A Detailed Analysis on Python

Written by Cole Cagle

I. Introduction

Python is considered one of the easiest programming languages for anyone to learn, whether they are new to programming or are familiar with other programming languages. Python has a large community that consists of users from all over the world. The community frequently has conferences and collaboration events for people to come together and learn more about the language or to complete projects. Python has been developed as open-source and can be used by anyone. As a result, there are many ways for people to showcase their creativity due to the large standard library and the thousands of third-party modules that have been developed. To learn more about Python and to get involved with the community, visit the link: Welcome to Python.org.

II. Category

Python is considered a multi-paradigm language due to it being a part of multiple categorizations. The first categorization to observe is that Python is an object-oriented programming language. Object-oriented programming involves the creation of objects and applying attributes and behaviors to the created objects. Essentially, this is taking real-world entities and converting them into software objects that contain data and have functionality.[1] An example that applies this definition is creating an object called “car”.

[1] Amos, David, “Object-Oriented Programming (OOP) in Python 3”

This “car” can have attributes such as color, model, and manufactured year. It also has the functionalities of being able to park, drive, and reverse directions. This is true for any cars in the real-world and this data and functionality can be converted and stored into a software object. The benefits of object-oriented programming are efficient development due to the large amount of libraries available and data is secure. The drawbacks are that sizes of the programs can be extremely large and the execution of the programs can be slow due to size.

Another categorization for the Python programming language is that it is considered procedural, or imperative. Procedural programming is when a program follows a sequence of instructions in order to execute. This type of programming takes a top-down approach due to the sequences needing to be defined in a correct and detailed manner.[2] One of the recommendations in procedural programming is to divide the instructions into blocks. It is the same approach that can be applied when dividing an essay into paragraphs. Each block should contain all the instructions needed for a function to be defined for execution. It is also important to note that data and functionalities are not combined and are treated as separate. The benefits of procedural programming is that the instructions are simple and easy to follow and it requires less memory. The drawbacks for procedural programming is that data is not secure and it is not ideal for complicated, or large-scale projects.

The last categorization for the Python language is that it is considered a functional language. Python is not often used in this manner but can be useful in certain scenarios. Functional programming involves using functions for execution. The output of these pure

[2] Programiz.Pro, “What is Procedural Programming? A Beginner’s Guide” (2022)

functions is based on the input that is received by them, from a user or another function.[3] Functions in Python have the ability to be treated and used like strings or integers. This allows for variables to be created and for functions to be assigned to a specified variable. It is important to note that strings can be combined together using the "+" operator or integers can produce a product using "*" operator. However, these operators are not defined for functions (when programming, it is important to understand the context of the situation). Python also allows for functions to be wrapped, or having inner and outer functions. An application of this in the real-world is the Chain Rule that is often taught in Calculus I when working with derivatives. Derivatives can be used to obtain the velocity or acceleration of objects and this calculation can be automated using Python.

III. Operators, Precedence, and Associativity

Python has a variety of operators that are defined in the language and can be used to achieve basic or complicated calculations. The first set of operators are arithmetic, which consists of: + (addition), - (subtraction), * (multiplication), / (division), % (modulus), ** (exponentiation), and // (floor division). These operators are appropriate when working with integers or floating point numbers for calculations. Whether the program that is being created needs to be able to find the sum of two numbers or find the antiderivative of a function, these operators can be used in defined functions in order to output the correct result. A recommended exercise to learn and understand these operators is to code a calculator that can solve a variety of functions. There are many examples created by users that can be seen by visiting: Search · calculator (github.com).

[3] Sturtz, John, "Functional Programming in Python: When and How to Use It"

The next set of operators are assignment. These operators are used to give variables a value, such as seen in Algebra when setting a variable equal to an integer (i.e. x = 6). The list of assignment operators is as follows: =, +=, -=, *=, /=, %=, //=, **=, &=, |=, ^=, >>=, and <<=. These operators will allow a user to create, access, and mutate the variables that are assigned values, which is an important feature in Python programming.[4] However, to be able to use these operators, an assignment statement must be created. There are three components to this statement: left-hand side, operator, and right-hand side. The left-hand side will consist of a variable, such as "x". The operator will consist of one of the aforementioned operators listed above. The right-hand side will consist of a value, which can be numeric or an object. These operators can be used to convert from Fahrenheit to Celsius, which is an important calculation in chemistry.

