University of
South Australia

Problem Solving and Programming

Problem Solving Process

Reference

- Robertson, A. *Simple Program Design*. 5th edition. 2006. Thomson.

Problem Solving Process

- Moving from problems to code…

- Use a systematic problem solving strategy:

1. Define the problem.
2. Develop an algorithm.
3. Test the algorithm for correctness.
4. Implement algorithm in chosen programming language (Python).
5. Test and verify program.

Problem Solving Process

- **1. Define the Problem.**
  - If you don't have a clear understanding of the problem, it is unlikely that you'll be able to solve it.
  - Steps to help you understand the problem:
    - Carefully read the problem until you understand what is required.
    - Divide the problem into:
      - Input
      - Output
      - Processing - List of steps/actions needed to produce the output
  - Underline words identifying the inputs and outputs.
    - Look for nouns
  - CAPITALISE words identifying processing actions.
    - Look for verbs

Problem Solving Process

- **1. Define the Problem.**
  - Strategy:
    - What is the desired outcome?
    - What do I need to know first to reach this goal?
    - What steps do I need to take? (look for verbs, in order)

Problem Solving Process

- **2. Develop an Algorithm.**
  - List the detailed set of instructions you need to perform to solve the problem.
  - The solution to any computing problem involves executing a series of actions in a specific order.
  - What is an algorithm?
    - An algorithm is like a recipe, a set of instructions that are:
      - Detailed
      - Precise
      - Ordered
    - A procedure for solving a problem in terms of:
      - The actions to be executed, and
      - The order in which these actions are to be executed.
  - The goal of an algorithm is to complete some task.
  - An algorithm is written in simple English.

Problem Solving Process

- **2. Develop an Algorithm.**
  - Correctly specifying the order in which the actions are to be executed is important.
  - An Exercise:
    - Write an algorithm to get out of bed and arrive at Uni for a lecture…

Problem Solving Process

- **2. Develop an Algorithm.**
  - Write an algorithm to get out of bed and arrive at uni for a lecture…

    Get out of bed
    Eat breakfast
    Take off pyjamas
    Take a shower
    Brush teeth
    Get dressed
    Drive to Uni
  - Suppose that the steps are performed in a slightly different order… trouble!!

    Get out of bed
    Take off pyjamas
    Get dressed
    Take a shower
    Brush teeth
    Eat breakfast
    Drive to Uni

Problem Solving Process

- **2. Develop an Algorithm.**
  - Design a solution and develop into an algorithm…
    - You may like to work backwards:

      What outputs are required?
      What data do you need to be able to calculate the outputs?
      Where can you get that data? (user inputs, constants, derived values, other programs).
  - You may use pseudocode to draft your solution…

Problem Solving Tool

- What is Pseudocode?
  - Pseudocode is an informal language that helps you develop algorithms.
  - Pseudocode is a tool used to plan your program before you start coding.
  - It helps you “think out” a program before attempting to write it in a programming language such as Python.
  - Pseudocode is a verbal description of your plan.
    - Similar to programming code.
    - Design an algorithm in pseudocode.
    - Structured, formalized, condensed English.
    - Should not resemble any particular programming language (ignore syntax).
  - You may use pseudocode in order to help you solve your problems.
  - You can design solutions without knowing a programming language.
  - Intended to help you create better computer programs.

Problem Solving Tool

- 3 Control Structures
  - Sequence
  - Selection
  - Repetition

- Pseudocode uses common words and keywords to symbolise these operations.

  For example:

Pseudocode:

```
IF time is greater than 7
  print 'time to go home'


k = 0
WHILE k is less than 3
  k = k + 1
  print k to screen
```

Python code:

```
if time > 7:
   print('Time to go home')


k = 0
while k < 3:
   k = k + 1
   print(k)
```

UniSA

Problem Solving Tool

- For example *(continued…)*:

Pseudocode:

```
k = 0
WHILE k < 3
   k = k + 1
   print k to screen
```

```
FOR k in 1 to 3
   print k to screen
```

Python:

```
k = 0
while k < 3:
    k = k + 1
    print(k)
```

```
for k in [1, 2, 3]:
   print(k)
```

UniSA

Problem Solving Process

- **2. Develop an Algorithm.**
  - Write down a set of steps to solve the problem.
  - Take the processing steps from step 1 (defining the problem).
  - Take the 3 basic control constructs:
    - Sequence
    - Selection
    - Repetition
  - Determine how the processing will be performed.
    - Sometimes a trial and error process.
  - Each processing step relates to **1 or more steps** in the algorithm.
  - Once an algorithm is fully developed and tested, implementing it in a particular programming language is relatively trivial.

Problem Solving Process

- **3. Test the algorithm for correctness.**
  - To detect errors early.
  - To make sure the algorithm is correct.

    i.e. produces the correct results.

