TUX GLOBAL INSTITUTE

Data science and Analytics

- Duration : 45 Hours
- Monday : 6:00pm to 8:00pm
- Wednesday : 6:00pm to 8:00pm
- Thursday : 6:00pm to 8:00pm

Python for Data Science

Note that we will focus on particular aspects of Python that would be important for someone who wants to

- ✓ load in some data sets,
- ✓ perform some computations on them,
- ✓ and plot some of the results.

Therefore, we will mostly be talking about Python's built-in data structures and libraries from the perspective of processing and manipulating structured and unstructured data.

Why Python?

The practice of data science involves many interrelated but different activities, including

- Accessing data,
- manipulating data,
- Computing statistical summaries or business metrics,
- plotting/grahing/visualizing data,
- Building predictive and explanatory models,
- evaluating those models, and finally,
- integrating models into productions systems

One option for the data scientist is to learn several different software packages that each specialize in one of these things, or to use a general-purpose, high-level programming language that provides libraries to do **all** these things.

Why Python?

Python is **an excellent choice** for this. It has a diverse range of open source libraries for just about everything the data scientist will do. Some of its highlights include:

- **Cross-platform** – high performance python interpreters exist for running your code on almost any operating system (Windows, Mac or Linux) or architecture.
- **Free** – Python and most of its libraries are both open source and free.
- **Simple** – It has efficient high-level data structures and a simple but effective approach to object-oriented programming.
- **Elegant syntax -** which, together with its interpreted nature, makes it an ideal language for scripting and rapid application development in many areas on most platforms

Python: Writing Pythonic Code

- ✓ You will often read questions on StackOverflow like, 'What is a more Pythonic way of doing X.'
- ✓ To know what that means, read **The Zen of Python.** Simply run `import this` on any Python interface.
- ✓ It is a description of its design principles, and code written using these principles is called 'Pythonic.'
- ✓ While there are typically multiple ways to crack a given problem, we will generally favor Pythonic solutions over shabby ones.

Python: Package Managers

Do read about **pip** and **conda** – both of which will act as your package/library managers.
To install libraries that aren't part of the Anaconda distribution, you will be using commands such as

- `pip install ggplot`
- `conda install ggplot`

What do you mean Python Basics?

Python Basic programming topics

- ✓ Basic Rules
- ✓ Declaring & Printing variables
- ✓ Objects, Methods, Attributes and Functions
- ✓ Using built-in functions
- ✓ Modules (Libraries)
- ✓ Data Types
- ✓ Basic Operators: Arithmetic using binary operators
- ✓ Dealing with Strings
- ✓ Control flow statements
- ✓ Control Flow with if, elif, else
- ✓ Loops
- ✓ Data Structures
- ✓ Working with Collections - List, Tuple, Set & Dictionary
- ✓ Functions – User defined functions
- ✓ Lambda functions
- ✓ Classes

Python: Basic Rules

Comments:
Any text preceded by the hash mark (pound sign) # is ignored by the Python interpreter.

Whitespace Formatting (Indentation):
Many languages like R, C++, Java, and Perl use curly braces to delimit blocks of code. Python uses whitespace indentation to make code more readable and consistent. A colon denotes the start of an indented code block after which all of the code must be indented by the same amount

One major reason th at whitespace matters is th at it results in most Python code looking cosmetically similar, which means less cogniti ve dissonance when you read a piece of code that you didn't wri te yourself.

Python uses indentation for blocks, instead of curly braces. Both tabs and spaces are supported, but the standard indentation requiresstandard Python code to use four spaces. For example

Declaring & Printing variables

Variable are dynamically typed, so no need to mention the variable types. Python interpreter can automa tically infer the type when the variables are initialized. The simplest directive in Python is the "print" directive- it simply prints out a line

```
In [1]: var1 = 2
        var2 = 5.0

In [2]: var1
Out[2]: 2

In [3]: type( var1 )
Out[3]: int

In [4]: type( var2 )
Out[4]: float
```

```
In [5]: print( var1 )
        2

In [6]: mystring = 'This is python'
        print( mystring )
        This is python

In [7]: print( var1, var2, mystring )
        2 5.0 This is python
```

There is difference between Python 2 and 3 for the print statement. In Python 2, the "print" statement is not a function, and therefore it is invoked without parentheses. However, in Python 3, it is a function, and must be invoked with parentheses.

Objects, Methods, Attributes and Functions

Every number, string, data structure, function, class, module, and so on exists in the Python interpreter is referred to as a **Python object**.

