Example 4: Use simple typecasting to switch between concatenation and summation sense of the “+” operator

```rust
// Rust
extern crate rand;
use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    let secret = rand::thread_rng().gen_range(1, 101);

    println!("Please guess secret number");
    println!("Hint: secret number is {}", secret);
    println!("Please input your number");
    let mut guess = String::new();
    io::stdin()
        .read_line(&mut guess)
        .expect("failed to read");
    println!("your guess is: {}", guess);

    match guess.cmp(&secret) {
        Ordering::Less => println!("too small"),
        Ordering::Greater => println!("too big"),
        Ordering::Equal => println!("you win!")
    }
}
```

```python
#Python
import random
# use six because input behaves differently in Python 3 and 2
import six

def main():
    secret = random.randint(1, 101)
    print("Guess secret number")
    print("Hint secret number is {}".format(secret))
    guess = six.moves.input("please input your number")
    print("Your guess is {}".format(guess))

    def compare(guess, secret):
        if guess == secret:
            print("you win")
        elif guess > secret:
            print("too big")
        elif guess < secret:
            print("too small")

    compare(guess, secret)

if __name__ == "__main__":
    main()
```

Python's expressiveness brings its mental and labor overhead down significantly. An open-source language with strong community support, Python sets up in seconds and can run lines of code directly in terminal via its interpreter. Due to Python's brevity, readability and developer-friendliness that makes is so quickly iterable, some estimates argue Python makes developers 5 to 10 times more productive.[8] This productivity will carry over to your organization, increasing its overall efficiency.

Example 5: Python enjoys forgiving yet informative error handling[7]

Python for Scripting and Prototyping: Biopharmaceutical Firm Case Study

Python's expressive, easily iterable nature makes it highly suited to rapid prototyping, proofs of concept and internal tools easily adaptable to shifting operational needs. For these essential reasons, Python is routinely used to facilitate tasks for a wide variety of businesses in numerous industries. The following case study offers key insights into why Python has the capacity to make such a profound difference in the way companies operate.

At one research biopharmaceutical firm with 54,000 employees worldwide, experimental chemists use computational models to identify compounds as potentially effective medicine. Historically, every prediction technique ran on a separate program, each with its own set of inputs, options and error handling. Unable to work with these concepts themselves, experimenters had to wait on computational chemists to run routine models on their behalf.

The firm wanted experimenters to run their own predictions, freeing the computational chemists to turn research over to their lab faster. Part of the company had a previously successful experiment in this regard by fronting their system of computational Perl scripts with a web-based interface. They used Python to refactor the backend, optimizing for reusability, maintainability and robustness. The firm understood Python was ideal for physical scientists without a specialized coding background. Python was designed to be easy to learn and use for the novice programmer yet powerful enough to tackle real-world challenges met by professional programmers. This results in a language that scales from short scripts written by chemists to large packages engineered by dedicated developers. (Refer to the "Scython" section on page XX for more on scientific computing in Python.)

Python's focus on learnability facilitated easy error handling aided the newly refactored process. Perl suppresses error messages except with explicit checks, adding keystrokes, cognitive overhead and more tricky tests. By contrast, Python raises an exception with any execution issue, including the complete stack traceback, helping quickly identify and fix problems without reliance on extra debugging tools. For instance, a prediction program, which normally returned numerical error values, would sometimes return simple "error" strings. Perl would automatically convert these results to 0 integers.

The previous system would recognize these returns as valid, propagating faulty results. On the other hand, due to Python's strong typing system, which clearly distinguishes between None, 0, and the empty string, the new process revealed the bug as soon as it occurred.

Furthermore, Python's typing gave the new process previously impossible levels of extensibility. To reuse results for molecular properties engaged in other property calculations, the program developed a manager, a dictionary-like object which caches property results and maps properties to their relevant prediction function. The now streamlined system could now handle all current and future prediction methods while remaining easily understandable and maintainable. Python's dynamic typing meant this system could mix properties represented as numbers, strings, containers or class instances while its strong typing fostered easy type checking to eliminate mismatch errors quickly. The process's user-defined predictions feature benefitted from Python's arithmetic expressions, which look almost exactly like scientific formula, combined with its built-in eval() function. Overall, the 5,600-line project took 3 months to develop, and another 3 to QA and document.

