ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Python in practice

Lesson 3: Sequence, selection, iteration

Semester 2021/22/2

Subject matter

1. Introduction
2. Variables, operators
3. Sequence, selection, iteration
4. Programming theses
5. Strings
6. Regular expressions
7. Files
8. Object-oriented programming
9. Multithread applications
10. GUI with tkinter
11. Communications
12. SQL basics
13. SQL queries
14. Example application

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Python in practice

Booleans & Statements

Logical values

- True or False
  - Represented by 1 bit; with values 0,1
- Informs about the thruth of a statement.
  - Example:

„This slide has a blue background.”	→	✓ True
„This text is written in red.”	→	✗ False
`10 > 2`	→	✓ True
`math.pi > math.e`	→	✓ True
`math.sin(x) < -1`	→	✗ False
`math.sin(math.pi/2) => 1`	→	✓ True
`5 + 5 == 10`	→	✓ True
`'A' >= 0x42`	→	✗ False
`len('Hello\n') == 6`	→	✓ True

`bool_var == True`

ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Statements

- Expressions can result in any data, even logical values.
- Expressions can be used in statements.
- The result of claims is always their truth.
- Statements may be included in expressions.

- Statements can also be made about the logical relationship of part-statements.
- Operatorations:
  - NOT (negation, inversion):
    - Gives the opposite of the truth of a statement
  - AND (conjunction):
    - It is only true if all the statements are true.
  - OR (disjunkction):
    - It is true if at least one of the statements is true

Statements

- Number of value set items: 2 (True or False)
- Number of combinations: 4
- Every combination has a result. → Number of results: 4
- The number of binary numbers that can be represented in 4 bits: 16

- There are 16 different 2-variable functions.
- The more complex functions can all be simplified to these 16.
- All 16 functions are made up of the 3 operations learned.

- The functions are identified by summing the results for all combinations to the corresponding numerical value

Logical relationships

		Never function	Negated disjunction	Inhibition	Negation	Inhibition	Negation	eXclusive OR Antivalence	Negated conjunction	AND Conjunction	XNOR Equivalence	Repeating function	Implication	Repeating function	Implication	OR Disjunction	Always function
$\boldsymbol{A}$	$\boldsymbol{B}$	$\boldsymbol{Y_0} = 0$	$\boldsymbol{Y_1} = \overline{A+B}$	$\boldsymbol{Y_2} = \bar{A}B$	$\boldsymbol{Y_3} = \bar{A}$	$\boldsymbol{Y_4} = A\bar{B}$	$\boldsymbol{Y_5} = \bar{B}$	$\boldsymbol{Y_6} = \bar{A}B + A\bar{B}$	$\boldsymbol{Y_7} = \overline{AB}$	$\boldsymbol{Y_8} = AB$	$\boldsymbol{Y_9} = AB + \bar{A}\bar{B}$	$\boldsymbol{Y_{10}} = B$	$\boldsymbol{Y_{11}} = \bar{A} + B$	$\boldsymbol{Y_{12}} = A$	$\boldsymbol{Y_{13}} = A + \bar{B}$	$\boldsymbol{Y_{14}} = A + B$	$\boldsymbol{Y_{15}} = 1$
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
		↑	↑		↑		↑	↑	↑	↑	↑	↑		↑		↑	↑

Operator precedence

First

Operator	Description
(expressions...), [expressions...], {key: value...}, {expressions...}	Binding or parenthesized expression, list display, dictionary display, set display
x[index], x[index:index], x(arguments...), x.attribute	Subscription, slicing, call, attribute reference
await x	Await expression
**	Exponentiation 5
+x, -x, ~x	Positive, negative, bitwise NOT
*, @, /, //, %	Multiplication, matrix multiplication, division, floor division, remainder 6
+, -	Addition and subtraction
<<, >>	Shifts
&	Bitwise AND
^	Bitwise XOR
|	Bitwise OR
in, not in, is, is not, <, <=, >, >=, !=, ==	Comparisons, including membership tests and identity tests
not x	Boolean NOT
and	Boolean AND
or	Boolean OR
if – else	Conditional expression
lambda	Lambda expression
:=	Assignment expression

Last

Python in practice

Control flow

Control flow

- The order in which instructions are executed.
- Categories:
    - **Sequential**
    - Jump
    - **Choice/selection** - Conditional branch
    - **Loop/iteration** – Executing until some condition is met
    - Subroutines
    - Stopping the program