Each object has an associated

- **type** (int, float, list, dict, str and so on…)
- attached functions, known as **methods**,
  - these have access to the object's internal data.
    - They can be called using the syntax: *obj.<method>(parameters)*
- **attributes** which can be accessed with the syntax: *obj.attribute*

"Functions are called using parentheses and passing zero or more arguments, optionally assigning the returned value to a variable: *result = f(x, y, z)*"

Let's discuss classes & objects later once we have done some basic topics in python

Built in functions

Functions comes with python base version, called built in functions.

Example: round()

```
In [12]: round( 1.234 )
Out[12]: 1
```

Round upto a number of decimal values

```
In [13]: round( 1.234, 2 )
Out[13]: 1.23
```

To invoke some functions that packaged need to be imported.

For example import a math function

```
In [14]: import math

In [15]: math.ceil( 1.2 )
Out[15]: 2

In [16]: math.floor( 1.2 )
Out[16]: 1
```

```
In [17]: abs( -1.2 )
Out[17]: 1.2

In [18]: # Get the variable type
         type( var1 )
Out[18]: int

In [19]: pow( var1 , 2 )
Out[19]: 4
```

Modules (Libraries) - Packages

Certain functions in Python are not loaded by default.

These include both features included as part of the language as well as third-party features that you download explicitly. In order to use these features, you'll need to **import the modules** that contain them.

>In Python a module is simply a .py file containing function and variable definitions. You

can import the module itself as: `import pandas`

But after this you'll have to always access its functions by prefixing them with the module name,

For example : `pandas.Series()`
Alternatively, we can provide an alias: `import pandas as pd`

This will save us some typing as we can then write `pd.Series()` to refer to the same thing.
Another option is to import frequently used functions explicitly and use them without any prefixes. For example,

```
from pandas import Series
```

Tip: Importing everything from a module is possible, but is considered bad practice as it might interfere with variable names and function definitions in your working environment.
So **avoid** doing things like: `from pandas import *`

Modules (Libraries) - Packages

Exploring built-in modules:

- ✓ Two very important functions come in handy when exploring modules in Python- the dir and help functions.
- ✓ We can look for which functions are implemented in each module by using thedir function
- ✓ When we find the function in the module we want to use, we can read about it moreusing the help function, inside the Python interpreter

Writing modules

- ✓ Writing Python modules is very simple. To create a module of your own, simply create a new .py file with the module name, and then import it using the Python file name (without the .py extension) using the import command.

Writing packages

- ✓ Packages are namespaces which contain multiple packages and modules themselves. They are simply directories, but with a twist.
- ✓ Each package in Python is a directory which **MUST** contain a special file called __init__.py. This file can be empty, and it indicatesthat the directory it contains is a Python package, so it can be imported the same way a module canbe imported.
- ✓ If we create a directory called foo, which marks the packagename, we can then create a module inside that package called bar. We also must not forget to add the__init_.py file inside the foo directory.

Data Types

Python supports two types of numbers – integers and floating point numbers. (It also supports complex numbers, which will not be explained in this tutorial).

Python has a small set of built-in types for handling numerical data, strings, boolean (True or False) values, and dates and time. These include

- *None* – The Python Null Value
- *str, unicode* – for strings
- *int* – signed integer whose maximum value is platform dependent.
- *long* – large ints are automatically converted to long
- *float* – 64-bit (double precision) floating point numbers
- *bool* – a True or False value

You could call the function *type* on an object to check if it is an int or float or string etc.
Type Conversion can be achieved by using functions like `int()`, `float()`, `str()` on objects of other types.

Basic Operators: Arithmetic using binary Operators

Just as any other programming languages, the addition, subtraction, multiplication, and division operators can be used with numbers. Most of the binary math operations and comparisons are as you might expect:

1 + 23; 5 - 7; 'This' + 'That'

Operation	Description
a + b	Add a and b
a - b	Subtract b from a
a * b	Multiply a by b
a / b	Divide a by b
a // b	Floor-divide a by b, dropping any fractional remainder
a ** b	Raise a to the b power
a & b	True if both a and b are True. For integers, take the bitwise AND.
a | b	True if either a or b is True. For integers, take the bitwise OR.
a ^ b	For booleans, True if a or b is True, but not both. For integers, take the bitwise EXCLUSIVE-OR.
a == b	True if a equals b
a != b	True if a is not equal to b
a <= b	True if a is less than (less than or equal) to b
a < b a > b	True if a is greater than (greater than or equal) to b
a >= b a is b	True if a and b reference same Python object
a is not b	True if a and b reference different Python objects

Arithmetic using Binary Operators

NOTE that Python 2.7 uses **integer division by default**, so that 5 / 2 equals 2. Almost always this is not what we want, so we have two options:

- Start your files with *from __future__ import division*
- Explicitly convert your denominator to a float as *5/float(2)*

However, if for some reason you still want integer division, use the // operator.

