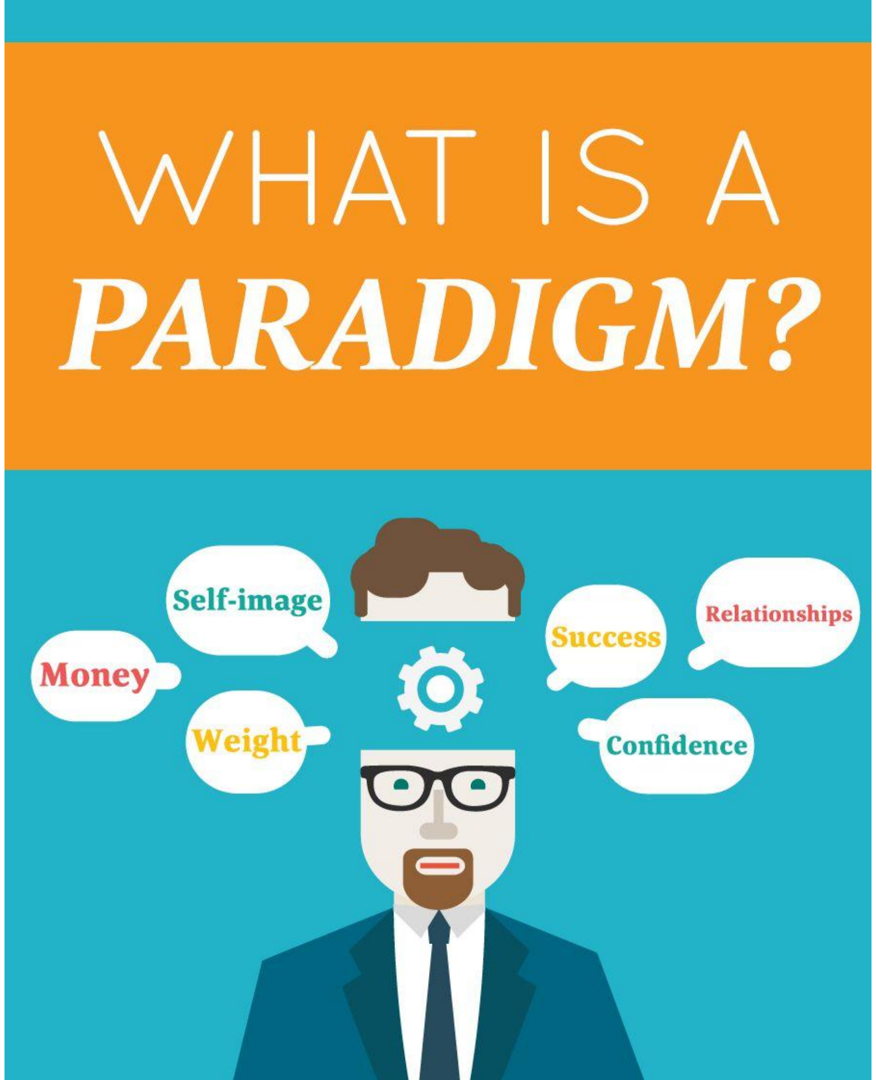
Programming Paradigms & Functional Programming