- Representations in documentation:
    - Flow chart: a graph of nodes and directed edges
        - illustrates the path of the control structures
    - Structogram: rectangles under one another in sequence or nested
        - illustrates the structure of the source code

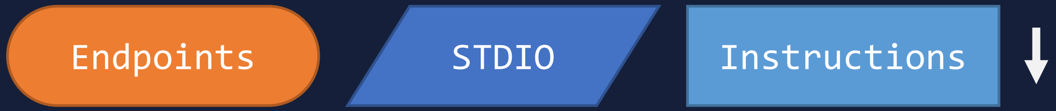
ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Sequential execution

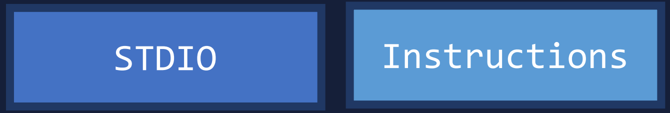
- Listing and executing instructions in the order listed.
- Linear programs

- Most common cases:
  - Reading and writing the standard streams
  - Assignment expressions (=)

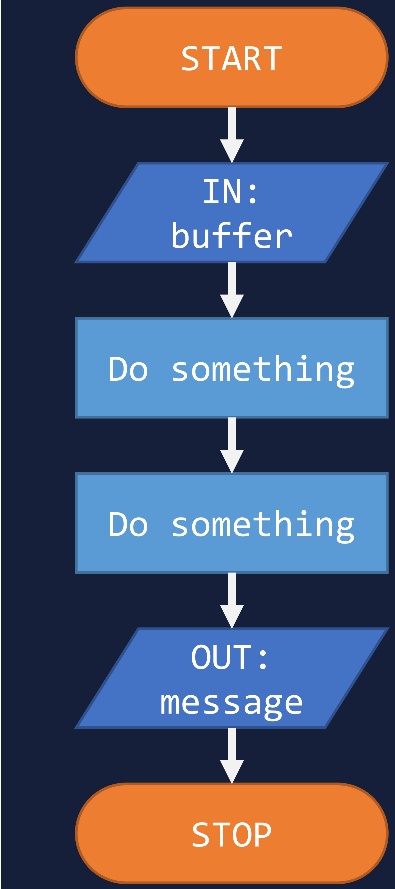
- Representation on a flowchart:



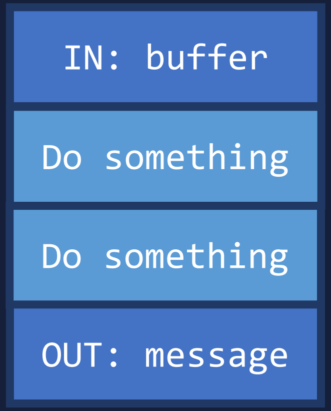
- Representation on a structogram:



Flowchart:



Structogram:

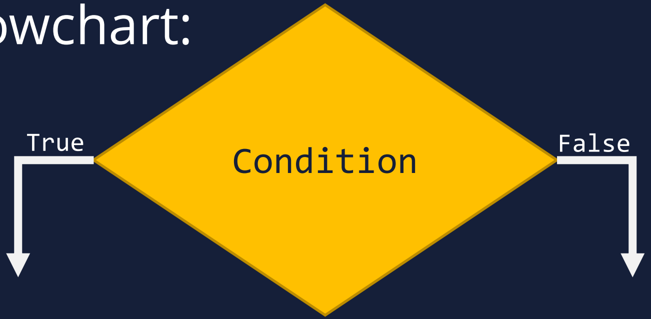


ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

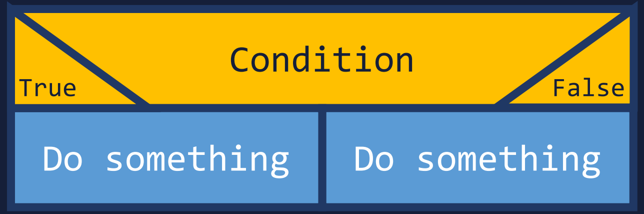
Selection

- Programmers can use two or more directional branches in a program.
- A clear condition must be met to select the appropriate branch.
- After evaluating the truth of the statement, exactly one branch may be valid. ‼

