THE KNOWLEDGE HOUSE

MODULE 5

PYTHON PROGRAMMING

OBJECTIVE

To read and understand the basics of Python Programming Fundamentals.

ANSWER THE FOLLOWING QUESTIONS IN A .TXT FILE:

WHAT_IS_A_PROGRAMMING

- Explain in your own words what is Python? (300 Words)
- Explain Data Python Data Types (200 Words)
- Explain in your own words What is a function? (250 Words)
- Explain the Python print statement (250 Words)
- Complete the attached notebook named pre_training.ipynb

MODULE 5
PYTHON PROGRAMMING

WHAT IS PROGRAMMING?

At its core programming is simply the act of writing commands to a computer so it can be used to store, retrieve, and manipulate data.
Computers process data in a different way than you and me. The human brain is good at doing many things well in an ambiguous environment. We have evolved our brains to be able to adapt to many different circumstances and because of this our brains are great at doing a bunch of things with incomplete information. However, because we aren't specialized in any one thing our brain is not great at doing things in the fastest, most consistent and most efficient ways possible. Computers are the opposite. They use specific instructions to do specific things in a specific environment. Any changes will cause the computer to break down. They can not deal with ANY ambiguity, everything needs to be defined on one level, all the way down to the basic units of computing: 0 and 1 or True or False.
All data we use, from the pictures on your phone to the videos you watch on Netflix, are basically structures built on this foundation. Where computers beat the human brain is in repetition and consistency.
Imagine if you were asked to file a bunch of reports away in a file cabinet by alphabetical order. If asked to do this for a stack of 100 papers it would take you a while, and there is a high chance of making an error in such mindless work. Now imagine having to do that for daily Google searches which consist of millions on millions of searches. A human could not do this, even if they worked through it there is an almost nonexistent chance there will be no mistakes. However, this is what google's servers do on a daily basis.
They do this because systems were programmed to do a specific thing, and can do this thing several times a second. The thing, in this case, is saving the entry in a database that can later be analyzed.
This is what programming is essentially, you telling the computer how to solve very specific problems for you. The goal is to write a program that solves the problem the most consistently in the fastest time.
Consistently: While computers are predictable in what they are made to do, the data we give them to work with often isn't. For example, oftentimes people may submit messy or simply incorrect data into a system. For example, if a field asks you for your name and you give that field a number, what should it do?
A poorly designed system might just take the response, whereas a better-designed one will be instructed to check if any of the entries contained numbers, and tell the user "Hey re-enter cuss it's wrong". Another example is capitalization, computers view 'a' and "A" as two separate letters. So "joe', and "Joe" might be entered as two separate names. A system that wants to check for duplicates and treat all data the same may want to make all capital letters lower case before checking for example.

MODULE 5
PYTHON PROGRAMMING

WHAT IS PROGRAMMING? CONT.

The concept of planning for user error is called planning for edge cases.

Speed: In addition to writing code that is able to perform its job with minimal errors taking into account user activity, applications need to be written with the minimal steps and maximum constraints possible. Every additional step slows the execution and takes more power to run. Computers are not magic; they run off electricity and heavier more complicated calculations take more energy.
Now the difference in speed between one way of running a calculation one way vs a more optimized way may be tiny. However in our example of saving google searches that calculation will run millions and millions of times. All those tiny incremental costs and additional bandwidth needed may be the difference between an entire site going down during a time when the most users are active (cyber Monday) or even if a crash does not happen it may cost millions in additional energy costs over time.

This is why writing good software makes someone very valuable.

DATA

If programming is the verb data is the noun.

Programming is the act of using a computer to manipulate data. All things on your computer that you interact with are data. The Binary (0&1) code is the DNA that comprises every tweet, email, image, video you see on the web, and every number you see in a report or chart you view online.
A picture is literally a set of instructions telling the computer to change the balance between the colors Green, Blue, and Red on the pixels on your screen. The same goes for video. Audio is the same thing with the speakers instead of the pixels. Your computer is literally reading a message saying "Make this noise in this timing".

