Lesson 1 Handout v0.1

1 Terminology

1.1 **data** – Has many meanings. But it can be generally thought of as both a collection of details and each individual detail. Such details can be numeric: like the amount of money in a bank account, textual: such as the title of a book, or any of many other possible details which describe something.

1.2 **memory** – Typically used to refer to the Random Access Memory (RAM) that most computers possess. RAM is normally volatile – that is its contents is lost when a machine is turned off. It can help to know that while most veteran programmers usually use the word ‘memory’ when talking about the volatile sotrage some people also use it to refer to non-volatile storage like hard drives and solid state drives.

1.3 **abstract** – Or ‘Abstraction’; refers to the practice of hiding the details of exactly how something is accomplished so that a programmer only needs to concentrate on the details that are directly relevant to to the program being written. In the case of variables, many programming languages allow the programmer to concentrate on the type of data and its lifespan without having to worry about the hardware it is stored in or its location.

1.4 **variable** – A common mechanism that is available in many programming languages which allows for storage of data for later use. The variable can be thought of as a method which can ‘abstract’ the use of machine storage by a program from all the messy details that would otherwise need to be accounted for.

1.5 **label** – Programmers often find themselves deciding on names for the components of the software they design. These names are sometimes referred to as ‘labels’.

1.6 **assignment statement / assignment operator** – An assignment statement is a statement within a program which assigns some data or other value to a variable. The assignment operator is usually the equal sine (‘=’).

1.7 **statement** – Any discrete piece of a program which contains enough infromation to be executed in and of itself. Usually statements occupy a single line of a program, although this is not always the case.

1.8 **literal** – When data is contained directly within the source code of a program the text which represents the actual value is called a literal.

1.9 **operator** – Operators are used to direct some kind of action. ‘+ - * / %’ are the addition, subtraction, multiplication, division, and modulus operators respectively. The equal sign is usually used as the ‘assignment operator’.

1.10 **interpreter** – There are several different processes available to take the source code a person writes and translate it into actions executed by a computer. For Python this is usually done through another program called an ‘interpreter’. Python’s interpreter takes the scripts (which are the standard form of ‘source code’ for Python) and it ‘interprets’ them into a virtual machine code. This virtual machine code

is then executed by a Python virtual machine. Some other programming languages which use a virtual machine include Java, C#, and Ruby.

1.11 **state of a variable** – A variable's state is usually the value it contains or points to. Some languages also recognize special states for variables which are independent of the dta typically contained or pointed to by them. One such state that Python recognizes is 'None', which means a truly empty variable as opposed to the value of zero.

1.12 **expression** – Generally, any statement which can evaluate to a value. This includes mathematical expressions.

1.13 **data type** – The form of informational details. Some of the data types available in Python include float, long-integer, short-integer, and string types.

1.14 **integer** – Any whole number. It can be either positive or negative. Some programming languages recognize more than one type of integer, usually differentiated by the amount of memory space they require. Python has no hard limit as to the size of the whole number its integers can represent.

1.15 **float** – A number which includes a 'fractional' or 'less-than one' portion. here 1 ½ is equivalent to 1.5 .

1.16 **string** – A collection of letters, number, punctuation symbols and similar. When thinking of a string it helps to consider them to be similar to a collection of characters that are 'strung' together.

1.17 **built-in function** – Functions can be thought of as mini-programs, or sub-programs. They can be 'called', sent input data, and often 'return' some data or value. A call to a function includes the name of the function followed by a pair of parenthesis containing whatever data is to be sent to the function.

1.18 **arguments** / **parameters** – The data and/or values sent to a function.

2 Programming Elements

2.1 Operators:

2.1.1 **=** - the equal sign causes a value on the right to be 'assigned' to the variable on the right.

2.1.2 **+ - * /** % - mathematical operators (note that *, /, and % are multiply, divide, and modulus respectively)

2.2 Python Keywords:

This section intentionally left blank.

2.3 Python Functions:

2.3.1 **print(***list of arguments***)** - built-in function that prints out whatever data is passed to it as parameter(s).

3 Program

```
print(“Hot Dogs!”)

numHotDogs = 35
print(“We have”, numHotDogs, “hot Dogs.”)

weightHotDogOz = 1.50
print(“These hot dogs weigh”, weightHotDogOz, “ounces.”)

brand_hotDog = “Piggy Toes”
print(“They are”, brand_hotDog, “brand hot dogs.”)


print()
numHotDogsPerPack = 12
costPerHotDogPack = 3.65


costPerHotDog =   costPerHotDogPack /  numHotDogsPerPack


print(“These hot dogs cost $”, costPerHotDogPack, “per pack.”)
print(“And with”, numHotDogsPerPack, “per pack they cost $”, costPerHotDog, “each.”)


print()
numBunsPerPack = 12
costPerBunPack = 1.49

costPerBun =  costPerBunPack /  numBunsPerPack

print(“The buns cost $”,  costPerBunPack, “per pack.”)
```