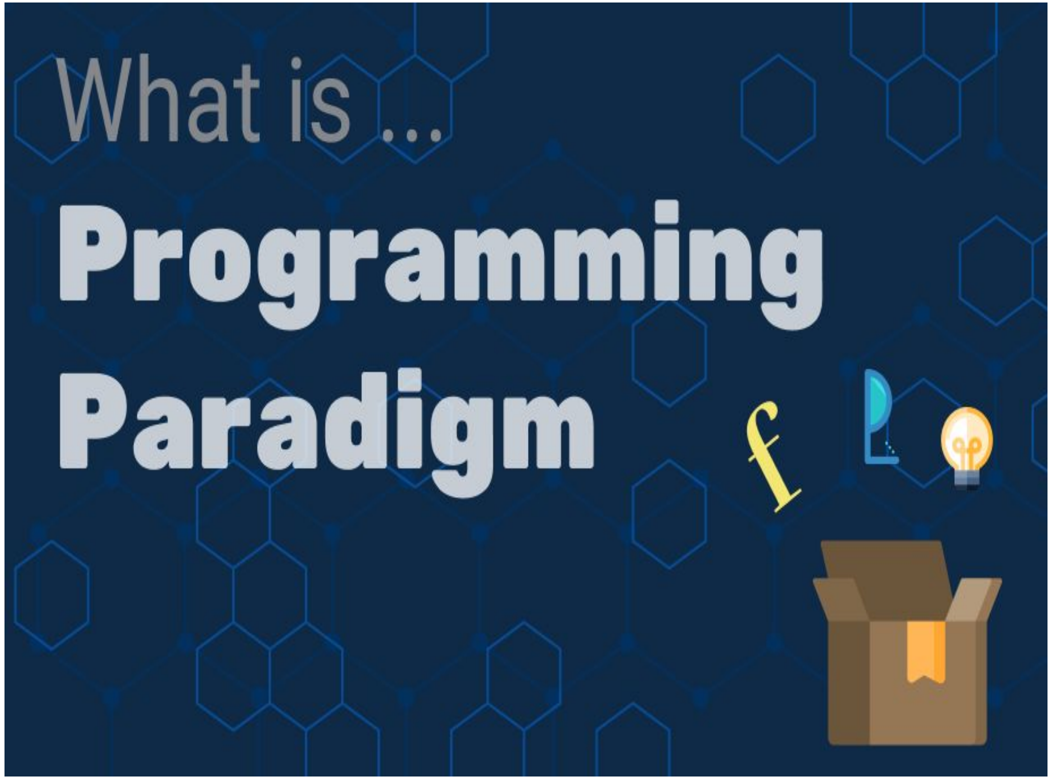
Ahmad Khalili

Omar Badawi

In the general sense, a paradigm is a set of concepts and assumptions that represent a view for a certain subject. In a way, it’s a framework that helps build a structure for a certain problem or set of problems



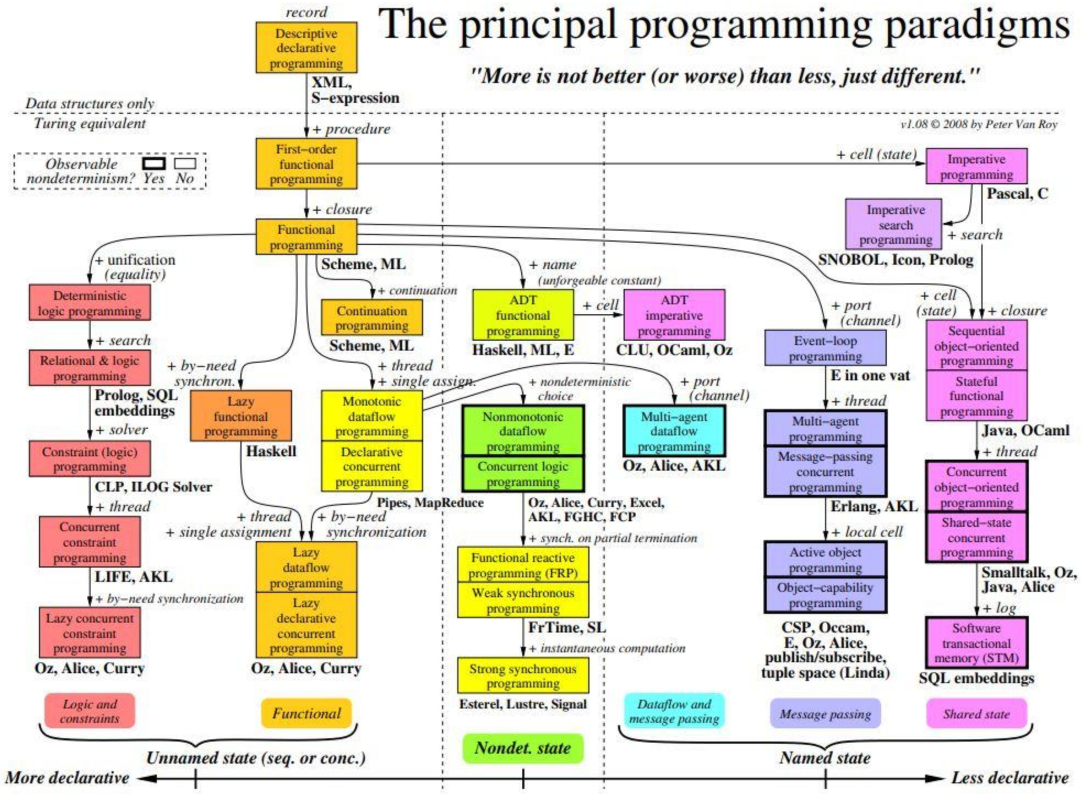
Programming paradigms, apart from also being paradigms, are classifications used to categorize programming languages based on the programming languages' paradigms' nature, such as model of execution, or code organization



Since we mostly can solve each problem with different paradigms and languages

The variation of programming paradigms allows us to choose the most suitable way to solve our problem