PYTHON FUNDAMENTALS

With hundreds of questions about how to get started with **Python for DS** on various forums, this post (and video series) is my attempt to settle all those questions. I'm a Python evangelist that started off as a Full Stack Python Developer before moving on to data engineering and then data science. My prior experience with Python and a decent grasp of math helped make the switch to data science more comfortable for me.

MODULE 5
PYTHON PROGRAMMING

PYTHON FUNDAMENTALS CONT.

So, here are the fundamentals to help you with programming in Python. Before we take a deep dive into the essentials, make sure that you have set up your Python environment and know how to use a Jupyter Notebook (optional).

A basic Python curriculum can be broken down into 4 essential topics that include:

- Data types (int, float, strings)
- Compound data structures (lists, tuples, and dictionaries)
- Conditionals, loops, and functions
- Object-oriented programming and using external libraries

Let's go over each one and see what are the fundamentals you should learn.

- **Data Types and Structures**
  The very first step is to understand how Python interprets data. Starting with widely used data types, you should be familiar with integers (int), floats (float), strings (str), and booleans (bool). Here's what you should practice.
  Type, typecasting, and I/O functions:
  - Learning the type of data using the type() method.

```
type('Harshit')

# output: str
```

  - Storing values into variables and input-output functions (a = 5.67)
  - Typecasting — converting a particular type of variable/data into another type if possible. For example, converting a string of integers into an integer:

```
astring = "55"
print(type(astring))

# output: <class 'str'>
astring = int(astring)
print(type(astring))

# output: <class 'int64'>
```

MODULE 5
PYTHON PROGRAMMING

PYTHON FUNDAMENTALS CONT.

But if you try to convert an alphanumeric or alphabet string into an integer, it will throw an error:

```
In [35]: astring = "5TOM"
         print(type(astring))

         <class 'str'>

In [36]: anum = int(astring)
         print(anum)

         ---------------------------------------------------------------------------
         ValueError                                Traceback (most recent call last)
         <ipython-input-36-9c6b57c73e9c> in <module>
         ----> 1 anum = int(astring)
               2 print(anum)

         ValueError: invalid literal for int() with base 10: '5TOM'
```

Once you are familiar with the basic data types and their usage, you should learn about **arithmetic operators and expression evaluations (DMAS)** and how you can store the result in a variable for further use.

```
answer = 43 + 56 / 14 - 9 * 2
print(answer)

# output: 29.0
```

Strings:
Knowing how to deal with textual data and their operators comes in handy when dealing with the string data type. Practice these concepts:

- Concatenating strings using +
- Splitting and joining the string using the split() and join()method
- Changing the case of the string using lower() and upper() methods
- Working with substrings of a string

Here's the **Notebook** that covers all the points discussed.

MODULE 5
PYTHON PROGRAMMING

PYTHON FUNDAMENTALS CONT.

- **Compound data structures (lists, tuples, and dictionaries)**

  Lists and tuples (compound data types):
  One of the most commonly used and important data structures in Python are lists. A list is a collection of elements and the collection can be of the same or varied data types.
  Understanding lists will eventually pave the way for computing algebraic equations and statistical models on your array of data.

  Here are the concepts you should be familiar with:

  - How multiple data types can be stored in a Python list.
  - Indexing and slicing to access a specific element or sub-list of the list.
  - Helper methods for sorting, reversing, deleting elements, copying, and appending.
  - Nested lists — lists containing lists. For example, [1,2,3, [10,11]].
  - Addition in a list.

  alist + alist

  # output: ['harshit', 2, 5.5, 10, [1, 2, 3], 'harshit', 2, 5.5, 10, [1, 2, 3]]
  Multiplying the list with a scalar:

  alist * 2

  # output: ['harshit', 2, 5.5, 10, [1, 2, 3], 'harshit', 2, 5.5, 10, [1, 2, 3]]

```
In [56]: alist*2
Out[56]: ['harshit', 2, 5.5, 10, [1, 2, 3], 'harshit', 2, 5.5, 10, [1, 2, 3]]

In [57]: alist + alist
Out[57]: ['harshit', 2, 5.5, 10, [1, 2, 3], 'harshit', 2, 5.5, 10, [1, 2, 3]]
```

Tuples are an immutable ordered sequence of items. They are similar to lists, but the key difference is that tuples are immutable whereas lists are mutable.

