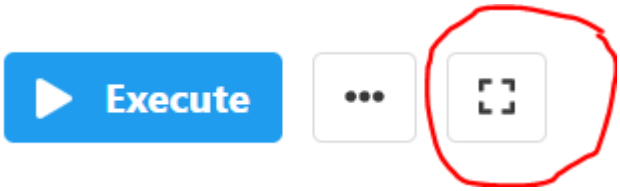
COBOL-Beginners-Course

A beginner's course in COBOL

Getting started

Open jdoodle for COBOL (right click and select Open link in new tab).

Click on the square



That gives a better layout.

Some words about COBOL

jdoodle gives us some COBOL code to start from.
Often when writing a new program, you don't start from scratch. At SEB, we generate lots of basic code from templates, and then fill in with business logic that makes the program unique.

When designing the COBOL language, it was important that the code should be similar to plain written English.
Therefore, the language is divided into divisions, sections, paragraphs and sentences.
Can you see DIVISION and SECTION somewhere in the code?
There are no paragraphs in this code, but every code line ends with a dot – sentences.
(At SEB, we only have a few special paragraphs and we do NOT end each code line with a dot.)

It's meant to be easy to read COBOL code.
As an example, code line: **ADD X Y GIVING Z.**

Even if you have never seen COBOL or any other code before, **can you guess what the program does?**
Click on the blue Execute button, and the program will run/execute.
What results did you get in the black Result box?

```
1  IDENTIFICATION DIVISION.
2  PROGRAM-ID. HELLO-WORLD.
3  DATA DIVISION.
4      WORKING-STORAGE SECTION.
5          77 X PIC 99.
6          77 Y PIC 99.
7          77 Z PIC 99.
8  PROCEDURE DIVISION.
9      SET X TO 10.
10     SET Y TO 25.
11     ADD X Y GIVING Z.
12     DISPLAY "X + Y = "Z.
13 STOP RUN.
14
```

WORKING STORAGE and PROCEDURE DIVISION

It says ***WORKING STORAGE*** in the code. Below *you declare / describe your variables*.

A variable is the name of a piece of memory. X PIC 99. means "variable X is two bytes memory with two digits". It describes what a field/value should look like.

PIC 99 means Picture 99 and can contain a numeric value with two digits. Can also be written PIC 9 (2).

PIC XX means Picture XX and can contain an alphanumeric value with two letters, two digits or mixed one letter and digit. Can also be written PIC X (2).

Under ***PROCEDURE DIVISION*** we have the code itself that will do something.
It ends here with **STOP RUN**. which must always remain (including the dot) at the end of the program.

Arithmetic

Remove all code under PROCEDURE DIVISION, except for STOP RUN.

Write or paste the following code instead:

```
MOVE 10 TO X
ADD 1 TO X
SUBTRACT 1 FROM X
MULTIPLY 6 BY X
DIVIDE 3 INTO X
DISPLAY "X="X
```

MOVE is used to move a value to a field. ”Move” value 10 to field X.
ADD, SUBSTRACT, MULTIPLY, DIVIDE, DISPLAY are other COBOL words used to perform various things in the program.

What will X contain when the program has been executed?
Click on the blue Execute button and we will see!

Arithmetic, continue

As we saw above, the result of a mathematical operation can be saved in another variable with `GIVING`.
We try to re-write our DIVIDE like this below PROCEDURE DIVISION:

```
DIVIDE X BY 3 GIVING Y REMAINDER Z
```

And our DISPLAY like this:

```
DISPLAY "X="X " Y="Y " Z="Z
```

What values will X, Y and Z contain? Click on Execute.
What does it mean that Z is zero?

When making more complicated mathematical expressions, you can use **`COMPUTE`**.

It’s possible to replace the entire previous code below PROCEDURE DIVISION with the following and still get the same result:

```
MOVE 10 TO X
COMPUTE Y = (((X + 1) - 1) * 6) / 3
DISPLAY "Y="Y
```

Paste the code above before STOP RUN, below PROCEDURE DIVISION.
Click on Execute. Did you get the same result?

Which code is easiest to read and understand?
Remove the arithmetic code in PROCEDURE DIVISION in preparation for the next task.

