Python crash course

Introduction

Python's Philosophy: simplicity, readability, community-driven development
Comparison with C#: compiled vs. interpreted, static vs. dynamic typing

Execution Model

C# - Compiled Language:

- **Definition**: C# is a compiled language, typically compiled into Intermediate Language (IL) code that runs on the .NET runtime (Common Language Runtime, CLR). The compilation step occurs before the code is executed.
- **Process**:
    - **Source Code**: Written in C#.
    - **Compilation**: Source code is compiled into IL by the C# compiler (`csc`).
    - **Execution**: The IL is then JIT (Just-In-Time) compiled into native code by the CLR and executed.
- **Advantages**:
    - **Performance**: Compiled code often runs faster because it is optimized for the target machine.
    - **Error Detection**: Many errors are caught during the compilation process, reducing runtime issues.
    - **Security**: Compiled binaries are harder to reverse-engineer than interpreted code.

Python - Interpreted Language:

- **Definition**: Python is an interpreted language, meaning the source code is executed line by line by the Python interpreter. While Python does compile code to bytecode (`.pyc` files), it is done at runtime and not ahead of time.
- **Process**:
    - **Source Code**: Written in Python.
    - **Interpretation**: The Python interpreter reads and executes the source code directly or compiles it to bytecode and then executes it.
- **Advantages**:
    - **Portability**: Python code can run on any system with a compatible interpreter without needing recompilation.
    - **Rapid Development**: The absence of a separate compilation step speeds up development and testing cycles.
    - **Ease of Debugging**: Since Python executes line by line, it’s easier to debug small sections of code in isolation.

Trade-offs:

- **C#**: The compilation step can slow down the edit-compile-test cycle, particularly in large projects.
- **Python**: Interpreted execution generally makes Python slower than C#, especially for CPU-bound tasks.

Typing

C# - Static Typing:

- **Definition**: In C#, the type of a variable is explicitly declared and checked at compile-time. This means that once a variable's type is declared, it cannot hold values of other types without explicit conversion.
- **Example**:
```
int number = 10;
string text = "Hello";
// number = text; // This would cause a compile-time error
```
- **Advantages**:
    - **Type Safety**: Prevents type-related errors at compile-time, leading to fewer runtime errors.
    - **Performance**: Static typing allows for optimizations by the compiler, resulting in generally faster execution.
    - **IDE Support**: Better autocompletion, refactoring tools, and error detection before running the program.

Python - Dynamic Typing:

- **Definition**: Python uses dynamic typing, where the type of a variable is determined at runtime based on the value assigned to it. Variables can change types during execution without explicit casting.
- **Example**:
```
number = 10
text = "Hello"
number = text  # No error; 'number' now holds a string
```
- **Advantages**:
    - **Flexibility**: Easier to write quick, concise code without needing to declare types.
    - **Conciseness**: Less boilerplate code, as there is no need for type declarations.
    - **Ease of Use**: Great for rapid development and prototyping, where the exact type may not be known upfront.

Trade-offs:

- **C#**: While type safety reduces errors, it can make the code more verbose and require more upfront planning.

- **Python**: The flexibility can sometimes lead to runtime errors that are only discovered during execution, which could have been caught at compile-time in a statically typed language.

Note:

Both C# and Python are strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them.

Development Environment and Ecosystem

C#:

- **Integrated Development Environment (IDE)**:
  - **Visual Studio**: The primary IDE for C# development, offering advanced features like IntelliSense, debugging, profiling, and more.
  - **Cross-Platform Development**: With .NET Core/.NET 5+, C# has gained significant cross-platform capabilities, but traditionally it was tied to Windows.
- **Build and Deployment**:
  - **MSBuild**: Used for compiling and building projects, supporting large-scale enterprise applications.
  - **NuGet**: Package management system that integrates tightly with Visual Studio for managing dependencies.

Python:

- **Integrated Development Environment (IDE)**:
  - **VS Code/PyCharm**: Popular IDEs/editors for Python, with plugins for linting, debugging, and version control.
  - **Cross-Platform**: Python is inherently cross-platform and can run on any system with the appropriate interpreter.
- **Build and Deployment**:
  - **Pip**: Python's package manager for installing and managing libraries.
  - **Virtual Environments**: Tools like `venv` and `virtualenv` are commonly used to manage project-specific dependencies.
  - **Deployment**: Python applications are often deployed using simple scripts, Docker containers, or cloud services like AWS Lambda, which cater well to Python's strengths in scripting and automation.

Trade-offs:

- **C#**: The .NET ecosystem is very powerful for enterprise-grade applications, but it can have a steeper learning curve and more overhead in setup compared to Python.
- **Python**: The ecosystem is more lightweight and flexible, ideal for scripting, automation, and rapid development, though it may lack some of the advanced tooling available in C#.

