
UiA
University of Agder
Norway

Lecture #2.2

Procedural-oriented PLC programming

MAS418

Programming for Intelligent Robotics and Industrial systems

Part II: PLC Software Development

Spring 2024

Daniel Hagen, PhD

UiA

Previous Lecture

Introduction to Part II -

PLC Software Development:

I. Introduction to PLC programming

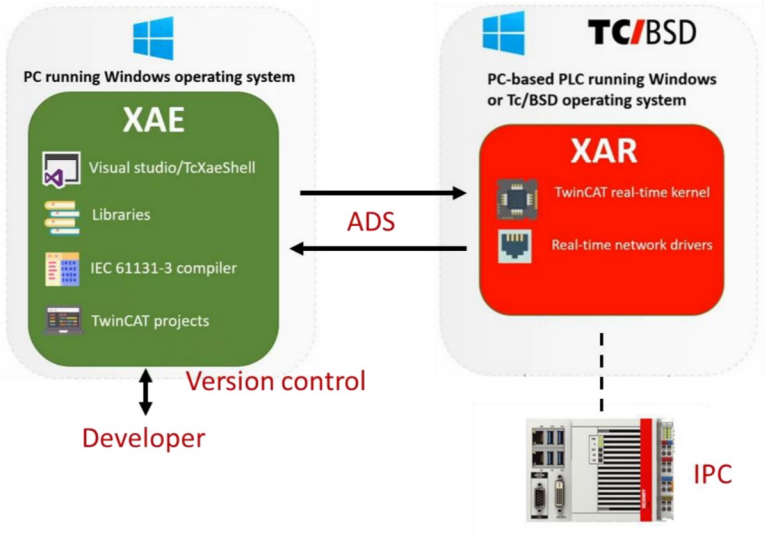
- SCADA system
- History
- IEC61131-3 standard
- Basic data types
- Version-control

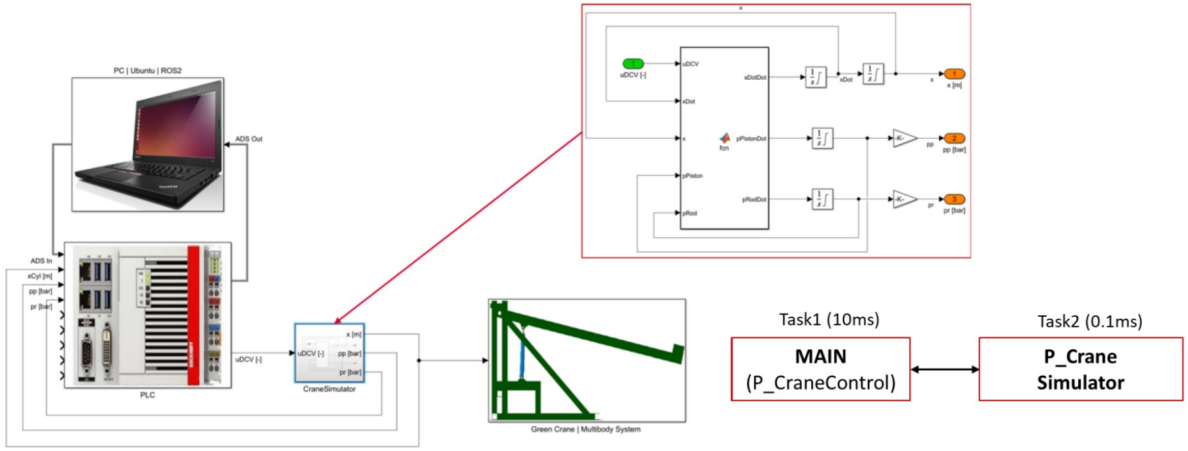
II. TwinCAT basics

- Introduction to TwinCAT
- Difference to SW running in OS (self-study)

III. Simulation in TwinCAT

- Simulink Example of Green Crane
- Green Crane simulator example
- Simplified hydro-mechanical model
- Time-domain simulation
- Create new task in TwinCAT
- Simulink PLC Coder

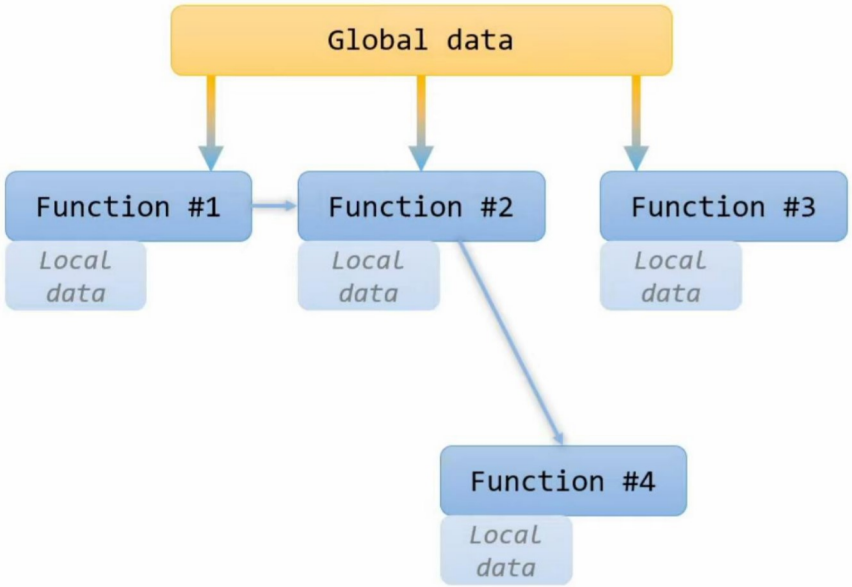




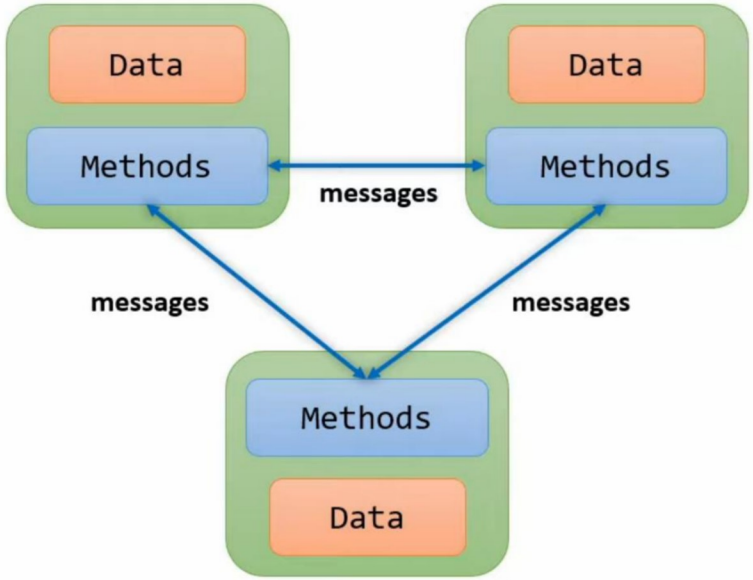
Imperative programming paradigms

- Within the computer science field there are two major imperative programming paradigms:

Procedural Oriented Programming (POP)

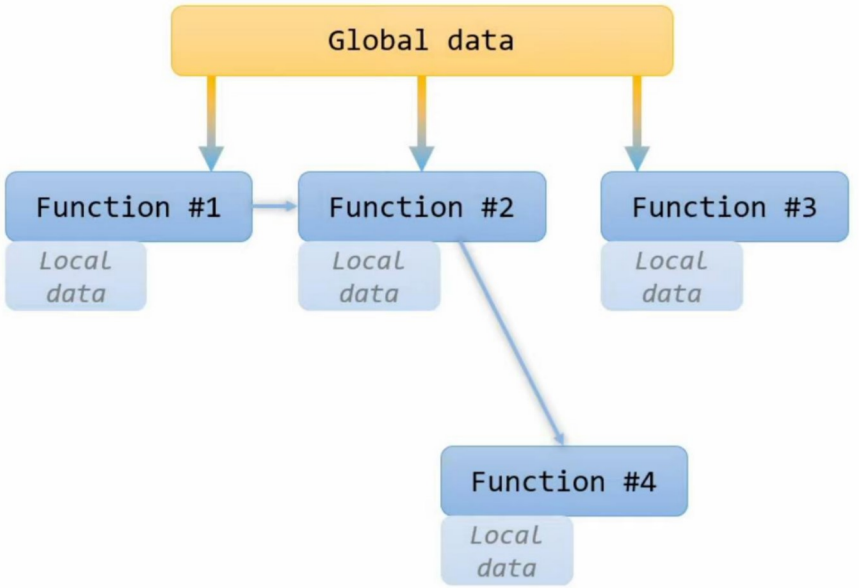


Object Oriented Programming (OOP)



