Learning Outcomes

At the end of the topic, the students must have:

1. Familiarized the different programming paradigms
2. Differentiated the different paradigms in terms of software development
3. Identified the top most useful programming paradigms

Content

Paradigm vs. Programming Paradigm

Paradigm can also be termed as method to solve some problem or do some tasks. **Programming paradigm** is an approach to solve problem using some programming language or also we can say it is a method to solve a problem using tools and techniques that are available to us following some approach.

A **programming paradigm** is a style, or "way," of programming.

Some Common Programming Paradigms:

1. Imperative

Programming with an explicit sequence of commands that update state.

Imperative programming (from Latin imperare = command) is the oldest programming paradigm. A program based on this paradigm is made up of a clearly-defined sequence of instructions to a computer.

Therefore, the source code for imperative languages is a series of commands, which specify what the computer has to do – and when – in order to achieve a desired result. Values used in variables are changed at program runtime. To control the commands, control structures such as loops or branches are integrated into the code.

Imperative programming languages are very specific, and operation is system-oriented. On the one hand, the code is easy to understand; on the other hand, many lines of source text are required

to describe what can be achieved with a fraction of the commands using declarative programming languages.

These are the best-known imperative programming languages:

- Fortran
- Java
- Pascal
- ALGOL
- C
- C#
- C++
- Assembler
- BASIC
- COBOL
- Python
- Ruby

The different imperative programming languages can, in turn, be assigned to three further subordinate programming styles – structured, procedural, and modular.

The **structured programming** style extends the basic imperative principle with specific control structures: sequences, selection, and iteration. This approach is based on a desire to limit or completely avoid jump statements that make imperatively designed code unnecessarily complicated.

The **procedural approach** divides the task a program is supposed to perform into smaller sub-tasks, which are individually described in the code. This results in programming modules which can also be used in other programs. The **modular programming** model goes one step further by designing, developing, and testing the individual program components independently of one another. The individual modules are then combined to create the actual software.

Imperative programming example

Imperative programming languages are characterized by their instructive nature and, thus, require significantly more lines of code to express what can be described with just a few instructions in the declarative style. In the following example, the aim is to output a list of first names:

Imperative programming (PHP)

```
$participantlist = [1 => 'Peter', 2 => 'Henry', 3 => 'Sarah'];
$firstnames= [];
foreach ($participantlist as $id => $name) {
    $firstnames[] = $name;
}
```

Advantages & Disadvantages

Advantages	Disadvantages
Easy to read	Code quickly becomes very extensive and thus confusing
Relatively easy to learn	Higher risk of errors when editing
Conceptual model (solution path) is very easy for beginners to understand	System-oriented programming means that maintenance blocks application development
Characteristics of specific applications can be taken into account	Optimization and extension is more difficult

2. Declarative

Programming by specifying the result you want, not how to get it.

There is no one specific definition of the paradigm, but all definitions agree on one thing: A characteristic feature of declarative programming languages is that they always describe the desired end result rather than outlining all the intermediate work steps. In **declarative programming**, the solution path to reach the goal is determined automatically. This works well, provided the specifications of the final state are clearly defined and an appropriate implementation procedure exists. If both of these conditions are met, declarative programming is very efficient.

Since **declarative programming** does not specifically describe the "how" but works at a very high level of abstraction, the programming paradigm also leaves room for optimization. If a better implementation procedure is developed, the integrated algorithm can identify and use it. This makes the paradigm futureproof. The procedure for how the result is to be achieved does not have to be set in stone when writing the code.

The best-known declarative programming languages are:

- Prolog
- Lisp
- Haskell
- Miranda
- Erlang
- SQL (in the broadest sense)

The different declarative programming languages can, in turn, be divided into two paradigms: **functional programming** languages and **logic programming** languages.

However, in practice, the boundaries are frequently blurred and elements of both imperative programming – with its sub-type's procedural, modular, and structured programming – and declarative programming are used to solve problems.

Declarative programming example

One of the strengths of declarative programming is its ability to describe problems more briefly and succinctly than imperative languages.

If we want to output a list of first names, in PHP this can be described with just one line of code using declarative programming – as the example shows – while the imperative method requires five lines.

