LABORATOIRE
D'INFORMATIQUE
INTELLIGENTE
BISKRA

COMPUTER SCIENCE 1

DR. ADEL ABDELLI

adel.abdelli@univ-biskra.dz

2024/2025

CHAPTER 2

ALGORITHM AND PROGRAM CONCEPTS

DR. ADEL ABDELLI

01/12/2024

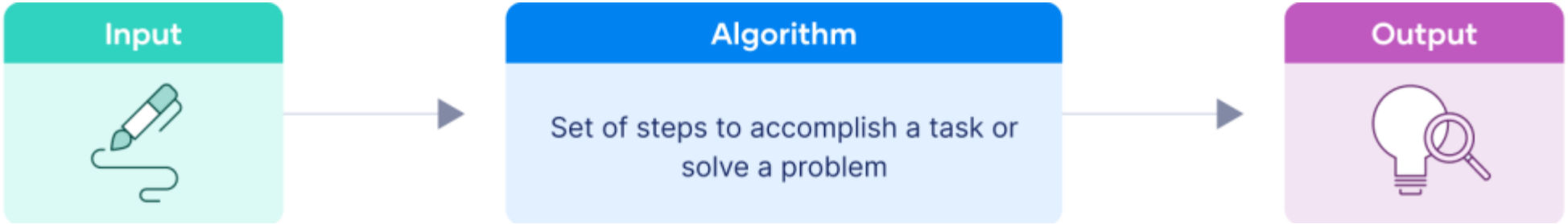
1. What's an Algorithm, Program, Programming language?

Definition 1: An algorithm is a finite sequence of elementary actions that allows solving a problem.

Definition 2: An algorithm is the result of decomposing a complex problem into elementary operations to be executed in several successive steps.

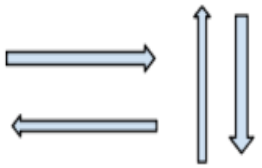
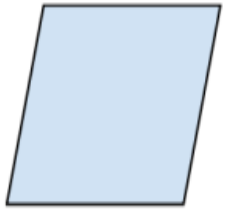
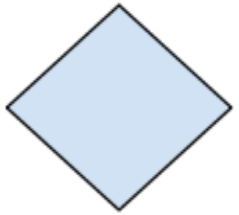
Definition 3: Program – It refers to the code (written by programmers) for any program that follows the basic rules of the concerned programming language.

Definition 4: programming language is a set of instructions for a computer to follow to perform a task.