The next set of operators are comparison. Comparison operators are used to compare values to each other. The operators are as follows: ==, !=, <, >, <=, and >=. Comparisons are a vital aspect to many programs because these results can determine the following function(s) that need to be executed or to return errors to the user. There are some mathematical formulas that cannot be solved if certain criteria is not met with regards to the values assigned to the variables. For example, if a program is created to solve the quadratic equation and to return the roots to the user, then the variable "a" in the denominator cannot be assigned "0". There needs to be a function implemented that will compare the input value for "a" and to make sure it is greater than "0" before the program is executed. Without this comparison, it will break the program.

Another set of operators found in Python are logical. The logical operators are "and", "or", and "not". These operators are used for taking conditional statements and

[4] Ramos, Leodanis Pozo, "Python's Assignment Operator: Write Robust Assignments" (2023)

combining them together. The output when using these returns either true or false. The “and” operator is true if both conditions are true and false if any condition is false. The “or” operator is true if only one of the conditions is true and the other is false. The “not” operator returns the inverse of the conditions, so if a condition is true it will return false or vice-versa. These operators are vital for programs that have multiple conditions they may need to be met in order to execute. An example is that in order to use the Pythagorean Theorem, two of the variables need to be known while the third remains unknown. A Python program could have conditional statements implemented that use logical operators to ensure that this requirement is met so the calculation can be returned correctly.

The next set of operators found in Python are bitwise. Bitwise operators are similar to logical and comparison operators but compare conditions that are in binary form. The operators are as follows: & (AND), | (OR), ^ (XOR), ~ (NOT), << (zero fill left shift), and >> (signed right shift). The & operator has the same properties as the logical “and” in that both bits are “1” if they are each “1”. The | operator has the same properties as the logical “or” in that both bits are “1” as long as one bit is “1”. The ^ operator is similar to the | operator, however, both bits are “1” if only one of them is “1”. If both are “1” then the bits will be set to “0”. The ~ operator is similar to the logical “not” in that it inverses the bits (1 returns 0, 0 returns 1). The << operator is used to add zeros to the right side of the bit sequence and leave off the leftmost bits (if two 0’s are added, remove the two leftmost bits). The >> operator is used to add the leftmost bit and remove the rightmost bits as needed (if the leftmost bit is 1 and two 1’s are added, then

remove the two rightmost bits). The application for these operators is notable in Discrete Mathematics when using truth tables for proofs.

The next type of operators in Python are membership. Membership operators are used for testing whether or not a sequence is stored in an object. There are only operators for this type, which are “in” and “not in”. The “in” operator is true if the sequence is in the object and the “not in” operator is true if the sequence is not in the object. A real-world application involving these operators is a sequence of names involving people that have appointments at a doctor’s office. If a person arrives at the office and has an appointment, then the appointment application will check using membership operators to see if this is true or not. If they have an appointment, then it will return true and allow them to check-in.

The last type of operators found in Python are identity. Identity operators are used to compare two objects to see if the objects are the same. There are only two operators for this type and they are “is” and “is not”. The “is” operator returns true if the objects are the same and the “is not” operator returns true if the objects are not the same. These operators are vital to programs that have many objects that may be similar but are not the same. An example of a program that could benefit from identity operators would be self-ordering machines that are appearing in restaurants. These operators can be implemented to ensure that orders are being correctly matched to the names of the customers.

The precedence of operators is applied in Python when expressions are implemented into a program and need to be calculated. An expression contains multiple operators and it is important that the Python interpreter is able to distinguish between the

order of operations. A popular phrase used in early academia to teach the order of operations is “Please excuse my dear aunt Sally”. This same phrase is applied by the Python interpreter to evaluate an expression. However, there are more levels of precedence due to Python having many types of operators that are not just limited to arithmetic type. The order of precedence from highest to lowest is: parentheses, function call, slicing, subscription, attribute reference, exponent, bitwise not, positive and negative, product and division, addition and subtraction, shifts left/right, bitwise and, bitwise xor, bitwise or, comparisons and identity and memberships, logical not, logical and, logical or, and lambda.[5] Each level listed is separated by a comma and there are many levels of precedence in Python. This can get complex and lead to the issues involving the order of operations when there are multiple operators on the same precedence level.