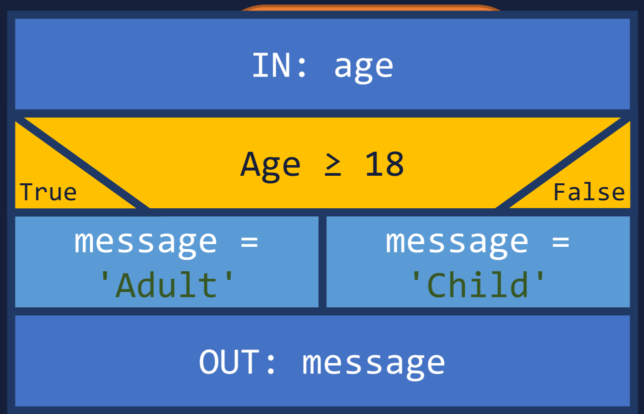
- Representation on a flowchart:

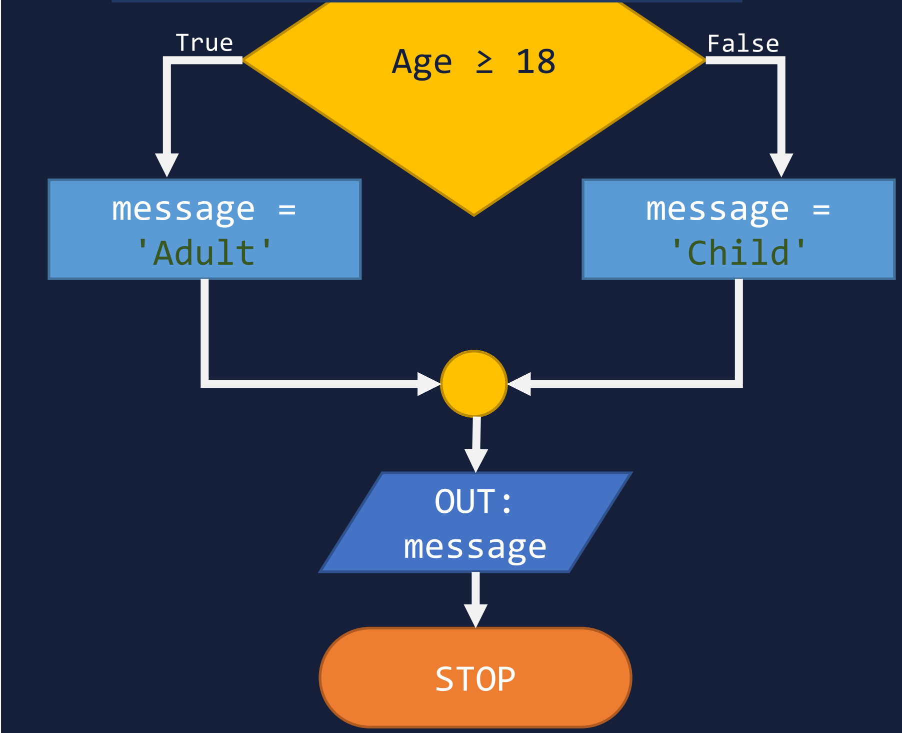


- Representation on a structogram:



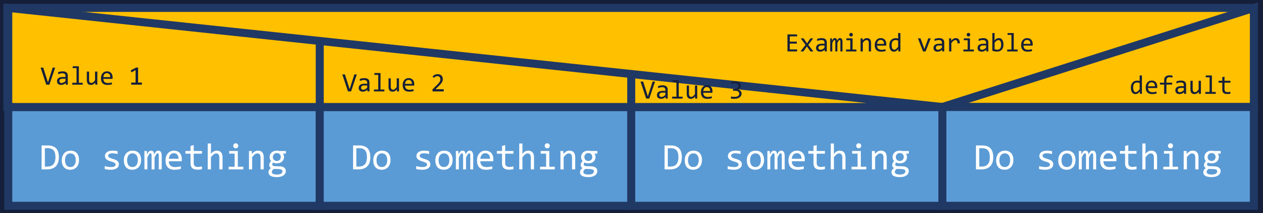
Structogram:
Flowchart:

