{POWER.CODERS}

Programming 101
Day 2

Agenda

Today we will learn about

- How to solve problems
- How programs are set up
- The process to write a program

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

Darkness phoebia

One family wants to get through a tunnel. Dad can make it in 1 minute, mama in 2 minutes, son in 4 and daughter in 5 minutes. Unfortunately, not more than two persons can go through the narrow tunnel at one time, moving at the speed of the slower one.

DARKNESS PHOEBIA

One family wants to get through a tunnel. Dad can make it in 1 minute, mama in 2 minutes, son in 4 and daughter in 5 minutes. Unfortunately, not more than two persons can go through the narrow tunnel at one time, moving at the speed of the slower one.

Can they all make it to the other side if they have a torch that lasts only 12 minutes and they are afraid of the dark?

TRY IT YOURSELF FIRST

Before you go to the next slide, try to solve the problem on your own first. Give yourself about 30 to 45 minutes.

- Understand: Visualize the problem. Ask the why?how?what?
- Plan: What are the constraints, the inputs, the steps?
- Divide: Break bigger problems into smaller ones.
- Conquer: Write pseudocode.

POSSIBLE SOLUTION

```
GetThroughTheTunnel
    Initialize maxTime as 12;
    Initialize time1 as 1, time2 as 2, time3 as 4, time4 as 5;
    Initialize maxPeople as 2;
    Initialize totalTime as 0;
    Initialize peopleAtStart as 4;
    Initialize ListOfPeople;

    Compare times with each other to sort them by speed
    => output is ordered list

    Select the 2FastestPeople (times)
    Select the 2SlowestPeople (times)

    Step 1: 2 fastest going
        IF 2FastestPeople <= maxPeople
            totalTime = totalTime + (slowest of the 2 fastest);
            peopleAtStart = peopleAtStart-2
        END IF
    Step 2: The fastest is going back
```

Add the sorting algorithm

Let's have a closer look to yesterday's exercise 1, finalize the pseudo-code and add how you would sort the times by speed.

Exercise 1

Allow the user to input digits, afterwards sort and print them in numerical order.

UNDERSTANDING THE PROBLEM

Understanding the Problem

Understanding the problem is **key to solving** it.

UNDERSTANDING THE PROBLEM

Understanding the problem is **key to solving** it.

Sometimes the first brief might not be enough.
Ask questions.

POSSIBLE QUESTIONS

- Is there a limit of inputs?
- Are there several input fields (one per digit) or one?
- If there is only one, how is the correct input? Is there a seperator per digit (e.g. comma and/or space)?

Can you think of more?

Possible questions

- Is there a limit of inputs? max. 10
- Are there several input fields (one per digit) or one?
- If there is only one, how is the correct input? Is there a seperator per digit (e.g. comma and/or space)?

Can you think of more?

Possible questions

- Is there a limit of inputs? max. 10
- Are there several input fields (one per digit) or one? Only 1
- If there is only one, how is the correct input? Is there a seperator per digit (e.g. comma and/or space)?

Can you think of more?

Possible questions

- Is there a limit of inputs? max. 10
- Are there several input fields (one per digit) or one? Only 1
- If there is only one, how is the correct input? Is there a seperator per digit (e.g. comma and/or space)? Official seperator is comma, space is optional

Can you think of more?

What are the constraints?

Try to find them yourself first. Take 10 minutes to find contraints and note them down.

What are the constraints?

Try to find them yourself first. Take 10 minutes to find contraints and note them down.

- Numbers, positive, non-decimal
- Digits, not greater than 9
- Input is string, needs to be converted
- max 10 digits
- min 1 digit

TEST INPUT

1,2,7,3 => true
1, 2, 7, 3 => true
01226354 => false

Expected output

```
1237 => true
1,2,3,7 => false
2371 => false
[1,2,3,7] => false
```

Before you start coding...

... always define expected input and output and visualize the steps to get from input to output.

Before you start coding...

... always define expected input and output and visualize the steps to get from input to output.

- Input: Input digits in one field, seperated by comma
- Process: Sorting algorithm
- Output: Print digits in numerical order

PSEUDOCODE

Have a look at yesterday's pseudocode again. Can you now optimize it with inputs and outputs in mind?

