Assignment PR002

1. Compare **UNIX philosophy** and with the **Zen of Python**. List down those principles which you think talk about the same aspect of software development. And list those, which you think conflict one another.

   The UNIX philosophy is:
   - Bottom-up not top-down
   - Pragmatic and grounded in experience
   - Not a formal design method
   - An implicit half-reflexive knowledge
   - Encourages sense of proportion and skepticism

   The rules:
   - Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new features.
   - Expect the output of every program to become the input to another, as yet unknown, program. Don't clutter output with extraneous information. Avoid strictly columnar or binary input formats. Don't insist on interactive input.
   - Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them.
   - Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you've finished using them.
   - Rule of Modularity: Write simple parts connected by clean interfaces.
   - Rule of Clarity: Clarity is better than cleverness.
   - Rule of Composition: Design programs to be connected to other programs.

- Rule of Separation: Separate policy from mechanism; separate interfaces from engines.
- Rule of Simplicity: Design for simplicity; add complexity only where you must.
- Rule of Parsimony: Write a big program only when it is clear by demonstration that nothing else will do.
- Rule of Transparency: Design for visibility to make inspection and debugging easier.
- Rule of Robustness: Robustness is the child of transparency and simplicity.
- Rule of Representation: Fold knowledge into data so program logic can be stupid and robust.
- Rule of Least Surprise: In interface design, always do the least surprising thing.
- Rule of Silence: When a program has nothing surprising to say, it should say nothing.
- Rule of Repair: When you must fail, fail noisily and as soon as possible.
- Rule of Economy: Programmer time is expensive; conserve it in preference to machine time.
- Rule of Generation: Avoid hand-hacking; write programs to write programs when you can.
- Rule of Optimization: Prototype before polishing. Get it working before you optimize it.
- Rule of Diversity: Distrust all claims for “one true way”.
- Rule of Extensibility: Design for the future, because it will be here sooner than you think.

The Zen of Python

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one-- and preferably only one --obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!

COMMONALITIES:

1. **Simple better than complex:** Both the philosophies talk about building simple code, and not complicating it by adding on extra features
2. **Beautiful is better than ugly:** Rule of clarity, modularity and simplicity is similar to this rule
3. **Errors should never pass silently:** Rule of failure is similar to this
4. **Sparse is better than dense:** Rule of separation

DIFFERENCES:

1. There should be one, and preferably only one obvious way to do it **v/s** Rule of Diversity.
2. Design and build software, even operating systems, to be tried early, ideally within weeks. Don't hesitate to throw away the clumsy parts and rebuild them **v/s** Now is better than never; although never is often better than *right* now.