Introduction to Python Programming

1. Introduction
   * 1. Python characteristics
     2. Anaconda
     3. IDE
     4. Python Environment Setup
2. Python Basic Data Structure
   * 1. Number
     2. Variables
     3. Strings
     4. Lists
     5. Dictionnaries
     6. Tuples
     7. Set
3. Python Comparison Operators
4. Python statements
   1. if, elif, and else Statements
   2. for loops
   3. while loops
   4. Useful operator
   5. List-dic comprehension
5. Methods and functions
   1. Methods
   2. Functions
   3. Nested Statements and Scope
   4. args and kwargs
   5. Lambda, Map, Filter, Reduce, Zip, all, any
6. Error and Exception handling

Slide \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Python was started as a hobby project in 1991 by Guido Van Rossum. At the begin, the purpose was to define a programming language for people whose day job has nothing to do with software development, but code mainly to handle data (data scientists). So far, the python user should concentrate on the problem rather than the language. The language has evolved enormously with the contribution of many developers/programmers. If you are new to programming, it is recommended to start with Python. Python interpreters are available on several operating systems such as Linux, macOS, and Windows.

Python has the following characteristics:

* Extensive libraries (i.e. reg. expressions, doc generation, CGI, ftp, web browsers, ZIP, WAV, cryptography, etc...) (wxPython, Twisted, Python Imaging library)
* Platform-independent: you can just copy over your code to another system, and it will auto-magically work (with python platform)
* Dynamic typing: In static-typed languages like C++, you have to declare the variable type and any discrepancy like adding a string and an integer is checked during compile time. In strongly typed languages like Python, it is the job of the interpreter to check the validity of the variable types and operations performed. Dynamic typing provides a lot of freedom, but simultaneously it makes your code risky and sometimes difficult to debug
* Garbage Collector: python will automatically free up space without you doing anything.
* Free & Open source: maintained by the community
* Simple: It is a minimalistic language in nature and reading a good python program should be like reading English (Pythonic philosophy )
* Portable. Python runs in where any c compiler can run
* Object-Oriented and Support procedural programming
* Extensible – in the sense that you can easily import other code
* Embeddable –you can easily place your code in non-python programs
* Extensive libraries
* Easy to Learn

Slide \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The terms interpreted or compiled is not a property of the language but a property of the implementation. In the case of compilation, the human understandable code is translated to machine understandable code. Machine code is the base level form of instructions that can be directly executed by the CPU. On the other hand, the interpretation executes a program by reading the source code a line at a time and doing what it says (shell like).

Python is an interpreted language and not a compiled one, although compilation is a step. Python code, written in .py file is first compiled to what is called bytecode which is stored with a .pyc or .pyo format.

So, an important aspect of Python’s compilation to bytecode is that it’s entirely implicit. You never invoke a compiler, you simply run a .py file. The Python implementation compiles the files as needed. This is different than Java, for example, where you have to run the Java compiler to turn Java source code into compiled class files. For this reason, Java is often called a compiled language, while Python is called an interpreted language. But both compile to bytecode, and then both execute the bytecode with a software implementation of a virtual machine.

Slide \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

There are multiple implementations of **Python language**. The official one is a byte code interpreted one. There are byte code JIT compiled implementations too. On-the-fly compilation, known as "Just In Time", is a mechanism that allows a virtual machine to optimize performance by compiling parts of the program instead of interpreting them.

Python source code (.py) can be compiled to different byte code also like **IronPython** (.Net) or Jython (JVM). The source code can also be JIT compiled using PyPy or Numba that translates a subset of Python and NumPy code into fast machine code.

It's worth it at this point to read up on how tracing JIT compilers work. Here's a brief explanation: The interpreter is usually running your interpreter code as written. When it detects a loop of code in the target language is executed often, that loop is considered "hot" and marked to be traced. The next time that loop is entered, the interpreter gets put in tracing mode where every executed instruction is logged. When the loop is finished, tracing stops. The trace of the loop is sent to an optimizer, and then to an assembler which outputs machine code. That machine code is then used for subsequent loop iterations.

This machine code is often optimized for the most common case and depends on several assumptions about the code. Therefore, the machine code will contain guards, to validate those assumptions. If a guard check fails, the runtime falls back to regular interpreted mode.