Key takeaways

- **Procedural-oriented PLC programming**
  - **Data types continue:**
    - Time
    - Pointers
    - Reference
    - Arrays
    - Enumeration
  - **Standard library:**
    - TON, TOF, R_TRIG, F_TRIG, CONCAT
  - Structures
  - Functions
  - Pass by value & pass by reference
  - **Instructions:**
    - Conditional statements
    - CASE instruction
    - FOR loops
    - WHILE loops



Overview

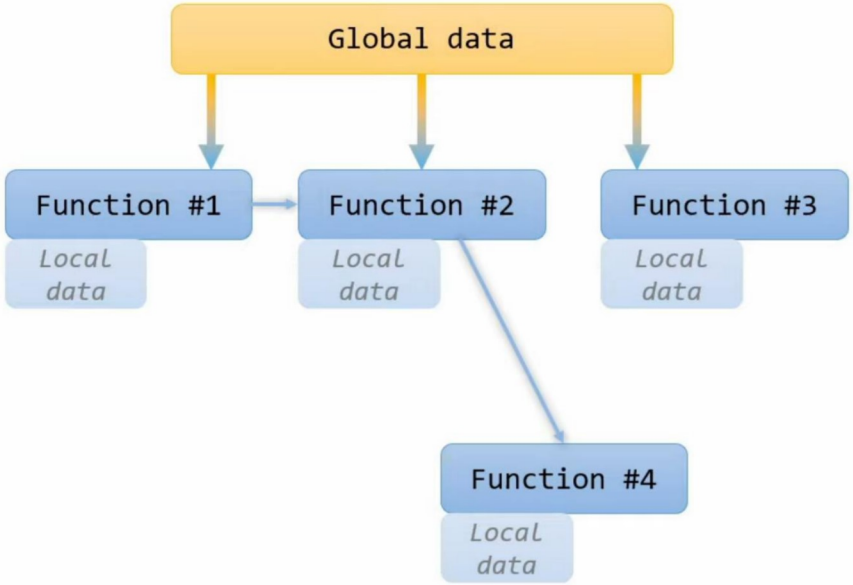
Introduction

Part I: Data types cont. & Std. library

Part II: Structures & Functions

Part III: Instructions

Summary



UiA

Part I: Data types cont. & Std. library

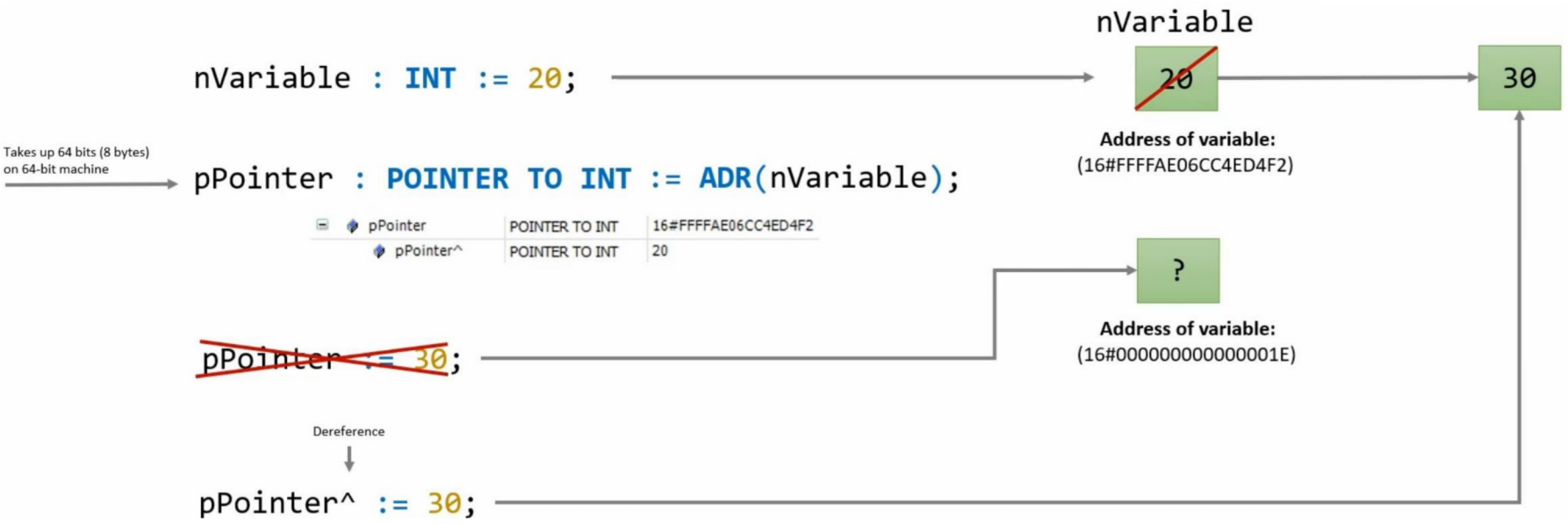
1. Time
2. Pointers
3. Reference
4. Arrays
5. Enumeration
6. Standard library

Time

TYPE	LOWER BOUND	UPPER BOUND	MEMORY CONSUMPTION	EXAMPLE
TIME	0 [ms]	4294967295 [ms]	32 bits (4 bytes)	`tVariable : TIME := T#1D4H5M2S132MS;`
TIME_OF_DAY (TOD)	0 (00:00:00:000) [ms]	4294967295 (23:59:59:999) [ms]	32 bits (4 bytes)	`tdVariable : TIME_OF_DAY := TOD#11:14:45.577;`
DATE	0 (01.01.1970) [s]	4294967295 (7 February 2106) [s]	32 bits (4 bytes)	`dDateVariable : DATE := D#1982-03-01;`
DATE_AND_TIME (DT)	0 (01.01.1970, 00:00) [s]	4294967295(7 February 2106, 6:28:15) [s]	32 bits (4 bytes)	`dtVariable : DATE_AND_TIME := DT#2013-04-18-15:31:19;`
LTIME	0 [ns]	213503d23h34m33s709ms551us615ns [ns]	64 bits (8 bytes)	`ltVar : LTIME := LTIME#213503D23H34M33S709MS551US615NS;`

https://www.youtube.com/watch?v=JSFvSu8uB9U&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=4

Pointers

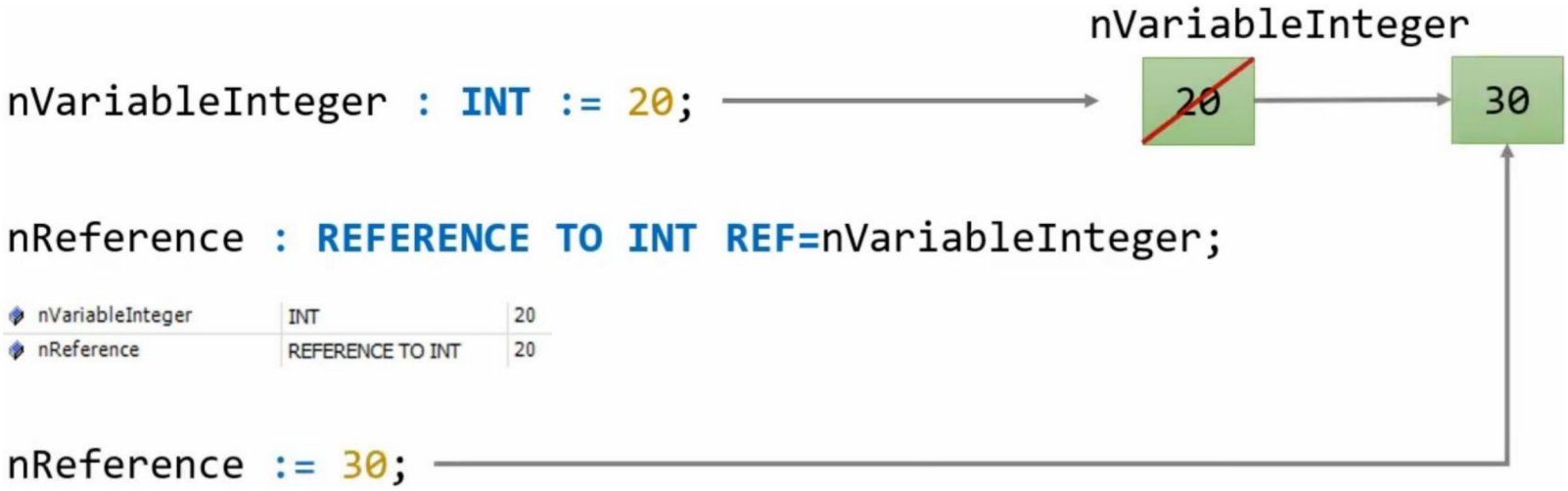



```
VAR
    dtVariable : DATE_AND_TIME := DT#2013-04-18-15:31:19;
    sHelloWorldString : STRING := 'Hello World!';
    pToString : POINTER TO STRING;
END_VAR

pToString := ADR(dtVariable);
```

pToString	POINTER TO STRING	16#FFFFAE06CC4ED474
pToString^	POINTER TO STRING	'Ç$11pQ$04'

https://www.youtube.com/watch?v=JSFvSu8uB9U&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=5

