**INTRODUCTION TO PYTHON**

Python is an extremely powerful language. It is a combination of C/C++ and Java .Therefore it can be used to tie large projects together or for rapid prototyping. It has a built-in GUI (graphical user interface) which lets the programmer make simple graphical interfaces with little effort. Python can also be used in a real time interpreter for testing code snippets before adding them into a normal executable.

Python is a scripting language. It’s not compiled to create the machine-readable code and that the code is “tied-into” another program as a control routine. Compiled languages, like C++, require the programmer to run the source code through a compiler before the software is can be used by a computer. Depending on the program’s size, the compilation process can vary from minutes to hours. Python can act as a glue between different programs. Python is often used as the scripting language for video games, while the heavy-duty work is performed by pre-compiled modules. Python can act in a call/response fashion, like taking controller input and passing it to the appropriate module.

Python is also considered a high-level language. Python has a built-in garbage collector so, a programmer don’t really need to worry about memory management and memory leaks, a common occurrence when using older languages such as C. The main significance of Python is readable code and enhancing programmer productivity. This is achieved by enforcing a strict way of structuring the code to ensure the reader can follow the logic flow and the programmer doesn’t have to worry about including a lot of different libraries or other source code to make his program work.

One of the main difference in Python is the use of whitespace .Unlike many other languages that require the programmer to use brackets, typically curly braces, i.e. “{}”, to identify different blocks of code. In Python, these code blocks are identified by different amounts of indentation.

**Versions of python**

Python 3.x breaks compatibility with programs written in 2.x versions. However, it is similar to 2.x version. It means that one has to be aware of the Changes to the language when before using version 3.0. Additionally, the

install base of 2.x is quite large and won’t be going away for quite some time. Most Linux distributions (and Mac OS X) still have older Python versions installed by default.

Python can be downloaded from the Python web site (for Windows) or it may already be installed on systems that uses a Mac, Linux, or \*BSD. However, the Unix-like operating systems, including OS X, may not have the latest version so one can upgrade, at least to version 2.6. Version 2.6 is a modification of 2.5 that allows use of both 2.x code and certain 3.0 functions. Essentially it lets to code in **legacy** style while still being able to use the latest features as desired, and testing which legacy features will be broken when moving to 3.0.

For those interested in using Python via a USB drive, you can use Portable Python. This is a self-contained Python environment that one can either run from the drive or install to your computer. This is useful for those who can’t or don’t want to install Python but would still like to use it.

To get the interactive interpreter running, open up a command prompt (or terminal) and type ‘python’ at the prompt. This will open a Python session, allowing one to work with the Python interpreter in an interactive manner. In Windows, just go to the Python file in All Programs and click it.

**Python approach**

The following features make python one of the best programming language. **1.Dynamic**

Python is a dynamic-typed language. Many other languages such as C/C++ and Java are static typed. A static typed language requires the programmer to explicitly tell the computer what type of thing each data value is. For example, in C if a variable that was to contain the price of something, one has to declare the variable as a float type. This tells the compiler that the only data that can be used for that variable must be a floating point number, i.e. a number with decimal point. If any other data value was assigned to that particular variable, the compiler will give an error while trying to compile the program.

Python doesn’t require this. One has to simply give variables names and assign values to them. The interpreter takes care of keeping track of what kinds of objects the program is using. This also means that one can change the size of the values while developing the program. Say for example you have another decimal number you need to add in your program. With a static typed language, one has to decide the memory size the variable can take while initialising that variable for the first time. A double is a floating point value that can handle a much larger number than a normal float. If a variable is declared to be a float but later on assigning a value that is too big to it, will cause the program to fail, one has to go back and change that variable to be a double.

In Python, it doesn’t matter. One can simply give it whatever number as they want. Python will take care of manipulating it as needed. It works for even derived values. For example, while dividing two numbers. One is a floating point number and one is an integer, Python realizes that it’s more accurate to keep track of decimals so it automatically calculates the result as a floating point number. Here’s how it would look like in the Python interpreter.

>>>9.0 / 3

3 .0

>>>8 / 4.0

2.0

From the above result, we can conclude that it doesn’t matter which value is on top or bottom, Python encounters that a float is being used and gives the output as a decimal value.

**2.Interpreted**

Many traditional languages are compiled i.e. the source code written by the developer is converted into machine language by the compiler. Compiled languages are usually used for low-level programming like device drivers and other hardware interaction and faster processing, e.g. video games.

