SOFTWARE ENGINEERING

Class 6

amrut@iiit.ac.in

INTRO TO PROGRAMMING PARADIGMS

TOPICS

- Overview of Programming Paradigms
- Imperative Programming Paradigms
- Declarative Programming Paradigms

OVERVIEW

OVERVIEW

How many Programming Languages are there?

OVERVIEW

How many Programming Languages are there?

~250 (as per TIOBE)

OVERVIEW

How many Programming Languages are there?

~250 (as per TIOBE)

~700 (as per Wikipedia)

OVERVIEW

How many Programming Languages are there?

~250 (as per TIOBE)

~700 (as per Wikipedia)

~8000 (as per HOPL)

OVERVIEW

How many Programming Languages are there?

~250 (as per TIOBE)

~700 (as per Wikipedia)

~8000 (as per HOPL)

Each of these languages has a few decisions behind it.

OVERVIEW

Programming Paradigm

A collection of decisions that language creator has taken which guide programmers' thoughts for programming of computers

OVERVIEW

Programming Languages

vs.

Programming Paradigms

OVERVIEW

Programming Language	Programming Paradigm
JAVA	Object Oriented
JavaScript	Object Oriented
Python	Object Oriented
Erlang	Functional
LISP	Functional
Go	Procedural
SQL	Declarative
Swift	Multi-Paradigm
Kotlin	Multi-Paradigm