Optimize your code now

Possible solution

```
SortNumbers
    INITIALIZE userInput AS String;
    INITIALIZE digitsList AS List;
    INITIALIZE output AS String;
    INITIALIZE counter;

    PROMPT "Please input minimum 2 digits and
        maximum 10 digits (0-9), separated by commas" AS userInput

    SPLIT userInput AT THE SEPERATOR "," TO digitsList
    // check for spaces, are there commas in the string

    FOR EACH item OF digitsList
        IF not a digit
            PROMPT ("Please make sure that you only add single digits (0-9)");
        END IF
    END FOR EACH

END SortNumbers
```

Another Example

Build a tip calculator.

Another Example

Build a tip calculator.

Do you have all the information you need to understand the problem?

Gather Requirements

Ask questions to find out all you need to know, e.g.

Gather Requirements

Ask questions to find out all you need to know, e.g.

- Can you explain how the tip should be calculated?

Gather requirements

Ask questions to find out all you need to know, e.g.

- Can you explain how the tip should be calculated?
- What's the tip percentage? Is it a fixed value or should the user be able to modify it?

Gather requirements

Ask questions to find out all you need to know, e.g.

- Can you explain how the tip should be calculated?
- What's the tip percentage? Is it a fixed value or should the user be able to modify it?
- What should the program display on the screen when it starts?

Gather requirements

Ask questions to find out all you need to know, e.g.

- Can you explain how the tip should be calculated?
- What's the tip percentage? Is it a fixed value or should the user be able to modify it?
- What should the program display on the screen when it starts?
- What should the program display for its output? Do you want to see the total and the tip?

Problem statement

Once your questions are answered, summarize the problem in a statement:

Problem statement

Once your questions are answered, summarize the problem in a statement:

”Create a simple tip calculator. The program should prompt for a bill amount and a tip rate. The program must compute the tip and then display both the tip and the total amount of the bill.

Example Output

For your understanding it is always helpful to have some example inputs and outputs in mind, before you start solving your problem.

Example Output

For your understanding it is always helpful to have some example inputs and outputs in mind, before you start solving your problem.

- What is the bill? CHF 200
- What is the tip percentage? 15
- The tip is CHF 30.00
- The total is CHF 230.00

Don't start coding yet

This is the time where programmers often jump ahead and start coding.

Don't start coding yet

This is the time where programmers often jump ahead and start coding.

Make sure your **program is thought through** probably, before you start.

Process to write a program

Discover inputs, processes and outputs

Even the simplest program has always inputs, processes and outputs. It is the **core definition** of a program.

Discover inputs, processes and outputs

Even the simplest program has always inputs, processes and outputs. It is the **core definition** of a program.

Go back to the problem statement and check for nouns and verbs. **Nouns** are usually your **inputs** and **outputs**, while **verbs** describe your **processes**.

Nouns in the problem statement

- bill amount
- tip rate
- tip
- total amount

Nouns in the problem statement

- bill amount **= input**
- tip rate
- tip
- total amount

Nouns in the problem statement

- bill amount **= input**
- tip rate **= input**
- tip
- total amount

Nouns in the problem statement

- bill amount **= input**
- tip rate **= input**
- tip **= output**
- total amount

Nouns in the problem statement

- bill amount **= input**
- tip rate **= input**
- tip **= output**
- total amount **= output**

VERBS IN THE PROBLEM STATEMENT

- prompt
- compute
- display

VERBS IN THE PROBLEM STATEMENT

- prompt
- compute
- display

Not every verb has to be a process. **prompt** and **display** define the GUI, how the user interacts with the program.

RESULT

- Inputs:
- Processes:
- Outputs:

Result

- Inputs: **bill amount, tip rate**
- Processes:
- Outputs:

Result

- Inputs: **bill amount, tip rate**
- Processes: **calculate the tip**
- Outputs:

Result

- Inputs: **bill amount, tip rate**
- Processes: **calculate the tip**
- Outputs: **tip amount, total amount**

Driving Design with Tests

Test-driven development (TDD) is a very common practice in software developement and more and more web application development.

Driving Design with Tests

Test-driven development (TDD) is a very common practice in software developement and more and more web application development.

The essence of TDD is to think about what the **expected result** of the program is **ahead of the time** and then work towards getting there.