Arithmetic using Binary Operators

Strings

"Many people use Python for its powerful and flexible built-in string processing capabilities. You can write string literal using either single quotes or double quotes, but multiline strings are defined with triple quotes. The difference between the two is that using double quotes makes it easy to include apostrophes

```
a = 'one way of writing a string' b = "another way"
c = """This is a multiline string"""
```

Strings are

- **sequences** of characters, and so can be treated like other Python sequences (for iteration)
- **immutable**, you cannot modify them in place without creating a new string
- can contain escape characters like \n or \t
  - there's a workaround if you want backslashes in your string: prefix it with **r** (for **raw**)
- **concatenated** by the + operator, try `'This' + ' and ' + 'That'`

Here I will highlight a few cool **string methods** as a teaser to what you can do with Python

```
my_str = 'a, b, c, d, e'
my_str.replace('b', 'B')
my_str.split(',') '-'.join(my_str.split(', '))
```

Arithmetic using Binary Operators

Strings Formatting:

Python uses C-style string formatting to create new, formatted strings. The "%" operator is used to format a set of variables enclosed in a "tuple" (a fixed size list), together with a format string, which contains normal text together with "argument specifiers", special symbols like "%s" and "%d".

Let's say you have a variable called "name" with your user name in it, and you would then like to print (out a greeting to that user.)

```
# This prints out "Hello, ALabs!“
name = “ALabs“
print("Hello,%s!" % name)
```

To use two or more argument specifiers, use a tuple (parentheses):

```
# This prints out “ALabs is 4 years old.“
name = “ALabs“
age = 4
print("%s is %d years old." % (name,age))
```

Arithmetic using Binary Operators

Strings Formatting:

Any object which is not a string can be formatted using the %s operator as well. The string which returns from the "repr" method of that object is formatted as the string.

For example:

```
# This prints out: A list: [1, 2, 3]
mylist = [1,2,3]
print("A list: %s" % mylist)
```

Here are some basic argument specifiers you should know:

- %s - String (or any object with a string representation, like numbers)
- %d - Integers
- %f - Floating point numbers
- %.<number of digits>f - Floating point numbers with a fixed amount of digits to the right of the dot.
- %x/%X - Integers in hex representation (lowercase/uppercase)

Dealing with strings - Examples

In [91]:
```
string0 = 'python'
string1 = "Data Science"
string2 = '''This is Data science
        workshop
        using Python'''
```

In [92]:
```
print( string0, string1, string2)
```

```
python Data Science This is Data science
      workshop
      using Python
```

In [93]:
```
string2.find( "Python" )
```

Out[93]: 53

In [94]:
```
string0.capitalize()
```

Out[94]: 'Python'

In [95]:
```
string0.upper()
```

Out[95]: 'PYTHON'

In [96]:
```
len( string2 )
```

Out[96]: 59

In [97]:
```
string2.split()
```

Out[97]: ['This', 'is', 'Data', 'science', 'workshop', 'using', 'Python']

In [98]:
```
string2.replace( 'Python', 'R')
```

Out[98]: 'This is Data science \n   workshop\n   using R'

Control Flow with if, elif, else

Python uses boolean variables to evaluate conditions. The boolean values True and False are returned when an expression is compared or evaluated.
"The if statement is one of the most well-known control flow statement types. It checks a condition which, if True, evaluates the code in the block that follows:

```
if x < 0:
    print 'It's negative'
```

An if statement can be optionally followed by one or more elif blocks and a catch-all else block if all of the conditions are False:

```
if x < 0:
    print 'It's negative' elif x == 0:
    print 'Equal to zero' elif 0 < x < 5:
    print 'Positive but smaller than 5' else:
    print 'Positive and larger than or equal to 5'
```

If any of the conditions is True, no further elif or else blocks will be reached.

```
In [28]: x = 10
         y = 12
         if x > y:
           print ("x>y")
         elif x < y:
           print ("x<y")
         else:
           print ("x=y")

         x<y
```

Compound Logic

We can write **compound logic** using boolean operators like **and, or**. Remember that conditions are evaluated left-to-right and will short circuit,i.e, if a True is found in an **or** statement, the remainingones will not be tested.

```
if 5 < 10 or 8 > 9:
     print 'The second condition was ignored.'
```

Youc an also write a **ternary if-then-else** on one line, which sometimes helps keep thingsconcise, These statements called as inline statements

```
parity = "even" if x % 2 == 0 else "odd"
```