Major Types of Paradigms


The principal programming paradigms
"More is not better (or worse) than less, just different."
v1.08 © 2008 by Peter Van Roy
record
Descriptive declarative programming
XML, S−expression
Data structures only
Turing equivalent
+ procedure
Observable nondeterminism? Yes No
First−order functional programming
+ cell (state)
Imperative programming
Pascal, C
+ closure
Imperative search programming
+ search
SNOBOL, Icon, Prolog
Functional programming
+ unification (equality)
Scheme, ML
+ name (unforgeable constant)
Deterministic logic programming
+ continuation
Continuation programming
Scheme, ML
ADT functional programming
+ cell
ADT imperative programming
Haskell, ML, E
CLU, OCaml, Oz
+ port (channel)
+ cell (state)
+ closure
Event−loop programming
E in one vat
Sequential object−oriented programming
Stateful functional programming
+ search
Relational & logic programming
Prolog, SQL embeddings
+ by−need synchron.
+ thread
+ single assign.
+ nondeterministic choice
+ port (channel)
Lazy functional programming
Haskell
Monotonic dataflow programming
Declarative concurrent programming
Pipes, MapReduce
Nonmonotonic dataflow programming
Concurrent logic programming
Oz, Alice, Curry, Excel, AKL, FGHC, FCP
Multi−agent dataflow programming
Oz, Alice, AKL
+ thread
Multi−agent programming
Message−passing concurrent programming
Erlang, AKL
Java, OCaml
+ thread
Concurrent object−oriented programming
Shared−state concurrent programming
Smalltalk, Oz, Java, Alice
+ solver
Constraint (logic) programming
CLP, ILOG Solver
+ thread
Concurrent constraint programming
LIFE, AKL
+ by−need synchronization
Lazy concurrent constraint programming
Oz, Alice, Curry
+ thread
+ single assignment
+ by−need synchronization
Lazy dataflow programming
Lazy declarative concurrent programming
Oz, Alice, Curry
+ synch. on partial termination
Functional reactive programming (FRP)
Weak synchronous programming
FrTime, SL
+ instantaneous computation
Strong synchronous programming
Esterel, Lustre, Signal
+ local cell
Active object programming
Object−capability programming
CSP, Occam, E, Oz, Alice, publish/subscribe, tuple space (Linda)
+ log
Software transactional memory (STM)
SQL embeddings
Logic and constraints
Functional
Nondet. state
Dataflow and message passing
Message passing
Shared state
Unnamed state (seq. or conc.)
Named state
More declarative
Less declarative

Imperative Programming

Imperative Programming is one of the oldest programming paradigms. It focuses on how a certain result is achieved by explicitly stating the process

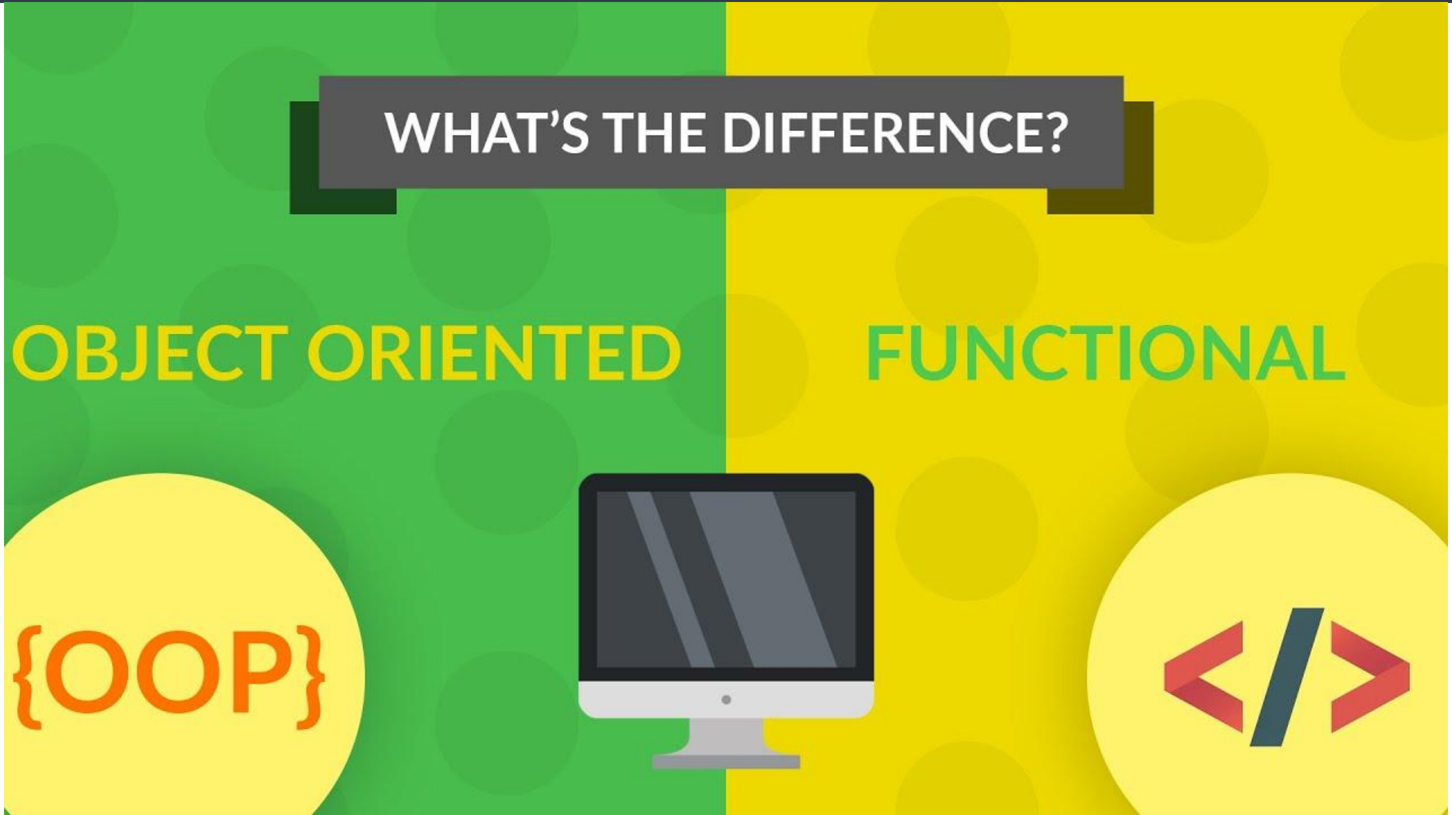
```
public static int[] DoubleArray(int[]
arr){
    for (int i = 0; i < arr.Length; i++){
        arr[i] = arr[i] * 2;
    }
    return arr;
}
```