This problem is avoided by the Python interpreter due to associativity. This allows for the order of operations to continue to be evaluated despite some operators sharing the same precedence. Nearly all of the operators in Python support left-to-right associativity with the only exception being the arithmetic operator for exponentials, **.[5] An example of this being applied can be seen using the expression, 3 * 8 / 4. The multiplication will occur first and update the expression to 24 / 4. Then the division will occur, which results in 6. Even though multiplication and division share the same precedence, associativity allows for the order of operations to continue. It is important to note that Python’s assignment and comparison operators are not supported by associativity and are considered non-associative operators.

[5] Jacob, Riya K., “Python Operator Precedence and Associativity”, PowerPoint

IV. Variation on the Classical Sequence

The classical sequence refers to the steps that are taken for a program to be created. It starts with using an editor to create and edit a source code file. The source code is then compiled into an assembly language file and assembled into an object file. A linker is used to execute the executable file and a loader runs the program in memory. Python has variations in this sequence.

One such variation of this sequence is that Python is compatible with integrated development environments (IDEs). A popular IDE that is used for programming with Python is Microsoft Visual Studio Code. In order to use Python on this IDE, a user will have to download and install a Python interpreter from the official Python website. The user will also need to download the Python extension that is available on the marketplace within VS Code and then locate the interpreter. The interface for VS Code is user-friendly and allows for a programmer to edit, run, and debug programs all in one place. This integration allows for the IDE to understand Python's syntax and identify errors as well as keep multiple versions of the program.

Another variation of the classical sequence is that Python uses an interpreter. Instead of translating the Python language into assembly language, the interpreter allows for the code to begin execution. This can be slower because the program will be run using the speed of the hardware, which can sometimes be outdated depending on the operating system being used.

V. Types

Python has various data types that can be implemented within programs. One such type that is found in the language are primitive types. There are four kinds of

primitive types in Python and they are: integers, booleans, strings, and floats. The base 10 number system is not the only integer type in Python. The other number systems that can be used are binary, hexadecimal, and octal. The boolean type contains either the value of true or false. This type is often implemented when working with logical operators. The string type is used to contain character data, or the letters of the alphabet. To define a string in Python, single or double quotes can be used as long as they match and form a pair. This type is often used to output messages to a user, such as results or errors upon program execution. The last primitive type is floats. This type is used to contain numeric values that are partial, such as fractions in decimal format. This type is often used in the implementation of programs that use arithmetic operators.

Another type that is found in Python are constructed types and these have to be imported into programs because they are in modules. The enum type is concerned with enumerations. This type allows for constants to be assigned names. Another constructed type is array. This type allows for multiple values to be stored into a single variable, and these values can be numeric or alphabetic. The next constructed type is lists. This type is used to store multiple values that can be of different data types into a single variable. This is different from the array type because the data types have to be the same in an array. The next constructed type is subtype. This type allows for related data types to be substituted for one another in a program. Another constructed type found in Python is tuples. This type allows for multiple entities to be stored in a single variable but in an ordered and unchangeable way. The next constructed type is dict, or dictionary. This type is similar to lists, however, the entities in the dictionary are accessed by using an index. The index locates a specified entity based on its position. The last constructed type in

Python is a set. This type allows for multiple entities to be stored in a single variable but in an unordered and unchangeable way.

Python allows programmers to make comments within their programs as a means to document it. These can be implemented by creating a new line and using the "#" symbol and follow it with text. These comments or annotations will be treated as static type and will not be executed when the program is running. This is recommended across the field of computer science when creating programs so that others who read the program later and want to modify it, can understand the goal and functions defined within the program.