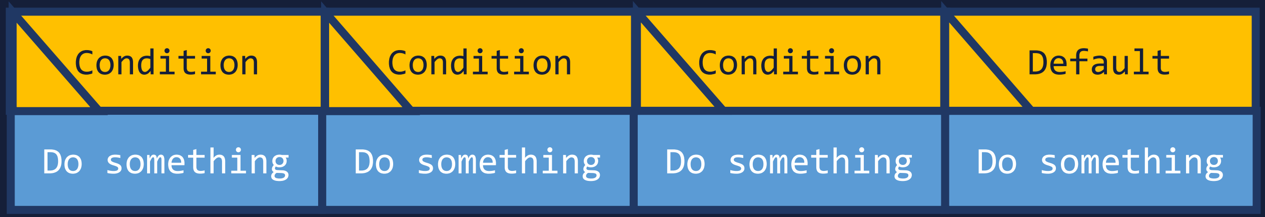




Multidirectional branching

- Selects the appropriate branch by listing the relevant values.
- Can only be used to test equality with fixed values. Cannot be used to examine relations and belonging to domains. ‼
- Common uses:
  - in computer games to distinguish the keys pressed by the player 🎮
  - it is also used by the interpreter when calculating expressions to identify the function of operators
- It cannot be represented in a flowchart, but it can be represented in two ways in a structogram:

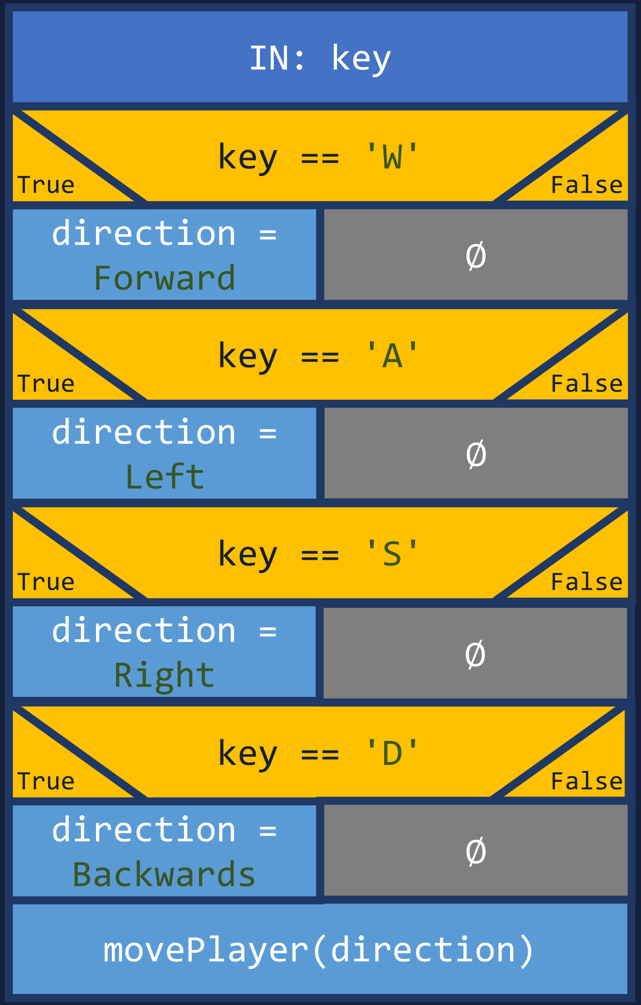




ÓBUDAI EGYETEM
ÓBUDA UNIVERSITY

Multidirectional branching

- C-like languages:
  - multidirectional branches → lookup table in memory.
  - the structure used in the code is called the "switch..case" branch
- Python:
  - Python does not have such a structure, instead it uses multiple bidirectional branches in a sequence. ‼