Declarative Programming

Declarative Programming is about telling the program what you need not how to get it , it helps to make the code much more readable and it hides the code complexity

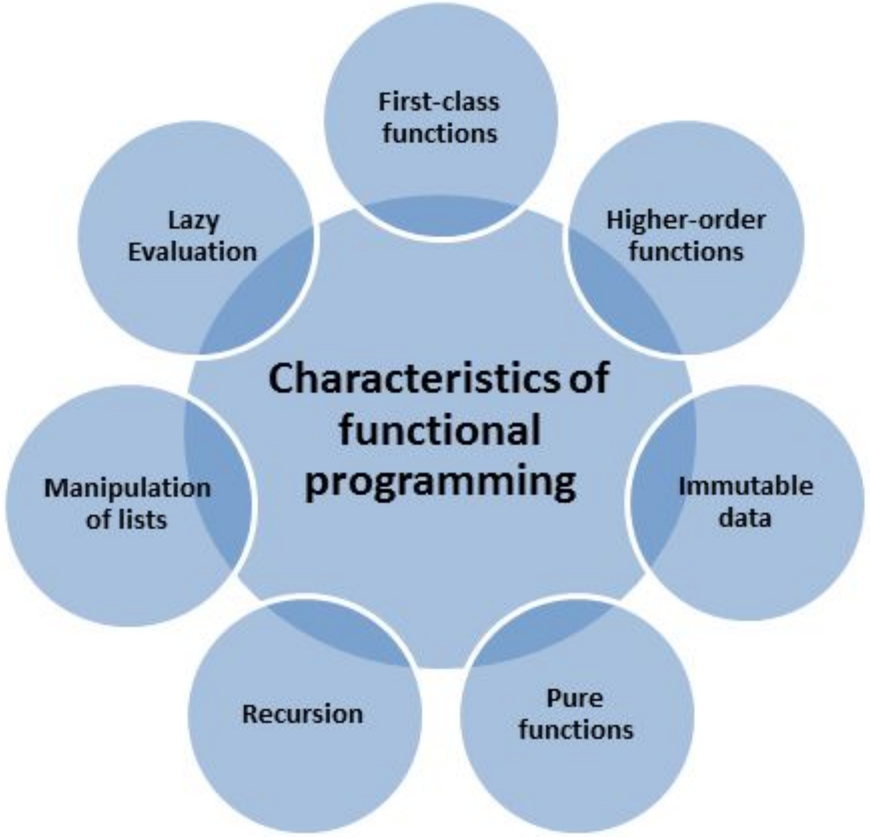
```
public static int[] DoubleArray(int[]
arr){
    Var result = arr.Select(x => x * 2 );
    return result;
}
```

Popular Programming Paradigms



Functional Programming

- ❑ It’s a Declarative Programming paradigm
- ❑ Uses Functions on data to reach the desired result
- ❑ No state or values changing !
- ❑ Full separation between data and functions