As the language is pre-converted to machine code, it can be processed by the computer much quicker because the compiler has already checked the code for errors and other issues that can cause the program to fail. The compiler won’t find all errors but it does help. The caution to use a compiler is that compiling can be a time consuming task, the actual compiling time can take several minutes to hours to complete depending on the program. If errors are found, the developer has to find and fix them and then rerun the compiler, this cycle continues until the program works correctly.

Python is considered as an interpreted language. It doesn’t have a compiler, the interpreter processes the code line by line and creates a byte code. Byte code is a language in between the machine code and the source code. Because of this in-between state, byte code is an ‘in-between language’ .Byte code is more transferable between operating systems than machine code, this helps Python be cross-platform. Java also uses byte codes.

However, as Python uses an interpreter rather than compiler, the code processing can be slower. The byte code still has to be analysed for use by the processor, which takes additional time. But the benefit is that the programmer can immediately see the results of his code. The programmer doesn’t have to wait for the compiler to decide if there is a syntax error somewhere that causes the program to crash.

**3.Protyping**

Because of interpretation, Python is used for rapid application development and program prototyping. For example, a simple program can be created in just a few hours and can be shown to a customer in the same visit.

Programmers can repeatedly modify the program and can view the results quickly. This allows them to try different ideas and find which one is best without investing a lot of time on dead-ends. This also applies to creating graphical user interfaces. Simple sketches can be laid out in minutes because Python not only has several different GUI libraries available but also includes a simple library (Tkinter) by default.

Another benefit of not having a compiler is that errors are immediately rectified by the Python interpreter. Depending on the developing environment, it will automatically read through the code as it is developed and notify the syntax errors. Logic errors won’t be pointed out but a simple mouse click will launch the program and final product will be shown. If something isn’t right, one can simply make a change and click the launch button again.

**4.Procedural programming**

Python is somewhat unique in that the user has two choices while developing the programs: procedural programming or object-oriented. As a matter of fact, one can mix the two in the same program. In brief procedural programming is a step-by-step process of developing the program in a somewhat linear fashion. Functions (sometimes known as subroutines) are called by the program at times to perform some processing, then control is returned back to the main program.

Object-oriented programming (OOP) is just programming with objects. Objects are created by distinct units of programming logic, variables and methods are combined into objects that do a particular thing. For example, one could model a robot and each body part would be a separate object, capable of doing different things but still part of the overall object. Object-oriented programming is also heavily used in GUI development.

Procedural programming is easier to learn as the thought process is mostly straightforward and essentially linear.

**Types and operators**

Python is based on the C programming language and it is familiar to C and C++ programmers. However, it makes programming a little easier because it’s not made to be a low-level language (it’s difficult to interact heavily with hardware or perform memory allocation) and it has built-in ‘garbage collection’ (it tracks references to objects and automatically removes objects from memory when they are no longer referenced), which allows the programmer to concentrate more about how the program will work rather than dealing with the computer.

**Syntax**

1. **Indentation**

Python forces the user to program in a structured format as the code block are determined by the amount of indentation used. Brackets and semicolons were used to show code grouping or end-of-line termination for the other languages. Python doesn’t require those instead indentation is used to signify where each code block starts and ends. Here is an example

x = 3

if x : # if x is true

y=1

if y : # if y is true

print "block 2"

print "block 1"

else : print "block 0"

Each indented line denotes a new code block. To understand the above code snippet, line 1 is the start of the main code block. Line 2 is a new code section, if ‘x’ has a value not equal to 0, then indented lines below it will be evaluated. Hence, lines 3 and 4 are in another code section, these lines will be evaluated if line 2 is true. Line 5 is yet another code section and is only evaluated if ‘y’ is not equal to 0. Line 6 is part of the same code block as lines 3 and 4, it will also be evaluated in the same block as those lines. Line 7 is in the same section as line 1 and is evaluated regardless of what any indented lines may do, in this case, this line will not be evaluated because both x and y are true.

The compound statements, like the if comparisons, are created by having the header line followed by a colon (‘:’). The rest of the statements are indented below it. The biggest thing to remember is that indentation determines grouping, if a particular code doesn’t work for some reason, double-check the statements which are indented.

**2.Multiple line spanning**

Statements can span more than one line if they are collected within braces such as parenthesis “()”, square brackets “[]”, or curly braces “{}”. Normally parentheses are used, When spanning lines within braces, indentation doesn’t matter, the indentation of the initial bracket used to determine which code section the whole statement belongs to. String statements can also be multi-line if one uses triple quotes.