Another feature involving Python is type inference. This allows for expressions to have static types that can be inferred from the order of operations. If there is an integer data type being multiplied by a floating data type, then Python will infer that the product will be a float data type. For example, 2 * 4.5 will have a product of 9.0 because a partial value is involved in the expression. Although, it would not be problematic for the "0" to be left off because "9" is a whole number, it could lead to issues with other expressions. An example where this could be an issue is 2 * 4.3, which has a product of 8.6. The "6" cannot be left off or else the product would be incorrect.

Python is a language that is not static type checked. Static type checking is when a type is determined for everything in the program, whether it be functions, variables, or objects. This is achieved before the program executes. Instead, Python is dynamically type checked. Dynamic type checking is when operands are checked with their associated operators at runtime of the program.

VI. Polymorphism

One of the essential characteristics with Python is polymorphism. Polymorphism allows an object that is created in code to have multiple actions that it can perform. Polymorphism is observable in the real-world, for example, take a person who is a college student on certain days and an employee at a business on other days. The person is the object and has the ability to perform different actions depending on their schedule. An important function of polymorphism is the ability to overload methods. This allows for different parameters to call the same method, however, Python does not support overloading.[6] Fortunately, there is a workaround to this issue, but it requires the programmer to define the logic of the method for each possibility based on the parameters that are being passed through it.

Python allows for operators to be overloaded. The addition operator (+) can be implemented in code and has the ability to perform different actions based on the data types it is given. For arithmetic purposes, the addition operator needs at least two numeric values, that can be either of type integer or float, and will return the sum of the values. In another instance, the addition operator can be used with string data types to combine two or more strings together. This allows for the addition operator to have compatibility with numbers and alphabetic characters.

The operator types that can be overloaded in Python are arithmetic, assignment, and comparison. In order to achieve this, magic methods must be implemented in code. These methods are extremely beneficial for programs that are written to perform a variety of calculations that depend on the data types that are presented. For a complete list of the magic methods in Python, visit the following link: Polymorphism in Python – PYnative.

[6] PYnative, “Polymorphism in Python” (2021)

VII. Scope

A security feature that is implemented in the Python programming language is scope. Anything that is created and defined within code has a scope, which is the accessibility given to the program to access classes, objects, or functions. The scope can range from a single block of code or the code in its entirety.[7] There are four types of scopes in Python, which are: local, global, built-in, and enclosing.

Local scope refers to a block of code, which is often a function. Variables that are created by a programmer only exist when they are called within a function. Once this function has been executed at runtime of the program, then the local scope will be terminated until it is executed again. An example of local scope is having two variables that are defined with given integer values. A block is created to take these defined variables and obtain the product of them. The output will be returned to the user and the local scope will be terminated until the user wishes to calculate another product.

Global scope refers to all the blocks of code, which are all the functions required for a working program. Python only allows for a single global scope per program and it refers to the entire program as the main module. The scope begins as soon as the program begins execution at runtime and ends when the program is terminated. Global scope will be applied again, if the program is executed once more. An example of global scope is having two variables that are given integer values. These integer values are then passed through four different blocks of code. One block will calculate and output the sum, a second block will calculate and output the difference, a third block will calculate and output the quotient, and the fourth block will calculate and output the product. Upon execution of these calculations, the program terminates along with the global scope.

[7] Raj, Aditya, “Python Scope”, (2021)

VIII. Memory Locations for Variables

Variables are used in Python to give a name to an object that has been defined within a program. To create a variable, the name of the variable needs to be set equal to a value. Python allows for variables to be either static or dynamic. Static variables remain the same throughout the execution of a program. Meanwhile, dynamic variables have the ability to change during execution of a program. This feature allows for variables to be given one type and then changed to another type. Since Python is considered an object-oriented language, variables are used to either reference or point to objects. This feature is important because an object is stored in a memory address and needs to be located when needed by the program. Variables serve as guides for locating objects in memory because they contain the memory address. Memory addresses are hexadecimal values that represent the addresses in memory. Each object has its own unique memory address, because if multiple objects could share a memory address then a program would confuse data and not execute correctly.