OVERVIEW

Programming Paradigms

- Imperative Programming Paradigms

OVERVIEW

Programming Paradigms

- Imperative Programming Paradigms
- Declarative Programming Paradigms

IMPERATIVE

IMPERATIVE

Imperative Programming

IMPERATIVE

Imperative Programming

- Explicit Order of execution

IMPERATIVE

Imperative Programming

- Explicit Order of execution
- Change of state

IMPERATIVE

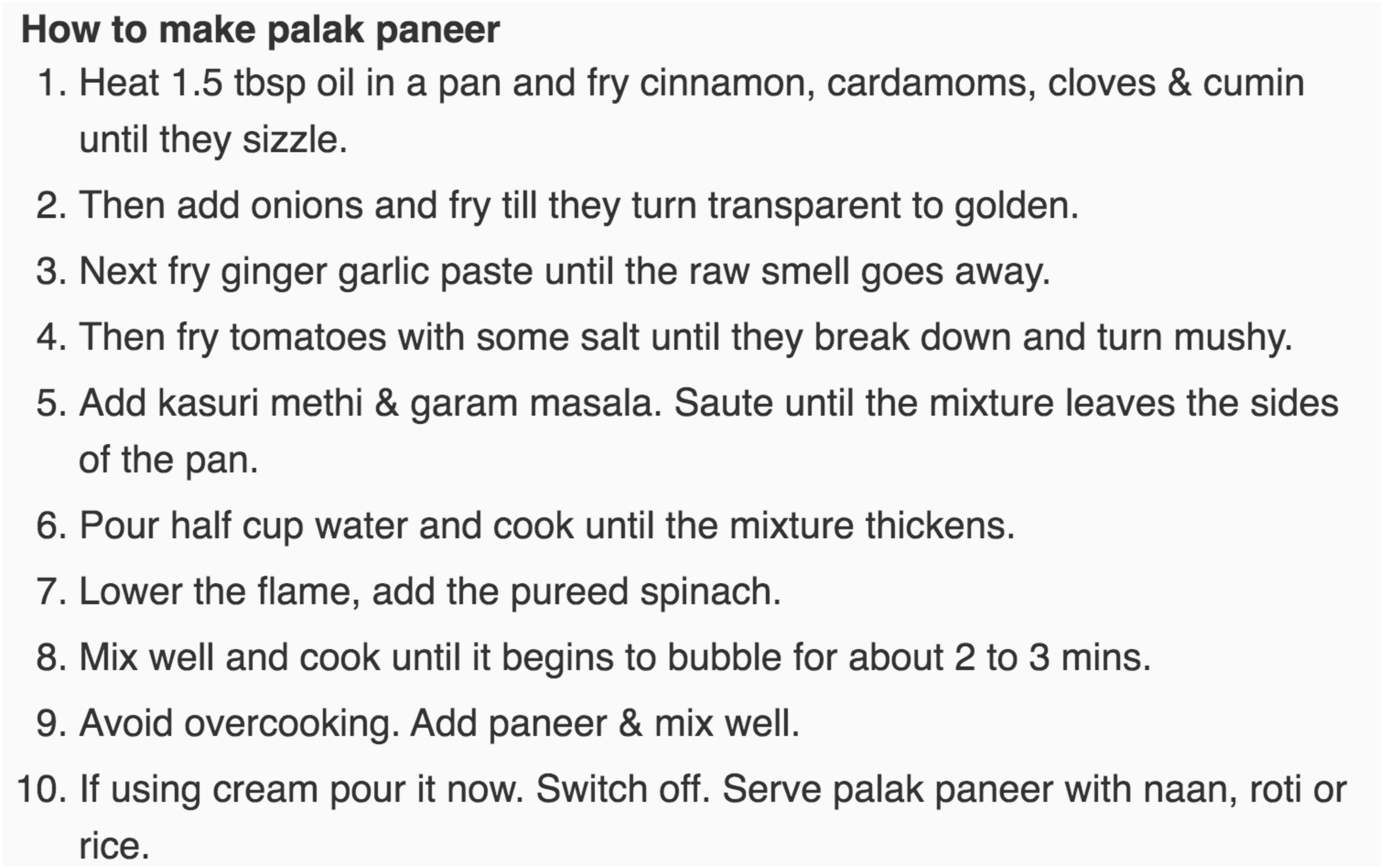
Imperative Programming

- Explicit Order of execution
- Change of state
- Explains the “how” of the program

IMPERATIVE

Examples

IMPERATIVE



How to make palak paneer

1. Heat 1.5 tbsp oil in a pan and fry cinnamon, cardamoms, cloves & cumin until they sizzle.
2. Then add onions and fry till they turn transparent to golden.
3. Next fry ginger garlic paste until the raw smell goes away.
4. Then fry tomatoes with some salt until they break down and turn mushy.
5. Add kasuri methi & garam masala. Saute until the mixture leaves the sides of the pan.
6. Pour half cup water and cook until the mixture thickens.
7. Lower the flame, add the pureed spinach.
8. Mix well and cook until it begins to bubble for about 2 to 3 mins.
9. Avoid overcooking. Add paneer & mix well.
10. If using cream pour it now. Switch off. Serve palak paneer with naan, roti or rice.

* *https://www.indianhealthyrecipes.com/palak-paneer-recipe-easy-paneer-recipes-step-by-step-pics/*

IMPERATIVE

Major Imperative Paradigms

- *Plain Imperative Programming*
- *Structured Programming*
- *Procedural Programming*
- *Object Oriented Programming*

IMPERATIVE – PLAIN IMPERATIVE

```
    numberlist = [1,2,3,4,5]
    result = 0
    current_index = 0
start:
    result = result + numberlist[current_index]
    if current_index < 5 goto next
    goto finished
next:
    current_index = current_index + 1
    goto start
finished:
    print result
```

IMPERATIVE - STRUCTURED PARADIGM

```
numberlist = [1,2,3,4,5]
result = 0

for current_index from 0 to 4
    result = result + numberlist[current_index]

print result
```

IMPERATIVE – PROCEDURAL PARADIGM

```
numberlist = [1,2,3,4,5]
call print_addition(numberlist)

procedure print_addition(numberlist)
    result = 0
    for current_index from 0 to 4
        result = result + numberlist[current_index]
    print result
```

