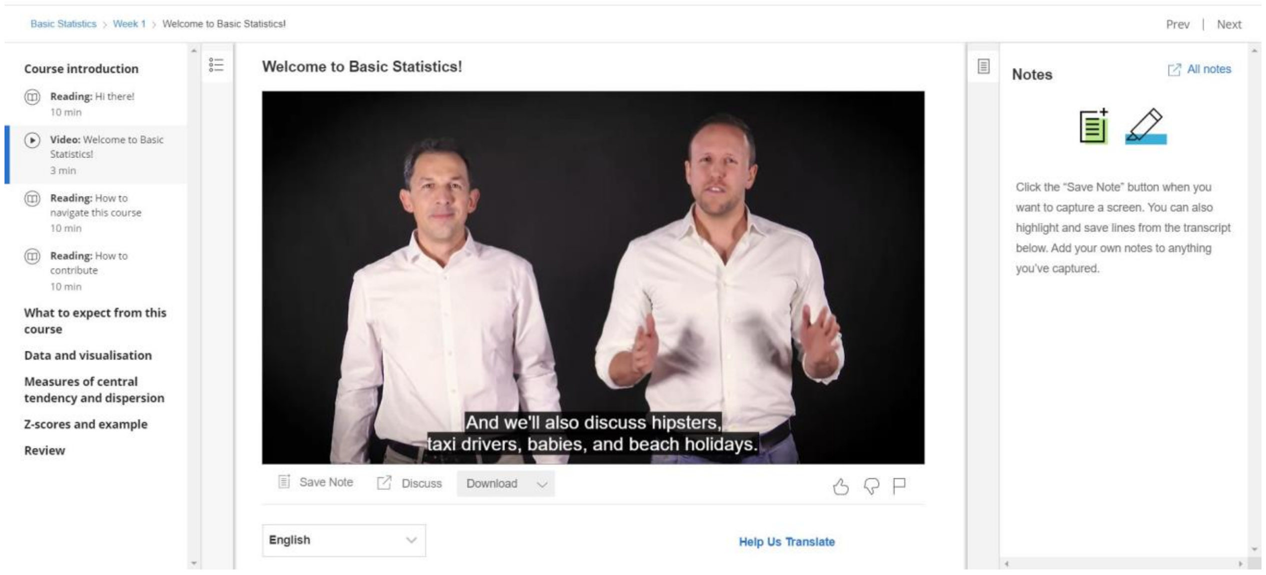
DAILY ASSESSMENT FORMAT

Date:	**21 JULY 2020**	**Name:**	**PAVITHRAN S**
Course:	**COURSERA**	**USN:**	**4AL17EC068**
Topic:	**STATISTICS**	**Semester & Section:**	**6TH B**
Github Repository:	**Pavithran**		

FORENOON SESSION DETAILS

Image of session



Basic Statistics > Week 1 > Welcome to Basic Statistics!
Prev | Next
Course introduction
Reading: Hi there! 10 min
Video: Welcome to Basic Statistics! 3 min
Reading: How to navigate this course 10 min
Reading: How to contribute 10 min
What to expect from this course
Data and visualisation
Measures of central tendency and dispersion
Z-scores and example
Review
Welcome to Basic Statistics!
And we'll also discuss hipsters, taxi drivers, babies, and beach holidays.
Save Note
Discuss
Download
English
Help Us Translate
Notes
All notes
Click the "Save Note" button when you want to capture a screen. You can also highlight and save lines from the transcript below. Add your own notes to anything you've captured.

Report – Report can be typed or hand written for up to two pages.

Why Statistics?

Statistical methods are mainly useful to ensure that your data are interpreted correctly. And that apparent relationships are really "significant" or meaningful and it is not simply happen by chance. Actually, the statistical analysis helps to find meaning to the meaningless numbers.

So, a "statistic" is nothing but some numerical value to that can describe certain property of your data set. There are few well know statistics are the average (or "mean") value, and the "standard deviation" etc. Standard deviation is the variability within a data set around the mean value. The "variance" is the square of the standard deviation. The linear trend is another example of a data "statistic".

Steps in the Data Analysis Process

Before staring Data Analysis pipeline you should know there are mainly five steps involved into it.

Step 1: Decide on the objectives or Pose a Question:

The first step of the data analysis pipeline is to decide on objectives. These objectives may usually require significant data collection and analysis.

