INTRODUCTION TO COMPUTER SCIENCE

Rutgers University

1. Algorithmic Thinking and Design



https://introcs.cs.rutgers.edu



INTRODUCTION TO
COMPUTER SCIENCE
Rutgers University

1. Algorithmic Thinking

- What is Algorithmic thinking?
- Algorithm development
- A Pseudo Language
- From Algorithm to pseudocode

Styles Adopted from Slides by Sedgewick and Wayne

A human being should be able to
change a diaper,
plan an invasion,
butcher a hog,
conn a ship,
design a building,
write a sonnet,
balance accounts,
build a wall,
set a bone,
comfort the dying,
take orders,
give orders,
cooperate,
act alone,
solve equations,
analyze a new problem,
pitch manure,
Need algorithmic thinking → program a computer,
cook a tasty meal,
fight efficiently, and
die gallantly.

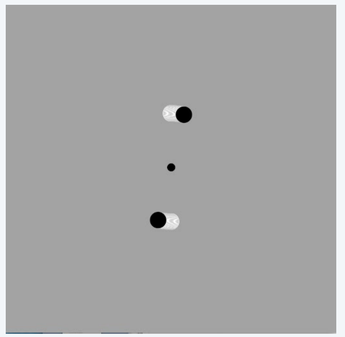
Specialization is for insects.

Robert A. Heinlein
Time Enough for Love (1973)

Learning to program

Learn how to tell a computer what you want it to do.

"Please simulate the motion of N heavenly bodies, subject to Newton's laws of motion and gravity."



Naive ideal: Natural language instructions.



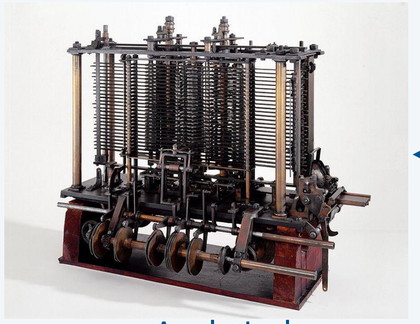
Prepackaged solutions (apps) are great when what they do is what you want.

Programming enables you to make a computer do anything you want. ← well, *almost* anything (stay tuned)

first programmer →



Ada Lovelace



Analytical Engine

← first computer

Programming: telling a computer what to do

Programming

- Is *not* just for experts.
- *Is* a natural, satisfying and creative experience.
- Enables accomplishments not otherwise possible.
- The path to a new world of intellectual endeavor.

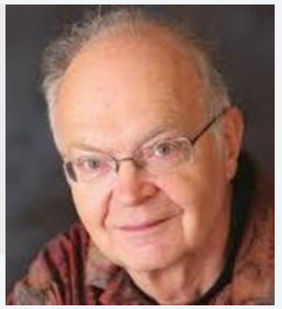
Challenges

- Need to learn what computers *can* do.
- Need to learn a programming *language*.



Telling a computer what to do

" Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do. "



– Don Knuth

Telling a computer what to do

Machine/Assembly language

- Easy for computer.
- Error-prone for human.

```
10: 8A00    RA ← mem[00]
11: 8B01    RB ← mem[01]
12: 1CAB    RC ← RA + RB
13: 9C02    mem[02] ← RC
14: 0000    halt
```

Adding two numbers

Natural language

- Easy for human.
- Error-prone for computer.



Actual newspaper headlines

—Rich Pattis

High-level language

- Some difficulty for both.
- An acceptable tradeoff.

```
for (int t = 0; t < 2000; t++)
{
    a[0] = a[11] ^ a[9];
    System.out.print(a[0]);
    for (int i = 11; i > 0; i--)
        a[i] = a[i-1];
}
```

Simulating an LFSR (see Prologue lecture)

But *which* high-level language?

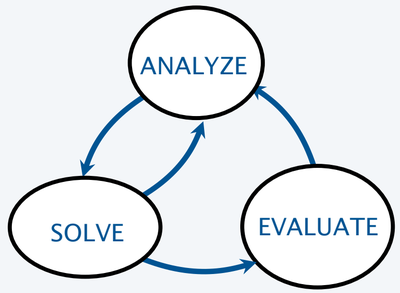


But first: Learn to think algorithmically.

