Procedural Vs Functional Programming

The main difference lies in their **philosophy regarding state and side effects.**

Procedural Programming (PP):

- **Focus:** How to achieve a result through a sequence of step-by-step instructions (procedures or functions) that **modify the program's state**.
- **Data:** Data is often separate from the code that operates on it. Functions frequently read from and write to **shared, mutable data structures** (like global variables).
- **State:** Emphasizes changing the program's state over time. The order of operations is crucial because each step can alter data that subsequent steps rely on.
- **Analogy:** A detailed cooking recipe where you have a list of steps to modify the ingredients in a single bowl on the counter. Each step changes the contents of that bowl directly.

Functional Programming (FP):

- **Focus:** What to compute by applying and composing **pure functions** that avoid changing state and mutable data.
- **Data:** Data is typically immutable. Instead of modifying existing data, functions take data as input and always **return new data** as output.
- **State:** Actively avoids shared, mutable state and side effects. Each function is self-contained and predictable.
- **Analogy:** A series of specialized kitchen gadgets where each gadget takes an ingredient, performs an action, and *produces a new, transformed ingredient*, leaving the original ingredient untouched. You pass the output of one gadget to the input of the next.

Table summarizing the key distinctions:

Feature	Procedural Programming (PP)	Functional Programming (FP)
Primary Focus	Sequence of operations; "how" to do it.	Evaluation of functions; "what" to compute.
Data Handling	Data and functions are separate.	Data and functions are closely related (functions act on data).
State	**Mutable state** is central; state changes are common.	**Immutable state** is preferred; state changes are avoided.
Side Effects	Functions often produce **side effects** (modify external state).	Functions are ideally **pure functions** (no side effects).
Function Use	Functions are sequences of commands.	Functions are first-class citizens (can be passed, returned).
Predictability	Can be harder to predict due to shared mutable state.	Highly predictable due to pure functions and immutability.
Concurrency	More challenging for concurrent execution (due to shared mutable state).	Easier for concurrent execution (no shared mutable state).
Debugging	Can be harder due to interdependencies and state changes.	Easier to debug due to isolated, predictable functions.