Chandler Scott

Term Paper Structures of Programming Languages: Python

ABC is a general-purpose programming language and programming environment that inspired the creation of Python. ABC's inspiration for Python is its greatest achievement. Python was conceptualized in the late 1980's by Guido van Rossum while he was working as an implementer on the team building language that is ABC. Guido van Rossum was inspired by his frustration with ABC and wanted to create a language that used all his favorite components from ABC and a few other scripting languages. Guido created a virtual machine, a simple parser, a simple runtime, replaced the curly braces with indentation, and started to create powerful data types built from applying his basic syntax. Some of these powerful datatypes include a hash table, a list, strings, and numbers.

The name "Python" did not come from the snake species python even though the logo is two snakes. The idea is from Guido van Rossum looking for a word that would appeal to C/Unix hackers and Guido being a big fan of Monty Python's Flying Circus.

Guido van Rossum published the first version of Python code (version 0.9.0) at alt.sources in February 1991(Lutz). This release included exception handling, functions, and the core data types mentioned earlier list, dictionary, string, and numbers. This language is also object-oriented and has a module system. A module is a software component or part of program that contains one or more routines.

Over the years, improvements were made to the Python language. Python version 1.0 was released in January 1994. Lambdas, maps, filters, and reduce were now available in Python. Six years later in 2000 Python version 2.0 was released that included a garbage collector and it was

supporting Unicode. Python continued to grow and constrict its competitors for another 8 years in the 2.x versions. Next to come was Python version 3.0, also known as Python 3000 or Py3K. This version of python is not compatible with the previous version of Python. An emphasis was placed on removal of duplicate programming constructs with the idea that "There should be one, and preferably only one, obvious way to do it." (Klein). Some of the changes in Python 3.0 include added functions, replacements for lists, simplification of rules for comparisons, certain ordering restrictions on lists, types must be comparable, return from float from division instead of integer, and "Text Vs. Data Instead of Unicode Vs. 8-bit" (Klein).

Python is an interpreted language. This means that the code is processed at runtime by an interpreter. You need not compile your Python program before executing it. PERL and PHP act in a similar way. Python is also interactive, meaning you can interact with the interpreter directly to write various Python programs.

As mentioned above Python's computational paradigm supported is object-oriented. This is technique of programming that encapsulates code within objects. An object-oriented system consists of a set of interacting or collaborating objects. Classes act as a template for creating objects of a specific type. Each object has a list of services, which are operations associated with an object that are visible to other objects also called clients. These services can be public or private methods in the class definition. In an object-oriented language such as Python, objects communicate through message passing.

Python has proven to be a good starting language for many programmers because it is easy to learn, easy to read, easy to maintain, and has a broad standard library. Python has an interactive mode, so you can debug specific chunks of code. Python can run on a variety of

platforms and allows you to add low-level modules to the interpreter. This allows you to customize various tools that are available within the language.

In this object-oriented language specific data types are supported. Some of these data types include lists, maps, strings, and numbers. Lists are just that, they are lists of some data-type. You can access and manipulate these data types at various positions. You can even have a list made up of lists. This is the approach I took to implement Conway's Game of Life in Python. Lists are part of the most basic data structure in Python, a sequence. Each element of a sequence is assigned a number for its position known as an index. First index of a list is accessed with a zero inside of square brackets. Even though python has six different types of sequences, the two most common are lists and tuples. Python has built in functions to manipulate these lists and return the length of a sequence, its largest value, or its smallest value.

In Python variables do not need explicit declaration to reserve memory space. Declaration happens automatically when you assign some sort of value to a variable. To assign a value to a variable you would use the equal sign (=). On the left side of the operand is the name of the variable and on the right is the value being stored in the variable. In Python you can assign the same value to multiple variables in one line of code. For example: z = y = x = a = 1 creates an integer object and all four of these variables z, y, x, and a are assigned to the same memory location. In all, Python has five standard data types numbers, string, list, tuple, and dictionary. Number data types store numeric values. When you assign a value to them this creates a number object. The numerical types supported in Python include int (signed integers), long (long integers, they can also be represented in octal and hexadecimal), float (floating point real values), and complex ("an ordered pair of real floating-point numbers denoted by x + yj, where x and y are the real numbers and j is the imaginary unit") (tutorialspoint.com). Numbers can be casted as

strings and vice versa. Strings which can be defined as a contiguous set of characters represented in quotation marks. Python allows for single or double quotes to denote a string. The plus sign is used for concatenation of two strings and the asterisk can be used for repetition of printing the string. A string and a number can both be parts of a list. “The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets” (tutorialspoint.com). One of the greatest benefits of a list is that Python does not require you to have matching types with the list. For example, you can have list made from integers and strings. You can append, remove, and replace elements in a list along with various other built-in functions for list manipulation and index access. Tuples are like lists and consist of values separated by commas. Unlike lists, tuples are enclosed within parenthesis. Tuples can be thought of as read-only lists because they cannot be manipulated/updated. Dictionaries in python are a kind of hash table and work like associative arrays. These dictionaries consist of key-value pairs. A dictionary is usually made up of a number and string value pair. “Dictionaries are enclosed by curly braces ({}) and values can be assigned and accessed using square braces ([])” (tutorialspoint.com). There is no concept of order amongst the elements in a dictionary. Elements themselves are not out of order, they are just simply unordered. But how are all these types assigned to variables?