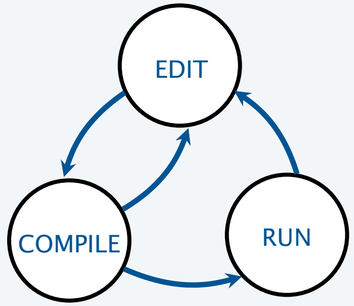
Algorithmic thinking

Learn how to analyze and solve computational problems at a level of abstraction .









A significant difference with coding : *No implementation details required.*

"coding" environment: using a set of pseudocode instructions to express a solution algorithmically.

An algorithm: A step-by-step process of solving a problem

Problem

- Find the sum of two numbers.

What we need: A vocabulary for describing an algorithm.

Algorithm

```
READ x
READ y
COMPUTE z AS x + y
DISPLAY z
```



1. Algorithmic Thinking

- What is Algorithmic thinking?
- Algorithm development
- A Pseudo Language
- From Algorithm to pseudocode

Algorithm development

as a three-step process, *with feedback*

1. ANALYZE

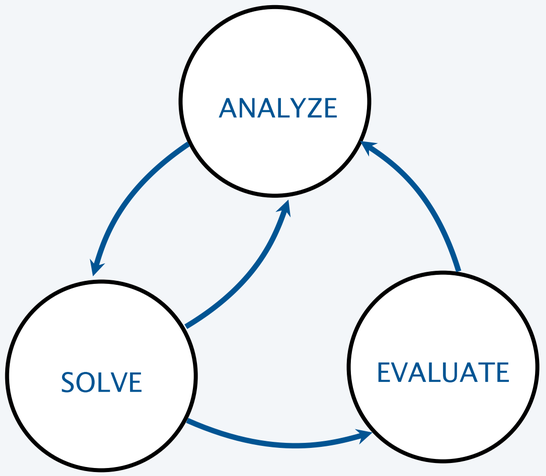
- Understand the problem you are solving.

2. SOLVE

- Develop a step-by-step algorithm
- Use a “language” to express it

3. EVALUATE

- Play a human computer to see how the algorithm works.
- Mistake? Go back to 1 and/or 2



Analyze, Solve, Evaluate

1. Problem 1 – Finding Duplicates

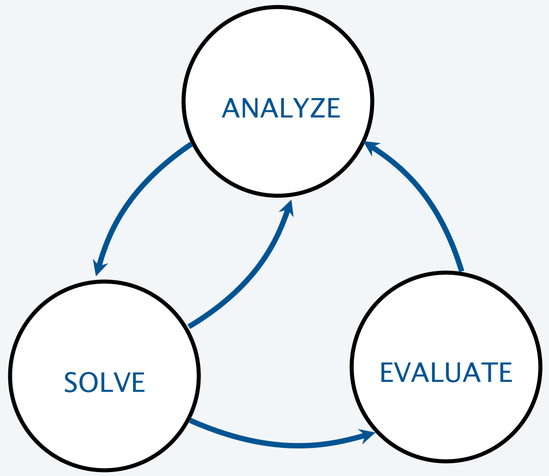
- Given a set of names, find out if there are duplicate names.

2. Problem 2 – Sorting a deck

- Sort a card deck by “natural order”

3. Problem 3 – Searching a list

- Find an item in a list

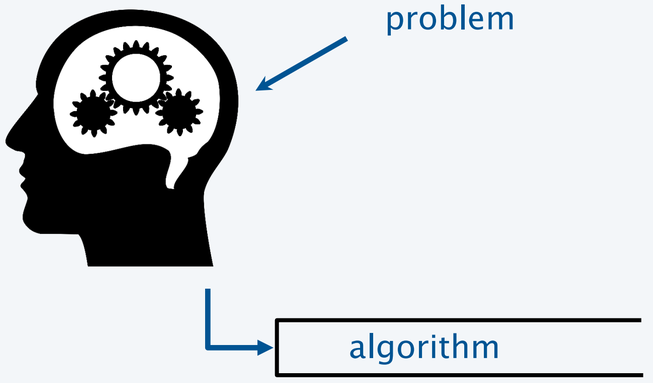


Computational thinking (CT)

is a set of **problem-solving methods** that involves expressing problems and their solutions.