**Python object types**

Like other programming languages, Python also has built-in data types that the programmer uses to create his program. The data types are the building blocks of the program. Depending on the programming language, different data types are available. Some languages, notably C and C++, have very primitive types, a lot of programming time is simply used up to combine these primitive types into useful data structures. Python does away with a lot of this difficult work. It already implements a wide range of types and structures, leaving the programmer more time to actually create the program. In c/c++ the programmer has to recreate the same data structure for every program.

Python has the following built-in types such as numbers, strings, lists, dictionaries, tuples, and files. Naturally, the user can build their own types if needed, but Python was created so that very rarely the user will have to roll their own. The built-in types are powerful enough to cover the vast majority of the code and are easily enhanced.

**Python numbers**

Python can handle normal long integers (max length determined based on the operating system like C), Python long integers (max length dependent on the available memory), floating point numbers (similar to C doubles), octal and hex numbers, and complex numbers ( the numbers with an imaginary component).

some examples of these numbers:

• integer: 2435, -62

• Python integer: 999999998L (In Python 3.x, all integers are Python integers)

• float: 3.24, 2e7, 5e-4

• octal: 012, 0454

• hex: 0xf33, 0X12FA

• complex: 2+4j, 3J, 5.0+3.5j

Python has the normal built-in numeric tools such as expression operators (\*, > >, +, <, etc.), math functions (pow, abs, etc.), and utilities (rand, math, etc.). For heavy number-crunching Python has the Numeric Python (NumPy) extension which has such things as matrix data types. It has to be installed separately on demand. It is heavily used in science and mathematical settings, as it is power and ease of use make it equivalent to Mathematica, Maple, and MatLab.

Logic operators are spelled out in Python rather than using symbols, for example logical AND is represented by “and”, logical OR is represented by “or”, and logical NOT uses “not” .

Operator level-of-precedence is same as that of C, but using parentheses is highly encouraged to ensure the expression is evaluated correctly and to enhance readability. Mixed types (float values combined with the integer values) are converted up to the highest type before evaluation, i.e. adding a float and an integer will cause the integer to be changed to a float value before the sum is evaluated

**Strings**

Strings in programming can be a simple text, individual characters, words, phrases, or complete sentences. They are one of the most common elements to use while programming, at least when it comes to interacting with the user. As they are so common, they are a native data type within Python, meaning they have many powerful capabilities built-in. Unlike other languages, the user don’t have to worry about creating these capabilities. This is good because the built-in ones have been tested many times and have been optimized for performance and stability

Strings in Python are different from most other languages. Firstly , there are no char types, only single character strings (char types are single characters, separate from actual strings, used for memory conservation). Strings also can’t be changed in-place, a new string object is created whenever the user wants to make changes to it, such as concatenation. This simply means the user have to be aware that he is not manipulating the string in memory, it doesn’t get changed or deleted while working with it.

some of the common string operations:

• s1 = ‘ ’ : empty string

• s2 = “knight’s” : double quotes

• block = “““ - ” ” ” : triple-quoted block

• s1 + s2 : concatenate (combine)

• s2 \* 3 : repeat the string a certain number of times

• s2[n] : index (the position of a certain character)

• len(s2) : get the length of a string

• “a %s parrot” % ‘dead’ : string formatting (deprecated in Python

3.x)

• “a {0} parrot”.format(“dead”) : string formatting (Python 3.x)

• for x in s2 : iteration (sequentially move through the string’s

characters)

• ‘m’ in s2 : membership (is there a given character in the string?)

**Statements**

The following is the listing of some of the Python statements.

Statement Role

Assignment Creating references

Calls Running functions

Print Printing objects

Print() Python 3.x print function

If/elif/else Selecting actions

For/else Sequence iteration

While/else General loops

Pass Empty placeholder

Break, Continue Loop jumps

Try/except/finally Catching exceptions

Raise Trigger exception

Import, From Module access

Def, Return Building functions

Class Building objects

.

**Assignment**

Assignment is basically putting the target name on the left of an equals sign and the object assigned to it on the right. There are only a few things to remember

• An assignment creates object references.

– It acts like pointers in C since it doesn’t copy objects, just refers to an object. Hence, one can have multiple assignments of the same object, (i.e. several different names referring for one object).

• Names are created when first assigned

– Names don’t have to be ‘pre-declared’, Python creates the variable name when it’s first created.

– Sometimes one may have to declare a name and give it an empty value, just like a “placeholder” for future use in the program.

In short assignment is just a name that can be reassigned to different objects. Since a name is just a reference to an object and doesn’t have to be declared, one can change it’s value to anything.