Python is a dynamically-typed language. Python like Java is also strongly typed language. This means that a type is bound to a variable at run-time. This variable is bound to a type once it is assigned a value. During execution of the Python file the variables type can change any number of times. Ruby, PHP, and JavaScript are also dynamically typed languages. “Languages with dynamic type binding are usually implemented using pure interpreters rather than compilers” (11, Nandigam).

Python supports arithmetic operators, comparison operators, assignment operators, logical operators, bitwise operators, membership operators, and identity operators.

Addition (+) allows you to add values on either side of the operator. Subtraction (-) subtracts right-hand operand from the left-hand operand. Multiplication (*) multiplies values on either side of the operator. Division (/) divides left hand operand by the right-hand operand. Modulus (%) divides left-hand operand by right hand operand and returns the remainder. Exponent (**) preforms exponential (power) calculation on the operators. Multiplication and Division take precedence over addition and subtraction. This can be manipulated with parenthesis.

Comparison operators include equals (==), does not equals (!= or <>), greater than (>), less than (<), greater than or equal to (>=), and less than or equal to (<=). These comparisons return true or false Boolean values within if statements.

Assignment operators in Python include = (assigns values from right side to operands to the left side), += (adds the right operand to the left operand and assign the result to the left operand), -= (subtracts the right operand to the left operand and assign the result to the left operand), *= (multiplies the right operand to the left operand and assign the result to the left operand), /= (divides the right operand to the left operand and assign the result to the left operand), %=, **=, and //=. The remaining three operators are “modulus and, exponent and, then finally floor division” (tutorialspoint.com).

Bitwise operators allow you to assign values to variables with their binary value. The symbol “&” copies a bit to the result if it exists in both operands. The symbol “|” copies a bit if it exists in either operand. The “^” copies the bit if it is set in one operand but not both. The

symbols “<<” stand for left shift and “>>” stands for right shift of bits. The “~” symbol returns the complement or the flipped bit sequence.

Logical operators in Python are “AND, OR, and NOT”. The “AND” operator says, “If bother the operands are true then condition becomes true. The “OR” operator says, “if any of the two operands are non-zero then condition becomes true”. The “NOT” operator is used to “reverse the logical state of its operand” (tutorialspoint.com).

Membership operators include “in” and “not in”. Pythons membership operators are used to test for membership in a sequence. The “in” operator evaluates to true if it finds a variable in the specified sequence and false otherwise. The “not in” operator evaluates to true if it does not find a variable in the specified sequence and false otherwise. Identity operators compare the memory locations of two objects. Two identity operators are “is” and “is not”. The operator “is” evaluates to true if the variables on either side of the operator point to the same object and false otherwise. The “is not” operator acts similar and is the opposite of the “is” operator.

Python operator’s precedence is as follows: First is exponentiation. Second are complement, unary plus and minus. Third are multiply, divide, modulo, and floor division. Fourth is addition and subtraction. Fifth is right and left bitwise shift. Sixth is bitwise ‘AND’. Seventh is bitwise exclusive ‘OR’ and regular ‘OR’. Eighth are comparison operators. Ninth are equality operators. Tenth are assignment operators. Eleventh are identity operators. Twelfth are membership operators. Lastly the logical operators are evaluated. Most of these operators can be used inside of loops and if, else if, or else statements.

A loop is a sequence of instructions that is continually repeated until a certain condition is reached. This can be done in Python using for loops, while loops, and nested loops. The while

loop repeats a statement or group of statements while a given condition is true. It tests the condition before the body of the loop. The for loop executes a sequence of statements multiple times and abbreviates the code that manages the loop variable. You can use a loop inside of another loop to create a nested loop. Loops have control statements break, continue, and pass. Break terminates the loop statement and transfers execution to the statement immediately following the loop. The continue statement causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating. The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute. Loops can be used inside of functions within the class in the Python file. Selection statements like the if statement, are often used throughout the code outside functions (at class level), in functions, and in loops.

Selection statements that are supported in Python include the if, elif (stands for else if), and else statements. The if statement simply means if the statement following the if is true, then do something. You can simulate an "if not true" by following the body of the if statement with an else statement. These two statements work together meaning that you cannot have an else without an if. You can have an if statement with no else statements. The else simply is the code that executes when the if statement is false and does not have a statement that goes alone with it, just a body. This means if not true, then else do this. If you want to create a certain condition after the if statement and "else if this is true, then do this chunk of code here". You can create nested if statements just like you created nested for loops by placing one if inside another. These if statements between the first if and else that belongs to the first if are called else if statements. You can do this by manipulating the indentation. Indentation of the if and else must be at the same level for the code to execute properly.