Input

We have come bit on our way of understanding of programming. A program becomes more valuable if it can interact with the outside world. The programmer can do this by letting the user give/submit values to the program. Values can be submitted to the program in a file with data, can be selected from a database, registered in a client application or with **ACCEPT**.

`ACCEPT` is used to enter values for the variables in this COBOL program.
But first we must have an appropriate variable. Below WORKING STORAGE, enter the following:

```
01 A-WORKAREA.
   05 A-DATE      PIC X(8).
```

Here we have described/declared a group variable A-ARBETSAREOR containing a single field , A-DATE, which is eight *alphanumeric* characters long (YYYYMMDD). A group variable holds several variables together, a group of variables. This is a good way to sort your variables in this way in COBOL.

```
ACCEPT A-DATE FROM DATE YYYYMMDD
DISPLAY "Today's date: "A-DATE
```

DATE is a built-in function in COBOL that retrieves today's date from the mainframe.
Remove all code under PROCEDURE DIVISION, except for STOP RUN.
Paste the code above before STOP RUN below PROCEDURE DIVISION.
Click on Execute.

Input, continue

Now we are going to retrieve a value from the user of the program. YOU are the user.

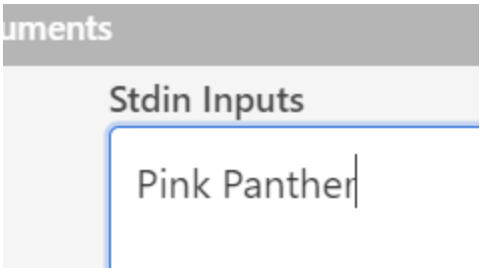
To the group A-WORKAREA below WORKING STORAGE we add:

```
   05 A-NAME      PIC X(40).
```

Write below PROCEDURE DIVISION:

```
ACCEPT A-NAME
DISPLAY A-NAME
```

Before running the program, a name must be entered. This is done in the **Stdin Inputs** box:



Click on Execute!

Add the variable A-AGE in the same way and retrieve the user's age in the code.
How should A-AGE be declared and where should it be placed in the code?
Where should the user enter/indicate their age?

Click on Execute. Have you received the new information?

A real program (kind of..)

Now we are going the write a real program. We will evaluate the user's age and determine if she's old enough to drive a car (at least 18), or not. Then we have to take in information about the user(*input*) and deliver a report(*output*). Enter the following below WORKING STORAGE:

```
01 EVALUATION.
   05 SUBJECT.
      10 FILLER    PIC X(8)  VALUE "REPORT".
      10 DATE-RED PIC 9999/99/99.
      10 FILLER    PIC X(4)  VALUE SPACE.
      10 NAME      PIC X(40).
   05 RESULT.
      10 FILLER    PIC X(25).
         88 CAR-NO  VALUE 'You can NOT drive a car.'.
         88 CAR-YES VALUE 'You can drive a car.'.
      10 FILLER    PIC X(45).
         88 SYS-NO  VALUE 'You are NOT allowed to shop at Systembolaget.'.
         88 SYS-YES VALUE 'You are allowed to shop at Systembolaget'.
```

We use the group "FILLER" in our report. "FILLER" takes up memory but cannot be manipulated. Which can be quite useful.

"88" is a real COBOL special. A function that allows you to switch between different values.
For example, between TRUE and FALSE.

In COBOL, as in all programming languages, you can choose the path in the program depending on the *conditions* with ***IF-THEN-ELSE***. It can look like this:

Add the following (IF to END-IF) to PROCEDURE DIVISION

```
IF A-AGE >= 18 THEN *> This is a comment regarding that ">=" means "greater than or equal to.
   SET CAR-YES TO TRUE
ELSE
   SET CAR-NO TO TRUE
END-IF
```

If you want to write little *less*, you can exclude "THEN", it still works.
If you want to write a little *more*, you can instead write the same thing like this:

```
IF A-AGE IS GREATER THAN OR EQUAL TO 18 THEN
```

Also add this to the code below PROCEDURE DIVISION:

```
MOVE A-DATE TO DATE-RED
MOVE A-NAME TO NAME
DISPLAY SUBJECT
DISPLAY RESULT
```

Click on Execute. How did it go?