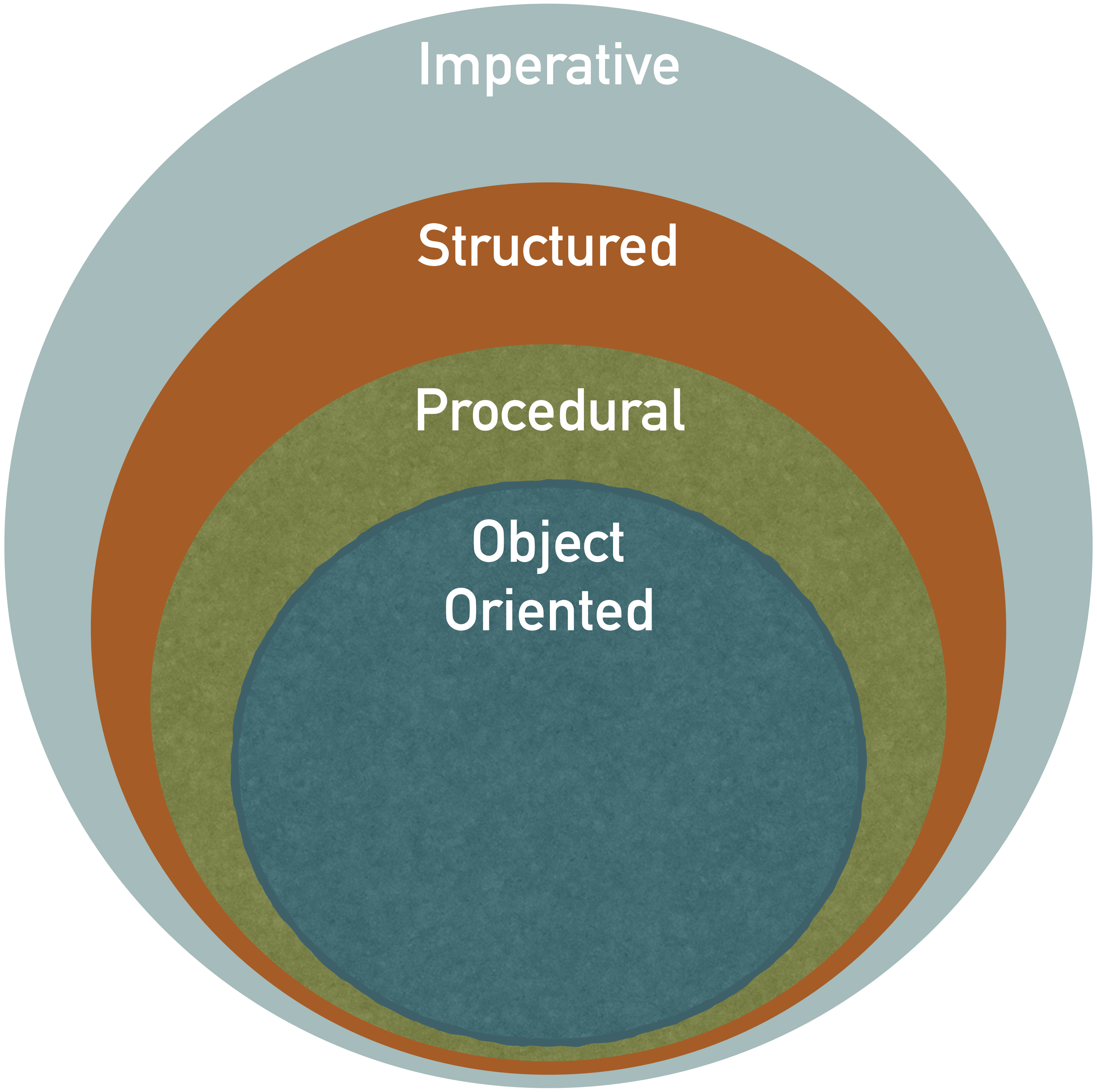
IMPERATIVE - OBJECT ORIENTED PARADIGM

```
numberlist = [1,2,3,4,5]
adder = new Adder
adder.initialize(numberlist)
adder.add_and_print()

class Adder
   method initialize(numberlist)
       numbers = numberlist

   method add_and_print()
       result = 0
       for current_index from 0 to numberlist.length
            result = result + numbers[current_index]
       print result
```

IMPERATIVE – SUMMARY



IMPERATIVE

Advantages

- *Most common paradigm used today*
- *Computing model more readily maps to the underlying hardware.*
- *Easy to code*
- *Gives more control to the programmer*