To define a function in Python you start with the keyword def, followed by the function name and parenthesis and ended with a colon. As mentioned above, indentation is key for creating a block of code within the function. You must make sure that the indentation of your function is properly done so that the function executes properly. Any code outside of the indentation of the defined function will not be part of that function. Any arguments wanted are placed within the parentheses next to the function name and are separated with commas. The return statement followed by some expression, exits the function and returns whatever the expression is. A return with no arguments is the same as return none. Parameters in the function have positional behavior and need to be informed in the same order that they were defined. In Python all parameters in a function are passed by reference. This means if you change what a parameter refers to within the function, the change also applies to the calling function. You can assign values to the arguments to given them default values. Required arguments are the arguments passed to a function in correct positional order. Keyword arguments are related to the function calls. When used, the caller identifies the arguments by the parameter name. Variables created in a function are called local variables. Global variables are created outside of the function and in the class itself. Then there are anonymous functions that are not declared using the keyword def. The lambda keyword is used instead to create small anonymous functions. These lambda functions can take any number of arguments but only return one value. The lambda functions are not allowed to contain commands or multiple expressions. Lambda functions are single line statements. These functions have scoping rules that apply to them and their variables.

Python has static scoping rules. If function f defines or deletes a variable name, then the variable name refers to a variable in the closure of function f. If the function f only uses f and

does not have a definition or deletion of f, then the variable f refers to what f means in the parent scope. The program keeps looking to the parent scope until a definition of f is found or you reach the global scope.

Modules are supported in Python as mentioned earlier. A module is a file containing Python definitions and statements. These are used to give the user a way to put definitions in a file and use them in a script or in an interactive instance of the interpreter. Definitions from a module can be imported into other modules or into the main module.

For example, look at the Python file fibo.py:

```
# Fibonacci numbers module

def fib(n):    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print b,
        a, b = b, a+b

def fib2(n):   # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

In this module you can see examples of how to define functions, assign variables, use operators, use built in functions such as append, an example of iteration, and the proper way to use a return statement at the end of the function. Also notice that the indentation of the two functions is the same, the code block is within the function indentation, and the while block code is within the while loops indentation. To import this module to the Python file, type “import fibo” at the top of the file.

While writing code, you may want to have the user interact with the program and enter some input. What happens when the user enters incorrect input? Since the program crashing is not a desired option, you must have a way to handle certain exceptions in the code. This is done with exception handling. Exception is a base class in Python for all exceptions. Various exceptions in python can be raised if the code runs into a problem such as Stop Iteration, System Exit, Standard Error, Arithmetic Error, Overflow Error, etc. There are twenty-nine standard exceptions that can be thrown in Python. Exception handling starts with an assertion. Assertions are carried out by the assert statement. This keyword, assert, was introduced in Python version 1.5. Programmers typically place assertions at the start of a function to check the input and make sure its valid and after the function call. When Python encounters an assert statement it evaluates the following expression, which is expected to be true. If the expression happens to be false, Python raises an Assertion Error exception. "Assertion Error exceptions can be caught and handled like any other exception using the try-except statement, but if not handled, they will terminate the program and produce a traceback" (tutorialspoint.com).

The following is an example of using assert in Python:

```
def KelvinToFahrenheit(Temperature):
   assert (Temperature >= 0),"Colder than absolute zero!"
   return ((Temperature-273)*1.8)+32
print KelvinToFahrenheit(273)
print int(KelvinToFahrenheit(505.78))
print KelvinToFahrenheit(-5)
```

According to Bernd Klein at www.python-course.eu, "Every object-oriented programming language would not be worthy to look at or use if it weren't to support inheritance" (Klein). Python supports inheritance and multiple inheritance. Classes in Python can inherit other classes.

The superclass is a class that the subclass inherits attributes and behavior methods from. The class that inherits these methods is referred to as the subclass.

An example of a super class and sub class interaction can be seen here:

```
class Person:
   def __init__(self, first, last, age): #this is the constructor
      self.firstname = first #assigning values
      self.lastname = last
      self.age = age
   def __str__(self):
      return self.firstname + " " + self.lastname + ", " + str(self.age)
class Employee(Person):
   def __init__(self, first, last, age, staffnum):
      super().__init__(first, last, age)
      self.staffnumber = staffnum
   def __str__(self):
      return super().__str__() + ", " + self.staffnumber
x = Person("Marge", "Simpson", 36)
y = Employee("Homer", "Simpson", 28, "1007")
print(x)
print(y)
```

You can see from this code you can see examples of:

- How to define a class
- How to use functions from the super class
- How to use return statements
- How to print variables
- How to use local and global variables
- Use of parameters
- Creation of objects and how to assign them to variables

That is the summary of Python. From this paper you learn how it was created, its basic rules, and you can see examples of how to apply these rules.

The previous code examples and information can be found at these sources:

https://www.python-course.eu/python3_history_and_philosophy.php

https://pythonconquerstheuniverse.wordpress.com/2009/10/03/static-vs-dynamic-typing-of-programming-languages/

https://www.tutorialspoint.com/python/python_variable_types.htm

https://mybb.gvsu.edu/bbcswebdav/pid-5904139-dt-content-rid-49236249_1/courses/GVCIS343.01.201920/NamesBindingsScopes.pdf