Solution can be translated to an algorithm

Convert the algorithm to a program that can be solved using a computer.



Problem ➔ CT ➔ Algorithm

Thinking Computationally

- Fundamental Skill for everyone, not just computer scientists.
- Integration of computational ideas in other disciplines.

Stages of CT

1. Abstraction problem formulation.
2. Automation expressing a solution that a machine can execute.
3. Analyses solution execution and evaluation.

Stay tuned for examples of computational thinking.

Learning how to think computationally

CT is a **critical skill** for the anyone in the future.

Formulate solutions with **clarity**

Organize the instructions in a way **a computer can follow**.



Most disciplines are computational

Communicating

- Language of constructs that help **formulate the solution**.
- Knowledge know how to **disseminate**.

Language of Communication

- Clear instructions on **how to express**.
- Complete language constructs **allow construction of general solution**.
- Logical use of construct to **accurately describe the computation**.

Computational thinking is a recurring theme in this course.



1. Algorithmic Thinking

- What is Algorithmic thinking?
- Algorithm development
- A Pseudo Language
- From Algorithm to pseudocode

Pseudo Language

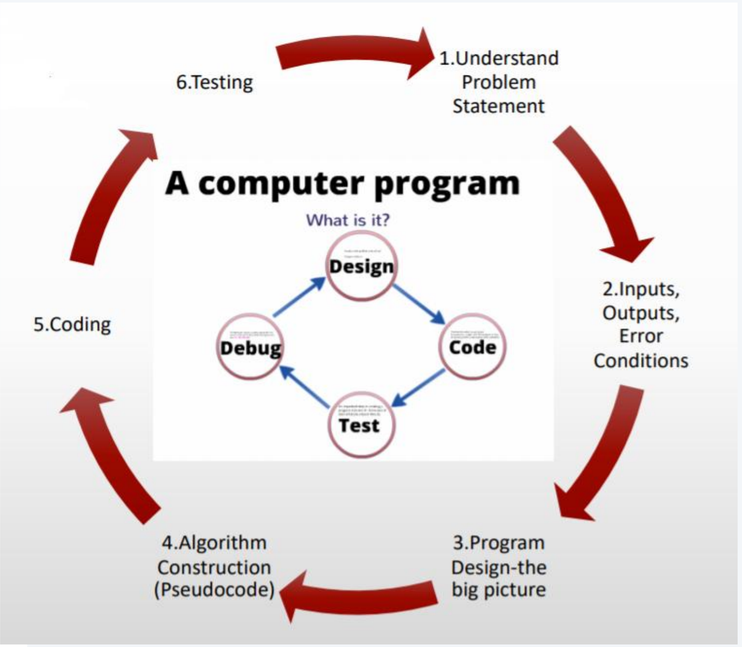
An informal high-level description of a computer program

Structure uses **same structural conventions** as a programming language.

Readability intended for **human reading** not machine reading.

Software Design Cycle

- Pseudo Language help **express the algorithm** in the design cycle.
- Coding help **avoid expensive errors** prior to coding.



Software Design Life Cycle

High-level components in Program Development

1. input how a program receives input data.
2. output how a program returns the results.
3. compute how a program performs its tasks.

Stay tuned on how input, output and compute work with a pseudo language and more details on converting an algorithm to pseudocode.