Driving Design with Tests

Test-driven development (TDD) is a very common practice in software developement and more and more web application development.

The essence of TDD is to think about what the **expected result** of the program is **ahead of the time** and then work towards getting there.

Doing that even before you start coding, often enables you to think **beyond the initial requirements**.

SIMPLE TEST PLANS

The easiest way to do that is creating simple test plans.

Simple test plans

The easiest way to do that is creating simple test plans.

A test plan lists the program's inputs and its expected result.

TEMPLATE FOR TEST PLAN

- Inputs:
- Expected result:
- Actual result:

TEMPLATE FOR TEST PLAN

- Inputs:
- Expected result:
- Actual result:

You start with writing down the program's inputs and expected output. After running your program you compare the expected result with the actual result.

TEST PLAN FOR OUR TIP CALCULATOR

- Inputs:

- Expected result:

TEST PLAN FOR OUR TIP CALCULATOR

- Inputs:
    - bill amount: 10

- Expected result:

Test plan for our tip calculator

- Inputs:
  - bill amount: 10
  - tip rate: 15
- Expected result:

Test plan for our tip calculator

- Inputs:
  - bill amount: 10
  - tip rate: 15
- Expected result:
  - tip: CHF 1.50

TEST PLAN FOR OUR TIP CALCULATOR

- Inputs:
  - bill amount: 10
  - tip rate: 15
- Expected result:
  - tip: CHF 1.50
  - total: CHF 11.50

What does it tell us so far?

What does it tell us so far?

- We have **2** inputs and **2** outputs.

What does it tell us so far?

- We have **2** inputs and **2** outputs.
- We need to convert our tip rate from a whole number to a decimal.

What does it tell us so far?

- We have **2** inputs and **2** outputs.
- We need to convert our tip rate from a whole number to a decimal.
- The output is formatted as a currency.

ONE TEST IS NOT ENOUGH

Work with more than one test.

One test is not enough

Work with more than one test.

Try to think about corner and edge cases.

Corner case

A problem or situation that occurs outside of normal operating parameters.

CORNER CASE

A problem or situation that occurs outside of normal operating parameters.

Think of it as a hard to reach design state that happens only when **multiple conditions** happen in some combination.

Corner case

A problem or situation that occurs outside of normal operating parameters.

Think of it as a hard to reach design state that happens only when **multiple conditions** happen in some combination.

Your program logic meets more than one boundary condition at once.

Edge case

A problem or situation that occurs only at an extreme (maximum or minimum) operating parameter, e.g. input.

Edge case

A problem or situation that occurs only at an extreme (maximum or minimum) operating parameter, e.g. input.

Your program logic meets one boundary condition.

Edge case

A problem or situation that occurs only at an extreme (maximum or minimum) operating parameter, e.g. input.

Your program logic meets one boundary condition.

Is it very useful to add edge cases to your tests, as they are usually the place where bugs appear.

Buddy exercise (30 min)

Come up with several test cases on your own (15 min), then come together with your buddy and discuss them (15 min).

Try to think about corner and edge cases.

For example: what happens if the bill amount is already a decimal? What happens if the tip rate is a decimal? What could be an edge case?

Constraints

Edge and corner cases can show you the constraints of your program. Also the program statement and test cases can visualize them.

Constraints

Edge and corner cases can show you the constraints of your program. Also the program statement and test cases can visualize them.

- Enter the tip as a percentage. The input should be 15 for "15%", not 0.15.

Constraints

Edge and corner cases can show you the constraints of your program. Also the program statement and test cases can visualize them.

- Enter the tip as a percentage. The input should be 15 for "15%", not 0.15.
- Both inputs have to be bigger than 0 and, of course, a number.

Constraints

Edge and corner cases can show you the constraints of your program. Also the program statement and test cases can visualize them.

- Enter the tip as a percentage. The input should be 15 for "15%", not 0.15.
- Both inputs have to be bigger than 0 and, of course, a number.
- Round fractions of a Rappen up to the next Rappen. Remember 5 Rappen is the smallest.

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**.

START WITH PSEUDOCODE

Although there is no right or wrong way to write pseudocode, there are widely used terms.

START WITH PSEUDOCODE

Although there is no right or wrong way to write pseudocode, there are widely used terms.