Reference



- Easier to use
- Type safe

```
dtVariable : DATE_AND_TIME := DT#2013-04-18-15:31:19;
nVariableInteger : INT := 122;
nReference : REFERENCE TO INT REF= dtVariable;
```

Cannot convert type 'DATE_AND_TIME' to type 'REFERENCE TO INT'

https://www.youtube.com/watch?v=JSFvSu8uB9U&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=5

Reference

- For more details about pointers and references, and how they differ to in C++, see the link below
- In general:
  - use **reference** when you can
  - and **pointers** when you have to

https://www.youtube.com/watch?v=JSFvSu8uB9U&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=5

Arrays

```
aOneArray : ARRAY[1..5] OF INT := [10, 20, 30, 40, 50];
```

```
aOneArray[2] := 133;
```


```
aOneArray : ARRAY[0..2] OF LREAL;
```

```
aOneArray : ARRAY[-1..1] OF STRING;
```

UiA

https://www.youtube.com/watch?v=JSFvSu8uB9U&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=5

Arrays

Alternative #1

```
a2dArray : ARRAY[1..2, 1..3] OF REAL;
```

a2dArray		
2dArray[1,1]	2dArray[1,2]	2dArray[1,3]
2dArray[2,1]	2dArray[2,2]	2dArray[2,3]

Alternative #2

```
a2dArray2 : ARRAY[1..2] OF ARRAY[1..3] REAL;
```

a2dArray2		
2dArray[1][1]	2dArray[1][2]	2dArray[1][3]
2dArray[2][1]	2dArray[2][2]	2dArray[2][3]

UiA

https://www.youtube.com/watch?v=JSFvSu8uB9U&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=5

Arrays

a2dArray	ARRAY [1..2, 1..3] OF REAL
a2dArray[1, 1]	REAL
a2dArray[1, 2]	REAL
a2dArray[1, 3]	REAL
a2dArray[2, 1]	REAL
a2dArray[2, 2]	REAL
a2dArray[2, 3]	REAL
a2dArray2	ARRAY [1..2] OF ARRAY [1..3] OF REAL
a2dArray2[1]	ARRAY [1..3] OF REAL
a2dArray2[1][1]	REAL
a2dArray2[1][2]	REAL
a2dArray2[1][3]	REAL
a2dArray2[2]	ARRAY [1..3] OF REAL
a2dArray2[2][1]	REAL
a2dArray2[2][2]	REAL
a2dArray2[2][3]	REAL

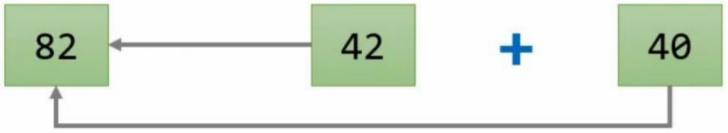
UiA

https://www.youtube.com/watch?v=JSFvSu8uB9U&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=5

Arrays


```
VAR
   aOneArray : ARRAY[1..5] OF INT := [10, 20, 30, 40, 50];
   nOneInt : INT;
END_VAR
```

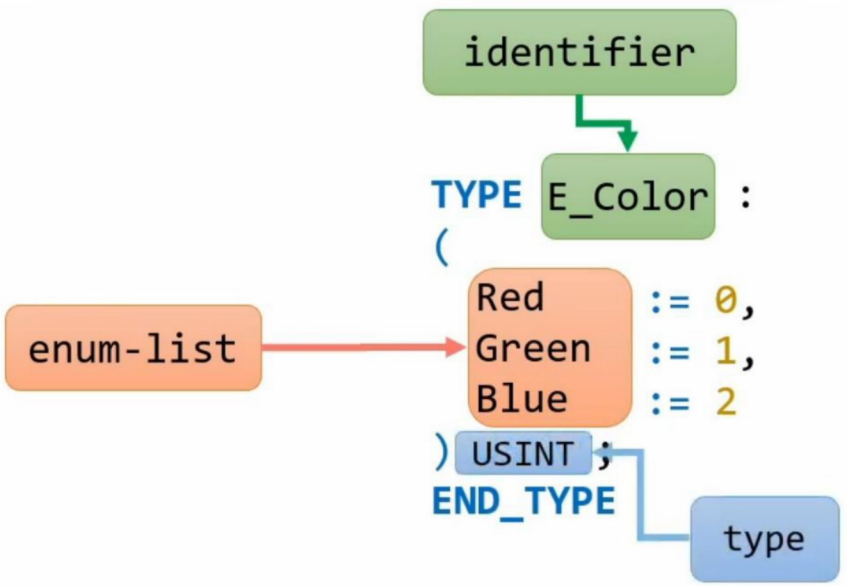
```
aOneArray[1] := 42;
nOneInt := aOneArray[1] + aOneArray[4];
```



UiA

https://www.youtube.com/watch?v=JSFvSu8uB9U&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=5

Enumeration




```
VAR
    eLightTower : E_Color;
END_VAR
```

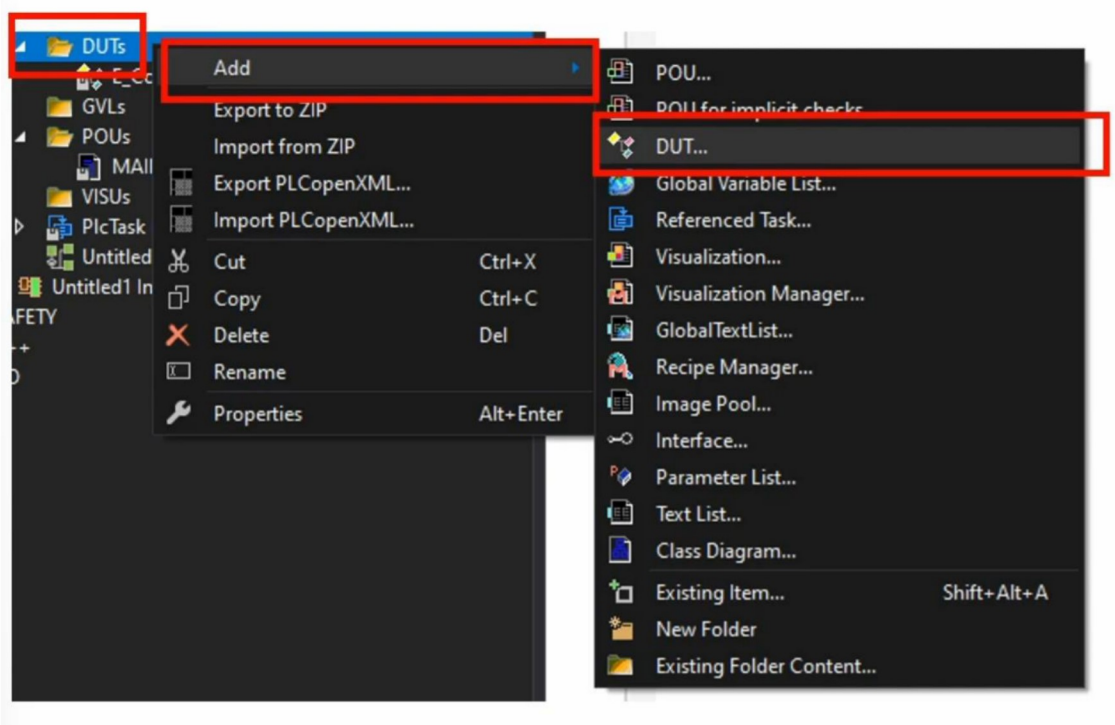
```
eLightTower := E_Color.Red;
```

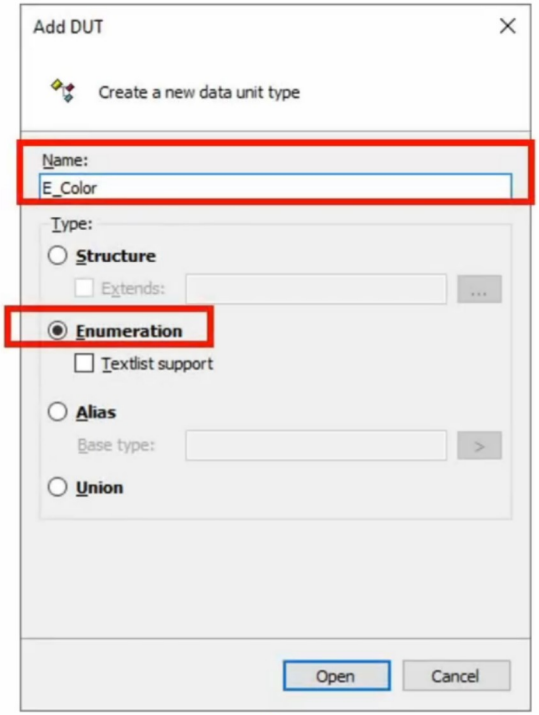
```
eLightTower := E_Color.
```

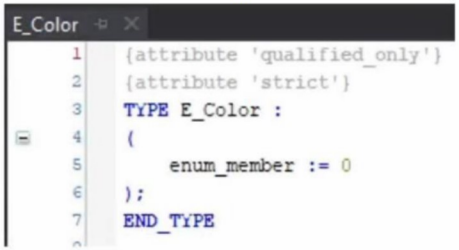
Blue
Green
Red

https://www.youtube.com/watch?v=JSFvSu8uB9U&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=4