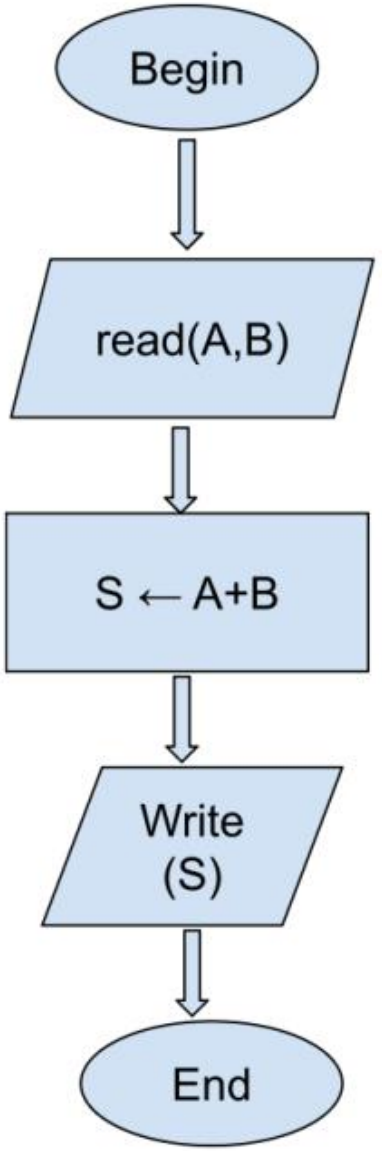


2. Flowchart Representation

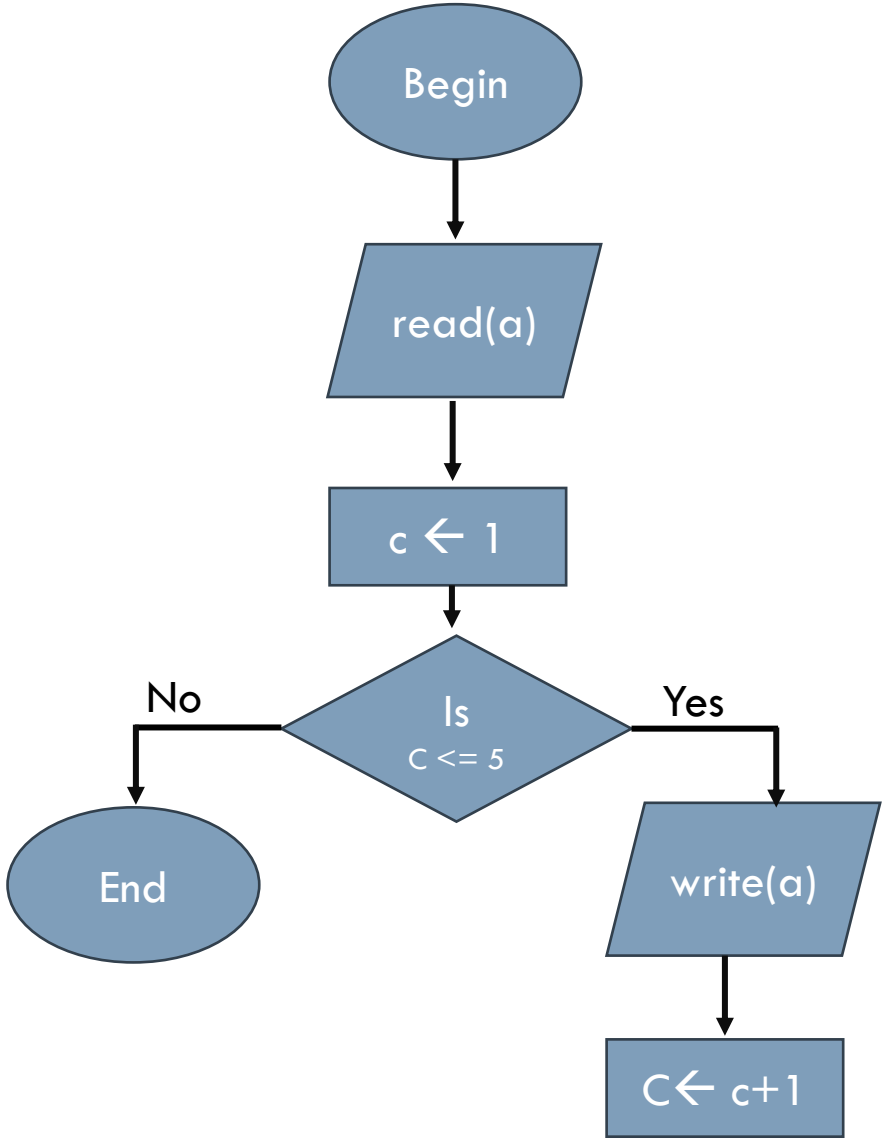
A flowchart is a conventional symbolic diagram that illustrates the steps of an algorithm and their relationships. We use flowcharts because a graphical representation helps with understanding. The flowchart is a functional diagram that presents the different parts of a program one after the other using graphic symbols to visualize the program's execution and the flow of data.

Name	Symbole	Definition
arrows		They indicate the direction of processing (up, down, left, right)
Begin / end		This symbol indicates the beginning or end of the flowchart
input/output		This symbol indicates input and output data
Treatment box		It indicates a specific treatment that can be performed
Test- decision box		It allows processing to be sent on one path or another, depending on the test result.

