**Analytical Analysis of a Programming Language**

*Name: Stephen Griffin*

*ID: 18482934*

*“Jack of all trades, master of none!”*

**Introduction**

The programming language that I will be analysing is the Python programming language. Python is a very popular and well-known language and is an example of a high-level language meaning it is closer to language spoken by humans than what the CPU reads. This makes it easy to understand, especially the syntax which is very neat and tidy. For the essay I will be looking at Python3 rather than 2.7 as this is considered the standard and most complete version.

The Python programming language was first published in 1991 by a man named Guido van Rossum.[[1]](#endnote-1) Guido had a number of goals for his new language, the first goal was for Python to be easy to use but also as powerful as other languages. Another goal was for Python to be open source, this meant people from around the world could contribute to its development and improvement. Now, 29 years later Python is ranked the third most popular programming language in the world according to the TIOBE index. It is narrowly behind C in first place and Java in second.[[2]](#endnote-2)

So why has Python become so popular? A huge part of Pythons success has been due to how easy it is to use. Someone who has never programmed before will need less time to learn it and start programming as they would in a language such as C or Java. For the same reasons as why it’s easy to learn, Python is easier than most to teach. Another benefit which some languages don’t have is that its multiplatform and can be used on most operating systems. [[3]](#endnote-3)

**Argument Passing**

There are two main ways in which programming languages perform evaluation when passing arguments to functions; call-by-value and call-by-reference. To understand the way in which Python does it we will look at both then compare to Pythons method. In the process of call-by-value, the result of the argument expressions evaluation is set to a variable in the function. This way it uses a local copy without changing the variable it took in. This is a default for programming languages such as C# and C++ [[4]](#endnote-4)

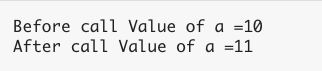
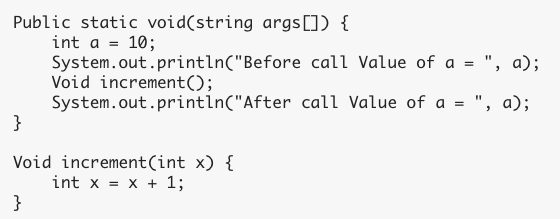
*Text

Description automatically generatedA picture containing text

Description automatically generated*

*An example of a function using call-by-value[[5]](#endnote-5) Output*

The other method, call-by-reference is when the function gets reference to the argument, instead of just a copy of the argument. The implication of this is that the function is then able to change the arguments value. This is seen as efficient with respect to space as instead of copying the variable and then working with that, we are changing the original. The downside of this method is that sometimes the variables that are being used can be changed by accident. This way of passing arguments is a default in the language Perl but is supported in languages such a C# and C++[[6]](#endnote-6)

**

*An example of a function using call-by-reference[[7]](#endnote-7) Output*

Finally, the way in which Python deals with passing arguments: call-by-object.Call-by-object is in some ways a combination of both methods. Object reference is used to pass the arguments to the function. What matters here is whether the arguments passed are immutable or mutable. When you pass a function immutable arguments such as integers and strings it performs similar to call-by-value. This is because these arguments can not be changed in the first place. When passing mutable arguments which can be changed, object reference is again used, for example a list where two things have to be considered: the lists contents can be changed in-place and if a new list is created with the same name, the old list stays the same. To summarize Python will use call-by-reference until the variable is being changed, then it changes to call-by-reference[[8]](#endnote-8)

**A picture containing scatter chart

Description automatically generatedText, letter

Description automatically generated**

*Example of call-by-value in Python[[9]](#endnote-9) Example of call-by-reference in Python*

**Object-oriented Programming(OOP)**

Object oriented programming is a way of structuring programs using objects. An example of an object might be a person. This person could have properties like a name, an age and a phone number. A big part of object oriented programming is the use of classes. A class is usually made up of functions which are called methods. A class can be viewed as a blueprint showing how to define something. For example a class person can say we need an age name and phone number but wont actually hold those variables, that job belongs to a class instance. An instance can be described as a class being used. Instances can be created like so: [[10]](#endnote-10)

A picture containing text

Description automatically generated

Now that we have looked at the basics of object oriented programming, does Python do it well? How does it compare to other languages in that respect? For this comparison I will be looking at Python versus Java. An important thing to note about Java is that it does not have functions in the way that Python does. Java works with methods which are part of a class. In Python we can create independent functions without classes. As you see above, we defined our attributes inside def \_\_init\_\_ . In Java this is done differently, we have to make a constructor because attributes must be defined before use.

**Text

Description automatically generated**

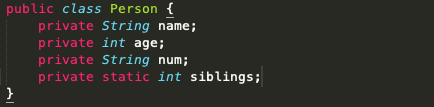
Class attributes are also somewhere where the two languages differ. For example lets say we want a variable for how many siblings our person has, in python we simply define a

variable inside the class but outside of our \_\_init\_\_ like so:

**Text

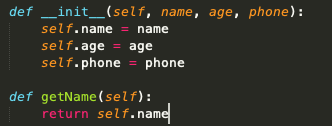
Description automatically generated**

This process is different in Java, it uses static attributes. Here is how we create our static attribute which is equivalent to a class attribute in Python.