Enumeration







https://www.youtube.com/watch?v=JSFvSu8uB9U&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=4

Standard library

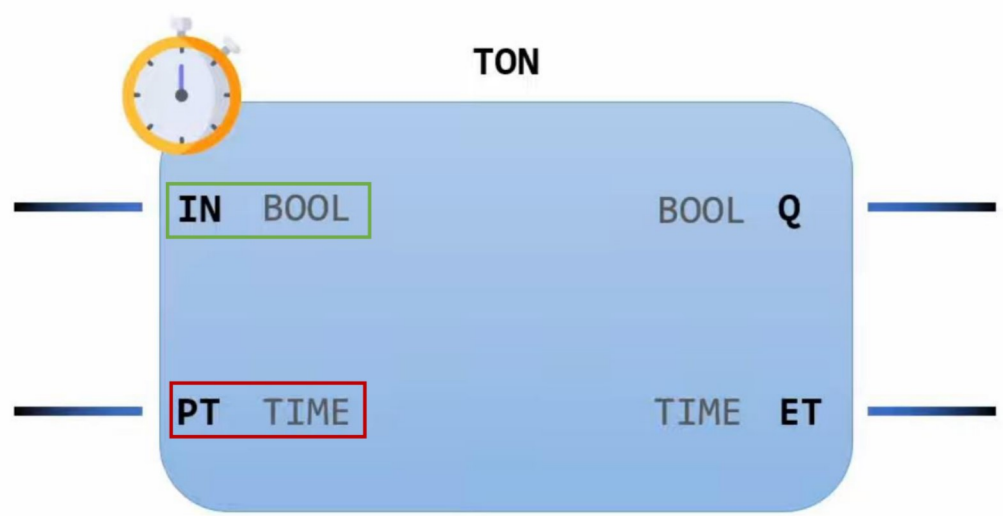
Some basic **PLC** functions and function blocks from the TwinCAT (Tc2_ Standard) library:

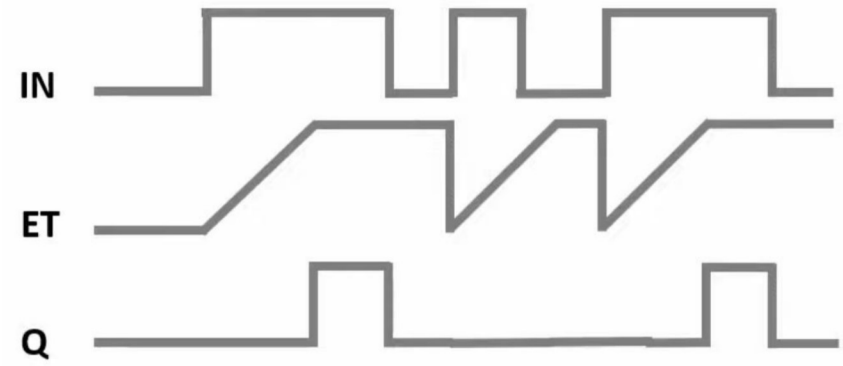
- **T**imer **ON** delay **– TON**
- Timer **OF**F delay **– TOF**
- **R**ising edge detection **– R_TRIG**
- **F**alling edge detection **– F_TRIG**
- **Concat**enate two STRINGs **– CONCAT**
  - One of many string functions

https://www.youtube.com/watch?v=n94JY-5-0jg&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=9

Standard library

- **TON**
  - Turns an output ON after a delay
  - The input **PT** is the time of the delay (wait period)
  - The input **IN** is used to reset the timer with a falling edge.
  - The output **ET** is the elapsed time that displays the amount of time since the start of the timer
  - The output **Q** is set to true after the timer has elapsed **ET** amount of time



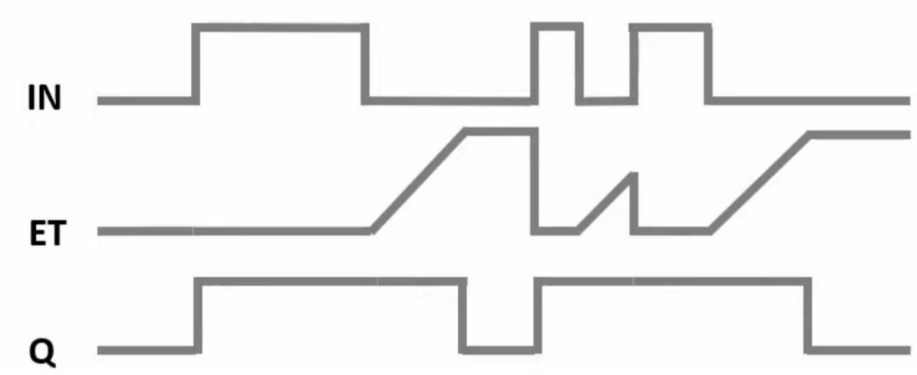


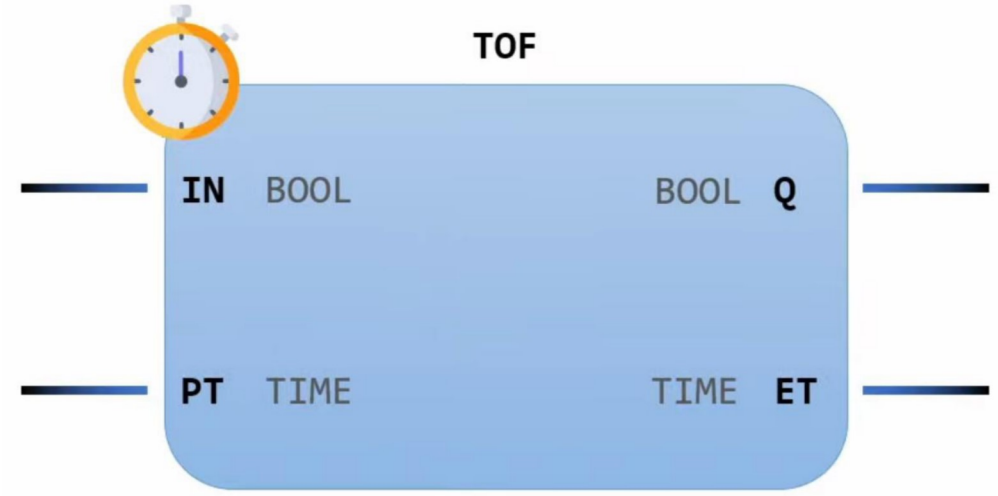

```
VAR
  fbTon : TON := (PT := T#3S);
  bBool : BOOL;
END_VAR

fbTon(IN := bBool);
```

https://www.youtube.com/watch?v=n94JY-5-0jg&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=9

Standard library

- **TOF**
  - The inverse of **TON**
  - Instead of having the timer elapsing when **IN** is high, it is counted when IN is low
    - i.e., when a falling edge is a trigger for the **FB**
  - Another difference is that once the timer has elapsed the output **Q** goes low as opposed to high in **TON**





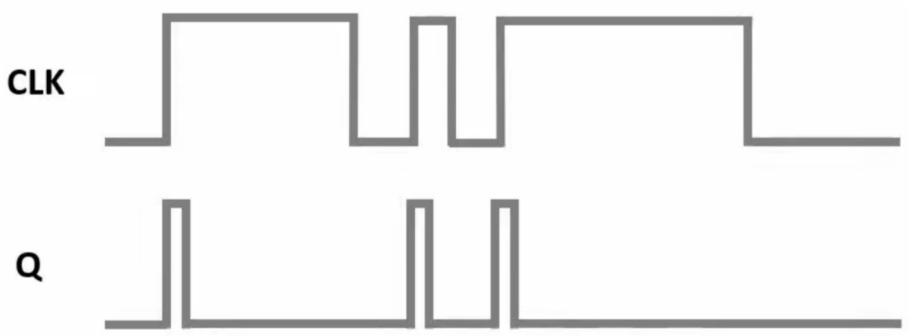

```
VAR
  fbTof : TOF := (PT := T#3S);
  bBool : BOOL;
END_VAR
```

```
fbTof(IN := bBool);
```

https://www.youtube.com/watch?v=n94JY-5-0jg&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=9

Standard library

- **R_TRIG**
  - Used to detect a **rising edge**
  - The input **CLK** is the signal to detect
  - The output **Q** is the edge detected
  - Each time the **FB** is called, **Q** will return a false until **CLK** has been false followed by a rising edge
    - i.e., the value true