  - Hand 'execute' your algorithm:
    - Work through your test cases and see if your algorithm handles these and gives the right answer.
    - Look for ambiguous or missing steps.
    - Look for steps that do a lot – these may need to be broken down further.

Problem Solving Process

- **3. Test the algorithm for correctness.**
  - To test if your approach to solving the problem will work, you need test cases where you know what the result or output should be.
  - You need one or more situations that are 'typical' and where the algorithm should work and you are able to specify what the algorithm should do.

1. Choose 2 simple sets of valid input values.
   - Select test data based on the requirements, not your algorithm
   - As you're not using a computer to calculate, keep values simple: 10, 20, 30 is easier than 3.75 2.89 and 5.31
2. Determine expected results.
3. Step through algorithm with first test data set.
4. Repeat with other test data set.
5. Check that results from steps 3 and 4 match expected results.

Problem Solving Process

- **3. Test the algorithm for correctness.**
  - Look for boundaries
    - Are there input ranges where there is no solution or the algorithm will not work?
    - Should the algorithm work differently for different ranges of inputs?
    - Look for special cases.
    - These are often associated with a boundary, such as first or last value of an input.

Problem Solving Process

- **4. Implement algorithm in chosen programming language.**
  - Create a Python solution.
    - Convert the algorithm into a Python solution.
  - Include comments in the solution.
    - Helps others follow your work (as well as yourself).
  - Generally better to do this incrementally – in small steps where you execute and test the code you have just added. This is important!

Problem Solving Process

- **4. Implement algorithm in chosen programming language.**
  - Follow good coding standards as you go.

    This includes:
    - Use of sensible and meaningful variable names.
    - Use of consistent indentation.
    - Leave some 'white space' to improve readability (i.e. blank lines, appropriate spacing between operators, etc).
    - Comment interesting / significant bits of code.
    - Use of comments to describe functions.

  - Tidy up your code as you go – keep it readable and take out redundant test code.

Problem Solving Process

- **5. Test and verify the solution / program.**
  - Compare to the hand solution / algorithm test.
  - Do your answers make sense?
  - Do they match your sample calculations?
  - Is your answer what was asked for?

Problem Solving Process

- Here's an example:

  Given a wall which has a width of 5 metres and a height of 10 metres, write a program that computes and displays the amount of paint required to paint the wall. Paint coverage is 10 square metres per litre.

Problem Solving Process

- 1. Define the problem

  Given a wall which has a width of 5 metres and a height of 10 metres, write a program that COMPUTES and DISPLAYS the amount of paint required to paint the wall. Paint coverage is 10 square metres per litre.

  You may like to restate the problem in your own words…

  Find the amount of paint required to paint a wall.

Problem Solving Process

- 1. Define the problem
  - Inputs

    Wall width width = 5 m

    Wall height height = 10 m

  - Output

    Quantity of paint (quantity in litres)

  - Processing

    Compute the area of a wall

    Compute amount of paint required

    Display amount of paint required

Problem Solving Process

- 2. Develop an algorithm.
  - The area of a rectangle can be calculated using the following formula:

    area = width x height

  - Compute the area of the wall

    area = width x height

    = 5 m x 10 m

    = 50 m^2

  - Compute the amount of paint required

    quantity = area / 10

    = 50 m^2 / 10 m^2

    = 5 litres of paint required

UniSA

Problem Solving Process

- 2. Develop an algorithm.
  - List the detailed set of instructions you need to perform to solve the problem.
    - The hand solution (if you have one) will help you do this.
    - The processing actions identified in step 1 will also help you do this.
  - Thus, the algorithm (detailed set of instructions) is as follows:

```
Set width to 10
Set height to 5
Compute area of wall  area = width x height
Compute quantity of paint  quantity = area / 10
Display the quantity of paint to the screen
```

Problem Solving Process

- 3. Test the algorithm for correctness.

- 4. Develop a Python solution.
  - Convert the algorithm into a Python solution.

```
width = 10
height = 5
area = width * height
quantity = area / 10
print('Quantity of paint required:', quantity)
```

Problem Solving Process

- 5. Test and verify the solution / program.
  - Compare to the hand solution (if you have one) or the testing from step 3.
  - Do your answers make sense?
  - Do they match your sample calculations?
  - Is your answer what was asked for?

Problem Solving Process

- Summary:
  - Analyse then Design then Test then Implement then Test.
  - Pseudocode is structured English.
    - Don't need to worry about syntax.
    - Focus on the most important aspect – design.
    - Good way to focus on solving the problem without getting stuck on the syntax/details of a programming language.
  - Let the problem statement guide you to the solution.
    - Identify the nouns (often input/output).
    - Identify the verbs (often the processing steps) in order.
  - Pseudocode can be translated into nearly any programming language.
  - Break up larger problems into a set of smaller ones.

End

Problem Solving Process