We are constantly riddled with problems in our daily lives that we want to solve to make our lives better. To solve these problems, we first need to identify the problem and form a plan of action, then execute that action and evaluate whether that action successfully solved the problem. In the following exercises, you will learn how to identify the problem, how to devise a plan, how to execute that plan and finally how to evaluate the solution.

The image above shows the general process of solving problems. Your initial plan may not be perfect, but through execution and validation you will be able to refine your solutions so that they are useful and efficient.

Okay, let’s do some examples to help you better understand the process of problem-solving.
Help Mario reach the flag

In the above diagram, how would you help Mario reach the flag? First, let’s understand the problem.

1. The problem:
The main problem is that Mario needs to reach the flag. However, there is a hole in the ground and two enemies in front of him. He needs to avoid the hole and the enemies and reach the flag. He can also kill the enemy by jumping on their head.

Therefore, we have identified the problem. Now we devise a plan.

2. Initial Plan:
A problem can be solved many ways, so your solution plan may be different from this one. The coherence and validity of the plan is determined after the execution and during evaluation. This is one possible solution for this problem:

1. Mario moves forward.
2. If there is a hole in the ground, jump over it.
3. Continue moving until there is an enemy in front of Mario.
4. Jump over the first enemy and move forward until there is a second enemy.
5. Jump over the second enemy and move forward until Mario reaches the flag.
6. When Mario reaches the flag, we have success.

3. Execute and Evaluate Initial Plan
If you visually trace the initial plan, you will see that the plan is valid as it solves the problem. However, if we look at the steps we encounter some questions about the problem:

- Do these steps work in other scenarios?
- Mario can also jump on the enemies’ head and kill them. That would also solve the problem. Can we create a solution that allows the user to choose how they want to solve the problem?

4. Revised plan:

1. Move Mario forward.
2. Check if there is a hole or enemy in front of him.
   a. If there is a hole, then jump over it.
   b. If there is an enemy in front of him, then jump over the enemy or jump on the enemy to kill it.
   c. If there is a flag, go to #4
3. Continue steps 1-3
4. END Mario reaches the flag.

In this second situation the initial plan does not work, so we need to make a plan that is going to be applicable to all scenarios. See step 3
Suppose you are traveling to London, where the temperature is measured in Celsius. Since we use Fahrenheit for temperature in the United States, you want to make a simple program that would take Celsius as input and convert the temperature to Fahrenheit.

1 **Understanding the problem:**
   
   I think the problem is that we need a way to convert Celsius temperature into Fahrenheit.

2 **Initial plan:**
   
   1. Take the input in Celsius.
   2. Convert the input to Fahrenheit.
   3. Write the output.

3 **Execution and Evaluation**

   Even though the instruction conveys the logic correctly, the steps are not very clear. If one must read through the steps, they could encounter questions like:
   
   - Where do I get the input from?
   - Where do I write the values for Celsius and Fahrenheit?
   - What is the formula for converting from Celsius to Fahrenheit?

4 **Revising the plan:**

   Rewrite the instructions addressing the errors found during the evaluation of our plan. Remember:
   
   1. The instructions are not just for you, other people should be able to find the answer using your plan.
   2. Your instructions must include obvious steps and sufficient information.

5 **Revised Plan**

   1. Read the temperature in Celsius
   2. Write the temperature in Celsius as a variable C.
   3. Write another variable called F which will store the value for the temperature in Fahrenheit.
   4. Use the conversion formula: 
      \[ F = \frac{9 \times C}{5} + 32 \]
   5. Write the value of F i.e. temperature in Fahrenheit.

**Pseudocode VS Instructions:**

We have seen from the previous examples that instructions can be quite helpful when creating a problem-solving plan. Pseudocode is a form of instructions that are specifically directed towards computer programmers. Therefore, we should be more specific when writing pseudocode. Below, we will practice some computing concepts with the help of pseudocode:

---

**Temperature Conversion**

Suppose you are traveling to London, where the temperature is measured in Celsius. Since we use Fahrenheit for temperature in the United States, you want to make a simple program that would take Celsius as input and convert the temperature to Fahrenheit.

1 **Understanding the problem:**

   I think the problem is that we need a way to convert Celsius temperature into Fahrenheit.

2 **Initial plan:**

   1. Take the input in Celsius.
   2. Convert the input to Fahrenheit.
   3. Write the output.

3 **Execution and Evaluation**

   Even though the instruction conveys the logic correctly, the steps are not very clear. If one must read through the steps, they could encounter questions like:
   
   - Where do I get the input from?
   - Where do I write the values for Celsius and Fahrenheit?
   - What is the formula for converting from Celsius to Fahrenheit?