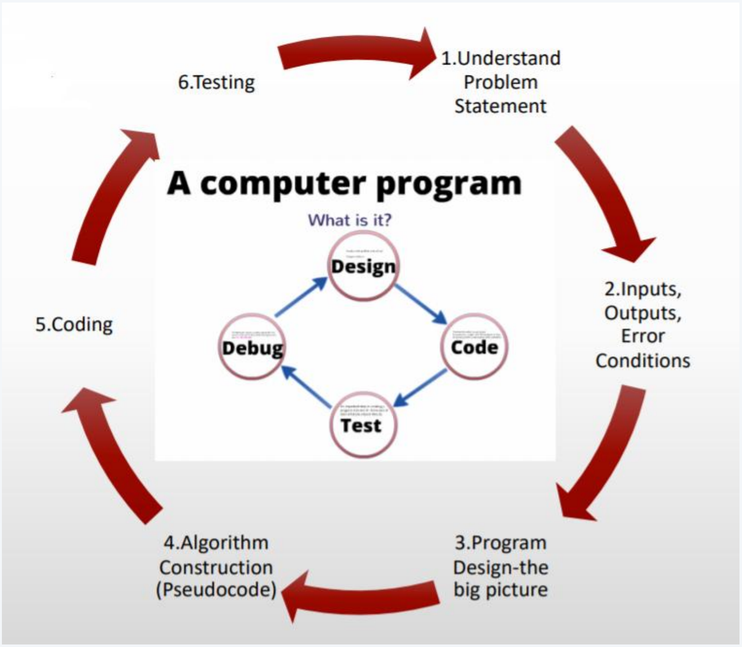
Properties of a Pseudo Language

A pseudo language enables high level description of a language

Implementation Details pseudocode **ignores implementation** details such as programming language.

Complete any program written in pseudo code **must be complete**.

Complete Program describes **entire logical process** of the solution and enable easy translation into an actual programming code.



Pseudocode Standards

- Not Universal pseudocode rules are **not universal**.
- Author-specific they depend on the author of a textbook or designer of the algorithm.
- **Our Standard** for the purpose of this course is that we adopt our “own” pseudo language.

We only use pseudocode as an easy transition into programming in Java.

Definitions

Define. Terms used in programming using a pseudo language

Definitions

<variable> an entity that may change its value (e.g. temperature)

<constant> a value that does not change (e.g. 4)

<expression> variables, constants and operators grouped together to show the value of something (e.g. x*5 + y). **Evaluates to a value.**

<Boolean expression> an expression that **evaluates to TRUE or FALSE** (e.g. n > 3)

<statement> a combination of expressions (e.g. COMPUTE x AS x + 3)

A Pseudo language vocabulary

I/O
READ
DISPLAY

Initialization computation
SET
COMPUTE

Flow control
IF-THEN
IF-THEN-ELSE
NESTED IF-THEN-ELSE

Repetition
WHILE-ENDWHILE
DO-WHILE
REPEAT-UNTIL
FOR-ENDFOR

Nested
NESTED-FOR
NESTED-FOR-IF

Increment/decrement
ADD
SUBTRACT

Comparisons
<
<=
>
>=
is
!=

Arithmetic Operations
+
-
*
/
%

Boolean operations
NOT
AND
OR

Terminate Program
HALT

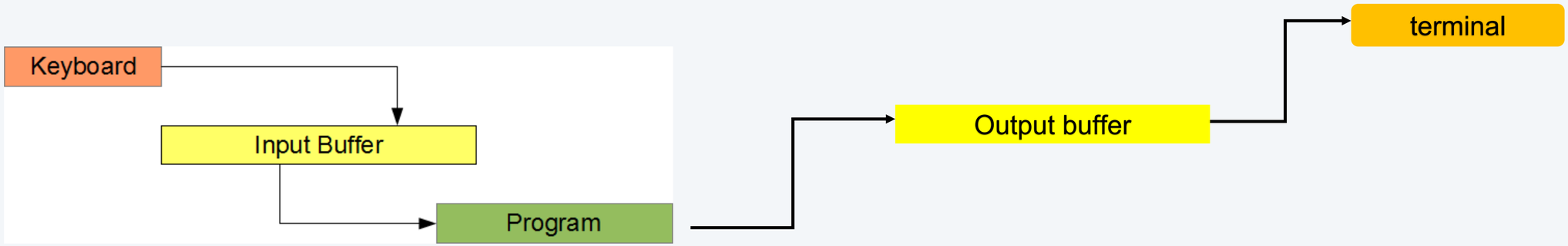
Your pseudocode will primarily consist of these constructs plus identifiers (names) that you make up. A *Pseudocode Reference Sheet* is posted for your use in this course.

