INTRODUCTION TO PYTHONNET

What is .NET?

INTRODUCTION:

In this tutorial, we are going begin our exploration of the Python for .NET (`pythonnet`) package, a powerful tool that allows us to access the CLR from Python. To fully grasp this topic, we will have to dedicate a substantial amount of time giving you background on .Net, it's history, how it's organized, and how it works.

Some of the ideas presented in this tutorial, might seem difficult to comprehend at first, but I assure as time goes on this won't be the case. To make the process of comprehending the ideas easier, I recommend you keep a few questions in the back of your head as you read through the material. For me, this made it easier to understand the "why" behind .Net. Additionally, it helped give purpose and context as to why certain tools and frameworks were developed and how they are designed to work.

When reading this tutorial, imagine you are developer who has been tasked with creating a new tool that is supposed to make the job of software development easier. In this role, ask yourself the following questions:

- How can we make the development process easier?
- How can we standardize the development process while still maintaining flexibility?
- How can we make tools language independent?
- How would the tool need to operate so that it can make the development process consistent?

With those questions in the back of your head, let's dive in begin to understand .NET!

ADDITIONAL NOTES:

Throughout this process, at least in the beginning, I'll usually phrase a question, provide a definition and possibly elaborate on the definition if I feel it doesn't provide the right detail. .NET was developed by Microsoft, most of the definitions I provide come straight from their website. I'll be providing links to each of the definition in the tutorial.

Let's start with the most basic question, **"What is .Net?".**

WHAT IS .NET?

.NET is an open source developer platform, created by Microsoft, for building many different types of applications. With .NET, you can use multiple languages, editors, and libraries to build for web, mobile, desktop, gaming, and IoT.

Additional Notes: It's open source, meaning anyone can use it for free and it's used for a wide variety of task. Additionally, .Net is the highest level of the of pyramid and the following items listed below all fall under .Net.

Link:
https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet

Now, the definition up above is a little misleading. It gives the impression that there is only one implementation of .NET, the reality is there is more than one implementation. The next question, naturally, is, **"Are there multiple versions of .NET?"**

ARE THERE MULTIPLE VERSIONS OF .NET?

There are various implementations of .NET. Each implementation allows .NET code to execute in different places—Linux, macOS, Windows, iOS, Android, and many more.

- .NET Framework is the original implementation of .NET. It supports running websites, services, desktop apps, and more on Windows.
- .NET Core is a cross-platform implementation for running websites, services, and console apps on Windows, Linux, and macOS. .NET Core is open source on GitHub.
- Xamarin/Mono is a .NET implementation for running apps on all the major mobile operating systems, including iOS and Android.
- .NET Standard is a formal specification of the APIs that are common across .NET implementations. This allows the same code and libraries to run on different implementations.

ADDITIONAL NOTES:

For .Net to be executed across multiple platforms, it must have multiple implementations. An implementation may just differ by the tools offered to it. For example, Xamarin/Mono provides extra tools and libraries for building and running on mobile apps.

To standardize all the different implementations of .NET they created a standard called ".NET Standard". With this standard, there are defined set of rules that allow for all the code and libraries to run the same across all the different implementations. This provides flexibility, different implementations, but consistency, a standardized API across the different implementations.

Link:
https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet

Normally, you must ask yourself a few questions when determining which implementation of .NET, you need to use. I'll list some of those questions down below, but till then because we will want to use Windows Specific APIs from pythonnet we will be using the .NET Framework implementation of .NET. This leads to our next question, **"What is .Net Framework?".**

WHAT IS .NET FRAMEWORK?

.NET Framework is a software development framework for building and running applications on Windows. .NET Framework is part of the .NET platform.

ADDITIONAL NOTES:

.Net Framework, is the framework that provides the template for you to build the software. By providing you libraries, which contain tools that help you do common things, you can dramatically reduce development time. .Net Framework simply bundles all these tools in the same package. It's also important to note that .NET Framework is ONLY AVAILABLE ON WINDOWS.

Link:
https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework

Now that we understand what the .NET Framework is we can begin to understand it's different components. These components provide services, like managing your app to executing your code. Our next question is **"What are the major components of .NET Framework?"**.

WHAT ARE THE MAJOR COMPONENTS OF THE .NET FRAMEWORK?

The two major components of .NET Framework are the Common Language Runtime and the .NET Framework Class Library.

The Common Language Runtime (CLR) is the execution engine that handles running applications. It provides services like thread management, garbage collection, type-safety, exception handling, and more.

The Class Library provides a set of APIs and types for common functionality. It provides types for strings, dates, numbers, etc. The Class Library includes APIs for reading and writing files, connecting to databases, drawing, and more.