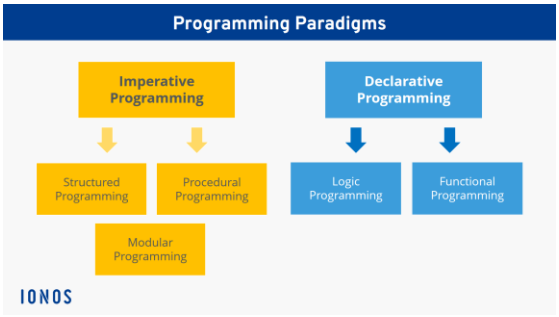
Declarative programming (PHP)

```
1 $firstnames = array_values($participantlist);
```

Advantages & Disadvantages

Advantages	Disadvantages
Short, efficient code	Sometimes hard to understand for external people
Can be implemented using methods not yet known at the time of programming	Based on an unfamiliar conceptual model for people (solution state)
Easy optimization as implementation is controlled by an algorithm	Hard to take characteristics of individual applications into account during programming
Maintenance possible independent of application development	

Imperative Vs Declarative Programming



Imperative programming languages differ from **declarative languages** on one fundamental point: **imperative programming** focuses on the "**how**", **declarative programming** on the "**what**".

But what does that mean**? Imperative programming languages** are composed of step-by-step instructions (how) for the computer. They describe explicitly which steps are to be performed in what order to obtain the desired solution at the end. By contrast, in declarative programming, the desired result (what) is described directly. This becomes clearer when using a cooking analogy for illustration: imperative languages provide recipes; declarative languages contribute photos of the finished meal.

In **declarative languages**, the source code remains very abstract in terms of the specific procedure. To get to the solution, an algorithm is used which automatically identifies and applies appropriate methods. This approach has numerous advantages: Programs can be written much more quickly, and applications are also very easy to optimize. If a new method is developed in the future, the abstract instructions in the source code mean that the algorithm can easily utilize the newer method.

Imperative programming (PHP)

```php
$participantlist = [1 => 'Peter', 2 => 'Henry', 3 => 'Sarah'];
$firstnames= [];
foreach ($participantlist as $id => $name) {
    $firstnames[] = $name;
}
```

Declarative programming (PHP)

```php
$firstnames = array_values($participantlist);
```

3. Structured (modular programming)

Programming with clean, goto-free, nested control structures.

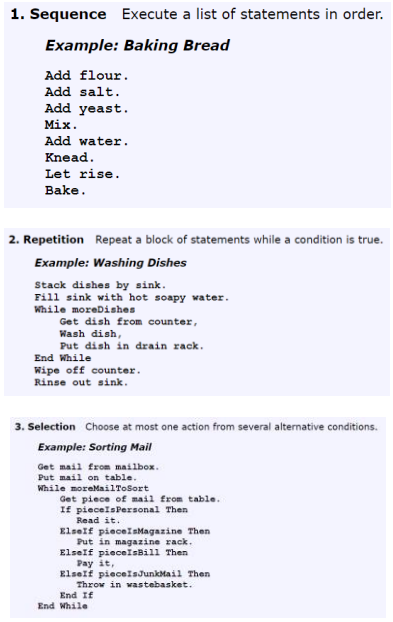
Structured programming is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.

Facilitates the creation of programs with readable code and reusable components. Where modules or elements of code can be reused from a library, it may also be possible to build structured

code using modules written in different languages, as long as they can obey a common module interface or application program interface (API) specification.

Structured programming encourages dividing an application program into a hierarchy of modules or autonomous elements, which may, in turn, contain other such elements. Within each element, code may be further structured using blocks of related logic designed to improve readability and maintainability. These may include case, which tests a variable against a set of values; Repeat, while and for, which construct loops that continue until a condition is met. In all structured programming languages, an unconditional transfer of control, or goto statement, is deprecated and sometimes not even available.

Example: Structured Programming in Everyday Life



1. Sequence Execute a list of statements in order.

Example: Baking Bread

```
Add flour.
Add salt.
Add yeast.
Mix.
Add water.
Knead.
Let rise.
Bake.
```

2. Repetition Repeat a block of statements while a condition is true.

Example: Washing Dishes

```
Stack dishes by sink.
Fill sink with hot soapy water.
While moreDishes
    Get dish from counter,
    Wash dish,
    Put dish in drain rack.
End While
Wipe off counter.
Rinse out sink.
```

3. Selection Choose at most one action from several alternative conditions.

Example: Sorting Mail

```
Get mail from mailbox.
Put mail on table.
While moreMailToSort
    Get piece of mail from table.
    If pieceIsPersonal Then
        Read it.
    ElseIf pieceIsMagazine Then
        Put in magazine rack.
    ElseIf pieceIsBill Then
        Pay it,
    ElseIf pieceIsJunkMail Then
        Throw in wastebasket.
    End If
End While
```

4. Procedural

Imperative programming with procedure calls.