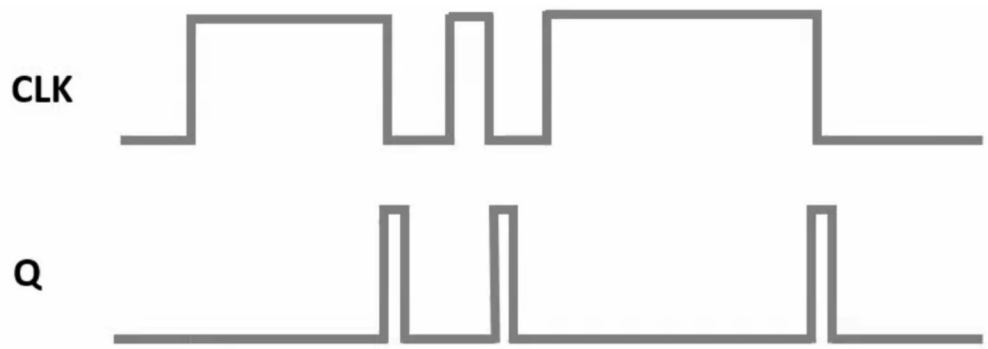



```
VAR
  fbRTrig : R_TRIG;
  bBoolean : BOOL;
END_VAR

fbRTrig(CLK := bBoolean);
```

https://www.youtube.com/watch?v=n94JY-5-0jg&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=9

Standard library

- **F_TRIG**
  - Works as **R_TRIG** but detects a **falling edge** instead of a **rising edge**
  - Each time the **FB** is called **Q** will return false until **CLK** has been true, followed by a falling edge
    - i.e., the value false

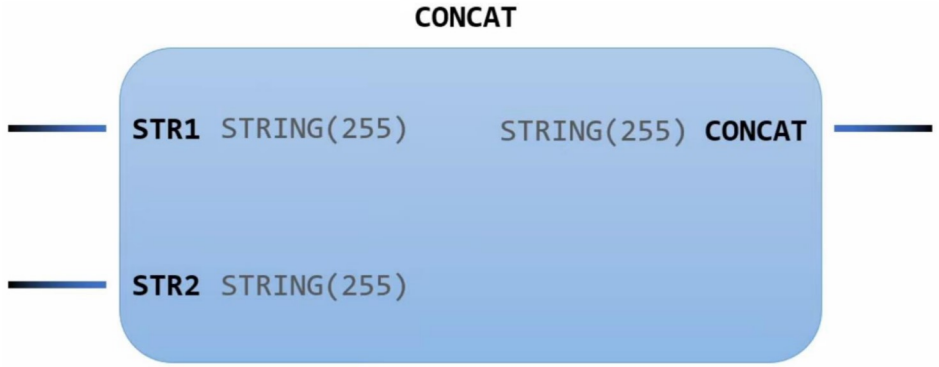



```
VAR
  fbFTrig : F_TRIG;
  bBoolean : BOOL;
END_VAR

fbFTrig(CLK := bBoolean);
```

https://www.youtube.com/watch?v=n94JY-5-0jg&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=9

Standard library

- **CONCAT**
  - Concatenates two **STRINGS** into one
  - **Strings** can be longer than 255 characters
  - The **CONCAT** function only works with strings up to 255 characters (pr input) and the resulting SRING is 255
  - There is a **FB** called **CONCAT2** that can take **STRINGS** of an arbitrary length and concatenate them




```
VAR
  sString1 : STRING(255) := 'Hello ';
  sString2 : STRING(255) := 'World!';
  sResultString : STRING(255);
END_VAR
```

```
sResultString := CONCAT(STR1 := sString1, STR2 := sString2);
```

→ 'Hello World!'

https://www.youtube.com/watch?v=n94JY-5-0jg&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=9

Part II: Structures & Functions

1. Introduction
2. Structures
3. Functions
4. Pass by value & pass by reference

UiA

Structures

```
aEventType : ARRAY[1..100] OF E_EventType;
aEventSeverity : ARRAY[1..100] OF TcEventSeverity;
aEventIdentity : ARRAY[1..100] OF UDINT;
aEventText : ARRAY[1..100] OF STRING(255);
aTimestamp : ARRAY[1..100] OF DATE_AND_TIME;
```

UiA

https://www.youtube.com/watch?v=zB_2x0NPMGc

Structures

- **Structure** is a used defined data type available in IEC 61131-3 that allows us to combine data items of **different** types

```
TYPE ST_Event :
STRUCT
    eEventType : E_EventType;
    eEventSeverity : TcEventSeverity;
    nEventIdentity : UDINT;
    sEventText : STRING(255);
    dtTimestamp : DATE_AND_TIME;
END_STRUCT
END_TYPE
```

```
aEventType : ARRAY[1..100] OF E_EventType;
aEventSeverity : ARRAY[1..100] OF TcEventSeverity
aEventIdentity : ARRAY[1..100] OF UDINT;
aEventText : ARRAY[1..100] OF STRING(255);
aTimestamp : ARRAY[1..100] OF DATE_AND_TIME;
```

```
VAR
    aEvents : ARRAY[1..100] OF ST_Event;
END_VAR
```

UiA

https://www.youtube.com/watch?v=zB_2x0NPMGc

Structures

- **Structure** is a used defined data type available in IEC 61131-3 that allows us to combine data items of **different** types

```
TYPE ST_Event :
STRUCT
    eEventType : E_EventType;
    eEventSeverity : TcEventSeverity;
    nEventIdentity : UDINT;
    sEventText : STRING(255);
    dtTimestamp : DATE_AND_TIME;
END_STRUCT
END_TYPE
```

```
VAR
    aEvents : ARRAY[1..100] OF ST_Event;
    sTempString : STRING(255);
END_VAR
```

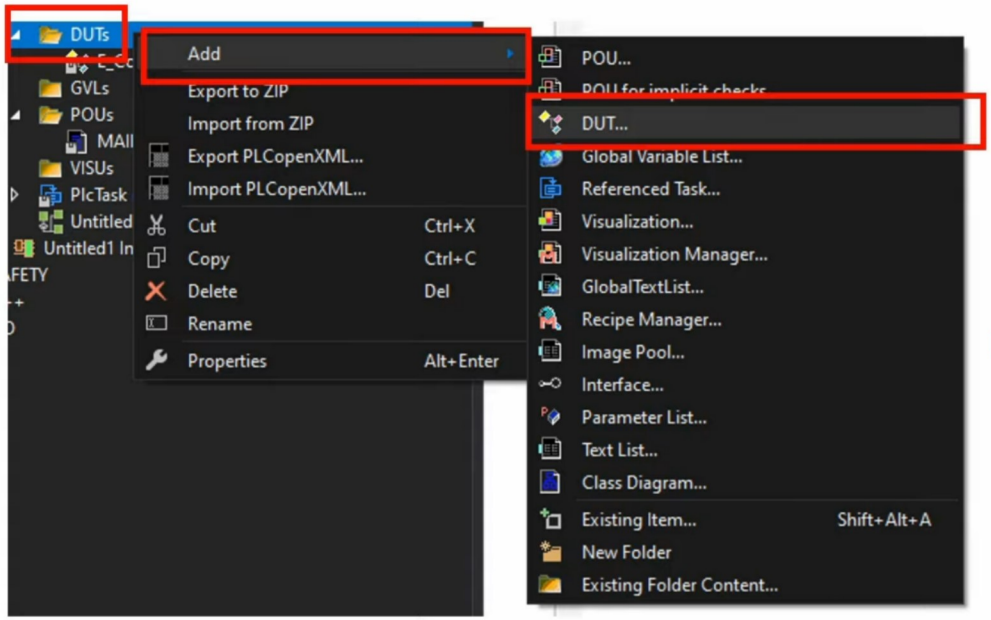
```
sTempString := aEvents[33].sEventText;
```

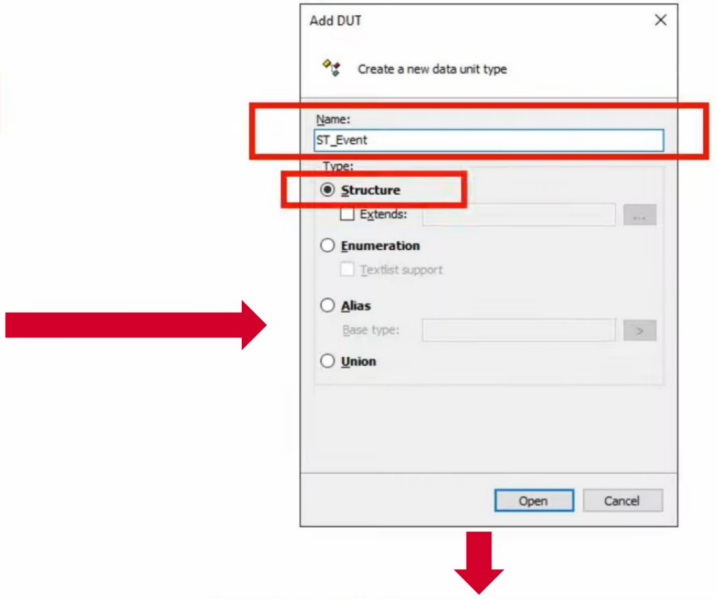
UiA

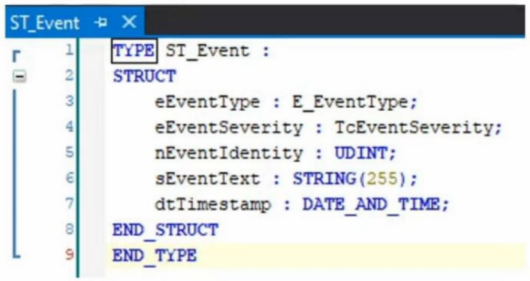
https://www.youtube.com/watch?v=zB_2x0NPMGc

