

{ POWER.CODERS }

Algorithms and programming principles

AGENDA

Today we will start with our recap quiz and then will learn about

- What an algorithm is
- Different programming principles
- Different types of programming

RECAP PROGRAMMING



WHAT IS PROGRAMMING?

WHAT IS PROGRAMMING?

Learning to program means learning how to **solve problems** using **code**.

WHAT IS PROGRAMMING?

Learning to program means learning how to **solve problems** using **code**.

➤ problem-solving = creative activity

WHAT IS PROGRAMMING?

Learning to program means learning how to **solve problems** using **code**.

- problem-solving = creative activity
- learning programming syntax = analytical activity

PROBLEM SOLVING



PROBLEM SOLVING

Solving a problem in programming can be defined as writing an original program that performs a particular **set of tasks** and meets all stated **constraints**.

PROBLEM SOLVING

Solving a problem in programming can be defined as writing an original program that performs a particular **set of tasks** and meets all stated **constraints**.

This set of tasks is called an **algorithm**.

WHAT IS AN ALGORITHM



ALGORITHM



ALGORITHM

An **algorithm** is a step-by-step set of operations that need to be performed.

ALGORITHM

An **algorithm** is a step-by-step set of operations that need to be performed.

If you take an algorithm and write code to perform those operations, you end up with a **computer program**.

IMPORTANT NOTE

An algorithm needs to be presented at their level of understanding and in a language they understand.

IMPORTANT NOTE

An algorithm needs to be presented at their level of understanding and in a language they understand.

"They" being **humans** as well as **machines**.

REAL-LIFE ALGORITHMS



REAL-LIFE ALGORITHMS

- > operating a laundry machine
- > playing a video game
- > baking a cake

LIKE A RECIPE



LIKE A RECIPE

> **Inputs** are the ingredients

LIKE A RECIPE

- **Inputs** are the ingredients
- **Procedure** is the list of steps to be followed

LIKE A RECIPE

- **Inputs** are the ingredients
- **Procedure** is the list of steps to be followed
- **Outputs** are the results of your recipe

EXAMPLE

The Euclidean algorithm

Find the greatest common divisor of 2 positive integers.

In math, the GCD of 2 or more positive integers is the largest positive integer that divides each of the integers.

Let's do it together.

WHAT DID WE DO?



1. Inputs: 2 positive integers, e.g. 40 (=number1) and 16 (=number2)
2. Procedure (=processes):
 1. Start
 2. Initialize variables for integers
 3. Initialize a variable for the output GCD
 4. Prompt for integer1 with "Put in your first positive number"
 5. Prompt for integer2 with "Put in your second positive number"
 6. Validate the variables to be positive integers
 7. Find the remainder: $\text{number1} \% \text{number2}$
 8. If the remainder == 0: then $\text{GCD} = \text{number2}$
 9. Else: then $\text{number1} = \text{number2}$, $\text{number2} = \text{Remainder}$, go back to step 3
 10. Display the output GCD of number1 and number2
 11. End
3. Outputs: their greatest common divisor (GCD)

REMEMBER THIS EXERCISE?



Validate a telephone number, as if written on an input form. Telephone numbers can be written as ten digits, or with dashes, spaces, or dots between the three segments, or with the area code parenthesized; both the area code and any white space between segments are optional.

INPUTS, PROCESSES AND OUTPUTS

A short, thick red horizontal line positioned below the word 'INPUTS'.

INPUTS, PROCESSES AND OUTPUTS

- **Inputs:** telephone number
- **Processes:** validate the format of the phone number
- **Outputs:** true or false

POSSIBLE PHONE NUMBERS



POSSIBLE PHONE NUMBERS

> 10 digits, e.g. 0797899236

POSSIBLE PHONE NUMBERS

- > **10 digits**, e.g. 0797899236
- > With **dashes**, e.g. 079-879-9236

POSSIBLE PHONE NUMBERS

- > **10 digits**, e.g. 0797899236
- > With **dashes**, e.g. 079-879-9236
- > With **spaces**, e.g. 079 879 9236

POSSIBLE PHONE NUMBERS

- > **10 digits**, e.g. 0797899236
- > With **dashes**, e.g. 079-879-9236
- > With **spaces**, e.g. 079 879 9236
- > With **dots**, e.g. 079.879.9236

POSSIBLE PHONE NUMBERS

- > **10 digits**, e.g. 0797899236
- > With **dashes**, e.g. 079-879-9236
- > With **spaces**, e.g. 079 879 9236
- > With **dots**, e.g. 079.879.9236
- > With **area code parenthesized**, e.g. (079) 879 9236 (optional)

POSSIBLE PHONE NUMBERS

- > **10 digits**, e.g. 0797899236
- > With **dashes**, e.g. 079-879-9236
- > With **spaces**, e.g. 079 879 9236
- > With **dots**, e.g. 079.879.9236
- > With **area code parenthesized**, e.g. (079) 879 9236 (optional)

Create test cases: Input and expected result.