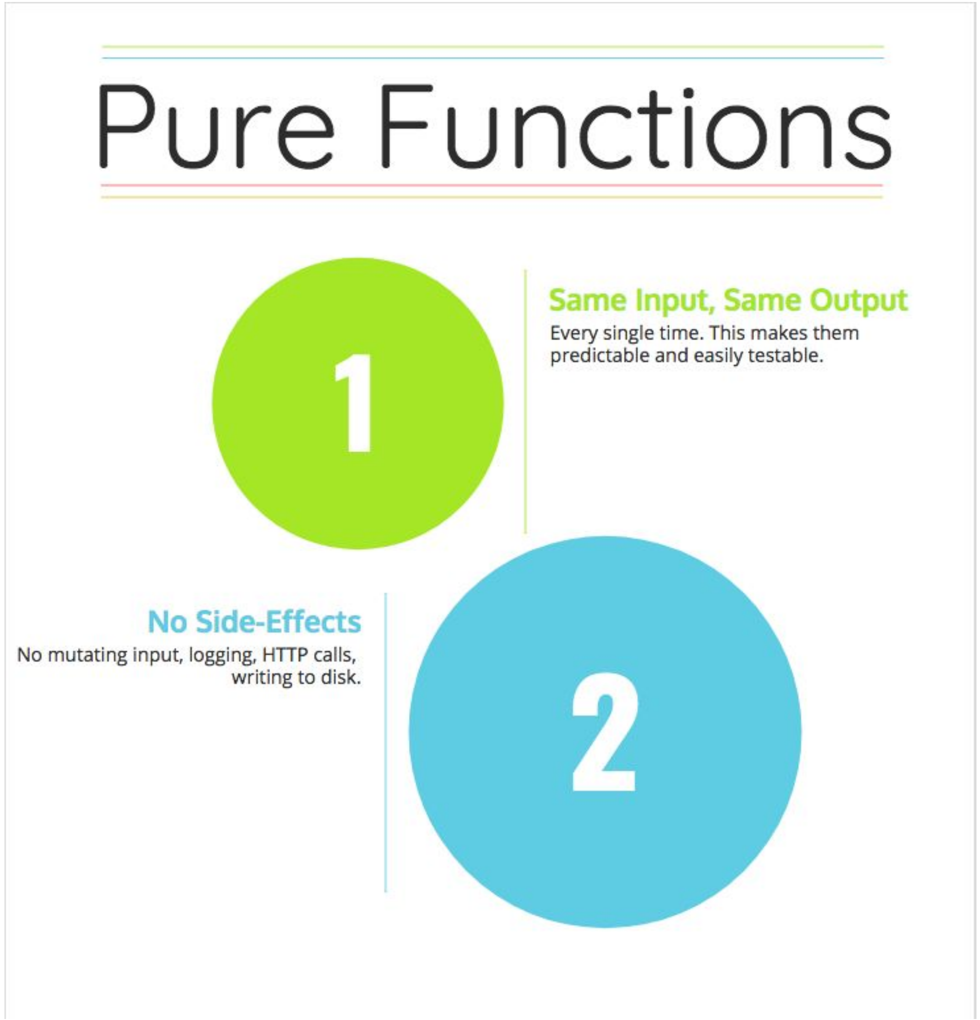


Immutability

- ❑ It’s about not changing data state or values after initializing it
- ❑ Instead of changing a variable value just create a new variable with the new value
- ❑ Immutability is great with multithreading since data values and states aren’t changing