Concepts to focus on:

- Indexing and slicing (similar to lists).
- Nested tuples.
- Adding tuples and helper methods like count() and index().

MODULE 5
PYTHON PROGRAMMING

PYTHON FUNDAMENTALS CONT.

Dictionaries

These are another type of collection in Python. While lists are integer indexed, dictionaries are more like addresses. Dictionaries have key-value pairs, and keys are analogous to indexes in lists.

Keys	Values
Key 1	Value 1
Key 2	Value 2
Key 3	Value 3
Key 4	Value 4

To access an element, you need to pass the key in squared brackets.

```
In [29]: # dictionary
         country_code = {'India': 1, 'USA': 2, 'China': 3}
         print(country_code)

         {'India': 1, 'USA': 2, 'China': 3}

In [30]: print(country_code['China'])

         3
```

Concepts to focus on:

- Iterating through a dictionary (also covered in loops).
- Using helper methods like get(), pop(), items(), keys(), update(), and so on.

Notebook for the above topics can be found **here**.

MODULE 5
PYTHON PROGRAMMING

PYTHON FUNDAMENTALS CONT.

- **Conditionals, Loops, and Functions**

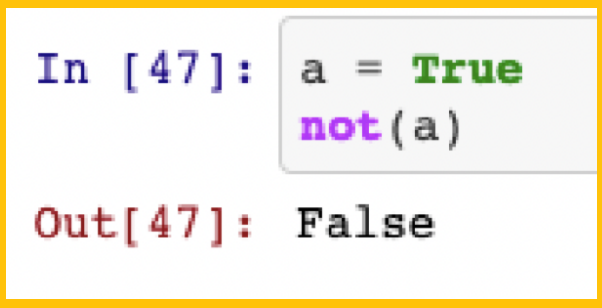
  Conditions and Branching
  Python uses these boolean variables to assess conditions. Whenever there is a comparison or evaluation, boolean values are the resulting solution.

```
x = True

print(type(x))

# output: <class bool>
print(1 == 2)

# output: False
```

  The comparison in the image needs to be observed carefully as people confuse the assignment operator (=) with the comparison operator (==).
  Boolean operators (or, and, not)

These are used to evaluate complex assertions together.

- Ior — One of the many comparisons should be true for the entire condition to be true.
- and — All of the comparisons should be true for the entire condition to be true.
- not — Checks for the opposite of the comparison specified.




```
score = 76
percentile = 83

if score > 75 or percentile > 90:
    print("Admission successful!")
else:
    print("Try again next year")

# output: Try again next year
```

Concepts to learn:

- if, else, and elif statements to construct your condition.
- Making complex comparisons in one condition.
- Keeping indentation in mind while writing nested if / else statements.
- Using boolean, in, is, and not operators.

MODULE 5
PYTHON PROGRAMMING

PYTHON FUNDAMENTALS CONT.

Loops

Often you'll need to do a repetitive task, and loops will be your best friend to eliminate the overhead of code redundancy. You'll often need to iterate through each element of a list or dictionary, and loops come in handy for that. while and for are two types of loops.

Focus on:

- The range() function and iterating through a sequence using for loops.
- while loops

```
age = [12,43,45,10]
i = 0
while i < len(age):
    if age[i] >= 18:
        print("Adult")
    else:
        print("Juvenile")
    i += 1

# output:
# Juvenile
# Adult
# Adult
# Juvenile
```

- Iterating through lists and appending (or any other task with list items) elements in a particular order

```
cubes = []
for i in range(1,10):
    cubes.append(i ** 3)
print(cubes)

#output: [1, 8, 27, 64, 125, 216, 343, 512, 729]
```

- Using break, pass, and continue keywords.