Examples:



Example 1: addition of two numbers



Example 2: showing a number “a” “n” times

3. The structure of an algorithm:

An algorithm is composed of 3 parts:

Algorithm Name; ⟶ header part: Is introduced by the word Algorithm followed by an algorithm name.

<Declaration part> ⟶ It declares all the objects used in the action part. It includes: constant and variables with their types and also functions and procedures

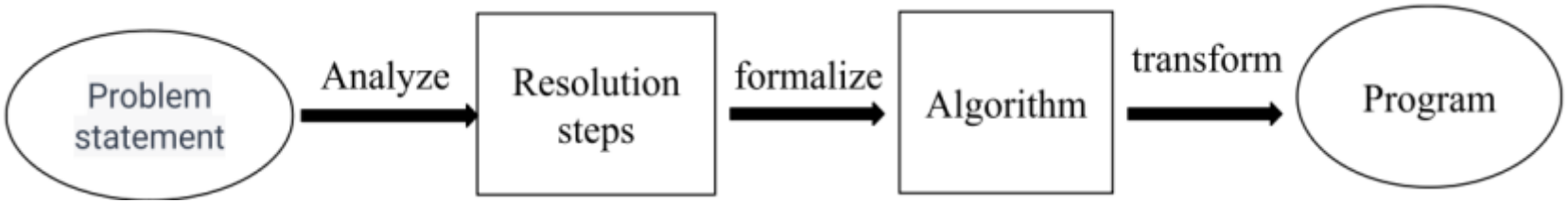
Begin

<action part> ⟶ here we specify the actions to be executed on the objects defined in the declaration part.

End

4. Steps in Problem Analysis

The analysis of a problem is carried out according to the following scheme:



Example Problem Statement: Write an algorithm that calculates the area of a rectangle.

Problem Analysis:

- Desired result: the area of a rectangle
- Required data: length and width
- Calculation rule: area = width * length

The Algorithm:

1. Read the width
2. Read the length
3. Area = width * length
4. Display the result (the area)

5. Data Structure:
5.1 Constants and variables:

Every object has three characteristics:

1. The name
2. The type
3. The value

- Each data (variable or constant) manipulated by an algorithm is designated by a unique name: Identifier.

- It is an alphanumeric string (containing only alphabetical characters [a-z, A-Z] and numbers [0-9]) in addition to the underscore character "_" (underscore), and it does not begin with a numeric character.

- An identifier is assigned to a single object. One can never use the same identifier for two different variables or constants.

5. Data Structure:
5.1 Constants and variables:

Example:
Among the following identifiers, indicate which ones are valid and which ones are not:

12x: Not valid (starts with a numeric character)

Total Price: Not valid (contains a space)

wall_height: Valid (contains allowed characters)

a1: Valid (alphanumeric)

a&b: Not valid (contains an invalid character “&”)

5.1.1 Constants:

A constant is an object with a fixed name, a fixed type, and a fixed value.

Declaration of constants:

Const constant_name = value

Example:

Const pi = 3.14

5.1.2 Variables:

A variable is an object with a fixed name, a fixed type, and a variable value.

Declaration of variables:

Var variable_name: type
Var a,b: integer
x: real

5.2 Elementary Data Types:

Elementary types are simple types with a single value. We distinguish the following types: integer, real, boolean, character, and string.

A type denotes the set of allowed values for an object and the set of operators allowed on objects of that type.

5.2.1 Integer Type:

Designates integer numbers. (e.g., 2, +2, -30), positive or negative; an integer occupies four bytes (32 bits) in memory.

5.2.2 Real Type:

Designates real numbers. (e.g., 2.5, +12.55, 2E5), a real number occupies eight bytes (64 bits) in memory.

5.1.2 Variables:

5.2.3 Boolean Type (Logical):

A variable of this type can contain one of the logical values true or false. A boolean occupies one byte (8 bits) in memory.