WHAT ARE THE VALIDATION STEPS?

WHAT ARE THE VALIDATION STEPS?

1. Remove the white space in the variable
2. Is the variable a number (=all digits)?
3. If yes, is the number of digits exactly 10? Then go to **step 7**
4. If no, is the variable empty? If yes, then go to **step 6**
5. If no, remove specific characters: dashes, spaces, dots and parentheses. Then go back to **step 2**
6. Not valid? Return error message
7. Valid? Return success message

WHAT ARE THE VALIDATION STEPS?

1. Remove the white space in the variable
2. Is the variable a number (=all digits)?
3. If yes, is the number of digits exactly 10? Then go to **step 7**
4. If no, is the variable empty? If yes, then go to **step 6**
5. If no, remove specific characters: dashes, spaces, dots and parentheses. Then go back to **step 2**
6. Not valid? Return error message
7. Valid? Return success message

Check your test cases against the validation.

WRITE YOUR PROGRAM



1. Start

9. End

WRITE YOUR PROGRAM

1. Start
2. Create a variable to receive the phone number

9. End

WRITE YOUR PROGRAM

1. Start
2. Create a variable to receive the phone number
3. Clear the variable in case it is not empty
-
-
-
-
-
9. End

WRITE YOUR PROGRAM

1. Start
2. Create a variable to receive the phone number
3. Clear the variable in case it is not empty
4. Prompt for the phone number
-
-
-
-
9. End

© 2006 The Authors
Journal compilation © 2006 Blackwell Publishing Ltd

- © 2021 Powercoders

WRITE YOUR PROGRAM

1. Start
2. Create a variable to receive the phone number
3. Clear the variable in case it is not empty
4. Prompt for the phone number
5. Store the response in the variable
6. Validate the stored response if it is a valid phone number
9. End


WRITE YOUR PROGRAM

1. Start
2. Create a variable to receive the phone number
3. Clear the variable in case it is not empty
4. Prompt for the phone number
5. Store the response in the variable
6. Validate the stored response if it is a valid phone number
7. Not valid? Go back to step 3 plus error message
9. End

WRITE YOUR PROGRAM

1. Start
2. Create a variable to receive the phone number
3. Clear the variable in case it is not empty
4. Prompt for the phone number
5. Store the response in the variable
6. Validate the stored response if it is a valid phone number
7. Not valid? Go back to step 3 plus error message
8. Valid? Return success message
9. End

HAVE ALL PROBLEMS ALGORITHMIC SOLUTIONS?



Many problems will require a combination of **algorithmic** and **heuristic** solutions.

HAVE ALL PROBLEMS ALGORITHMIC SOLUTIONS?

Many problems will require a combination of **algorithmic** and **heuristic** solutions.

Heuristic solutions emerge from trial and error based on knowledge and experience.

HAVE ALL PROBLEMS ALGORITHMIC SOLUTIONS?

Many problems will require a combination of **algorithmic** and **heuristic** solutions.

Heuristic solutions emerge from trial and error based on knowledge and experience.

Real-life examples: Winning a chess game, making a speech at a ceremony or convention.
In programming typical problems are getting computers to speak a language or recognize patterns (Machine learning and artificial intelligence).

A QUICK WORD ON COMPLEXITY

When writing algorithms the complexity or performance is a **key factor to consider**: cost efficient, space efficient and fast. In computer science the **Big O notation** is used to describe the performance, or better the **worst-case scenario**.

BUDDY EXERCISE (45 MIN)

I am thinking of an integer between 1 and 100. Your task is to find this number by asking me questions of the form "Is your number higher, lower or equal to x" for different numbers x.

- Describe input and output
- List the steps of an algorithm that solves the guessing problem
- How many steps do you need?
- Can you think of a mathematical formula your algorithm is based on?

PROGRAMMING PRINCIPLES



MANIPULATING DATA



Moving data and playing with it is the foundation of programming, e.g. send login credentials to a web sever.