- **Initialize:** set an initial value

Start with pseudocode

Although there is no right or wrong way to write pseudocode, there are widely used terms.

- **Initialize:** set an initial value
- **Prompt:** for user input

Start with pseudocode

Although there is no right or wrong way to write pseudocode, there are widely used terms.

- **Initialize:** set an initial value
- **Prompt:** for user input
- **Display:** for output on the screen

Try it yourself first

Before you go to the next slide, try to write the pseudocode on your own first. Start with very general descriptions of your steps. Give yourself about 30 to 45 minutes.

TIP CALCULATOR IN PSEUDOCODE

```
TipCalculator
    Initialize billAmount to 0
    Initialize tip to 0
    Initialize tipRate to 0
    Initialize total to 0

    Prompt for billAmount with "What is the bill amount?"
    Prompt for tipRate with "What is the tip rate?"

    convert billAmount to a number
    convert tipRate to a number

    tip = billAmount * (tipRate / 100)
    round tip up to nearest 5 rappen
    total = billAmount + tip

    Display "Tip : CHF" + tip
    Display "Total : CHF" + total
End
```

What did we do?

What did we do?

- We set up **variables** with default values

What did we do?

- We set up **variables** with default values
- We asked for **user input**

What did we do?

- We set up **variables** with default values
- We asked for **user input**
- We did some **conversions** and math

What did we do?

- We set up **variables** with default values
- We asked for **user input**
- We did some **conversions** and math
- We displayed the calculated **output** on the screen

Improve the program

The next step would be to improve the code, e.g.

IMPROVE THE PROGRAM

The next step would be to improve the code, e.g.

- Check for edge and corner cases

Improve the program

The next step would be to improve the code, e.g.

- Check for edge and corner cases
- Split the programs into functions

Improve the program

The next step would be to improve the code, e.g.

- Check for edge and corner cases
- Split the programs into functions
- Format the output as floats (just to be sure)

IMPROVE THE PROGRAM

The next step would be to improve the code, e.g.

- Check for edge and corner cases
- Split the programs into functions
- Format the output as floats (just to be sure)

We will discuss data types, variables and functions in detail next week.

Writing the code

The final step would be to **write** the code, **comment** it properly, **test** and **debug** it right away and finally **revise** and **optimize** your code for quality and performance.

One word about comments

Never comment on **what** the code is doing, only write comments that explain **why**.

BAD EXAMPLE

No need to understand the code of the example yet, just have a look at the comments.

```
// This function checks whether a number is even
let function f(x){
    // compute x modulo 2 and check whether it is zero
    if (x%2 == 0){
        // the number is even
        return true;
    } else {
        // the number is odd
        return false;
    }
}
```

Good example

Notice that the name of the functions and the variables are self-explanatory. Comments are not needed.

```
let function is_divisible(number, divisor){
    return number%divisor == 0;
}

let function is_even(number){
    return is_divisible(number, 2);
}
```

Optimizing code

Better naming and a better task breakdown make the comments from the bad example (what comments) obsolete.

OPTIMIZING CODE

Better naming and a better task breakdown make the comments from the bad example (what comments) obsolete.

Revise your code just as you would revise an essay: Sketch, write, delete, reformulate, ask others what they think. Repeat until only the crispest possible expression of your idea remains.

Exercises

For each exercise follow this process:

- List inputs, process and outputs
- Write test cases
- Find constraints
- Write the algorithm in pseudocode

1. Say hello (30 min)

Create a program that prompts for your name and prints a greeting using your name.

2. Counting the number of characters (60 min)

Create a program that prompts for an input string and displays output that shows the input string and the number of characters the string contains.

3. Printing quotes (45 min)

Create a program that prompts for a quote and an author. Display the quotation and author as shown here:
[Author] says, "[Quote]" (Replace [Author] and [Quote] with the actual values.

4. Mad Lib (45 min)

Definition: Mad Libs are a simple game where you create a story template with blanks for words. You or another player then construct a list of words and place them into the story, creating an often silly or funny story as result.

More info on next page

4. Mad Lib (45 min)

Program statement: Create a simple mad-lib program that prompts for a noun, a verb, an adverb, and an adjective and injects those into a story that you create.

Still have time and energy?

Finish open exercises in HTML and CSS. Work on your website project.