**Expressions/Calls**

Python expressions can be used as statements since the result won’t be saved, expressions are usually used to call functions or methods and for printing values at the interactive prompt.

**Printing**

Printing in Python is very simple. Using print writes the output to the C stdout stream, it normally goes to the console unless one redirects it to another file.

Python has 3 streams for input/output (I/O). sys.stdout is the standard output stream: it is normally send to the monitor but can be rerouted to a file or other location. sys.stdin is the standard input stream: it normally receives input from the keyboard but can also take input from a file or other location. sys.stderr is the standard error stream: it only takes errors from the program.

The print statement can be used with either the sys.stdout or sys.stderror stream. This allows one to maximize efficiency.

Printing, by default, adds a space between items separated by commas and adds a line feed at the end of the output stream. To suppress the line feed, just add a comma at the end of the print statement.

Python 3.x replaces the simple print statement with the print() function. This is to make it more powerful, such as allowing overloading.

**if Tests**

One of the most common control structures used to run into in other programs, is the if conditional block. Simply put, one asks a yes or no question, depending on the answer different things happen.

Unlike C, Pascal, and other languages, there isn’t a switch or case statement in Python. One can get the same functionality by using if/elif tests, searching lists, or indexing dictionaries. Since lists and dictionaries are built at runtime, they can be more flexible.

**while Loops**

while loops are a standard workhorse of many languages, the program will continue doing something while a certain condition exists. As soon as that condition fails, the loop stops. The Python while statement is similar to other languages.

Here’s the main format

while <test >: #loop test

<code block> #loop body

else : #optional else statement

<code block> #run if didn’ t exit loop with break

The break and continue statements works the exact same as in C. The equivalent of C’s empty statement (a semicolon) is the pass statement, and Python includes an else statement for use with breaks.

**for Loops**

The for loop is a sequence iterator for Python. It will work on nearly anything: strings, lists, tuples, etc.

The main format is below

for <target> in <object >: #assign object items to target

<statements>

if <test >: break #exit loop now, skip else

if <test >: continue #go to top of loop now

else :

<statements> #if a break is not hit

when the for loop starts, it looks at the first item in the list. This item is given a value of 0 (many programming languages start counting at 0, rather than 1). Once the code block is done doing its processing, the for loop looks at the second value and gives it a value of 1. Again, the code block does it’s processing and the for loop looks at the next value and gives it a value of 2. This sequence continues until there are no more values in the list. At that point the for loop stops and control proceeds to the next statement in the program.

Related to for loops are range and counter loops. The range() function auto-builds a list of integers for the user. It is used to create indexes for a for statement but one can use it anywhere.

Counter loops simply count the number of times the loop has been processed. At the end of the loop, a variable is incremented to show that the loop has been completed. Once a certain number of loops have occurred, the loop is executed and the rest of the program is executed.

**pass Statement**

The pass statement is simply a way to tell Python to continue moving, nothing to see here. I many of the cases, the pass statement is used while initially writing a program. The user may create a reference to a function but haven’t actually implemented any code for it yet. However, Python will be looks for something within that function. Without having anything to process, the Python interpreter will give an exception and stop when it doesn’t find anything. In this case if the user simply puts a pass statement in the function, it will continue on without stopping

**break and continue Statements**

These two statements affect the flow control within a loop. When a

particular condition is met, the break statement breaks out of the

loop, effectively ending the loop prematurely (though in an expected

manner). The continue statement short circuits the loop, causing

flow control to return to the top of the loop immediately.

**try, except, finally and raise Statements**

try creates a block that attemptsto perform an action. If that action fails, the except block catches any exception that is raised and does something about it. finally performs some last minute actions, regardless of whether an exception was raised or not. The raise statement manually creates an exception.

**import and from Statements**

These two statements are used to include other Python libraries and modules that a user wants to use in the program. This helps to keep the program small (the user don’t have to put all the code within a single module) and “isolates” modules (the user only imports what he needs). Import actually calls the other libraries or modules while from makes the import statement selective: the user only import subsections of a module, minimizing the amount of code brought into your program.

**def and return Statements**

These are used in functions and methods. Functions are used in procedural-based programming while methods are used in object-oriented programming. The def statement defines the function/method. The return statement returns a value from the function or method, allowing the user to assign the returned value to a variable.

**Class Statements**

These are the building blocks of OOP. class creates a new object. This object can be anything, whether an abstract data concept or a model of a physical object, e.g. a chair. Each class has individual characteristics unique to that class, including variables and methods. Classes are very powerful and currently the big thing in most programming languages.