****

This comparison is a good example of how Python is not as suited to object oriented programming as Java is. Java allows for attributes to be defined as private, meaning they limit access from outside the class. You can use public methods to get access to private variables. This limitation of access is ideal for object oriented programming because it often stops you from accessing and overriding variables accidently. Python doesn’t support the same idea of having public and private, everything is considered public. [[11]](#endnote-11) This is a good example of how Python has tried to be the jack of all trades, it has left out a very useful feature in order to make it easier to understand.

Another difference is the use of ‘self’ in Python versus the use of ‘this’ in Java. Self in python is needed if you want to access an attribute. If we don’t use self, Python creates a local variable. As we can see here, self is being used to access the attribute ‘name’.



In Java, the use of ‘this’ is usually not necessary in the way that self is in Python. As long as there are no other variables with the same name, ‘this’ doesn’t need to be added. This method will set the name variable without the use of ‘this’.

**Graphical user interface, text

Description automatically generated**

**Type Checking**

In Java every interface and class is a type, two objects implementing the same interface are considered the same type. This feature of Java allows for the implementation of Polymorphism as you can interchangeably use different classes. Python does type checking very differently, it uses a much more simple concept called “duck typing”. This follows the rule that “if it walks like and duck and quacks like a duck, then it’s a duck”. Python looks at how objects behave in order to determine their type.[[12]](#endnote-12) Here is an example of duck typing in Python with its output. Calling len() on the created object returns the value of the \_\_len\_\_ method which as you can see from output is 50.

**Text

Description automatically generated**

**Operator Overloading**

Operator Overloading is one example of a functionality that Python has that some other languages do not, which is unusual as Python tries to include a lot without necessarily doing it better than other languages. Operator overloading makes use of Pythons magic methods, two of which we have already seen, \_\_init\_\_ and \_\_len\_\_. For example if you want to add two objects that are classes you can use operator overloading. Here I have extended the Person class to include an example of operator overloading.

**Text

Description automatically generated**

**Code Execution**

A common criticism of Python is that it is slower than other languages at executing code. Python is often praised for how dynamic it is and its versatility, but that comes at a cost of slower execution. There are a few reasons as to why code execution takes longer in Python, one of them is the GIL(Global Interpreter Lock). This is a global lock on the interpreter for when a running thread is working with a shared resource. The lock exists to stop multiple threads trying to change a resource at the same time. Programs can take longer to run because the interpreter is temporarily being locked to stop the threads clashing.[[13]](#endnote-13) This argument only applies to programs which have implemented concurrency.

Another reason Python suffers from slower execution is that its dynamically typed. This means that Python has the concept of types but variable types are dynamic. The opposite of this is a statically typed language, C and Java are examples of this because you have to state the variable type when you create it.[[14]](#endnote-14) Here is an example of creating variables in Python.

****

As this runs, a second variable with the same name and type string is created, the memory which was created for the first a is also deallocated. Converting types and comparing types is a costly operation, Python checks the type every time it writes or reads a variable.

**Reference Counting & Garbage Collection**

To look at these two Python memory management concepts we must briefly look at CPython. CPython is the reference implementation of Python. Most Python users use this implementation but there are a few less common ones. [[15]](#endnote-15) Its purpose is to compile the Python code you write into bytecode then interpret it.

The idea of reference counting is that whenever you create an object in Python, there is a C object created that has a Python type and a reference count. Every time your program references a given object, the reference count in increased. This also applies to when an object is dereferenced, the reference count is decreased.[[16]](#endnote-16) If at some point the reference count of a given object drops to 0, the memory that was assigned to it is deallocated. Here is an example where I print the reference count of a string I created.

Graphical user interface, application

Description automatically generated

The other main memory management concept used in Python is garbage collection. In the following example an instance of the Garbage class is created on itself, then the instance is deleted. The only problem is, it hasn’t been deleted from memory because the reference count didn’t hit zero as it has a reference to itself. This problem is solved by the garbage collector, one which basic reference counting cannot solve.[[17]](#endnote-17)

**Text

Description automatically generated**

The idea of garbage collection is that an object starts in a first generation, if garbage collection executes on a certain generation and an object remains, it moves to the next generation and so on up until the third generation. For each generation, the garbage collector as a limit of objects, whenever the limit is reached a collection process begins, objects which survive have their generation moved on. [[18]](#endnote-18) Garbage collection differs from reference counting in that the programmer can actually initiate a garbage collection process and also disable it. This can be done with the “gc” module, here is some code using the module to get a count of how many objects are in each generation currently:

**Graphical user interface, application

Description automatically generated**

**Disorganised Build System**

The build system is another big problem people often have with the Python programming language. For most projects you will want to import dependencies that you will use in your programs. There are a lot of Python versions so when working in a group, it’s best to use a non system-installed version which has been decided on. Python modules are generally packaged in two ways, “wheel” or “egg”. Wheel is considered standard for packaging. After the development stage of a project, when it comes to building, there can be problems because a given module might have a number of dependencies needed to compile, but where ever the project is being built could rely on a C compiler in order to run your module. These dependencies can be put into a container image, but you end up with artifacts that take up a lot of space.[[19]](#endnote-19) Overall the installing of packages in Python is not clean, for example importing a module often needs a large amount of supporting code.