IN: key

key == 'W'
True | False
direction = Forward | ∅

key == 'A'
True | False
direction = Left | ∅

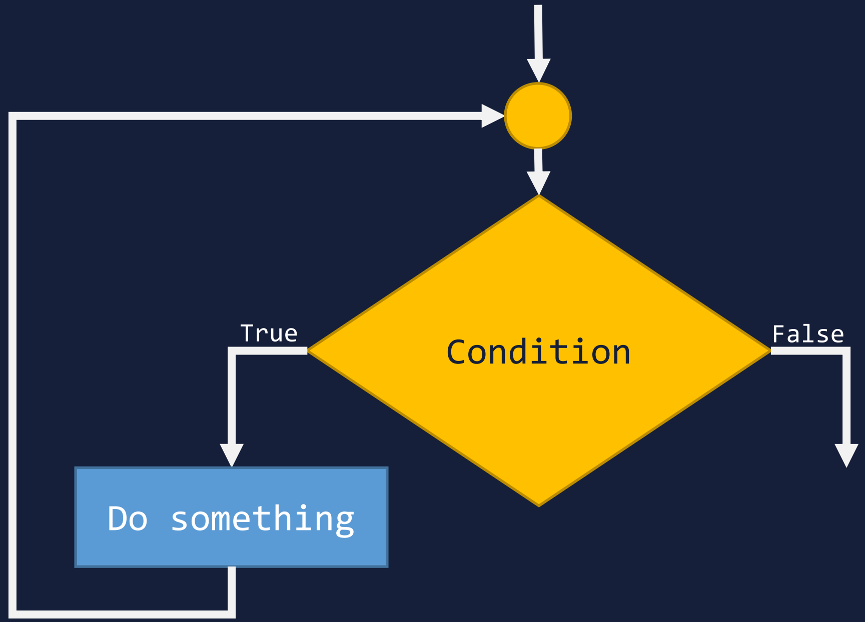
key == 'S'
True | False
direction = Right | ∅

key == 'D'
True | False
direction = Backwards | ∅

movePlayer(direction)

Iteration

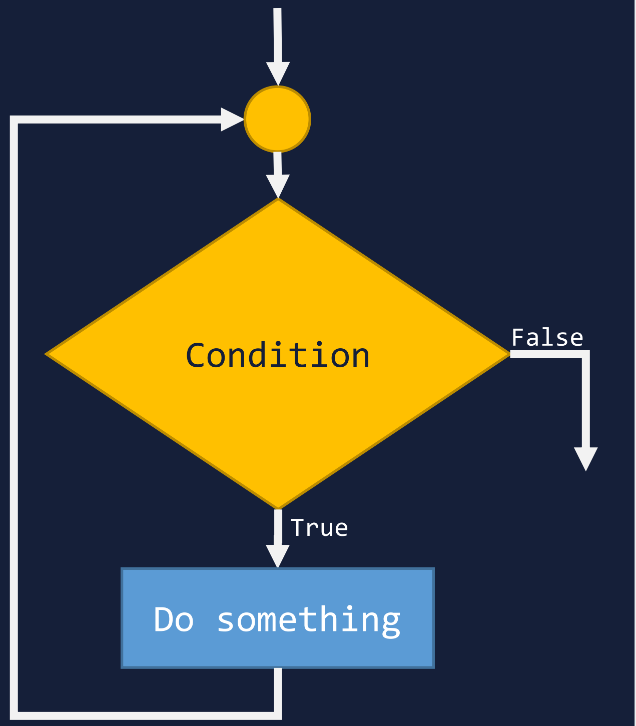
- An instruction at a branch can even be a JUMP that points to a previously run part of the program, which is thus executed again.
- A clear condition must be met to repeat the execution of the program snippet.



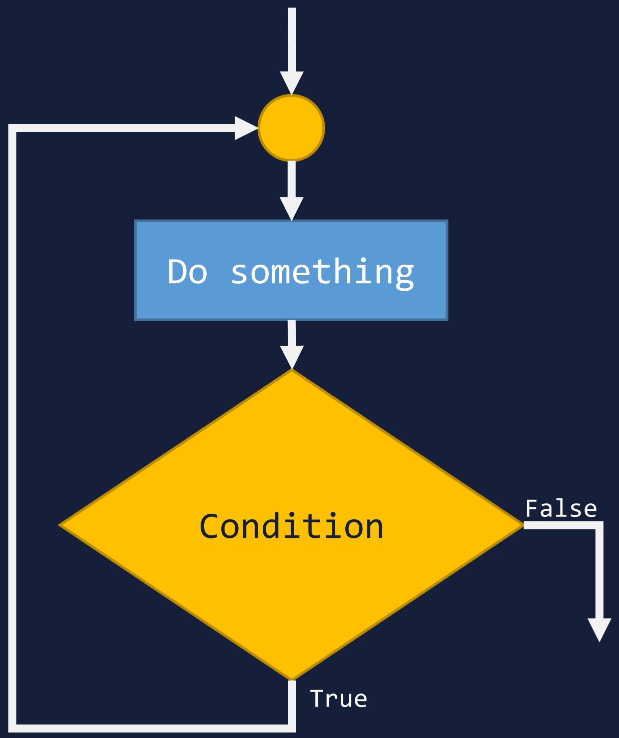
Iteration

- Types:
    - Pre-test loop:
        - the loop condition is tested before entering the loop.
    - Post-test loop:
        - the test of the loop condition occurs after the body of the loop has been carried out one time
        - the body of the loop are always executed at least one time
    - For loop:
        - Repeats actions a pre-determined number of times