INPUT and OUTPUT (I/O)

Q. How does input and output work?

A. Input. Read data from memory (RAM) or from an input device (keyboard/disk)

Output. Display/Write results to an output device (terminal/disk/printer)



There are variations of input/output that we will discuss later in the course

I/O - READ and DISPLAY

Q. How does READ and DISPLAY work (in pseudo language)?

A. READ. Reads a value into a variable

DISPLAY. Outputs a variable or constant

READ <variable>

DISPLAY <variable/constant>

Syntactical form

READ temperature

DISPLAY temperature

An Example

INITIALIZE

Q. How do we initialize a variable (a placeholder) in a program?

A. Using the SET statement in the pseudo language.

SET <variable> **TO** <constant | variable>

Syntactical Form

SET x **TO** 10

SET x **TO** y

An Example

COMPUTE

Q. How do we compute an expression in a program?

A. Using the COMPUTE statement in the pseudo language.

COMPUTE <variable> **AS** <expression>

Syntactical Form

COMPUTE x **AS** 10 * 12 + 20

COMPUTE x **AS** y + z

An Example

Flow Control - One-way Branching

Q. How does one-way branching work?

A. IF-THEN. One-way branching based on the condition being TRUE

```
IF <Boolean-expression> THEN
    <statement/statements>
ENDIF
```

Syntactical Form

```
IF (number > 0) THEN
    DISPLAY positive
ENDIF
```

An Example

Flow Control - Two-way Branching

Q. How does two-way branching work?

A. IF-THEN-ELSE. Two-way branching is based on Condition being TRUE or FALSE

```
IF <Boolean-condition> THEN
    <statement/statements>
ELSE
     <statement/statements>
ENDIF
```

Syntactical form

```
IF temperature < 40 THEN
    DISPLAY cold day
ELSE
    DISPLAY mild day
ENDIF
```

An Example

Flow Control - Multi-way Branching

Q. How does multi-way branching work?

A. NESTED IF-THEN-ELSE. multi-way branching based on Condition being TRUE or FALSE and more

```
IF <Boolean-condition> THEN
    <statement/statements>
ELSE
   IF <Boolean-condition> THEN
        <statement/statements>
   ELSE
       <statement/statements>
   ENDIF
ENDIF
```

A syntactical form

```
IF temperature < 40 THEN
    DISPLAY cold day
ELSE
    IF temperature >= 80 THEN
        DISPLAY hot day
    ELSE
        DISPLAY mild day
    ENDIF
ENDIF
```

3-way branching example

Iterations - WHILE loop

Q. How does WHILE loop work?

A. Initialization. (optional)

Boolean Expression. Evaluate a Boolean expression

IF TRUE. Execute a sequence of statements

Repeat

<initialization statements>

WHILE <Boolean-expression>

<statements if Boolean expression is true>

ENDWHILE

Syntactical Form

```
SET n TO 5
SET sum TO 0
WHILE n > 0
  COMPUTE sum AS sum + n
  SUBTRACT 1 FROM n
ENDWHILE
```

An example

Iterations - FOR loop

Q. How does FOR loop work?

A. Initialization. Evaluate an initialization statement

Boolean Expression. Evaluate a Boolean expression

IF TRUE. Execute a sequence of statements

Execute an increment statement

Repeat

```
FOR <iteration bounds>
    <statements>
ENDFOR
```

Syntactical Form

```
FOR each day of the week
  DISPLAY day of the week
ENDFOR
```

An Example

Iterations – REPEAT-UNTIL loop

Q. How does REPEAT-UNTIL loop work?

A. Initialization. (optional)

Execute a sequence of statements

Boolean Expression. Evaluate a Boolean expression

IF FALSE Repeat

```
REPEAT
    <statements>
UNTIL <Boolean expression>
```

Syntactical Form

```
SET n TO 10
REPEAT
   COMPUTE n AS n + 1
UNTIL n > 100
```

An Example

ADD and SUBTRACT

Q. How do we **increment** or **decrement** a variable?

A. Using the ADD or SUBTRACT statement in the pseudo language.

ADD <variable/constant> **TO** <variable>

SUBTRACT <variable/constant> **FROM** <variable>

Syntactical Form

ADD 1 **TO** x

ADD y **TO** x

SUBTRACT 1 **FROM** x

SUBTRACT x **FROM** y

Example

NESTED-FOR STATEMENTS

Q. How do we nest FOR statements ?

A. NESTED-FOR. Repeat a FOR loop within another FOR loop

```
FOR <iteration bounds>
   FOR <iteration bounds>
        <statements>
   ENDFOR
ENDFOR
```