5.2.4 Character Type:

Designates the set of uppercase and lowercase alphabetical letters, numbers, special characters (e.g., '?', '!', ')', '*', ...), and the space character. A character occupies one byte (8 bits) in memory.

5.2.5 String Type:

Designates the set of character strings that we can form by grouping characters. (e.g., 'night', 'Color').

6.Operators:

Here are the operators used in algorithmic syntax categorized into assignment, arithmetic, relational, and logical.

6.1 Assignment Operators:

‘←‘ (Assignment)

6.2 Arithmetic Operators:

‘+‘ (Addition)
‘-‘ (Subtraction)
‘*‘ (Multiplication)
‘/‘ (Division)

6.Operators:

6.3 Relational Operators:

‘==‘ (Equal to)
'!=‘ (Not equal to)
‘<‘ (Less than)
‘>‘ (Greater than)
‘<=‘ (Less than or equal to)
‘>=‘ (Greater than or equal to)

6.4 Logical Operators:

‘&&‘ (Logical AND)
‘||‘ (Logical OR)
‘!‘ (Logical NOT)

7 Basic Actions (Input/Output/Assignment):

7.1 Assignment:

This action allows us to assign a value resulting from the evaluation of an expression E to a variable V.

The type of the expression must be compatible with the type of the variable V. This action is denoted by:

$V \leftarrow E$

Which means evaluate E and store the result in the memory location called V.

Example: $C \leftarrow A + B$

7.2 Read operation

This action allows us to input data using the keyboard. It is denoted by:

Read(x)

Which means place the data entered on the keyboard into the memory location x.

7.3 Write operation:

This action allows us to communicate a result or a message to the user via the screen. It is denoted by:

Write(E)

Which means evaluate the expression E and display the result.

Algorithm 1: Addition

```
var  A, B, C: integer
begin
read (A)
read (B)
C ← A + B
write ('the addition result is',C )
end
```

Example Addition of two integers.

Exercise:

Provide the step-by-step execution of this algorithm:

```
Algorithm algo
Var A, B: integer
Begin
A ← 5
B ← 9
Write('A+B')
Write(A+B)
end
```

Solution:

It will display on the screen 'A+B' and then 14.

Translation of algorithms into Python programs:

Translating algorithms into Python programs involves expressing the logical steps of the algorithm in a way that the Python interpreter can understand and execute. Here's a general guide on differences between Python and algorithm syntax:

Algorithm	python
contain header part contain the keyword **Algorithm** followed by the name of algorithm	no header part

contain declaration part	Python is dynamically typed, so we don't need to declare variable types explicitly.
	x =5 # x is an integer
	x = "hello" # Now x is a string
	x = 3.14 # Now x is a float

read ()	input ()
write()	print ()
Assignment A <– 5	A=5
Types Integer Real string boolean	 int float str bool
Operations mod div exp	 % // **

What's Python ?

Interactive
interact with the interpreter directly to write your programs

High-level
Humain-friendly: very easy to understand and learn

Interpreted
implementations execute instructions directly and freely, without previously compiling a program into machine-language instructions

Dynamically-typed
where the interpreter assigns variables a type at runtime based on the variable's value at the time

Garbage-collected
(automatic mangement Of memory – allocation & free up memory)

Object-oriented
technique of programming that encapsulates code within objects

Python features

Portable
run on a wide variety of hardware platforms

GUI Programming

Easy-to-learn

Scalable

Interactive Mode
allows interactive testing and debugging of snippets of code

Easy-to-read

Extendable
You can add low-level modules to the Python interpreter

A broad standard library

Easy-to-maintain

Databases

Python Variable Types

• Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.