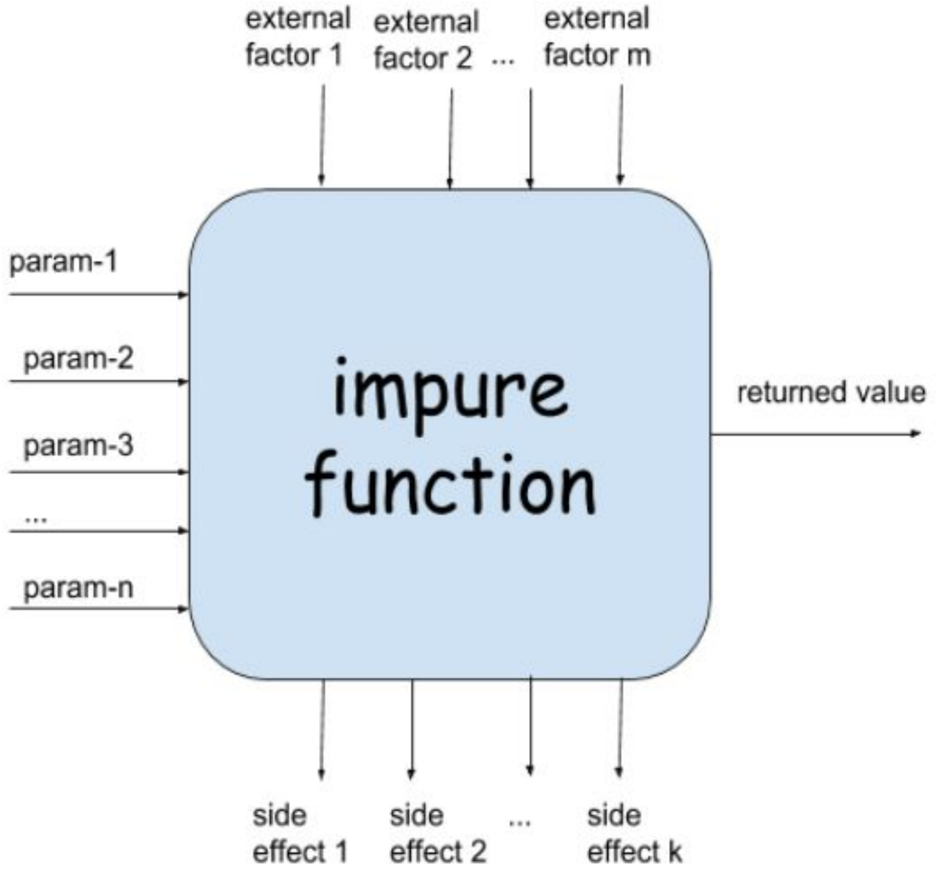
Pure Functions

These are functions that explicitly state their behavior through their inputs and outputs using the function’s body. This makes them easier to trace and debug since they don’t have any side effects.


Pure Functions
1
Same Input, Same Output
Every single time. This makes them predictable and easily testable.
No Side-Effects
No mutating input, logging, HTTP calls, writing to disk.
2

Impure Functions

These are functions that implicitly state their behavior through their inputs and outputs using variables from outside the function’s scope. These functions are typically harder to read because of the affect to/from outside-variables



First-Class Functions

These are functions that are treated as objects which can be stored in variables, passed around to other functions, or even having their values returned inside Higher-Order Functions

```
def shout(text):
    return text.upper()


yell = shout
```

Higher-Order Functions

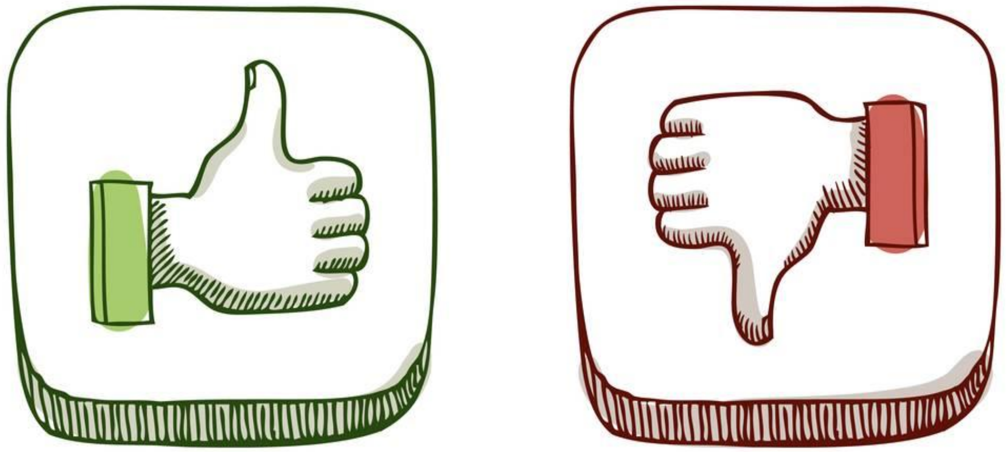
They are functions that use other functions as inputs (parameters), or return the values of other function’s values

```python
def create_adder(x):
    def adder(y):
        return x + y

    return adder
```