Step 2: What to Measure and How to Measures

Measurement generally refers to the assigning of numbers to indicate different values of variables. Suppose, through your research you are trying to find if there was a relationship between height and weight of human, it would make sense to measure the height and weight of dogs using a scale.

Step 3: Data Collection

Once you know what types of data you need for your statistical study then you can determine whether your data can be gathered from existing sources/databases or not. If data is not sufficient the you have to collect new data. Even if you have existing data, it is very important to know how the data was collected? This will helps you to understand you ca determine the limitations of the generalizability of results and conduct a proper analysis.

The more data you have, the more better correlations, building better models and finding more actionable insights is easy for you. Especially data from more diverse sources helps to do this job easier way.

Step 4: Data Cleaning

This is another crucial step in data analysis pipeline is to improve data quality for your existing data. Too often Data scientists correct spelling mistakes, handle missing values and remove useless information. This is the most critical step because junk data may generate inappropriate results and mislead the business.

Step 5: Summarizing and Visualizing Data

Exploratory data analysis helps to understand the data better. Because a picture is really worth a thousand words as many people understand pictures better than a lecture. Likewise, Measures of Variance indicate the distribution of the data around the center. Correlation refers to the degree to which two variable move in sync with one another.

Step 6: Data Modeling

Now build models that correlate the data with your business outcomes and make recommendations. This is where the unique expertise of data scientists becomes important to business success. Correlating the data and building models that predict business outcomes

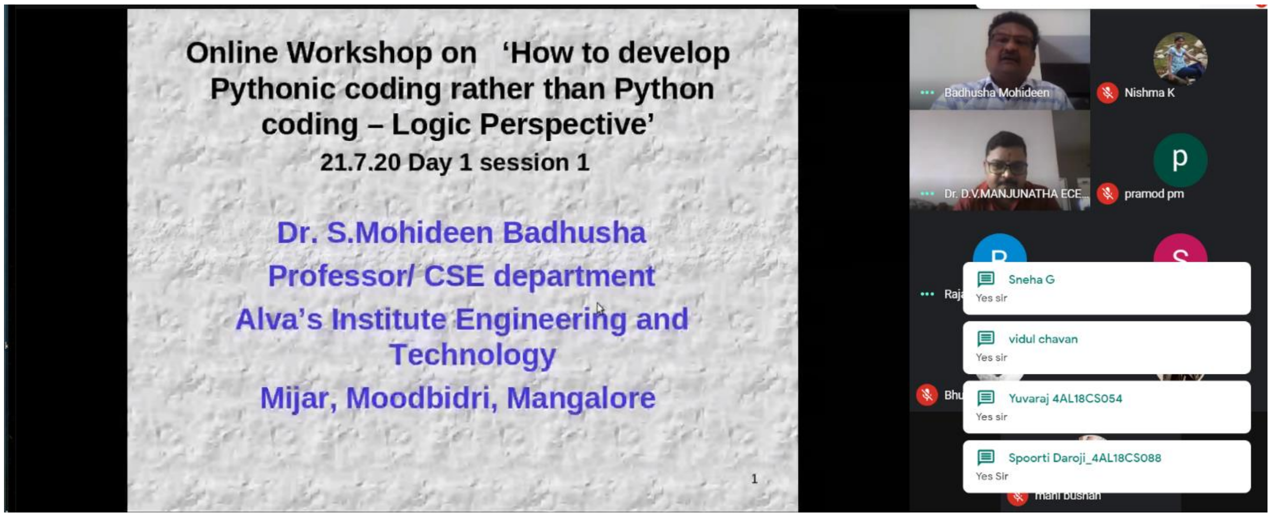
Step 7: Optimize and Repeat

The data analysis is a repeatable process and sometime leads to continuous improvements, both to the business and to the data value chain itself.

Now you know steps involved in Data Analysis pipeline. Before advancing to more sophisticated techniques, I suggest starting your data analysis journey with the following statistics fundamentals –

Here is a road map for getting started with Data Analysis. Before starting any statistical data analysis, we need to explore data more and more. To explore data below topics are very useful.