Procedural Programming may be the first programming paradigm that a new developer will learn. Fundamentally, the procedural code is the one that directly instructs a device on how to finish a task in logical steps. This paradigm uses a linear top-down approach and treats data and procedures as two different entities. Based on the concept of a procedure call, Procedural Programming divides the program into procedures, which are also known as routines or functions, simply containing a series of steps to be carried out.

Simply put, Procedural Programming involves writing down a list of instructions to tell the computer what it should do step-by-step to finish the task at hand.

Advantages and Disadvantages of Procedural Programming

Procedural Programming comes with its own set of pros and cons, some of which are mentioned below.

Advantages

- Procedural Programming is excellent for general-purpose programming
- The coded simplicity along with ease of implementation of compilers and interpreters
- A large variety of books and online course material available on tested algorithms, making it easier to learn along the way
- The source code is portable, therefore, it can be used to target a different CPU as well
- The code can be reused in different parts of the program, without the need to copy it
- Through Procedural Programming technique, the memory requirement also slashes
- The program flow can be tracked easily

Disadvantages

- The program code is harder to write when Procedural Programming is employed
- The Procedural code is often not reusable, which may pose the need to recreate the code if is needed to use in another application
- Difficult to relate with real-world objects
- The importance is given to the operation rather than the data, which might pose issues in some data-sensitive cases
- The data is exposed to the whole program, making it not so much security friendly

Example (Javascript):

```
let accounts = [];

function account(name, balance = 300){
  accounts.push({
    name: name,
    balance:  balance
  });
}

function getAccount(name){
  for(let i = 0; i < accounts.length; i ++){
    if(accounts[i].name === name){
      return accounts[i];
    }
  }
}

function deposit(name, amount){
  let account = getAccount(name);
  account.balance = account.balance + amount;
}

function withdraw(name, amount){
  let account = getAccount(name);
  account.balance = account.balance - amount;
}

function transfer(payer, beneficiary, payment){
  let payerAccount = getAccount(payer);
  withdraw(payerAccount.name, payment);
  let beneficiaryAccount = getAccount(beneficiary);
  deposit(beneficiaryAccount.name, payment);
}
```

For example, to develop a simple Bank Account App procedurally:

- Creating an account for an individual (account)
- Getting an account to deposit or withdraw funds (getAccount, deposit, withdraw)
- Transferring funds between two different accounts (transfer)
- Recording all changes that occur with all accounts (accounts)

5. Functional

Programming with function calls that avoid any global state.

Functional Programming is the use of pure high-order functions to develop applications. This involves a focus on creating code that avoids changing state and mutating data.

Example (Javascript):

1.
```
//higher order function that returns a function

function tacoFactory(taco){
    return function (string){
        return taco + " " + string
    }
}

let bestTacoEver = tacoFactory("cactus taco")

console.log(bestTacoEver("is the one taco to rule them all"));
// => cactus taco is the one taco to rule them all
```

2.
```
const hi = name => `Hi ${name}`

const greeting = hi;

greeting("Stranger")   //
```

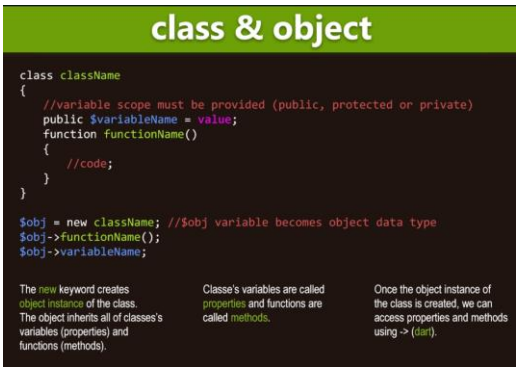
6. Object-Oriented

Programming by defining objects that send messages to each other. Objects have their own internal (encapsulated) state and public interfaces.

Object orientation can be:

a. **Class-based**: Objects get state and behavior based on membership in a class.
b. **Prototype-based**: Objects get behavior from a prototype object.

Example (PHP):



7. Event-Driven Programming

Programming with emitters and listeners of asynchronous actions.

In computer programming, event-driven programming is a programming paradigm in which the flow of the program is determined by events such as user actions (mouse clicks, key presses), sensor outputs (embedded), or message passing from other programs or threads (e.g., using Application Programming Interface or API).