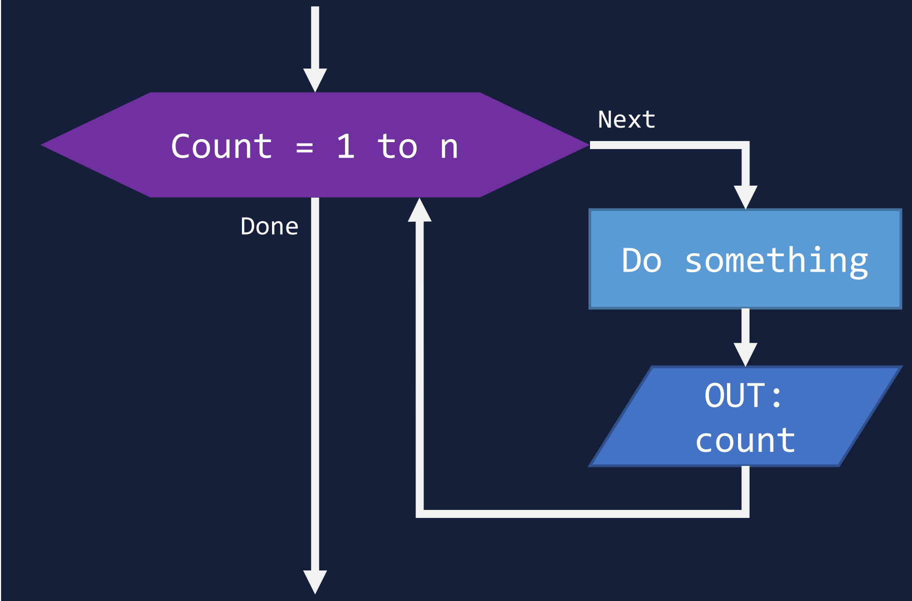
Loops in a flowchart



Pre-test loop

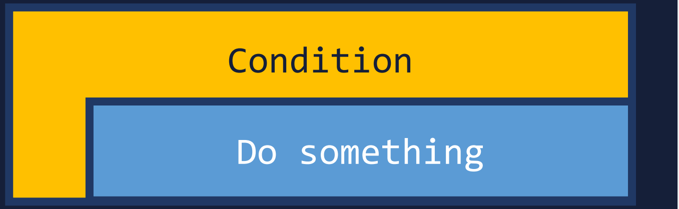


Post-test loop



For loop

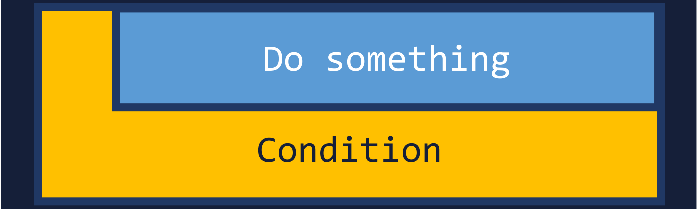
Loops in a structogram



Condition

Do something

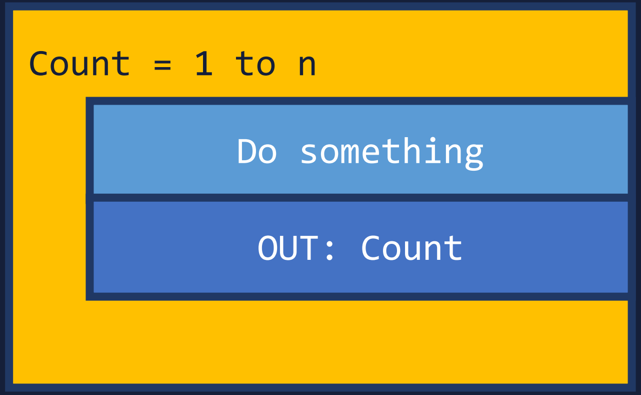
Pre-test loop



Do something

Condition

Post-test loop



Count = 1 to n

Do something

OUT: Count

For loop