WRITE FOR A HUMAN AUDIENCE



You program in teams, not on your own. Be as clear as possible and as simple as possible. Clean code is more important than clever code.

FORCED SIMPLICITY



Good programming is about keeping the underlying logic expressed simply, so that it can be readable to others and your future self.

” Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.

—The Zen of Python

KISS PRINCIPLE



Keep it **s**imple, **s**tupid.

KISS PRINCIPLE



Keep it **s**imple, **s**tupid.

Start small and simplify your program as much as possible, before you even start. The more complex the code, the more bugs and errors will turn up, maintaining the code as well as modifying it will be more difficult.

DRY PRINCIPLE

Don't repeat yourself.

DRY PRINCIPLE

Don't repeat yourself.

Avoid duplicated and repeated code and data. Usually you can do that by abstracting and pulling code into functions, components and modules you only write once and re-use.

YAGNI PRINCIPLE



You **aren't gonna need it.**

YAGNI PRINCIPLE

You **aren't gonna need** it.

Never code a functionality that you may need in the future. You might not need it after all and it makes your code more complex.

REFACTOR YOUR CODE

There are many more principles, but the bottom line is: especially as an inexperienced programmer **code rarely comes out right the first time.**

Revisit, rewrite or even redesign parts of your code base based on programming principles.

TYPES OF PROGRAMMING (PROGRAMMING PARADIGMS)

DECLARATIVE PROGRAMMING



Declares **what** is being done rather than how it should be done.

DECLARATIVE PROGRAMMING

Declares **what** is being done rather than how it should be done.

Example: SQL

IMPERATIVE PROGRAMMING



Focuses on **how** a task is done rather than what is being done.

IMPERATIVE PROGRAMMING



Focuses on **how** a task is done rather than what is being done.

Examples: Java, JavaScript, Ruby

PROCEDURAL PROGRAMMING



Uses **procedures**, also known as routines, subroutines or functions, to operate on data structures and carry out tasks.

PROCEDURAL PROGRAMMING

Uses **procedures**, also known as routines, subroutines or functions, to operate on data structures and carry out tasks.

Examples: Java, JavaScript, C, Pascal, Basic

OBJECT-ORIENTED PROGRAMMING

Involves **building objects** with data attributes and programming subroutines. Code reusability and inheritance are main concepts of today's dominant paradigm.

OBJECT-ORIENTED PROGRAMMING

Involves **building objects** with data attributes and programming subroutines. Code reusability and inheritance are main concepts of today's dominant paradigm.

Examples: Java, JavaScript, Python, PHP, C, C++, Ruby

MULTI-PARADIGM LANGUAGES



Programming languages, like JavaScript, Java, and many more support **more than one** programming paradigm, in order to allow the **most suitable programming style** for a task.

ONLINE RESSOURCES

- Principles of algorithmic problem solving (book)
- A beginner's guide to Big O notation
- Fundamental programming principles
- 7 common programming principles
- 10 basic programming principles
- First principles of programming

EXERCISES



WHICH NUMBER IS BIGGER?

- Declare 2 variables, both numbers
- Compare which number is greater
- Log the output, e.g. "The greater number of 5 and 10 is 10."
- Add an output for the else statement, e.g. "The smaller number of 5 and 10 is 5."

Let's code it together (15 min)

RETIREMENT CALCULATOR



Create a program that determines how many years you have left until retirement and the year you can retire. It should prompt for your current age and the age you want to retire and display the output as shown in the example that follows on the next slide.

Time estimates: 1 hour 30 minutes

RETIREMENT CALCULATOR

What to do

- List inputs, outputs, processes in a text file.
- What are the constraints? Any edge case to consider? Note in your text file.
- Write the steps of your algorithm in pseudo-code in a second text file.
- Write the program in JavaScript. Find out how to get the current year from your computer.
- Push all 3 files (1 js and 2 txt) to your Github account.

RETIREMENT CALCULATOR

Example:

- > What is your age? 25
- > At what age would you like to retire? 65
- > You have 40 years left until you can retire.
- > It's 2020, so you can retire in 2060.

Your computer knows what the current year is. Figure out how to do that in JavaScript.

TIP CALCULATOR

Create a tip calculator in JavaScript. Take the pseudocode we created last week and turn it into JS. Step by step.

Time estimates: 45 minutes

WORK ON YOUR PROJECT