Use Cases

- **C#**:
    - **Enterprise Applications**: C# is widely used in large-scale enterprise applications, desktop applications (especially with Windows Forms or WPF), and game development (using Unity).
    - **Performance and scalability**: Ideal when type safety, performance, and scalability are critical.
    - The .NET ecosystem is very powerful for enterprise-grade applications, but it can have a steeper learning curve.
- **Python**:
    - **Scripting and Automation**: Python excels in automation, scripting, data analysis, and AI/ML tasks.
    - **Ease of Learning and Use**: Often chosen for rapid prototyping and development, scientific computing, web development (with frameworks like Django and Flask) or educational purposes due to its simplicity.
    - The ecosystem is more lightweight and flexible, though it may lack some of the advanced tooling available in C#.

Installation

python, virtual env / anaconda, VS Code or another editor
work with packages

useful commands:

cmd:

```
conda create -n condaEnvName python=3.12 --no-default-packages
conda activate condaEnvName
conda install pip
pip freeze > requirements.txt
pip install -r requirements.txt
```

VS Code:

```
ctrl+shift+P to open VS Code's command palette
-> Python: Select Interpreter
```

Basics

Few simple samples of a python code.

Python Basics

- **Syntax**: Indentation, significant whitespace
- **Data Types and Variables**:

    - Primitive data types: `int, float, str, bool`
    - Mutable vs. immutable types: `int, float, str, bool,tuple` vs. `list, dict, set`
    - Implicit, explicit type casting
- **Operators**: Arithmetic, logical, comparison, assignment operators.

Control Structures

- **Conditional Statements**: `if`, `elif`, `else` (`if`, `else if`, `else` in C#)
- **Loops**:
    - `for` and `while` loops
    - `break`, `continue`, `pass` (`break`, `continue` in C#)
- **Comprehensions**: List comprehensions (LINQ in C#)

Functions

- **Defining Functions**: `def` keyword, function signatures, default arguments
- **Lambdas**: Anonymous functions (similar to delegates or lambda expressions in C#)
- **Scope and Namespaces**: Global vs. local scope, `global` and `nonlocal` keywords

Object-Oriented Programming (OOP)

- **Classes and Objects**:
    - Defining classes with `class, __init__` method (similar to constructors in C#), instance vs. class variables
- **Inheritance and Polymorphism**:
    - Single and multiple inheritance
- **Special Methods**:
    - `__str__`, `__repr__`, `__eq__`, etc.
- **method overloading and overriding** (differences compared to C#)

Error Handling

- **Exceptions**:
    - `try`, `except`, `else`, `finally`, creating custom exceptions (similar to `try-catch` in C#)

File I/O

- **Reading and Writing Files**: `open()`, `read()`, `write()`, `with` statement for context management.
- **Handling JSON and CSV**: `json` and `csv` modules.

Modules and Packages

- **Importing Modules**: `import`, `from ... import`, aliasing `import ... as ...`
- **Creating Modules**: Organizing code into reusable modules (vs. C# namespaces)
- **Key Libraries**:
    - `numpy`, `pandas` for data manipulation
    - `matplotlib` for plotting
    - `pyodbc` for databases connection (using ODBC drivers) and SQL queries
    - `requests` for HTTP requests
    - `typing` for type hints and annotations, enabling static type checking
    - `abc` for abstract classes, enforcing method implementation in subclasses, supporting object-oriented design principles

Advanced Topics

- **Decorators**: An equivalent to attributes in C# but more powerful
- **Metaclasses**: Customizing class creation
- **Concurrency**: `asyncio`, threading, and multiprocessing (vs. async/await, threading in C#)

Best Practices

- **Code Style**: PEP 8 guidelines
- **Testing**: Writing unit tests with `pytest` (or `unittest`)
- **Official Python Documentation**: https://docs.python.org/3/

Acceptance code

Introduce and show acceptance tests, show other python vs C# differences, try a common addition of a new operation and change of the barcode handover from NotEvaluated to NotEvaluatedDetails

Python does not support method overloading

```
class SqlOverview(SqlReadOperation):
    def analyze_sql_output(self, sql_output: List[dict]) ->
ValidationResult:
        return self.analyze_sql_output(self, sql_output,
fail_if_output_not_empty: False)

    def analyze_sql_output(self, sql_output: List[dict],
fail_if_output_not_empty: bool) -> ValidationResult:
        #the code...
```

Questions and feedback

acceptance code readability, complexity, understanding ?

- refactoring needed
- some brainstorming about readability, sustainability (maintenance) ?