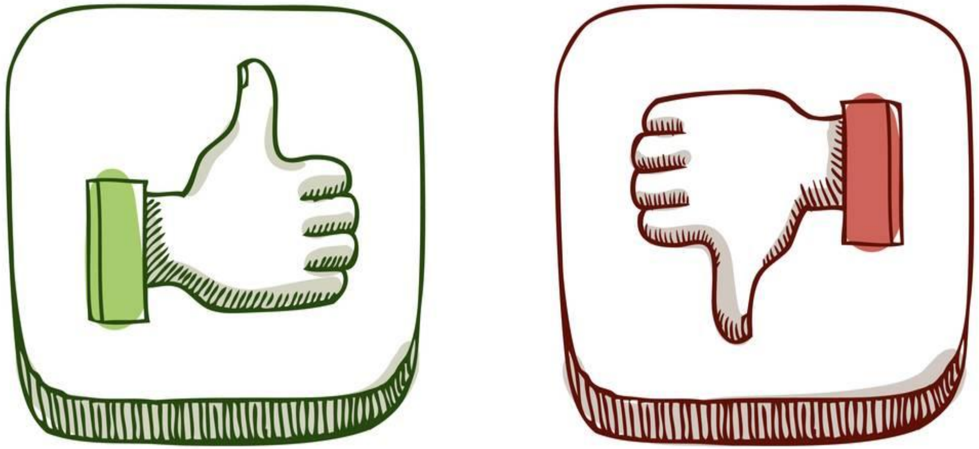
Advantages of Functional Programming

- ❏ Higher reusability
- ❏ Easy to debug and read
- ❏ More testable



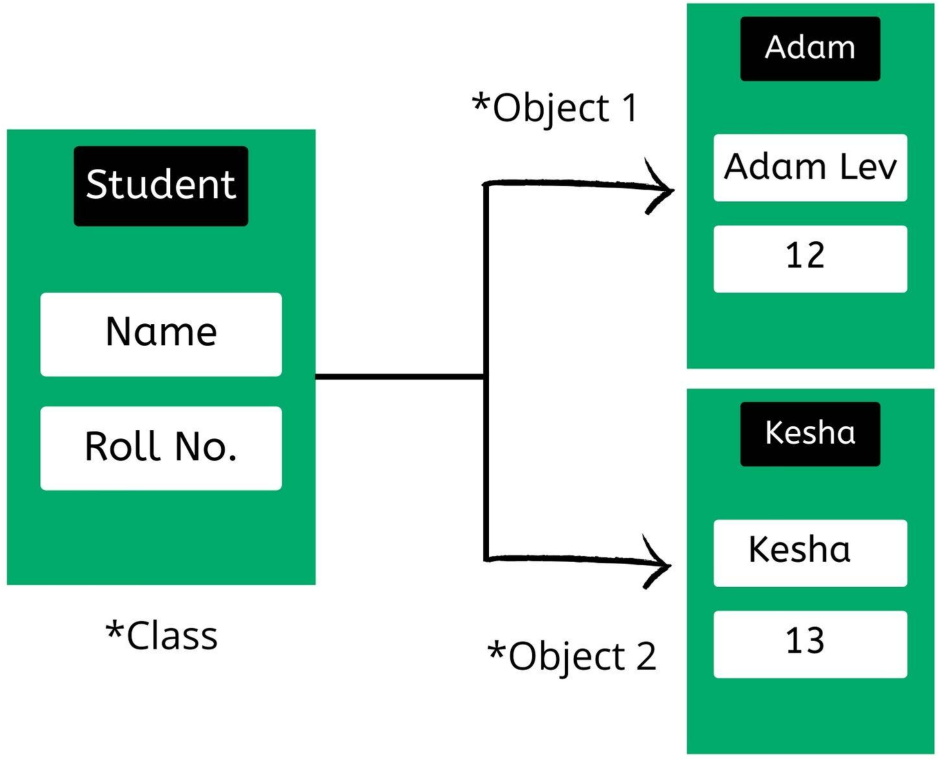
Disadvantages of Functional Programming

- ❏ Steep learning curve
- ❏ It may be complex sometimes due to immutability
- ❏ Smaller community and less tools



Object-Oriented Programming

It’s an Imperative-style programming paradigm that centers around objects and classes. Objects being the smallest unit in design and classes being the blueprints of what houses these objects along with their related methods