A real program(kind of..), continue

Add to the program to also be able to report if the user is old enough to shop at Systembolaget.
Extra exercise: Can you add an IF to the IF we already have, to be able to report this correctly?

Having IF in IF is called *nested IF*. In this case, it may not be that complicated, but it can quickly become a bit messy and difficult to read code if you have several things to keep track of in your program.
As in other languages (where it may be called *switch* or *select*) you can "choose path" in COBOL depending on the value of a variable, such as A-AGE.
In COBOL you do this with **EVALUATE**:

```
EVALUATE A-AGE
WHEN 0 THRU 17
   SET CAR-NO  TO TRUE
   SET SYS-NO  TO TRUE
WHEN 18
WHEN 19
   SET CAR-YES TO TRUE *> We come here for both 18 and 19!
   SET SYS-NO  TO TRUE
WHEN OTHER
   SET CAR-YES TO TRUE
   SET SYS-YES TO TRUE
END-EVALUATE
```

EVALUATE in COBOL is powerful. If you write `EVALUATE TRUE` you can use optional conditions with optional variables.

```
EVALUATE TRUE
WHEN A-NAME = 'Pink Panther'
   MOVE 'You are Pink Panther! You are above the law!' TO RESULT
WHEN A-AGE < 18
   SET CAR-NO  TO TRUE
   SET SYS-NO  TO TRUE
WHEN A-AGE >= 18 AND < 20
   SET CAR-YES  TO TRUE
   SET SYS-NO   TO TRUE
WHEN OTHER
   SET CAR-YES  TO TRUE
   SET SYS-YES  TO TRUE
END-EVALUATE
```

Try your nested IF or replace the IF with EVALUATE as above. **Click on Execute.**
Try to change age and name so that you get different output in your report.

PERFORM, LENGTH OF and Reference Modification

Now we will try to print the user's name *vertically* after the report. Then we will loop/spin through the entire variable and print one character at a time on its own line. **How big is the memory for a variable?**
This can of course be seen in WORKING STORAGE, but we want to write robust code that works even if the variable declaration should change. Therefore, we use the **LENGTH OF variable** which gives the number of bytes of memory that the variable has. We can test this with this line of code below PROCEDURE DIVISION:

```
DISPLAY LENGTH OF A-NAME
```

Click on Execute. What happened?

If you want to access a few characters in a string, you can specify it by adding a parenthesis to the variable and pointing out which characters you want to work with, like this:

```
DISPLAY A-NAME(3:5)
```

Which in this example means that we print characters number 3-7 of NAME. We start with the third character and take a total of five characters.

If you want to get hold of all characters from a position X, you write variable (X:).
This is called *Reference Modification* in *COBOL*.

Note that in COBOL we start *indexing* with 1 and not 0 as in many other languages.
To create a loop in COBOL, **PERFORM** is used. But where you in other languages loop *as long as* a condition is met, in COBOL you loop *until* a condition becomes true. So not "while" but instead "UNTIL" i.e.
`PERFORM UNTIL X > LENGTH OF NAMN.` **What does it mean?**

So now we want to step forward the variable X, which is our position in NAME, and print the character there. We do that with the following code:

```
PERFORM VARYING X FROM 1 BY 1 UNTIL X > LENGTH OF A-NAME
   IF A-NAME(X:) = SPACE *> If the rest of A-NAME is empty ...
      EXIT PERFORM     *> ... then we jump out of the loop
   END-IF
   DISPLAY A-NAME(X:1)   *> We print a character in A-NAME
END-PERFORM
```

FizzBuzz (extra exercise)

Now you have learned enough to solve the classic programming nut FizzBuzz.

Let the user enter any (positive) integer, N. For all n=1,2,3….,N print n. But, if n is evenly divisible by 3, print "Fizz". If n is evenly divisible by 5 print "Buzz". If n is evenly divisible by *both* 3 and 5, print "FizzBuzz". It should look like this for N=15:

```
01
02
Fizz
04
Buzz
Fizz
07
08
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
```