When objects are created and assigned a variable name, there is a reference count for the object throughout the program. This count refers to the amount of times that an object stored in memory is referenced or needed in the program. An object is also given a unique numeric value that is called an “identifier”. Objects cannot share these values or else the program would fail to execute. After the count for references reaches “0”, the identifier for an object is freed and can be assigned to another object. This feature helps provide optimization when the interpreter begins reading code.

Variables can be either short or long, however, it is recommended that variables are names that can be recognized and understood by programmers, in the event that

modifications are needed. Variables can be letters of the alphabet, either upper or lowercase as well as digits. Python is case sensitive so it is important to pay attention to the cases being used when creating variables. However, it is important to note that variable names cannot begin with a digit. Underscore symbols can be used as well if needed to serve as a space between words. The three most popular methods for variable names involve Camel Case, Pascal Case, and Snake Case. An example of Camel Case being implemented is “pythonProgrammingLanguage”. Pascal Case is extremely similar but is something such as “PythonProgrammingLanguage”. Lastly, Snake Case being implemented is “python_programming_language”. Snake Case is usually the preferred convention when creating variables.

IX. Memory Management

Global variables that are in a program are stored in heap memory. Meanwhile, local variables are stored in stack memory. Methods are also stored in stack memory. Stacks and heaps are two important data structures that allow for data to be stored and retrieved by a program. Python allocates and deallocates memory for a program, and the amount depends on the size. This allocated memory is then divided between objects and non-objects. Memory allocation is vital to the success of a program due to its various components that need to be located quickly for execution.

The stack is a LIFO data structure, which means it is “Last In, First Out”. When methods are created in a program and begin execution, the last method will be the first one to execute and create output, and the first method will be the last one to execute and create the final output. An example of a stack in the real world is a stack of plates. The last plates that are added to the stack will be on the top and will be the first plates to be

used again. Meanwhile, the plates on the bottom will be the last ones used. The implementation of the stack for memory management allows for order to be kept when a program begins executing.

The heap data structure is a binary tree. There are two types of heaps that are used, max-heap and min-heap. With max-heap, the largest values are stored at the top and the smaller values are stored towards the bottom of the tree. Min-heap is the opposite of max-heap when storing values. The implementation of a heap in memory management provides order, however, it is partial. This is important for locating variables that reference objects across multiple blocks of code.

X. Parameters

In Python, parameters are variables that are located within functions as a part of their definitions. Another term closely associated with this is “arguments”. Arguments are the values referenced by variables. Both parameters and arguments are able to pass information to a function. Functions will contain a required amount of parameters or arguments needed in order to execute. If the specified and required parameters are not sent to the function as defined, then the function will not execute and cause the program to output errors. Python also has a beneficial feature that allows for parameters to be given names and assigned values that can allow parameter passing to be executed as needed by the function.

XI. Conclusion

The Python programming language is an extremely beneficial language to learn in the field of computer science. It has endless capabilities and has a large and active

community that continues to give support to beginners and experts. Python can be found in almost every industry in the world of business. Some of the popular applications that have been developed with Python are Netflix, Uber, and Spotify, to just name a few. Python developing can serve as an excellent career choice or as a means to create the next generation of life-changing applications.

XII. References

1. Amos, David, “Object-Oriented Programming (OOP) in Python 3” Object-Oriented Programming (OOP) in Python 3 – Real Python

2. Programiz.Pro, “What is Procedural Programming? A Beginner’s Guide” (2022) What is Procedural Programming Paradigm? (programiz.pro)

3. Sturtz, John, “Functional Programming in Python: When and How to Use It” Functional Programming in Python: When and How to Use It – Real Python

4. Ramos, Leodanis Pozo, “Python’s Assignment Operator: Write Robust Assignments” (2023) Python's Assignment Operator: Write Robust Assignments – Real Python

5. Jacob, Riya K., “Python Operator Precedence and Associativity”, PowerPoint Microsoft PowerPoint - OP_Python (littleflowercollege.edu.in)

6. PYnative, “Polymorphism in Python” (2021) Polymorphism in Python – PYnative

7. Raj, Aditya, “Python Scope”, (2021) Python Scope - PythonForBeginners.com