As the above example demonstrates, Python has the ability to adapt to an organization's unique needs. Its built-in error-checking and prediction tools are intuitive and powerful, allowing even novice programmers to jump in and tackle important business challenges.

Python Performance

In basic terms, Python achieves great success at quickly adapting critical components of internal and enterprise systems to changing needs. However, the language possesses vast power beyond scripting and rapid prototyping with huge teams developing large-scale, sophisticated, general purpose applications in every space. Recall from the introduction that Python's shortcomings manifest mostly in compute-bound applications. Applications such as Google's Search and App Store tend to use Python as "glue" for communication among the more computationally intensive system components. By contrast, YouTube, by any metric a compute-intensive application at scale, runs primarily on Python. A conscientious developer can always achieve performant software in Python, often in less time and usually with less code than in Java or C++ due to its penchant for quick iteration.[9]

Generally speaking, Python development teams must balance programming efficiency with performance. Based on our experience, labor will usually present a project's largest cost until its later stages of growth. Therefore, Python's productivity puts it at a potential advantage over more performant languages when starting a project. To strategize language use as a project scales, such that huge dev teams the likes of Dropbox, PayPal, JPMorgan and Bank of America can use Python as a core technology, we need to examine its execution.

Python's speed will depend on its implementation. In its native implementation, CPython, programs are compiled into bytecode and then executed with an interpreter. This gives Python applications more platform independence since translation happens on the final machine. In fact, Java's virtual machine (JVM) interprets Java bytecode into native machine code to make its applications effective on a cross-platform basis. Interpretation also lends flexibility to the development process: changes to source code execute immediately, exposing bugs and integrating new features quicker. The edit-interpret-debug development cycle outperforms the edit-compile-run-debug cycle.

However, interpretation generally executes slower than compilation since it must translate code into machine code subroutines. Unpacking Python objects takes more computational steps than executing assembly, adding runtime overhead. To overcome this, Python's PyPy implementation uses just-in-time (JIT) compilation, which dynamically analyzes code to execute it using an optimal balance of ahead-of-time compilation and interpretation. PyPy approaches and in some cases outperforms compiled languages.[10]

Additionally, Python's numpy package can outperform corresponding naive implementations in C/C++ for fundamental algorithms like matrix multiplication[1112] and standard deviation.[13]

Performance analyses should also consider type system. Dynamically typed languages can only reveal type errors at runtime, slowing code inspection and debugging for sufficiently complex systems in most cases. To avoid type errors, type checks must occur at each runtime instance rather than once at compile time, adding computational overhead. The Cython implementation uses static typing to combine its performance gains with Python's expressiveness.

In order for your business to achieve the best results, it's critical to use side-by-side performance benchmarking to compare optimized CPython, PyPy, and Cython implementations against other compiled or statically typed languages. For compute-intensive components not fast enough in Python, you can choose more performant language to rewrite in where savings from performance gains surpass the cost of lower productivity.

Specialized Libraries & Frameworks

PyPI, Python's package index, contains over 80,000 modules, which Python interfaces with in "a human-friendly way."[14] In essence, this means your organization can save time by reusing code from these modules to make Python even more productive.