Disadvantages

- *Difficult to debug because of mutability*
- *Difficult to read/maintain*
- *Difficult to create concurrent/parallel system*
- *Making mistakes is easier*

DECLARATIVE

DECLARATIVE

Declarative Programming

- Implicit Order of Execution

DECLARATIVE

Declarative Programming

- Implicit Order of Execution
- Abstract out state mutation

DECLARATIVE

Declarative Programming

- Implicit Order of Execution
- Abstract out state mutation
- Explains the “what” of the program

DECLARATIVE

Declarative Programming

- Implicit Order of Execution (vs. explicit)
- Abstract out state mutation (vs. state changes)
- Explains the “what” of the program (vs. “how”)

DECLARATIVE

Examples

DECLARATIVE

Palak Paneer is Indian cottage cheese in smooth creamy spinach gravy

DECLARATIVE

Major Declarative Paradigms

- *Plain Declarative Programming*
- *Functional Programming*
- *Logic Programming*

DECLARATIVE – PLAIN DECLARATIVE

```
<html>
<body>

<h1>This is my heading</h1>

<p>This is a <b>bold</b> paragraph</p>

</body>
</html>
```

DECLARATIVE – FUNCTIONAL PROGRAMMING

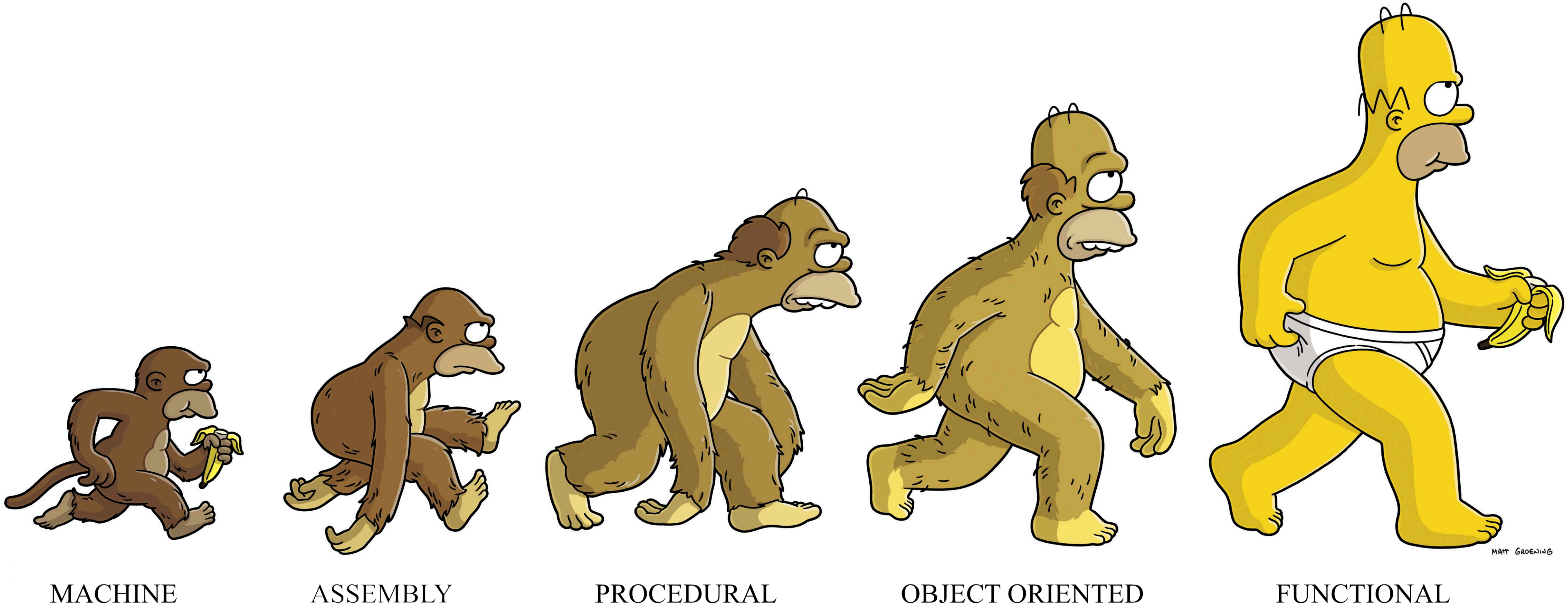
```
numberlist =:= [1,2,3,4,5]
add x, y =:= x+y
result =:= reduce(numberlist, 0, add)
```