0 s

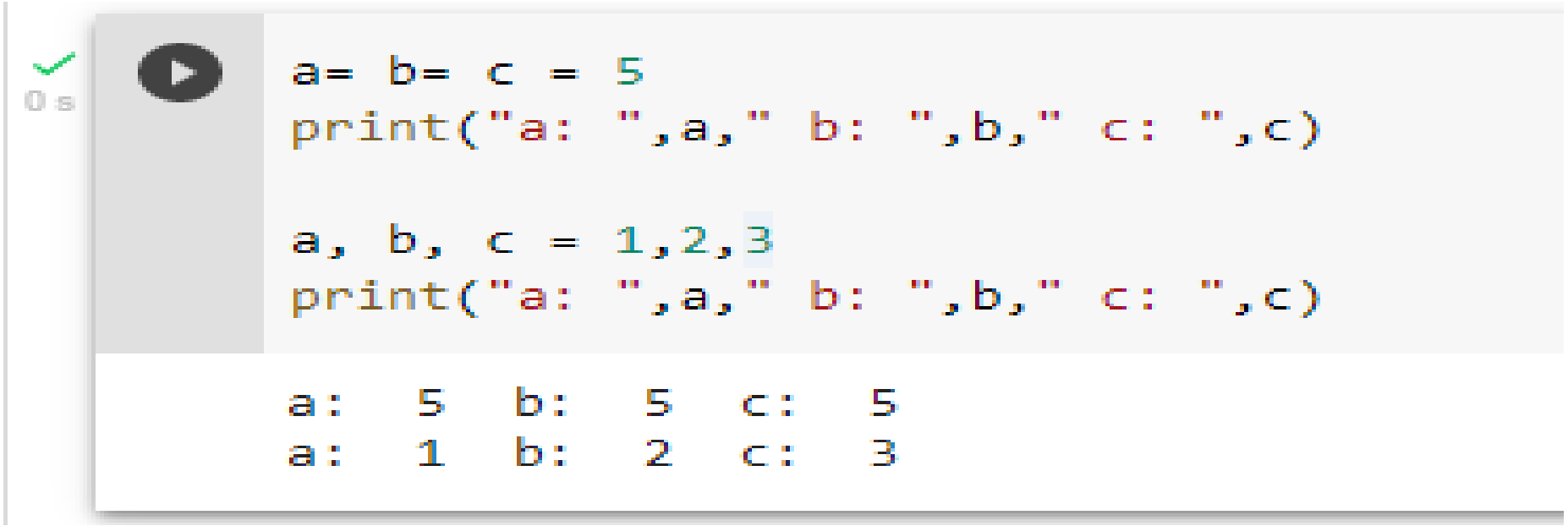
```
number = 5
name = "Oman"
average = 15.5

print(number)
print(name)
print(average)
```

```
5
Oman
15.5
```

Python Variable Types

Multiple Assignment:

Python allows you to assign a single value to several variables simultaneously, and you can also assign multiple objects to multiple variables.



Python syntax

1. Python Identifiers:

- An identifier starts with a letter **A to Z** or **a to z** or an **underscore** (_) followed by zero or more letters, underscores and digits (0 to 9).

- Python does not allow punctuation characters such **as @, $, and %** within identifiers.

- Python is a **case sensitive** programming language. Thus, **Manpower** and **manpower** are two different identifiers in Python.

Python syntax

2. **Reserverd words:** (keywords)

These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Python syntax

3. Lines and Indentation

- Python doesn't provide brackets to denote code blocks for flow control or the declaration of classes and functions.
- Line indentation, which is strictly regulated, is used to identify blocks of code.
- The indentation is vary in size, but each sentence in the block must be indented by the same number of spaces.

```
def roomState(room): #function to show the state of the room
  for i in range(len(room)):
    for j in range(len(room[i])):
      if room[i][j] == 1:
        print("room is dirty")
        return 1
  print("room is clean")
  return 0
```

Python syntax

4. Multi-Line Statements

- In Python, statements often conclude on a new line. The line continuation character (\) can be used in Python to indicate that the line should continue, though. For instance:

```
total = item_one + \
        item_two + \
        item_three
```

- The line continuation character is not required for statements included in the [], {}, or (). For instance:

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

Python syntax

5. Quotation in Python:

As long as the same kind of quotation appears at both the beginning and the end of the string, Python supports single ('), double ("), and triple (''' or """) quotes to signify string literals.