Structures

- **How to create a struct:**





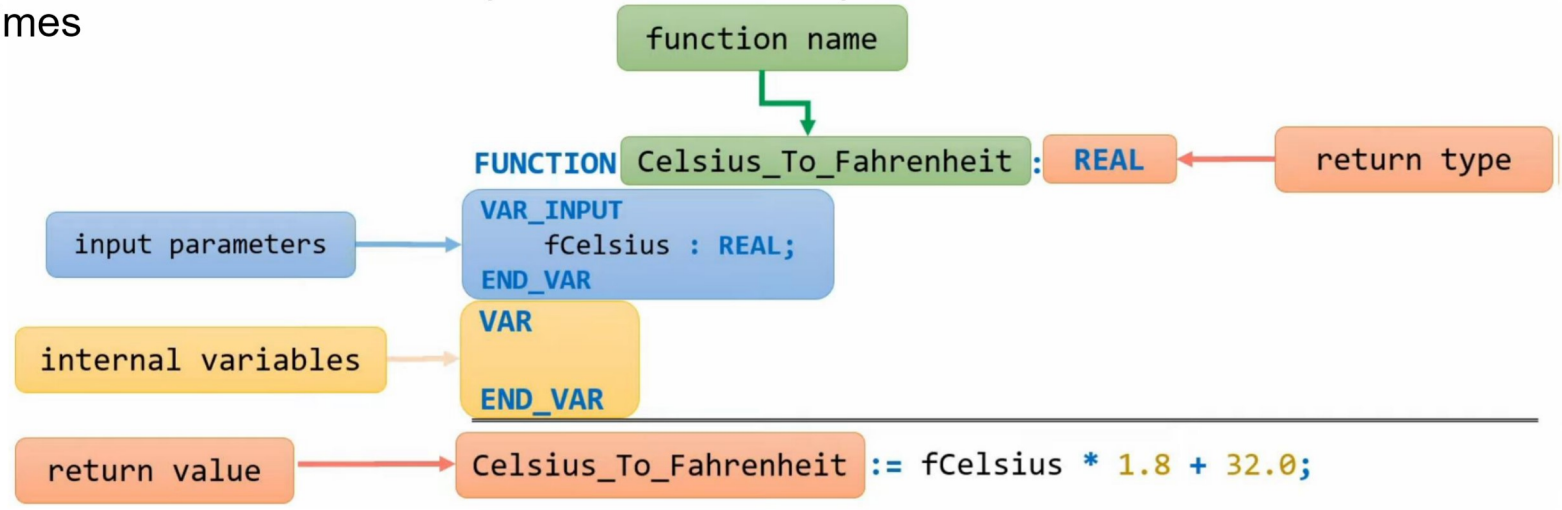


- Just as with enumerations the scope of **structures** is global

https://www.youtube.com/watch?v=zB_2x0NPMGc

Functions

- Functions are used to perform certain actions and they are important for reusing code
  - Define the code once, and you can use it many times




```
PROGRAM MAIN
VAR
    fFahrenheit : REAL;
END_VAR
```

41 ← `fFahrenheit := Celsius_To_Fahrenheit(fCelsius := 5.0);`

https://www.youtube.com/watch?v=zB_2x0NPMGc

Functions

```
FUNCTION Greater_Smaller
VAR_INPUT
    nNumber1 : INT;
    nNumber2 : INT;
END_VAR
VAR_OUTPUT
    nGreater : INT;
    nSmaller : INT;
END_VAR
```

output parameters

```
IF nNumber1 > nNumber2 THEN
    nGreater := nNumber1;
    nSmaller := nNumber2;
ELSE
    nGreater := nNumber2;
    nSmaller := nNumber1;
END_IF
```

```
PROGRAM MAIN
VAR
    nReturnGreater : INT;
    nReturnSmaller : INT;
END_VAR
```

```
Greater_Smaller(nNumber1 := 8,
                nNumber2 := 15,
                nGreater => nReturnGreater,
                nSmaller => nReturnSmaller);
```

nReturnGreater	nReturnSmaller
15	8

https://www.youtube.com/watch?v=zB_2x0NPMGc

Functions

```
FUNCTION SwapNums
VAR_IN_OUT
    nNumber1 : INT; 5
    nNumber2 : INT; 15
END_VAR
VAR
    nNumberTemp : INT;
END_VAR
```

input & output parameters at same time!

```
5  nNumberTemp := nNumber1; 5
15 nNumber1 := nNumber2; 15
5  nNumber2 := nNumberTemp; 5
```

```
PROGRAM MAIN
VAR
    nA : INT := 5;
    nB : INT := 15;
END_VAR
```

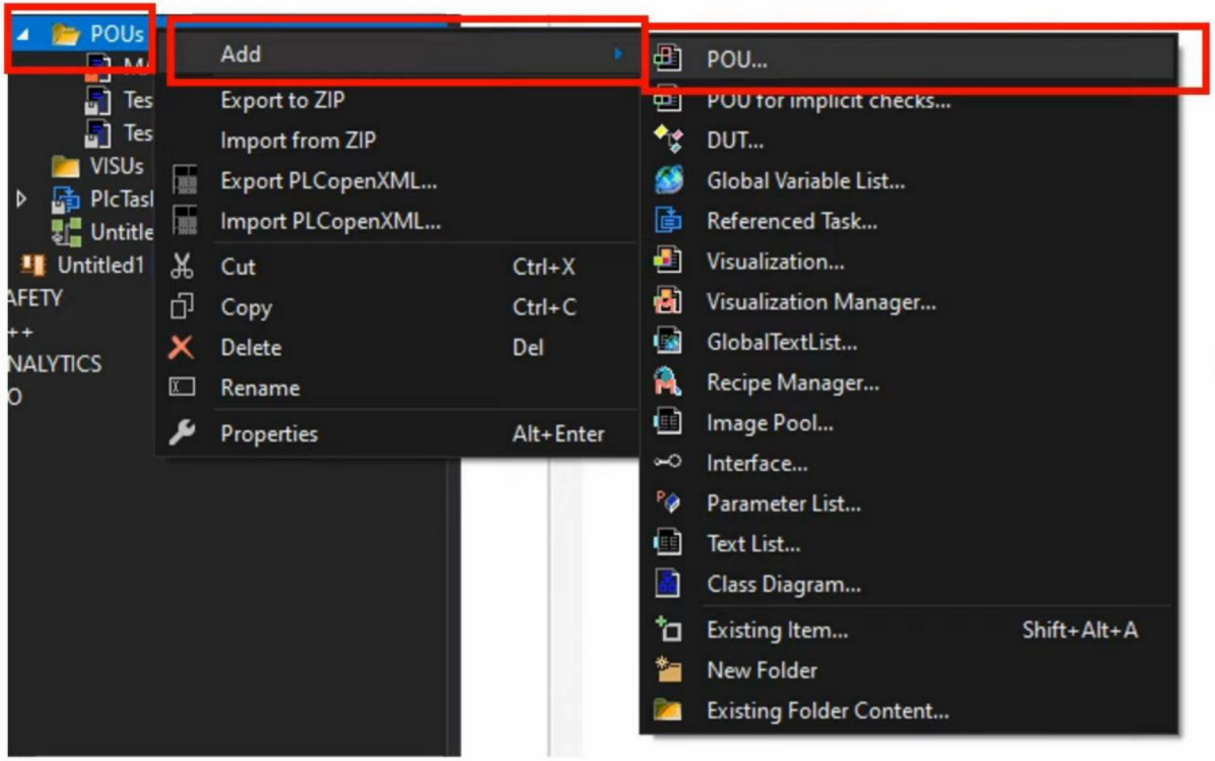
```
SwapNums(nNumber1 := nA,
         nNumber2 := nB);
```