Paradigm Purity

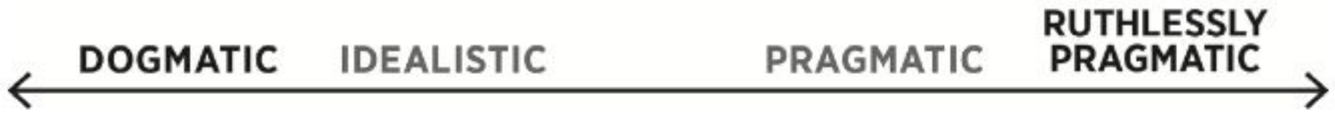
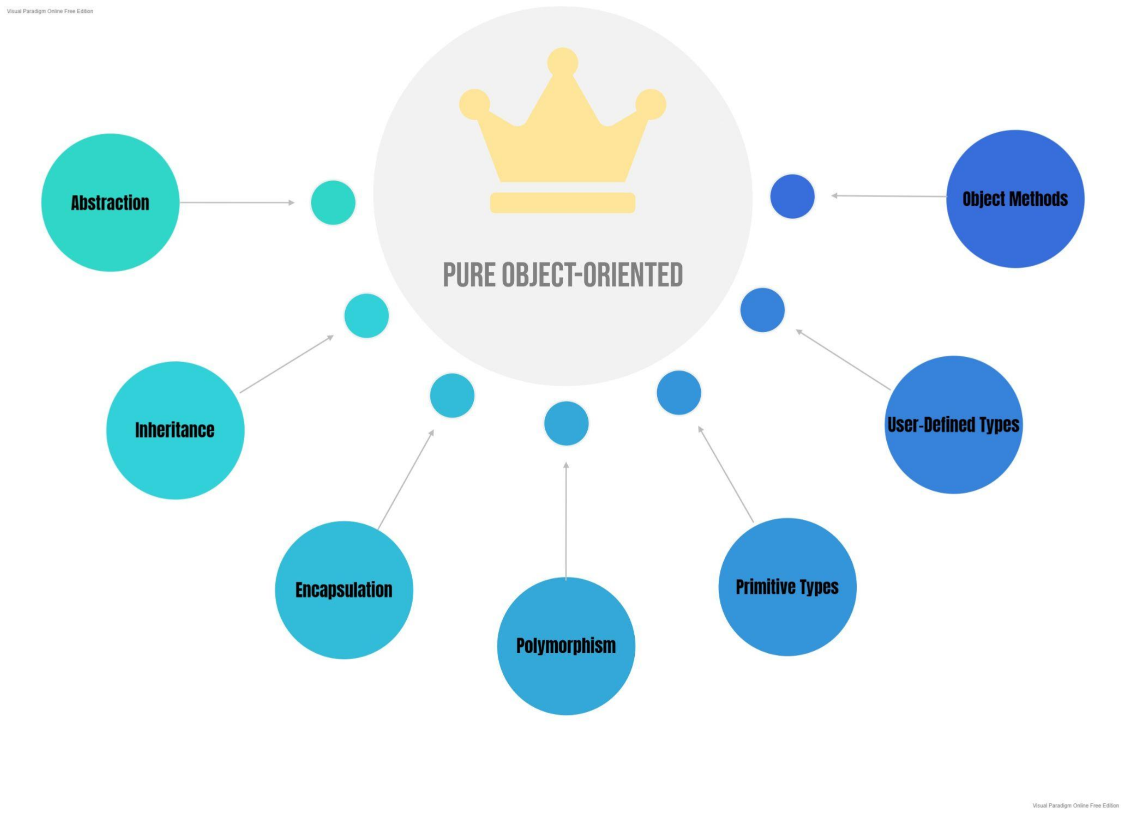


Fig. 4: The Spectrum from Idealism to Pragmatism

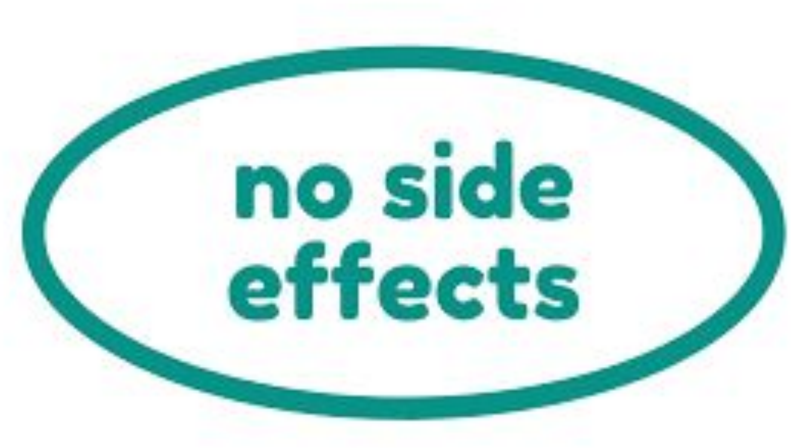
Purely Object-Oriented

For a language to be considered "Pure" Object-Oriented, there needs to be 7 requirements met



Purely Functional Programming

For a language to be considered "Pure" Functional programming , there needs to be one main requirement met.



Other Paradigms

Procedural Programming

Procedural Programming is an Imperative programming paradigm which is centered around procedures/functions. While Object-Oriented applications are divided into objects, Procedural ones are divided by functions

Logic Programming

Logic programming is a declarative programing paradigm that is based on mathematical logic expressions. These expressions define facts and rules. Queries are a widely used thing in logical programing to reach results using the rules that were already defined

Conclusion

- There is no “ultimate” programming paradigm
- Some programming languages are multi-paradigm
- OOP & FP are the most commonly used paradigms
- Each paradigm has its ups and downs as we’ve seen before
- Purity differs from one paradigm to another
- Being pure doesn’t necessarily mean better

Resources

- Introduction of Programming Paradigms - GeeksforGeeks
- What is Functional Programming? Tutorial with Example (guru99.com)
- Programming Languages: Principles & Paradigms
- Classification of Principal Programming Paradigms
- Higher Order Functions in Python - GeeksforGeeks
- https://syndicode.com/blog/pros-and-cons-of-functional-programming/
- Differences between Procedural and Object Oriented Programming - GeeksforGeeks
- https://betterprogramming.pub/when-and-when-not-to-use-functional-programming-73dbcb5d0a85