To extend the string across many lines, use triple quotes. For instance, all of the following are permitted:

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

Python syntax

6. Comments in Python

1. Using #:

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.
```

2. Using triple-quoted string:

```
'''
This is a multiline
comment.
'''
```

Python syntax

7. Multiple Statements on a Single Line

- The semicolon (;) permits several statements on a single line. Here is an example of the semicolon in use:

```
import numpy as np; a = np.zeros([1,3]); print(a)
```

```
[[0. 0. 0.]]
```

Input Function in Python:

Input function is that reads data from the users of our program, it's like read() function in the algorithm.

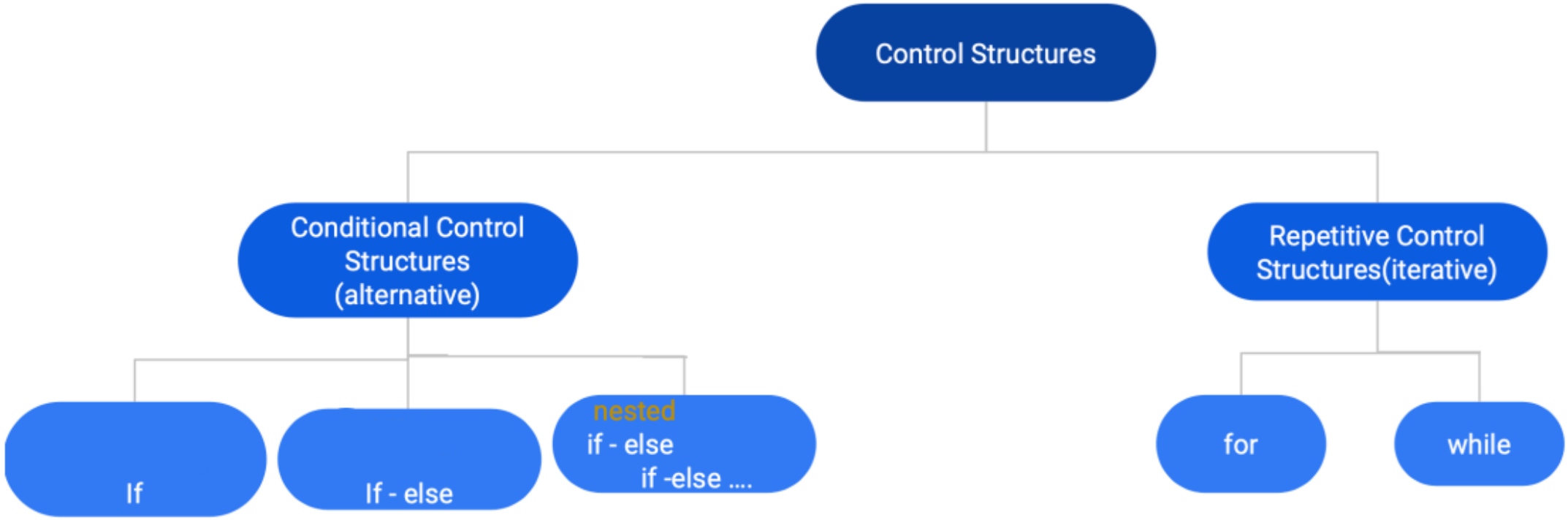
Input function reads all the data entered by the users as "Text", so, if we need to do arithmetic operation on the numbers that the user entered, we can't do that directly because even the numbers are read as text by the input function.

To solve that Python creators developed function that converts a text into an integer or a real number,
Example:

```
print("enter any text you want")
text1 = input()
print(text1)

print(" enter an integer number")
integer1 = int(input())
print(integer1)

print(" enter a real number")
real1 = float(input())
print(real1)
```

Control Structures

There are two types of control structures: Conditional control structures and Repetitive control structures.



Conditional Control Structures

The main conditional control structure types are:

1. **if statement** - Allows you to execute code based on the evaluation of a condition.

Algorithm syntax:

```
if (condition) then
      Actions
End if
```

Python syntax:

```
if (condition):
    Actions
```

Conditional Control Structures

2. **if/else statement** - Allows you to execute one code block if a condition is true, or an alternate code block if the condition is false.

Algorithm syntax:

```
If condition then
        Actions
else
       Actions
EndIf
```

Python syntax:

```
if (condition):
    Actions
else:
    Actions
```

This means that if the condition is true, the first block of actions is executed, and if it is false, the second block of actions is executed.

Notes:

The condition is a boolean (logical) expression.

Conditional Control Structures

2. if/else statement

Example of an algorithm (“checking if number is even”):

Algorithm 2: oddEven

```
var  num : integer
begin
write (enter your number)
read (num)
if num % 2 = 0 then
    write (The number is even)
else
    write (The number is odd)
endIf
end
```

Conditional Control Structures

2. if/else statement

Exercise:

Provide an algorithm that performs the division of two real numbers A/B.
And you must check first if B is not 0 (We can't divide on 0)

Algorithm 3: Division

```
var  A,B : real
begin
write (enter the value of A and B)
read (A,B)
if B = 0 then
    write ('Division impossible')
else
    write (A / B)
endIf
end
```

Conditional Control Structures

3. **Nested if statement** - Allows you to check multiple conditions and execute code for the first condition that evaluates to true

Algorithm syntax:

```
if condition then
    Actions
else
    if condition then
      Actions
    else
      Actions
      ....
    EndIf
EndIf
```

Python syntax:

```
if (condition) :
    actions
elif (condition) :
    actions
    ....
else :
    actions
```

Conditional Control Structures

3. Nested if statement

Example: a python program that categorizes numbers into positive, zero, or negative.

```
# Read a number from the user
number = float(input("Enter a number: "))

# Check whether the number is positive, zero, or negative
if number > 0:
    print("Positive")
elif number == 0:
    print("Zero")
else:
    print("Negative")
```

Conditional Control Structures

3. Nested if statement

Example: a python program that check if x is greater than y or not, and if so, it checks if y divides completely x or not.

Splitting the problem:

- The possible cases:

- x > y: and y divides completely x
- x > y: and y doesn't divide completely x
- x < y: x is less than y (so, logically we can't divide completely x/y)
- x = y: this case happens when the condition x>y and x<y aren't satisfied

```
x = int(input("Enter first number: "))
y = int(input("Enter second number: "))

if x > y:
  print(x, "is greater than", y)

  if x % y == 0:
    print(y, "divides", x, "completely")
  else:
    print(y, "does not divide", x, "completely")

elif x < y:
  print(y, "is greater than", x)

else:
  print("Both numbers are equal")
```

Conditional Control Structures

Exercise 1:

Write a Python program that takes a student's score as input and prints a message based on the following conditions:

1. **Failing Grade**: If the score is below 50, print "Fail".
2. **Passing Grade**:
    - If the score is between 50 and 64, print "Pass".
    - If the score is between 65 and 79, print "Good".
    - If the score is between 80 and 89, print "Very Good".
    - If the score is 90 or higher, print "Excellent".
3. **Invalid Input**: If the score is below 0 or above 100, print "Invalid score".

Use if, elif, and else statements with and and or conditions to accomplish this.

Conditional Control Structures

Exercise 2:

Write a Python program that takes two numbers and an operator (+, -, *, /) as input. Use if statement to perform the corresponding arithmetic operation and display the result.

Exercise 3:

Write a Python program that takes three integer inputs representing the sides of a triangle and outputs whether it is an equilateral (متساوي الأضلاع), isosceles (متساوي الساقين), or scalene triangle (مختلف الأضلاع).

REPETITIVE CONTROL STRUCTURES

DR. ADEL ABDELLI

01/12/2024

Repetitive Control Structures

In an algorithm and programming, certain instructions may need to be repeated, and it would not be very efficient to rewrite them. Loops have been introduced to model repetition.

Example:
Let's say we want to print "Hello" 10 times, so, we need to write the following:

Algorithm syntax