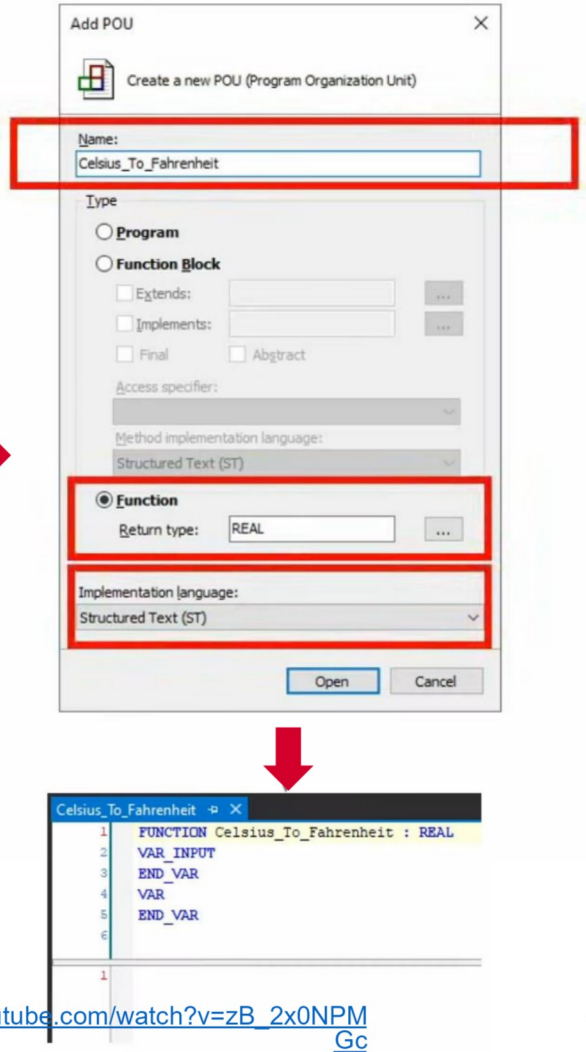
nA	nB
15	5

https://www.youtube.com/watch?v=zB_2x0NPMGc

Functions

- **To create a function:**

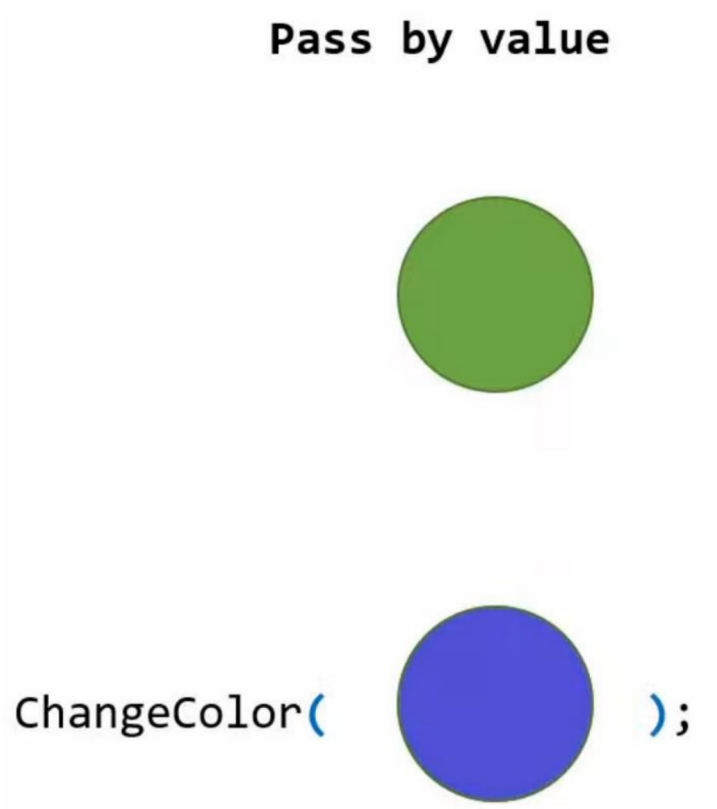


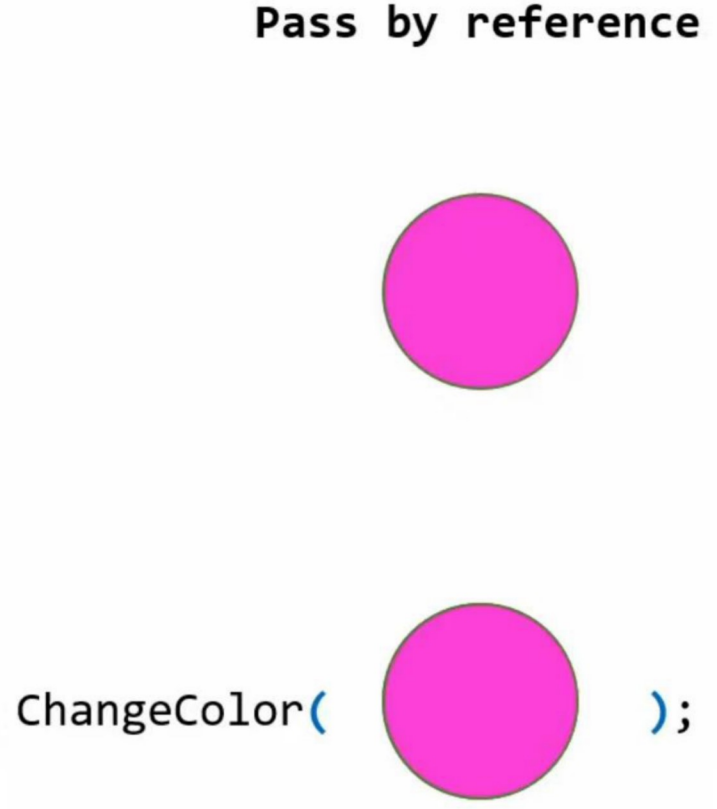


https://www.youtube.com/watch?v=zB_2x0NPMGc

Pass by value & pass by reference

- When passing values into your function, you can do so either by value or by reference





https://www.youtube.com/watch?v=zB_2x0NPMGc

Pass by value & pass by reference

Pass by value

IEC 61131-3

```
FUNCTION UpdateEventTimestampWithSystemTime
VAR_INPUT
    stEvent : ST_Event;
END_VAR
VAR_OUTPUT
    stEventOut : ST_Event;
END_VAR
```

```
stEventOut := stEvent;
stEventOut.dtTimestamp := …(GetSystemTime…)
```

Pass by reference

IEC 61131-3

```
FUNCTION UpdateEventTimestampWithSystemTime
VAR_IN_OUT
    stEvent : ST_Event;
END_VAR
```

```
FUNCTION UpdateEventTimestampWithSystemTime
VAR_INPUT
    stEvent : REFERENCE TO ST_Event;
END_VAR
```

```
stEvent.dtTimestamp := …(GetSystemTime…)
```

https://www.youtube.com/watch?v=zB_2x0NPMGc

Pass by value & pass by reference

- **When should we use pass by value and when should we use pass by reference?**
  - Pass by value means you are making a copy in memory of the parameters value that is passed in a copy of the contents of the actual parameter
  - The penalty for this can get quite extensive if the data that has to be copied is big
- In general:
  1. Pass by **value** when the function does not want to modify the parameter and the value is easy to copy (small parameter → ints, floats, bools, etc. → most a few words)
  2. Pass by **value** when the function does not want to modify the parameter and the value is expensive to copy (structure events example etc.)
  3. Pass by **reference** when the function does want to modify the parameter and the value is expensive to copy

UiA

https://www.youtube.com/watch?v=zB_2x0NPMGc

Part III: Instructions

1. Introduction
2. Conditional statements
3. CASE instruction
4. FOR loops
5. WHILE loops

UiA

Introduction

- **Instructions** are used to control the flow of our program
- **In this part we will go through the following:**
  - **Conditional (IF/ELSE) statements**
  - **CASE instruction**
  - **FOR loops**
  - **WHILE loops**

Conditional statements

- Use **IF** to specify a block of code to be executed, if a specified conditions is true

```
IF condition THEN
    // block of code to be executed if the condition is true
END_IF
```

- Use **ELSE** to specify a block of code to be executed, if the same condition is false

```
IF condition THEN
    // block of code to be executed if the condition is true
ELSE
    // block of code to be executed if the condition is false
END_IF
```

- Use **ELSIF** to specify a new condition to test, if the first condition is false

```
IF condition1 THEN
    // block of code to be executed if condition1 is true
ELSIF condition2 THEN
    // block of code to be executed if condition1 is false and condition2 is true
ELSE
    // block of code to be executed if condition1 is false and condition2 is false
END_IF
```