DECLARATIVE – FUNCTIONAL PROGRAMMING

```
void example() {
    int[] numberList = new int[]{1, 2, 3, 4, 5};
    int result = Arrays.stream(numberList).reduce(0, this::add);
    System.out.println(result);
}

private int add(int x, int y) {
    return x + y;
}
```

DECLARATIVE – FUNCTIONAL PROGRAMMING



DECLARATIVE - LOGIC PROGRAMMING

```
// Facts
father(rheagar, jon)
mother(lyanna, jon)
father(aerys, rheagar)

// Rules
grandparent(X, Z) :- parent(X, Y), parent(Y, Z)
parent(X, Y) :- father(X, Y)
parent(X, Y) :- mother(X, Y)

// Query
?- grandparent(Q, jon)
```

DECLARATIVE

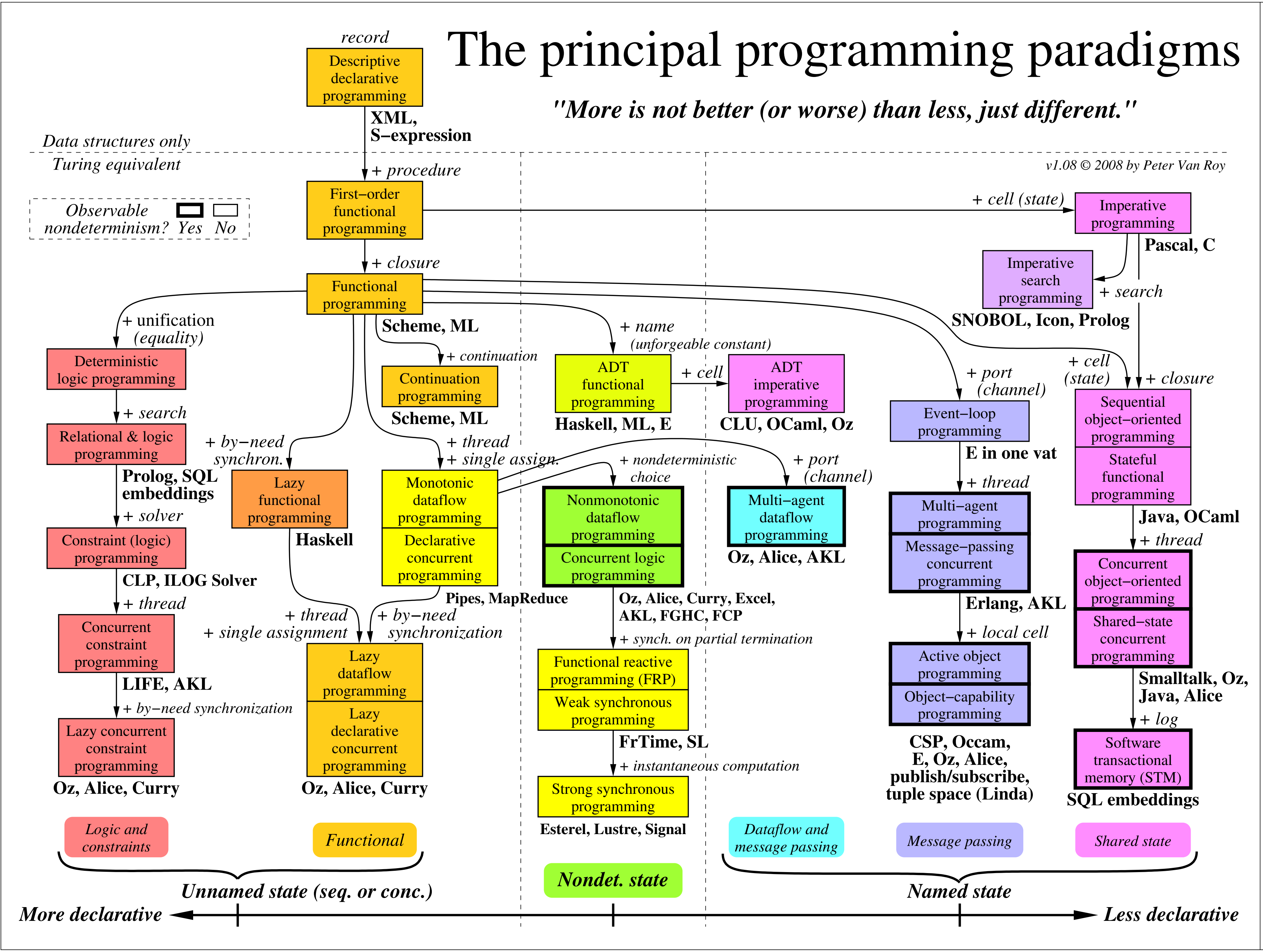
Advantages

- *More languages are supporting functional programming today*
- *Immutability makes it easier to debug*
- *Easy to read and understand*
- *Concurrency/parallelism is easy*

Disadvantages

- *Not familiar to a lot of people*
- *May not be highly efficient*

REFERENCES


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

Explanations

See "Concepts, Techniques, and Models of Computer Programming".

The chart classifies programming paradigms according to their kernel languages (the small core language in which all the paradigm's abstractions can be defined). Kernel languages are ordered according to the creative extension principle: a new concept is added when it cannot be encoded with only local transformations. Two languages that implement the same paradigm can nevertheless have very different "flavors" for the programmer, because they make different choices about what programming techniques and styles to facilitate.

When a language is mentioned under a paradigm, it means that part of the language is intended (by its designers) to support the paradigm without interference from other paradigms. It does not mean that there is a perfect fit between the language and the paradigm. It is not enough that libraries have been written in the language to support the paradigm. The language's kernel language should support the paradigm. When there is a family of related languages, usually only one member of the family is mentioned to avoid clutter. The absence of a language does not imply any kind of value judgment.

