CA341 - Assignment 3

Analytical Analysis of a Programming Language

Adrian Irwin - 20415624

Introduction

For my analytical analysis of a programming language, I have decided to analyse Python, specifically Python 3. Python is an interpreted high-level language that is dynamically typed and has garbage collection. Python does not stick to one specific programming paradigm and supports multiple paradigms.

Interpreted Language

Python is an interpreted language but uses a hybrid implementation of an interpreter and a compiler. A hybrid interpreter doesn't explicitly go through each line of the code and unpack them into a series of actions and then execute them. Instead, the interpreter first checks the code to see if there are any syntax errors[1], if there are none it compiles the code into bytecode. Unlike a compiled language the resulting code can still have errors after being compiled into bytecode. The interpreter will then execute the code using the translated bytecode. With the use of an interpreter, if an error occurs the interpreter is more easily able to give us the location of where the error occurred because the interpreter stays with the code for the execution of the code.[2]

Dynamic Typing

In contrast to other popular languages such as C++ or Java, which are statically typed, Python is dynamically typed. This means that when defining variables the type associated with the variable does not have to be specified. Static typing is more efficient and secure but is less flexible and is harder to work with when the data types that will be used are not known.

Example: Creation of an integer variable with the value of 12 in Python

```python
number = 12
```

As seen in the above example, when creating a variable the type does not have to be declared, while in statically typed languages the type would have to be explicitly defined before the variable name. Although this makes the code easier to learn and understand this leads to the program being less efficient and slower. However, it is more flexible and allows for the program to be more dynamic and adaptable to changes in the data types that are used.

The dynamic typing of variables is done at runtime and not at compile time. As Python uses a hybrid interpreter implementation that only checks for syntax errors before translating the code into bytecode, the interpreter won't find the errors until it finally runs the code. This can lead to errors in the program if the wrong type is used. For example, if a string is passed to a function that expects to take an integer, the program will crash during the running of the program.

Memory Management

Python uses automatic memory management. The use of automatic memory management over memory management defined by the developer allows the developer to spend more time focusing on how the program will work and run instead of trying to find the most efficient way to manage its memory. Automatic memory management gives the programmer less control over how memory is allocated but does lead to not having to deal with memory leaks or dangling pointers.

The main automatic memory management method used by Python is called reference counting[3]. For each object that is created the interpreter will create a count for how many times it is referenced and this number will be associated with the object in memory. If an object is referenced the number is incremented, if it is dereferenced the number is decremented. Once the number of references to an object reaches zero, the memory associated with the object is deallocated. As seen below and shown in the Python Developer's Guide[4], Python implements a function in the sys library for a developer to see how many references an object has, this function adds one to the overall value as the function includes its own reference to the object:

```
variable = []  # create a new empty list
sys.getrefcount(variable)
Output = 2
reference = variable # create a new reference to the same list
sys.getrefcount(variable)
Output = 3
del reference # delete the reference
sys.getrefcount(variable)
Output = 2
```

The other memory management method used by Python is called the cyclic garbage collector which is used for reference cycles which is ‘the main problem with the reference counting scheme’[4]. If an object creates a reference to itself then if we try to remove the reference to it we are not able to reduce the reference count to zero. As the reference count never reaches zero, the memory associated with the object is never deallocated. This is known as a reference cycle. The cyclic garbage collector focuses on objects that can contain other objects. The garbage collector has a threshold for the amount of object allocations and deallocations before it runs the garbage collector.

Similarly to Java’s HotSpot JVM, which divides the heap into three generations where objects are moved into older generations the longer they live, Python’s cyclic garbage collector also implements generations. There are only three generations and every object starts in the first generation, if an object survives the collection they are moved to the next generation where the generation is checked for garbage collection less often. The final generation is where objects are surveyed the least often out of the three, likewise, Java’s garbage collector keeps the longest living objects in its third generation.

Programming Paradigms

Python has the ability to support multiple programming paradigms. This essentially means that Python can be used to write code in many different styles. Python supports the object-oriented, impure functional, procedural and imperative programming paradigms[5]. This allows the programmer to write code in a way that is most efficient and effective for the task at hand.

As an example, we will run through how Python implements parts of the object-oriented paradigm. Python allows the programmer to write code into a class that can be instantiated into objects which all have the same variables and methods. The main difference between object-oriented programming in Python and other languages is that creating private variables is not possible, there is a convention in Python where variables can be treated as private by adding two underscores before the variable name.[6]
To create a class that inherits another class in Python, the parent class is passed as an argument to the child class that is being created. To also inherit all of the parent’s variables and functions, super().__init__(args) is added to the __init__() function of the child.

Example: Creating a child class from a parent class and inheriting its variables and functions.

```python
class Parent:
    def __init__(self, name):
        self.name = name

class Child(Parent):
    def __init__(self, name, age):
        super().__init__(name)
        self.age = age
```

Conclusion

In conclusion, Python is a dynamically typed interpreted language with automatic memory management. Python also allows the user to write code in multiple different styles that would be most effective for them at that time. Python may have disadvantages in some of these areas but it also has advantages in other areas that outweigh its disadvantages. Regardless of its disadvantages, many people use Python due to it being easy to learn and use but also being a powerful language at the same time.

References

[1] A. N. Kamthane and A. A. Kamthane, Programming and problem solving with Python. Chennai, India: McGraw Hill Education (India) Private Limited, 2020.

[2] M. L. Scott, Programming language pragmatics. Amsterdam; San Francisco: Morgan Kaufmann, 2016.

[3] Python Software Foundation, “Design and History FAQ,” Python 3.11.0 documentation, Accessed: Dec. 03, 2022 [Online] https://docs.python.org/3.11/faq/design.html#how-does-python-manage-memory

[4] P. G. Salgado, “Garbage Collector Design,” Python Developer’s Guide, Accessed: Dec. 03, 2022 [Online] https://devguide.python.org/internals/garbage-collector/

[5] N. Thaker and A. Shukla, “Python as Multi Paradigm Programming Language,” International Journal of Computer Applications, vol. 177, no. 31, pp. 38–42, Jan. 2020, doi: 10.5120/ijca2020919775.

[6] Python Software Foundation, “9. Classes,” Python 3.11.0 documentation, Accessed: Dec. 03, 2022 [Online] https://docs.python.org/3.11/tutorial/classes.html#tut-private