ADDITIONAL NOTES:
The CLR has a lot more to it then just itself, it's part of a much bigger infrastructure that we will cover next. At this point you just need to know that the CLR is just the engine that runs the app and provides the services. The Class Library is your set of prebuilt tools you have at your disposal. This is an extensive library that contains numerous objects, and we will be using it heavily in pythonnet.

Link:
https://dotnet.microsoft.com/learn/dotnet/what-is-dotnet-framework

Okay at this point we should have a good understand of .NET, it's implementations, and its components. Let's start talking about the components. You read up above, that .NET Framework leverages the CLR to run your apps and provide services. Naturally you might be asking, "How it does this?". Well there are a few topics you need to understand to answer the question.

The CLR is part of the CLI (Common Language Infrastructure), this infrastructure provides all the specifications necessary for .NET to work. The simple question is, **"What is the CLI?"**

WHAT IS THE COMMON LANGUAGE INFASTRUCTURE (CLI)?

Common Language Infrastructure is a standard developed by Microsoft that allows the use of multiple different high-level languages on different computer platforms without rewriting the code for any specific architecture.

CLI consists of 4 features:

1. Common Type System (CTS)

It defines the programming types and operations supported by the .NET runtime engine. It defines rules that languages must follow, which help ensure that objects written in different languages can interact with each other. It defines the rules that ensure that the data types of objects written in various languages can interact with each other.

2. Metadata

It describes all classes and class members that are defined in the assembly and the classes and class members that the current assembly will call from another assembly. A CLI Language compiler will generate the metadata and store this in the assembly containing the CIL. When the run-time executes CLI, it will check to make sure that the metadata of the called method is the same as the metadata that is stored in the calling method. This ensures that a method can only be called with exactly the right number of parameters and exactly the right parameter types. Metadata can be added to the code through attributes.

3. Common Language Specification (CLS)

For programming languages to communicate effectively, targeting CLI is not enough. There must be a common set of standards to which every .NET language must adhere. This common set of language features is called the Common Language Specification (CLS).

4. Virtual Execution System (VES)

The VES is responsible for loading and executing programs that are written for the .NET runtime. The purpose of the VES is to provide support that is required to execute the Common Intermediate Language (CIL) instruction set.

All compatible languages compile to Common Intermediate Language (CIL), which is an intermediate language. When the code is executed, the platform-specific VES will compile the CIL to the machine language according to the specific hardware and operating system.

ADDITIONAL NOTES:

CLI is the standard, we know that standards provide consistency by defining rules that everyone must adhere to. The rules apply to a programming language that wants to be CLI compliant, if you don't follow the rules you aren't compliant and therefore can't be used in CLI. The four rules are as follow:
Follow a Common Type System. A common type system provides a way to make objects language neutral. This means we can use them in different languages if they follow CTS.

Provide Metadata:
This helps describe your objects, and will make sure that activities, like calling a method, will only work if the right number of parameters are passed through.

Follow the CLS:
This way you can effectively communicate with the CLI
Be able to be executed by the VES.

If you don't compile to CIL you can't be run by the VES. If you can't compile to CIL then you're not following the rules up above.

Link:
https://www.c-sharpcorner.com/blogs/overview-of-common-language-infrastructure

You noticed at the end of the definition that code you write targeting the CLI is compiled into something called Common Intermediate Language (CIL). With CIL we can allow for language independence, because code we write in one language, say C#, can be compiled to CIL and then run on the CLR. Let's ask the question, **"What is CIL?"**

WHAT IS THE COMMON INTERMEDIATE LANGUAGE?

What is "Intermediate Language" (or IL for short)? It is a product of compilation of code written in high-level .NET languages. Once you compile your code written in one of these languages, you will get a binary that is made from IL. It is important to note that the IL is independent from any specific language that runs on top of the runtime; there is even a separate specification for it that you can read if you're so inclined.

Once you produce IL from your high-level code, you will most likely want to run it. This is where the CLR takes over and starts the process of Just-In-Time compiling, or JIT-ing your code from IL to machine code that can be run on a CPU. In this way, the CLR knows exactly what your code is doing and can effectively manage it.

Intermediate Language is sometimes also called Common Intermediate Language (CIL) or Microsoft Intermediate Language (MSIL).

ADDITIONAL NOTES:
This was a simple definition, the CIL is just the result of taking high-level .NET languages, and compiling it with CLI. Once you have CIL you can run it using the CLR. It's important to note that the compiled code is stored in assemblies (.dll or .exe files). The CLR takes these files, uses the JIT compiler to turn it into machine code, and then run it on the specific architecture.