State is the ability to remember information, or more precisely, to store a sequence of values in time. Its expressive power is strongly influenced by the paradigm that contains it. We distinguish four levels of expressiveness, which differ in whether the state is unnamed or named, deterministic or nondeterministic, and sequential or concurrent. The least expressive is functional programming (threaded state, e.g., DCGs and monads: unnamed, deterministic, and sequential). Adding concurrency gives declarative concurrent programming (e.g., synchrocells: unnamed, deterministic, and concurrent). Adding nondeterministic choice gives concurrent logic programming (which uses stream mergers: unnamed, nondeterministic, and concurrent). Adding ports or cells, respectively, gives message passing or shared state (both are named, nondeterministic, and concurrent). Nondeterminism is important for real−world interaction (e.g., client/server). Named state is important for modularity.

Axes orthogonal to this chart are typing, aspects, and domain−specificity. Typing is not completely orthogonal: it has some effect on expressiveness. Aspects should be completely orthogonal, since they are part of a program's specification. A domain−specific language should be definable in any paradigm (except when the domain needs a particular concept).

Metaprogramming is another way to increase the expressiveness of a language. The term covers many different approaches, from higher−order programming, syntactic extensibility (e.g., macros), to higher−order programming combined with syntactic support (e.g., meta−object protocols and generics), to full−fledged tinkering with the kernel language (introspection and reflection). Syntactic extensibility and kernel language tinkering in particular are orthogonal to this chart. Some languages, such as Scheme, are flexible enough to implement many paradigms in almost native fashion. This flexibility is not shown in the chart.

https://www.info.ucl.ac.be/~pvr/paradigmsDIAGRAMeng108.pdf

REFERENCES

- *GOTOs considered harmful (https://homepages.cwi.nl/~storm/teaching/reader/Dijkstra68.pdf)*
- *https://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf*
- *https://en.wikipedia.org/wiki/Lambda_calculus*
- *https://en.wikipedia.org/wiki/First-order_logic*
- *https://www.amazon.com/Programming-Languages-Principles-Paradigms-Undergraduate/dp/1848829132*