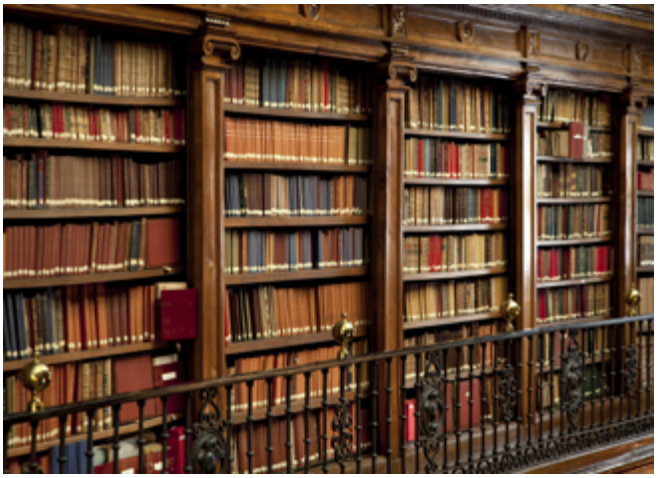
Scython

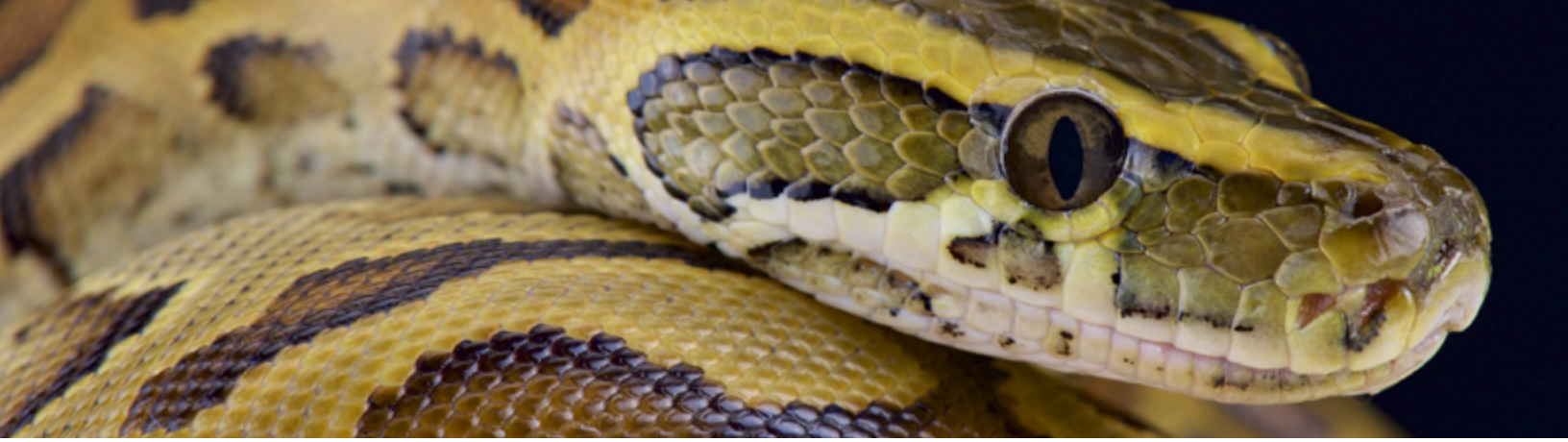
The term "Scython" is often used to refer to Python's rich collection of packages for data analysis and scientific computing. Python has grown in use for data science with 54% of O'Reilly Data Science Survey respondents saying they chose Python in 2016 versus 51% in 2015.[15] The standard library sqlite3 facilitates easy transition from SQL for relational database programming. Numpy, Scython's foundational third-party library, which provides an optimized N-dimensional array object and C/C++ and Fortran integration, comes in the SciPy stack along with pandas for data mining, scikit-learn for machine learning and other core packages.

In order to provide maximum benefits to your business, you can use networkx for graph storage, analysis and visualization or nltk to analyze text with natural language processing. Python's extensive set of data science libraries have led firms like Continuum and Enthought to focus solely on Python and DataCamp to offer a dedicated Python course. The approach taken by these large corporations can also apply to your business, offering further advantages for company performance.

Embedded Systems

These data crunching abilities, its ease of use and cross-compatibility, have made Python the fastest growing language for embedded systems and the Internet of Things, including the official language for Raspberry Pi.[16] Since embedded systems are often developed by young coders or hobbyists, Python's superlative popularity in computer science education indicates this growth will continue. Your company can also be strengthened by the embedded systems that are integral to Python's foundation. Consider employing Python's readability of script communication to and between systems for user configuration or automated testing.