Date:	**21 JULY 2020**	**Name:**	**PAVITHRAN S**
Course:	**PYTHON**	**USN:**	**4AL17EC068**
Topic:	**How to develop Pythonic coding rather Than Python coding**	**Semester & Section:**	**6TH B**
Github Repository:	**Pavithran**		

AFTERNOON SESSION DETAILS
Image of session 

Online Workshop on 'How to develop Pythonic coding rather than Python coding – Logic Perspective'
21.7.20 Day 1 session 1

Dr. S.Mohideen Badhusha
Professor/ CSE department
Alva's Institute Engineering and Technology
Mijar, Moodbidri, Mangalore

1

Sneha G
Yes sir

vidul chavan
Yes sir

Yuvaraj 4AL18CS054
Yes sir

Spoorti Daroji_4AL18CS088
Yes Sir

Report – Report can be typed or hand written for up to two pages.

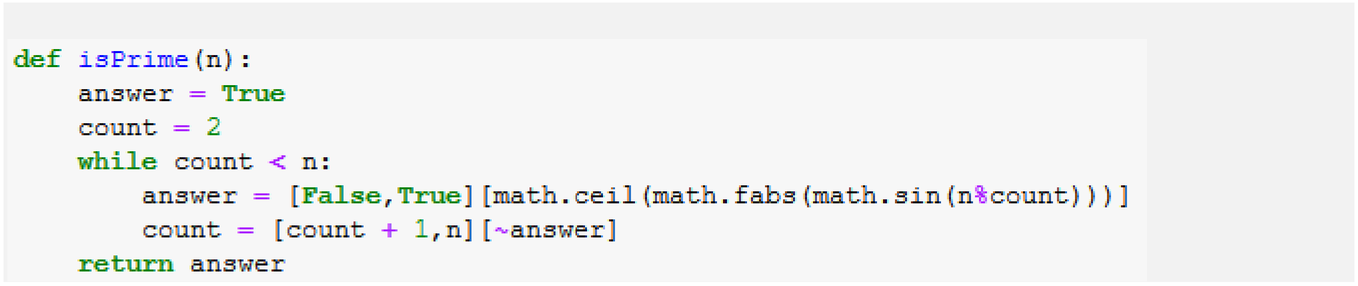
Pythonic — A bionic Python?

As with nearly every programming language, there are certain stylistic and conventional guidelines that are accepted by the Python community to promote unified, maintainable, and concise applications that are written the way the language intended them to be written. These guidelines range from proper variable, class, and module naming conventions, to looping structures, and even the proper way to wrap lines of code. The name "Pythonic" was coined to describe any program, function, or block of code that follows these guidelines and takes advantage of Python's unique capabilities.

Why does all of this matter? This question is open to many interpretations, but a few key reasons why you should care come down to the clarity, efficiency, and credibility of the code. Let's break these down further.

Clarity

The clarity of your code is paramount to your success if you want to be a developer. As you grow in the field, you will likely work with others at some point in time, which will require peers to read your code. If your code is written poorly, it can be a nightmare for others to decipher your intentions, even in short chunks.



```
def isPrime(n):
    answer = True
    count = 2
    while count < n:
        answer = [False,True][math.ceil(math.fabs(math.sin(n%count)))]
        count = [count + 1,n][~answer]
    return answer
```

The definition of insanity

Does this code work? Yup. Does the function name describe the function's purpose? Sure does. Is it easy to identify what this code is supposed to accomplish if you changed the function name? Probably not without spending an hour analyzing it.

As is the case of every beginning developer I have known (myself included), there is a commonly-held mentality of “it works — don’t touch it” when it comes to code. The moment we can write something that solves our problem, we are afraid of doing anything to the code in fear that we will break everything and be unable to fix it again.

I would encourage any developer to break this mentality as early as possible (this goes for all languages). Even if you created the poorly-written code yourself, it is often difficult to return to it a week, month, or even a year later and attempt to unravel its mystery. To make matters worse, if you can’t decipher the code yourself, how do you expect fellow teammates or collaborators to uncover the meaning?

By writing programs the way the language was intended, developers should naturally be writing code that looks similar to that of their peers. This makes it easy to understand, easy to share, and easy to update.