- Use **CASE** to specify many alternative blocks of code to be executed

UiA

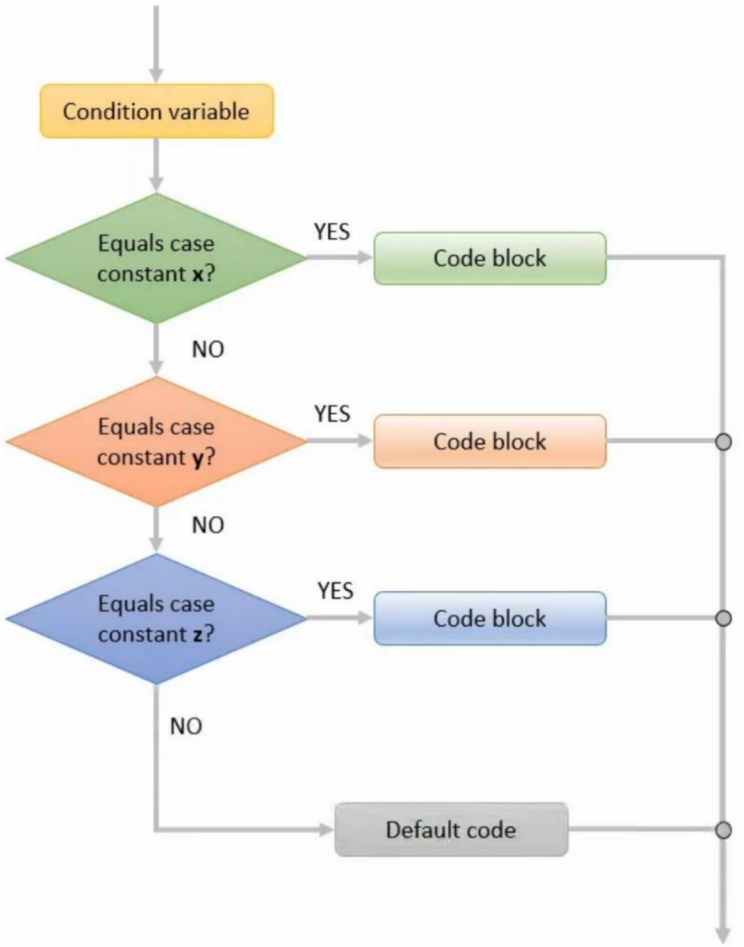
https://www.youtube.com/watch?v=pnLgRNb09qE&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=8&t=1s

CASE instruction

- The **CASE** statement is used to perform different action based on a condition variable

```
CASE variable OF
    x :
        // block of code
    y :
        // block of code
    z :
        // block of code
ELSE
    // block of code
END_CASE
```

- The value of the variable is compared with the value of each case
- If there is a match, the associated block of code is executed
- If there is no match the **ELSE** code block is executed.



https://www.youtube.com/watch?v=pnLgRNb09qE&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=8&t=1s

FOR loops

- **FOR**-loops can execute a block of code a number of times

Instead of writing:

```
VAR
  nSum : INT;
  aArray : ARRAY[1..5] OF INT;
END_VAR
```

```
nSum := nSum + aArray[1];
nSum := nSum + aArray[2];
nSum := nSum + aArray[3];
nSum := nSum + aArray[4];
nSum := nSum + aArray[5];
```

You can write:

```
VAR
  nSum : INT;
  nCounter : INT;
  aArray : ARRAY[1..5] OF INT;
END_VAR
```

```
FOR nCounter := 1 TO 5 BY 1 DO
    nSum := nSum + aArray[nCounter];
END_FOR
```

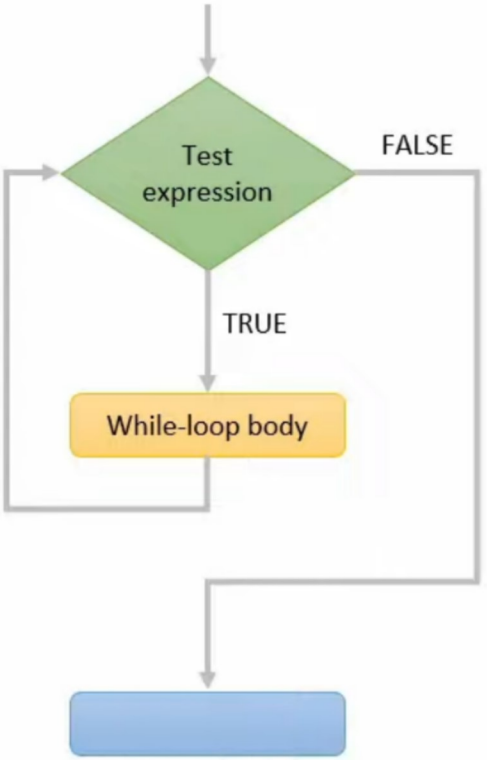
Optional

https://www.youtube.com/watch?v=pnLgRNb09qE&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=8&t=1s

WHILE loops

- The **WHILE**-loop loops through a block of code as long as a specified condition is true

```
VAR
   nSum : INT;
   nCounter : INT := 1;
   aArray : ARRAY[1..5] OF INT;
END_VAR

WHILE nCounter < 6 DO
    nSum := nSum + aArray[nCounter];
    nCounter := nCounter + 1;
END_WHILE

// block of code
```



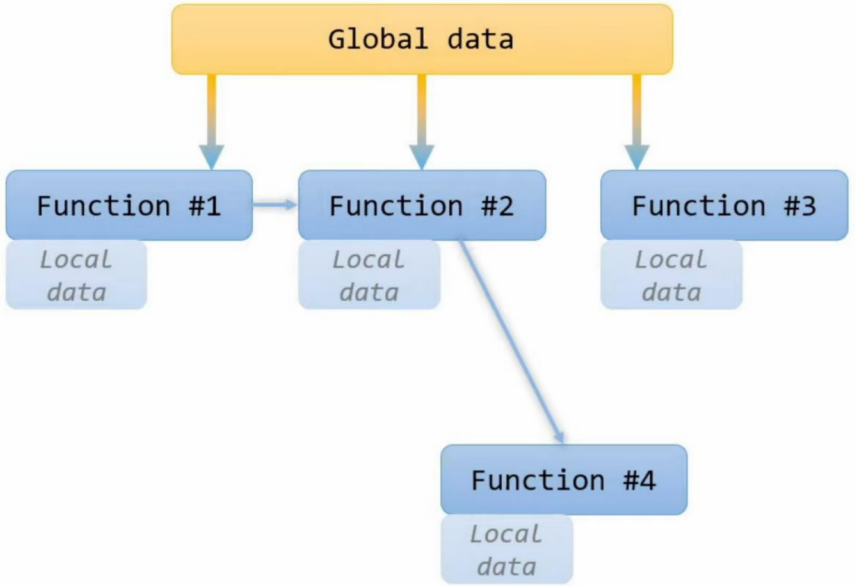
https://www.youtube.com/watch?v=pnLgRNb09qE&list=PLimaF0nZKYHz3I3kFP4myaAYjmYk1SowO&index=8&t=1s

Summary

UiA

Summary

- In **Part I** you have learned how to allocate memory in the PLC and how to use some of the most popular function blocks from the standard PLC library
- In **Part II** we looked into how we can organize data into structures and actually make something useful with that data using functions
- In **Part III** we looked into **Interfaces**
  - Conditional statements
  - CASE instruction
  - FOR loops
  - WHILE loops



Next Lecture

Object oriented PLC Programming:

I. **Presentation of application**
II. **Function Blocks**
III. **Interfaces**

- **Homework:**
  - PLC programming using TwinCAT 3 - Function blocks & interfaces (Part 6a/18)
  - PLC programming using TwinCAT 3 - Function blocks & interfaces (Part 6b/18)

Januar 2024

Uke	Ma	Ti	On	To	Fr	Lø	Sø
1	1	2	3	4	5	6	7
2	8	9	10	11	12	13	14
3	15	16	17	18	19	20	21
4	22	23	24	25	26	27	28
5	29	30	31				

1.1: 1. nyttårsdag

Februar 2024

Uke	Ma	Ti	On	To	Fr	Lø	Sø
5				1	2	3	4
6	5	6	7	8	9	10	11
7	12	13	14	15	16	17	18
8	19	20	21	22	23	24	25
9	26	27	28	29			

Mars 2024

Uke	Ma	Ti	On	To	Fr	Lø	Sø
9					1	2	3
10	4	5	6	7	8	9	10
11	11	12	13	14	15	16	17
12	18	19	20	21	22	23	24
13	25	26	27	28	29	30	31

24.3: Palmesøndag, 28.3: Skjærtorsdag, 29.3: Langfredag, 31.3: 1. påskedag

April 2024

Uke	Ma	Ti	On	To	Fr	Lø	Sø
14	1	2	3	4	5	6	7
15	8	9	10	11	12	13	14
16	15	16	17	18	19	20	21
17	22	23	24	25	26	27	28
18	29	30					

1.4: 2. påskedag

Mai 2024

Uke	Ma	Ti	On	To	Fr	Lø	Sø
18			1	2	3	4	5
19	6	7	8	9	10	11	12
20	13	14	15	16	17	18	19
21	20	21	22	23	24	25	26
22	27	28	29	30	31		

1.5: Offentlig høytidsdag, 9.5: Kristi Himmelfartsdag, 17.5: Grunnlovsdag, 19.5: 1. pinsedag, 20.5: 2. pinsedag

Juni 2024

Uke	Ma	Ti	On	To	Fr	Lø	Sø
22						1	2
23	3	4	5	6	7	8	9
24	10	11	12	13	14	15	16
25	17	18	19	20	21	22	23
26	24	25	26	27	28	29	30

Self-study | Part #1 | Part #2 | Part #3 | Exam

Lab exercise

Lab Exercise #2.2 – Procedural-oriented PLC programming