4 **Revising the plan:**

   Rewrite the instructions addressing the errors found during the evaluation of our plan. Remember:
   
   1. The instructions are not just for you, other people should be able to find the answer using your plan.
   2. Your instructions must include obvious steps and sufficient information.

5 **Revised Plan**

   1. Read the temperature in Celsius
   2. Write the temperature in Celsius as a variable C.
   3. Write another variable called F which will store the value for the temperature in Fahrenheit.
   4. Use the conversion formula: 
      \[ F = \frac{9 \times C}{5} + 32 \]
   5. Write the value of F i.e. temperature in Fahrenheit.

**Pseudocode VS Instructions:**

We have seen from the previous examples that instructions can be quite helpful when creating a problem-solving plan. Pseudocode is a form of instructions that are specifically directed towards computer programmers. Therefore, we should be more specific when writing pseudocode. Below, we will practice some computing concepts with the help of pseudocode:
Suppose you are designing an elevator with very simple logic: it has three buttons corresponding to three floors; when a user presses a button, it moves the elevator to that floor. How would you write the pseudocode for this program?

Remember the guidelines for problem-solving!

1. **Understanding the problem:**
   
   We are developing an elevator program, where a user can press a number from 1 to 3 and the elevator will move to the floor corresponding to that number.

2. **Initial plan:**
   
   1. Show the user the floor numbers.
   2. Get the floor number from the user.
   3. Move the elevator to that floor number.

3. **Execution and Evaluation**
   
   Reading through the initial plan, we encounter some questions like:
   
   - What are the floor numbers?
   - Where do I show the numbers?
   - Where do I store the user’s choice?
   - Where should I write the output?
   - How does the user know whether the floor has been reached?
Revised plan:

1. PRINT: “Please enter the floor number you want to go to:
   1: First Floor
   2: Second Floor
   3: Third Floor”.
2. CREATE variable called userChoice to store the user’s choice.
3. STORE the number the user entered into userChoice.
4. IF userChoice is equal to 1
   a. Move the elevator to first floor
   b. PRINT: “You have reached the first floor.”
ELSE IF userChoice is equal to 2
   a. Move the elevator to second floor.
   b. PRINT: “You have reached the second floor.”
ELSE IF userChoice is equal to 3
   a. Move the elevator to third floor
   b. PRINT: “You have reached the third floor.”

Execution

When the program runs, each step is executed as such:

1. The program prints, “Please enter the floor number you want to go to:
   1: First Floor.
   2: Second Floor.
   3: Third Floor.”
2. A variable called userChoice is created.
   a. Assume, the user entered: 2
3. userChoice = 2.
4. Since, userChoice = 2, the elevator will move to second floor and print: “You have reached the second floor”

Re-Revised Plan

This new plan is much better but while executing new questions emerged:

- What if the user enters a number that is not within 1 to 3, or what if it is not even a number?
- What if the user made a mistake, and needs to go to a different floor? This code runs just one time. Can we let the user confirm that they are indeed in the correct floor?

Try to solve the above questions and write a more detailed plan:
Driving around cities in the United States, you may have seen large water towers, like the one pictured. These water towers provide water to your house. Each of these towers can contain millions of gallons of water. Let’s say you are designing a program that starts filling the reservoir when it’s empty and stops when it’s full. Can you design a pseudocode for this program?

1. **Understanding the problem:**
   We are developing a program that detects when a water tower is not full and fills it with water.

2. **Initial plan:**
   1. Check if the reservoir is not full.
   2. If there is space.
      a. Fill with water.
   3. Continue step 2 until the reservoir is full.

3. **Execution and Evaluation**
   Reading through the initial plan we encounter some questions like:
   - What is the capacity of the reservoir?
   - How do we check whether the reservoir is not full?
   - How does the user know that the reservoir is full?

4. **Revised plan:**
   1. CREATE a variable called reservoirCap to store how much water a reservoir can hold
   2. PRINT: “Please enter the capacity of the reservoir:”
   3. STORE the value the user entered into reservoirCap.
   4. CREATE a variable called waterLevel
   5. STORE the value 0 in waterLevel.
   6. CHECK WHILE (waterLevel is not equal to reservoirCap)
      a. Add one gallon of water.
      b. ADD 1 to the waterLevel.
   7. ENDWHILE
   8. PRINT: “The reservoir is full.”

5. **Execution**
   When the program runs each step is executed as such:
   1. A variable called reservoirCap is created.
   2. The program writes: “Please enter the capacity of the reservoir:”
      a. Assume, the user entered: 10
   3. reservoirCap = 10
   4. A variable called waterLevel is created.
   5. waterLevel = 0
   6. Since waterLevel = 0 and reservoirCap = 10, the program adds one gallon of water and adds 1 to waterLevel.
   7. Step 6 continues while waterLevel is not equal to reservoirCap.
   8. The system writes: “The reservoir is full.”