*An example of a build failure[[20]](#endnote-20)*

**Mobile Development**

With the popularity of smart phones in recent years, mobile app development suitability is an important factor to consider when looking at a programming language. One of the biggest downsides to Python is that neither Apple iOS nor Android support it as one of their official programming languages for their platforms. Python can still be used to develop applications on these two major platforms but it would require a huge amount of extra effort and time when compared to making a mobile application in Java for Android or Swift for iOS.[[21]](#endnote-21) Another reason why Python isn’t considered for mobile development is speed. Due to their size, mobile devices have a lot more limitations on processing power and memory than a Desktop PC for example. Faster languages such as Java work better to provide mobile users with a more responsive environment as applications will execute faster.

**Development Times**

One of the major advantages to working in Python is the development times. Some companies might consider delivery speed as one of the most important factors when it comes to choosing a language as they may be working to strict deadlines, here Python flourishes. Due to the diversity of Python it can be used throughout the various development stages of building an application. For example it can be used to build the backend, then test it with Pythons UnitTest features. If too many other languages had to be used at the different stages, it might be necessary to employ a lot more developers with different skill sets, whereas if you are using Python at multiple stages of the development process, you could complete it with a smaller team. This translates into a cheaper project as costs for developers are down. Here is a visual representation of how long it takes to write a string processing application in different languages, as you can see Python is second quickest.[[22]](#endnote-22)

**Chart, bar chart

Description automatically generated**

**Conclusion**

In conclusion, it is fair to consider Python as a “jack of all trades and master of none” due to its eagerness to be versatile. Python looks weak when execution times are compared to other languages and also doesn’t support Object Oriented Programming as well as other languages such as Java. Pythons speed problems has also had the knock-on effect of making it unsuitable to mobile application development. With these disadvantages stated, there are also some very positive characteristics such as quick development times, operator overloading, and it’s simplicity to understand, learn and teach.

1. Python Institute <https://pythoninstitute.org/what-is-python/> [↑](#endnote-ref-1)
2. TIOBE Index <https://www.tiobe.com/tiobe-index/> [↑](#endnote-ref-2)
3. Python Institute(why python) <https://pythoninstitute.org/what-is-python/> [↑](#endnote-ref-3)
4. Python Course(call by value) <https://www.python-course.eu/passing_arguments.php> [↑](#endnote-ref-4)
5. Guru 99(call by value diagram) <https://www.guru99.com/call-by-value-vs-call-by-reference.html> [↑](#endnote-ref-5)
6. Python Course(call by reference) <https://www.python-course.eu/passing_arguments.php> [↑](#endnote-ref-6)
7. Guru 99(call by reference diagram) <https://www.guru99.com/call-by-value-vs-call-by-reference.html> [↑](#endnote-ref-7)
8. Python Course <https://www.python-course.eu/passing_arguments.php> [↑](#endnote-ref-8)
9. Python Course(python diagram) <https://www.python-course.eu/passing_arguments.php> [↑](#endnote-ref-9)
10. Real Python <https://realpython.com/python3-object-oriented-programming/> [↑](#endnote-ref-10)
11. Real Python(OOP java vs python) <https://realpython.com/oop-in-python-vs-java/> [↑](#endnote-ref-11)
12. Real Python <https://realpython.com/oop-in-python-vs-java/> [↑](#endnote-ref-12)
13. Real Python <https://realpython.com/python-memory-management/> [↑](#endnote-ref-13)
14. Medium <https://medium.com/hackernoon/why-is-python-so-slow-e5074b6fe55b> [↑](#endnote-ref-14)
15. CPython Wiki <https://en.wikipedia.org/wiki/CPython> [↑](#endnote-ref-15)
16. Stackify <https://stackify.com/python-garbage-collection/> [↑](#endnote-ref-16)
17. Stackify(generational gc) <https://stackify.com/python-garbage-collection/> [↑](#endnote-ref-17)
18. Stackify <https://stackify.com/python-garbage-collection/> [↑](#endnote-ref-18)
19. Medium <https://medium.com/@natemurthy/all-the-things-i-hate-about-python-5c5ff5fda95e> [↑](#endnote-ref-19)
20. Medium(build error) <https://medium.com/@natemurthy/all-the-things-i-hate-about-python-5c5ff5fda95e> [↑](#endnote-ref-20)
21. NetGuru <https://www.netguru.com/blog/python-vs-java-comparison> [↑](#endnote-ref-21)
22. Monterail <https://www.monterail.com/blog/is-python-slow> [↑](#endnote-ref-22)