MODULE 5
PYTHON PROGRAMMING

PYTHON FUNDAMENTALS CONT.

List Comprehension

A sophisticated and succinct way of creating a list using an iterable followed by a for clause. For example, you can create a list of 9 cubes as shown in the example above using list comprehension.

```
# list comprehension
cubes = [n** 3 for n in range(1,10)]
print(cubes)
# output: [1, 8, 27, 64, 125, 216, 343, 512, 729]
```

Functions

While working on a big project, maintaining code becomes a real chore. If your code performs similar tasks many times, a convenient way to manage your code is by using functions.
A function is a block of code that performs some operations on input data and gives you the desired output. Using functions makes the code more readable, reduces redundancy, makes the code reusable, and saves time.

Python uses indentation to create blocks of code. This is an example of a function:

```
def add_two_numbers(a, b):
    sum = a + b
    return sum
```

We define a function using the def keyword followed by the name of the function and arguments (input) within the parentheses, followed by a colon.
The body of the function is the indented code block, and the output is returned with the return keyword.
You call a function by specifying the name and passing the arguments within the parentheses as per the definition.

```
In [66]: add_two_numbers(5,11)

Out[66]: 16
```

More examples and details **here.**

MODULE 5
PYTHON PROGRAMMING

PYTHON FUNDAMENTALS CONT.

- **Object-Oriented programming and using external libraries**

  We have been using the helper methods for lists, dictionaries, and other data types, but where are these coming from?
  When we say list or dict, we are actually interacting with a list class object or a dict class object.
  Printing the type of dictionary object will show you that it is a class dict object.

```
In [7]: adict = {'US': 1897897}
        print(type(adict))

        <class 'dict'>
```

These are all pre-defined classes in the Python language, and they make our tasks very easy and convenient.
Objects are instances of a class and are defined as an encapsulation of variables (data) and functions into a single entity. They have access to the variables (attributes) and methods (functions) from classes.
Now the question is, can we create our own custom classes and objects? The answer is YES.

Here is how you define a class and an object of it:

```
class Rectangle:
    def __init__(self, height, width):
        self.height = height
        self.width = width

    def area(self):
        area = self.height * self.width
        return area

rect1 = Rectangle(12, 10)

print(type(rect1))

# output: <class '__main__.Rectangle'>
```

MODULE 5
PYTHON PROGRAMMING

PYTHON FUNDAMENTALS CONT.

You can then access the attributes and methods using the dot(.) operator.

```
In [12]: rect1.height
Out[12]: 12

In [13]: rect1.width
Out[13]: 10

In [14]: rect1.area()
Out[14]: 120
```

Using External Libraries/Modules

One of the main reasons to use Python for data science is the amazing community that develops high-quality packages for different domains and problems. Using external libraries and modules is an integral part of working on projects in Python.
These libraries and modules have defined classes, attributes, and methods that we can use to accomplish our tasks. For example, the math library contains many mathematical functions that we can use to carry out our calculations. The libraries are .py files.

You should learn to:

- Import libraries in your workspace

```
In [16]: import math

In [17]: print(type(math))
         <class 'module'>
```

MODULE 5
PYTHON PROGRAMMING

PYTHON FUNDAMENTALS CONT.

Using External Libraries/Modules Cont.

- Using the help function to learn about a library or function

```
In [19]: help(math)

Help on module math:

NAME
    math

MODULE REFERENCE
    https://docs.python.org/3.7/library/math

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x
```

- Importing the required function directly.

```
In [27]: from math import log, pi
         from numpy import asarray

         log(100,10)

Out[27]: 2.0
```

- How to read the documentation of the well-known packages like pandas, numpy, and sklearn and use them in your projects