Efficiency

Back when I was interning in college, one of my fellow interns I met on the job told me “don’t bother writing something that’s already been done in Python, because you won’t be able to write something better.” While I was originally frustrated by this depressing thought, I eventually realized there was some truth to his statement. Python has been around for nearly three decades at this point and has quickly become one of the most popular languages by developers around the world. Python is also known for containing an abundance of libraries that can do almost anything you want or need. Many of these libraries and features see thousands of members creating updates over several years, squeezing as much performance out of every line of code as possible. While you are certainly welcome to writing your own optimal string comparison function, chances are what you come up with won’t be any faster than what already exists, and the time spent developing the new function could have been spent working on the actual problem you are attempting to solve. In general, look for a built-in function or data type that achieves what you are looking for. Chances are, this will be the fastest way to complete a task. If not, check if there are any libraries or packages that can be installed which do what you need. If you still don’t have a solution, now’s the time to create your own!

Credibility

For anyone who first learned how to program in a language other than Python, it’s generally clear which language the developer came from. Take the following problem as an example:

- Find the sum of all numbers between 10 and 1,000

A C (or C++) developer would probably write something along the following lines:

```
int a = 10;
int b = 1000;
int total_sum = 0;while (b >= a) {
    total_sum += a;
    a++;
}
```

A direct Python re-write of this would look very similar:

```
a = 10
b = 1000
total_sum = 0while b >= a:
    total_sum += a
    a += 1
```

While the above statement will yield the expected output, most Python developers would throw a fit over this code, complaining that it isn’t Pythonic and doesn’t leverage the language’s power. Starting fresh, here’s how you can solve the problem the Pythonic way:

```
total_sum = sum(range(10, 1001))
```

This single line of code generates the exact same result as above (for the record, I *did* intend to write `1001` in the code as Python’s `range` command has an inclusive lower bound and a non-inclusive upper bound, meaning the lower number will be a part of the loop, while the higher number will not). If you were to write Python code using the first example, your credibility as a Python developer would go down as the Python community is very passionate about writing code following the guidelines. Here’s another example:

- Determine if a particular string is in an array

For most non-Python developers, the first solution would probably look something like this:

```
#include <stdbool.h>
char * arr[] = {"apples", "oranges", "bananas", "grapes"};
char * s = "cherries";
bool found = false;
int len = sizeof(arr) / sizeof(arr[0]);for (int i = 0; i < len; i++) {
    if (!strcmp(arr[i], s)) {
        found = true;
    }
}
```

As before, a direct Python translation would be:

```
arr = ["apples", "oranges", "bananas", "grapes"]
s = "cherries"
found = False
size = len(arr)for i in range(0, size):
    if arr[i] == s:
        found = True
```

As I’m sure you guessed, there’s a much simpler way to write this in Python:

```
arr = ["apples", "oranges", "bananas", "grapes"]
found = "cherries" in arr
```

No matter which method you choose above, `found` will always evaluate to `False` (or `false`) in the end. The last choice, however, is the clear champion when it comes to Pythonic code. It is concise and easily understandable. Even those that have never read Python (or any code for that matter) have a chance at comprehending the intention of this last block unlike the previous two.

The final example is one of my favorite tools in Python, **list comprehension**. This technique allows you to embed a loop inside a list to create a new list. Consider the following:

- Double the value of every even value in an array

First, here’s the C code:

```
int[] arr = { 1, 2, 3, 4, 5, 6 };
int length = sizeof(arr) / sizeof(arr[0]);for (int i = 0; i < length; i++) {
    if (arr[i] % 2 == 0) {
        arr[i] *= 2
    }
}
```

And the direct Python translation:

```
arr = [1, 2, 3, 4, 5, 6]
length = len(arr)for i in range(0, length):
    if arr[i] % 2 == 0:
        arr[i] *= 2
```

Now the Pythonic way:

```
arr = [1, 2, 3, 4, 5, 6]
arr = [x * 2 if x % 2 == 0 else x for x in arr]
```

This might look funny at first if you have never seen list comprehension in action. I’ve found it’s often easiest to look at list comprehension from right to left. First, it iterates through every element in the list `for x in arr`, then it checks if the element is even `if x % 2 == 0`. If so, it doubles the number `x * 2`, and stays the same if not `else x`. Whatever the element ends up as, it gets appended to a new list. In our case, we are overwriting the original value of `arr` with the new list. These are just a few common ways to make code Pythonic. You likely noticed that all of these examples involved loops of some sort. While there are many ways to write Pythonic code, a great practice is to ask yourself if you truly need a loop or if it can be replaced with an idiomatic substitute. If you care about your credibility in the software world and want to proudly call yourself a Python developer, make sure you know and use some of these techniques when applicable in your code.