Web Frameworks

Many widely used web frameworks, including Django, Flask, Tornado and Pyramid, run on Python. Django powers Eventbrite, Instagram (80 million users) and Disqus. It serves 8 billion page views a month, 45k requests a second[17] and scaled from 250M to 500M users on the same 100 boxes, which run on Django. Twilio, Netflix, LinkedIn and Uber have selected Flask for microservices and internal applications. Quora, bit.ly and hipmunk chose Tornado. Pinterest (20M users) combines all three, running Tornado in Django instances[18] and employing Flask for its API. Pyramid helps power cars.com, Yelp, and Mozilla.[19] Your company can also benefit from the model set by these major corporations, taking full advantage of the web frameworks for improved strategic positioning.

Python for Your Organization

In conclusion, the next time you try to innovate or improve a business process at your organization, write the technology to power it in Python. Its ease of use, adaptability, and growing popularity will ensure that your entire workforce, from experienced professional developers to fresh technical hires to intrapreneurs in non-technical departments, will love to build, maintain and engage with this versatile tool. Designed to optimally balance learnability, expressiveness and power at optimal levels, Python can quickly take experimental solutions from rapid prototype to full-fledged product in the same language. All of these features combine seamlessly into a language that offers unparalleled productivity. Coding your next project in Python will bring its labor overhead, giving your innovation strategy a competitive advantage both now and for the foreseeable future.

About Sphere Software

With over a decade of proven success delivering innovative custom software solutions, Chicago-based Sphere builds solutions in software, web, mobile and big data analytics while ensuring the highest level of customer service and satisfaction. Sphere provides unparalleled project management and thought leadership, forming a true partnership with each and every client. In addition, Sphere's deep technical acumen includes Python, Ruby on Rails, Scala, Go, Polymer, Clojure, .NET, Java, Node.js, React.js (to name just a few), and a broad range of industry experience as well.

Sphere believes in complete transparency, open collaboration and communication, including open client access to the same internal collaboration tools used by our development team. Furthermore, Sphere provides full client access to project status updates and source code.

To find out how Sphere can help your business reach peak performance, contact us on our website at www.sphereinc.com

1. http://pypl.github.io/PYPL.html
2. http://blog.codeeval.com/codeevalblog/2016/2/2/most-popular-coding-languages-of-2016
3. https://www.codeschool.com/blog/2016/01/27/why-python/
4. http://stackoverflow.com/questions/27222193/clean-code-for-sequence-of-map-filter-reduce-functions
5. https://www.stat.washington.edu/~hoytak/blog/whypython.html
6. https://blog.startifact.com/posts/older/what-is-pythonic.html
7. https://pawelmhm.github.io/python/static/typing/type/annotations/2016/01/23/typing-python3.html
8. https://pythonconquerstheuniverse.wordpress.com/2009/10/03/python-java-a-side-by-side-comparison/
9. https://www.paypal-engineering.com/wordpress/wp-content/uploads/2014/12/cpp_py_medium.png
10. https://morepypy.blogspot.com/2011/08/pypy-is-faster-than-c-again-string.html
11. http://stackoverflow.com/questions/10442365/why-is-matrix-multiplication-faster-with-numpy-than-with-ctypes-in-python
12. http://stackoverflow.com/questions/41365723/why-my-python-numpy-code-is-faster-than-c
13. http://notes-on-cython.readthedocs.io/en/latest/std_dev.html
14. https://opensource.com/life/16/8/python-vs-cc-embedded-systems
15. https://www.continuum.io/blog/developer-blog/learning-python-data-science-cheat-sheets
16. https://opensource.com/life/16/8/python-vs-cc-embedded-systems
17. https://blog.disqus.com/scaling-django-to-8-billion-page-views
18. http://www.revolunet.com/static/django-success-stories/#10
19. https://trypyramid.com/community-powered-by-pyramid.html