Event-driven programs can be written in any programming language, although some languages (Visual Basic for example) are specifically designed to facilitate event-driven programming, and provide an integrated development environment (IDE) that partially automates the production of code, and provides a comprehensive selection of built-in objects and controls, each of which can respond to a range of events. Virtually all object-oriented and visual languages support event-driven programming. Visual Basic, Visual C++ and Java are examples of such languages.

A visual programming IDE such as VB.Net provides much of the code for detecting events automatically when a new application is created. The programmer can therefore concentrate on issues such as interface design, which involves adding controls such as command buttons, text boxes, and labels to standard forms (a form represents an application's workspace or window). Once the user interface is substantially complete, the programmer can add event-handling code to each control as required.

The change in emphasis from procedural to event-driven programming has been accelerated by the introduction of the Graphical User Interface (GUI) which has been widely adopted for use in operating systems and end-user applications. It really began, however, with the introduction of object-oriented (OO) programming languages and development methodologies in the late 1970s. By

the 1990's, object-oriented technologies had largely supplanted the procedural programming languages and structured development methods that were popular during the 70s and 80s.

The link between object-oriented programming and event-driven programming is fairly obvious. For example, objects on a Visual Basic form (usually referred to as controls) can be categorised into classes (e.g. "Button", "TextBox" etc.), and many instances of each can appear on a single form. Each class will have attributes (usually referred to as properties) that will be common to all objects of that type (e.g. "BackgroundColour", "Width" etc.), and each class will define a list of events to which an object of that type will respond. The methods (event-handlers) to handle specific events are usually provided as templates to which the programmer simply has to add the code that carries out the required action.

Top programming languages in 2016 and their respective programming paradigms (Samuel, 2017)

Number	Programming Language	Main Programming Paradigm(s)
1	Java	Object-Oriented, Imperative, Event-driven with GUI, Concurrent, Functional, Generic, & Reflection
2	C	Imperative (Procedural and Structural)
3	C++	Imperative, Object-Oriented
4	Python	Imperative, Object-Oriented, Functional, Event-Driven with GUI, Concurrent, Reflection, & Meta programming
5	C#	Object-Oriented, Imperative, Event-Driven with GUI, Functional, Concurrent, Generic, & Reflection
6	R	Functional, Object-Oriented, Event-Driven with GUI, Imperative, Reflective, & Array
7	PHP	Imperative, Object-Oriented, Event-Driven with GUI, Functional, & Reflection
8	Java Script	Scripting

Top Programming Language Worldwide in 2021 according to PYPL.

PYPL Index: The PYPL PopularitY of Programming Language Index is created by analyzing how often language tutorials are searched on Google. The index is updated once a month.

PYPL Index (Worldwide)

Aug 2021	Change	Programming language	Share	Trends
1		Python	29.93 %	-2.2 %
2		Java	17.78 %	+1.2 %
3		JavaScript	8.79 %	+0.6 %
4		C#	6.73 %	+0.2 %
5	↑	C/C++	6.45 %	+0.7 %
6	↓	PHP	5.76 %	-0.0 %
7		R	3.92 %	-0.1 %
8		Objective-C	2.26 %	-0.3 %
9	↑	TypeScript	2.11 %	+0.2 %
10	↓	Swift	1.96 %	-0.3 %
11	↑	Kotlin	1.81 %	+0.3 %
12	↓	Matlab	1.48 %	-0.4 %
13		Go	1.29 %	-0.2 %
14	↑↑	Rust	1.21 %	+0.2 %
15	↓	VBA	1.16 %	-0.1 %
16	↓	Ruby	1.02 %	-0.1 %
17		Scala	0.79 %	-0.1 %
18	↑	Ada	0.77 %	+0.2 %
19	↓	Visual Basic	0.75 %	+0.0 %
20		Dart	0.68 %	+0.2 %

NOTE: Paradigms are not meant to be mutually exclusive; a single program can feature multiple paradigms!

References

https://cs.lmu.edu/~ray/notes/paradigms/

https://www.ionos.com/digitalguide/websites/web-development/imperative-programming/

Samuel, M. S. (2017). An insight into programming paradigms and their programming languages. Journal of Applied Technology and Innovation, 1(1), 37-57.

https://searchsoftwarequality.techtarget.com/definition/structured-programming-modular-programming

https://condor.depaul.edu/sjost/nwdp/

https://hackr.io/blog/procedural-programming

https://levelup.gitconnected.com/functional-object-oriented-procedural-programming-644feda5bcfc

https://www.technologyuk.net/computing/software-development/software-design/event-driven-programming.shtml