```
Write("Hello")
Write("Hello")
Write("Hello")
….
..
.
Write("Hello")
```

Python syntax

```
print("Hello")
print("Hello")
print("Hello")
….
..
.
print("Hello")
```

Conditional Control Structures

The solution is to use a loop. There is two types of loops, For and While loop.

1. The for loop:

When the number of repetitions is known, the for loop uses a counter to calculate the number of times the loop's block of instructions is repeated.

Algorithm syntax

```
for i←0 to i<10 do
   write("Hello")
endFor
```

Python syntax

```
for i in range(0,10):
    print("Hello")
```

Conditional Control Structures

In Python, the range function is used to generate a sequence of numbers. It is commonly used with for loops to iterate over a sequence of numbers. The basic syntax of the range function is as follows:

n: (Optional) The starting value of the sequence (default is 0).

m: The end value of the sequence (exclusive).

p: (Optional) The step or increment between values (default is 1).

Conditional Control Structures

Example:

Write an algorithm/Python code that calculates the factorial of a positive integer according to the formula:
Fact(x) = 1 * 2 * 3 * ... * x.

Algorithm:

Algorithm 5: Factorial

```
var  x, fact, i: integer
begin
read (x)
fact ← 1
for i ← 1 to x+1 do
    fact ← fact * i
endFor
write (fact)
end
```

Python:

```
a = int(input('enter a number'))
f = 1
for i in range(1,a+1):
    f = f * i
print('the factorial of',a, 'is: ',f)
```

Conditional Control Structures

2. While Loop:

A `while` loop allows the repetition of a block of instructions as long as a certain condition remains true.

a) Algorithm syntax :

```
While Condition Do
        Instruction 1;
        Instruction 2;
        ...
        Instruction n;
EndWhile
```

b) Python syntax :

```
While (condition):
        Instruction
```

If the condition is true, the group of instructions is executed, and then the "EndWhile" returns execution to the condition. This process continues until the condition is evaluated as false.

Conditional Control Structures

2. While Loop:

Note:

•The group of instructions may never be executed if the loop's condition is not initially satisfied.

Therefore, be cautious about loops that are never executed.

•Inside the loop, it is imperative to make changes so that the condition becomes false at some point.

Otherwise, if the condition always remains true, it will lead to an infinite loop, which is strictly

forbidden in algorithms.

Conditional Control Structures

2. While Loop:

Example:

Write an algorithm/Python code of guessing game, where the user have to guess our secret number which is between 0 and 10.

Algorithm:

```
Agorithm SN
Var a, secret_number: integer
Begin
    write('enter a number')
    read(a)
    while a ≠ secret_number do
        write('wrong number! enter another number')
        read(a)
    EndWhile
    Write('you won! you found the secret number')
END
```

Python:

```
a = int(input("enter a number"))
secret_number = 5
while (a != secret_number):
    a = int(input("wrong number! enter another number"))

print("you won! you found the secret number")
```

Conditional Control Structures

2. While Loop:

Example 2:

Writing a program to print numbers from 1 to 10 using a while loop.

Algorithm:

```
Agorithm exp2
Var i : integer
Begin
    i ← 1
    while i <= 10 do
        write(i)
        i ← i + 1
    EndWhile
END
```

Python:

```
i = 1
while i<= 10:
    print(i)
    i = i+1
```

Conditional Control Structures

2. While Loop:

Exercise: (Countdown Timer)

Create a countdown timer starting from a number entered by the user.

Solution:

```
n = int(input("Enter the starting number for countdown: "))
while n > 0:
    print(n)
    n = n - 1
print("Time's up!")
```

THANK YOU FOR YOUR ATTENTION