Link:
https://docs.microsoft.com/en-us/dotnet/standard/managed-code

Alright, let's cover the final portion, **"What is the CLR?".**

WHAT IS THE CLR?

The .NET Framework provides a run-time environment called the common language runtime, which runs the code and provides services that make the development process easier.

Compilers and tools expose the common language runtime's functionality and enable you to write code that benefits from this managed execution environment. Code that you develop with a language compiler that targets the runtime is called managed code; it benefits from features such as cross-language integration, cross-language exception handling, enhanced security, versioning and deployment support, a simplified model for component interaction, and debugging and profiling services.

ADDITIONAL NOTES:
The CLR is the engine in the car that makes it go. Pythonnet integrates with the CLR so we can run our Python code in the .NET Framework. Additionally, the CLR provides many services behind the scenes so that way we can expedite the development process.

Link:
https://docs.microsoft.com/en-us/dotnet/framework/get-started/index#Introducing

Now, you may have forgotten this already, but I'll make sure to remind you. We know that two major components of .NET framework our CLR and the Class Library. We've covered the CLR so let's move on to Class Libraries.

WHAT ARE CLASS LIBRARIES?

.NET implementations include classes, interfaces, delegates, and value types that expedite and optimize the development process and provide access to system functionality. To facilitate interoperability between languages, most .NET types are CLS-compliant and can therefore be used from any programming language whose compiler conforms to the common language specification (CLS).

.NET types are the foundation on which .NET applications, components, and controls are built. .NET implementations include types that perform the following functions:

- Represent base data types and exceptions.
- Encapsulate data structures.
- Perform I/O.
- Access information about loaded types.
- Invoke .NET Framework security checks.
- Provide data access, rich client-side GUI, and server-controlled, client-side GUI.

.NET provides a rich set of interfaces, as well as abstract and concrete (non-abstract) classes. You can use the concrete classes as is or, in many cases, derive your own classes from them. To use the functionality of an interface, you can either create a class that implements the interface or derive a class from one of the .NET classes that implements the interface.

ADDITIONAL NOTES:
The Class Library provides the tools you need to do for everyday task, like reading files or declaring data types. These Class libraries are available across all .NET implementations. It's hard to tell from the definition up above, but there are three categories of Class Libraries. In the definition above we covered the .NET Standard Class Libraries.

Link:
https://docs.microsoft.com/en-us/dotnet/standard/class-libraries

The other two class libraries are platform specific and portable Class Libraries. Let's walk through them.

WHAT IS PLATFORM SPECIFIC LIBRARIES?

Platform-specific libraries are bound to a single .NET implementation (for example, .NET Framework on Windows) and can therefore take significant dependencies on a known execution environment. Such an environment will expose a known set of APIs (.NET and OS APIs) and will maintain and expose expected state (for example, Windows registry).

Developers who create platform specific libraries can fully exploit the underlying platform. The libraries will only ever run on that given platform, making platform checks or other forms of conditional code unnecessary (modulo single sourcing code for multiple platforms).

Platform-specific libraries have been the primary class library type for the .NET Framework. Even as other .NET implementations emerged, platform-specific libraries remained the dominant library type.

ADDITIONAL NOTES:
Because we will be using the .NET Framework heavily in Pythonnet we will be using a lot of Window specific APIs like System.Data and System.Web.

Link:
https://docs.microsoft.com/en-us/dotnet/standard/class-libraries

WHAT ARE PORTABLE CLASS LIBRARIES?

Portable libraries are supported on multiple .NET implementations. They can still take dependencies on a known execution environment; however, the environment is a synthetic one that is generated by the intersection of a set of concrete .NET implementations. This means that exposed APIs and platform assumptions are a subset of what would be available to a platform-specific library.

You choose a platform configuration when you create a portable library. These are the set of platforms that you need to support (for example, .NET Framework 4.5+, Windows Phone 8.0+). The more platforms you opt to support, the fewer APIs and fewer platform assumptions you can make, the lowest common denominator. This characteristic can be confusing at first, since people often think "more is better", but find that more supported platforms results in fewer available APIs.

Many library developers have switched from producing multiple platform-specific libraries from one source (using conditional compilation directives) to portable libraries. There are several approaches for accessing platform-specific functionality within portable libraries, with bait-and-switch being the most widely accepted technique at this point.

ADDITIONAL NOTES:
I call these libraries the in-between of Platform-Specific and .NET Standard. With Portable Class Libraries you can use them across implementations/platforms, but you get a subset of the functionality you would get if you had chosen to use a Platform Specific library.

Link:
https://docs.microsoft.com/en-us/dotnet/standard/class-libraries