Syntactical Form

```
FOR each week of month
   FOR each day of week
        DISPLAY week of month,  day of week
   ENDFOR
ENDFOR
```

example

NESTED-FOR-IF STATEMENT

Q. How do we nest FOR and IF statements ?

A. NESTED-FOR-IF. Check IF condition within a FOR loop

```
FOR <iteration bounds>
  IF <Boolean expression> THEN
    <statement/s>
  ELSE
     <statements>
  ENDIF
ENDFOR
```

Syntactical Form

```
FOR day of the month
  IF day is even THEN
     DISPLAY even day
  ELSE
     DISPLAY odd day
  ENDIF
ENDFOR
```

An Example

Comparison operators

Fundamental operations that are defined for each primitive type allow us to *compare* values.

- Operands: two expressions of the same type.
- Result: a value of type boolean.

operator	*meaning*	true	false
is	equal	2 is 2	2 is 3
!=	not equal	3 != 2	2 != 2
<	less than	2 < 13	2 < 2
<=	less than or equal	2 <= 2	3 <= 2
>	greater than	13 > 2	2 > 13
>=	greater than or equal	3 >= 2	2 >= 3

Examples

non-negative discriminant?	(b*b - 4.0*a*c) >= 0.0
beginning of a century?	(year % 100) is 0
legal month?	(month >= 1) AND (month <= 12)

% is the modulus operator, the remainder: 15 % 2 = 1



1. Algorithmic Thinking

- What is Algorithmic thinking?
- Algorithm development
- A Pseudo Language
- From Algorithm to pseudocode

Problem. Ellen and Ana are moving together. Determine how many cats they will have in total?

Solution Design.

Find out how many cats each person has.

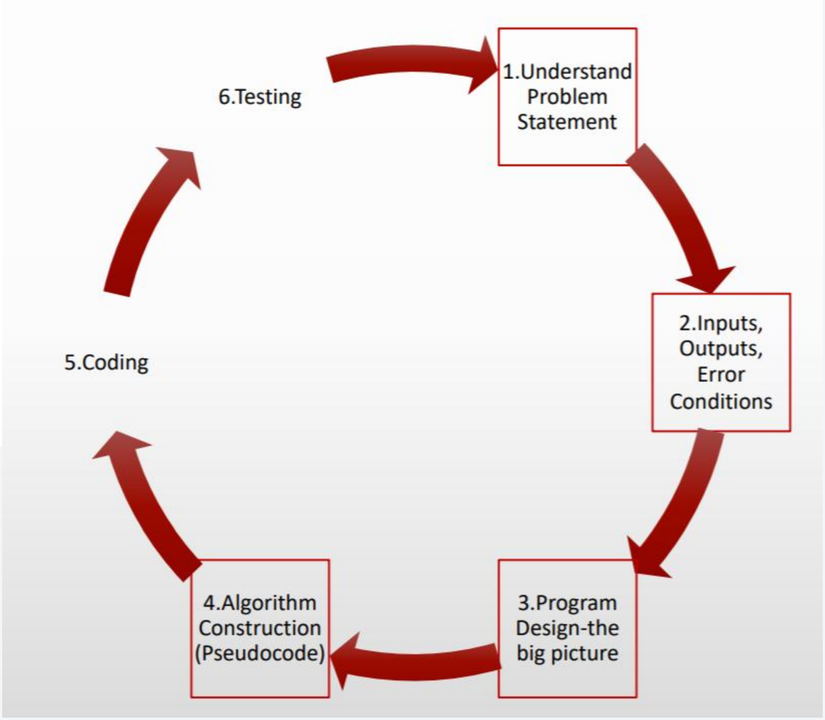
Compute and display the sum.

Identify.

Inputs. Ellen's number of cats, Ana's number of cats

Output. The total cats

Error conditions. Bad input (e.g. Ana's cats is -5)



Example Program 1

Problem. Ellen and Ana are moving together. Determine how many cats they will have in total?

Solution Design.

Find out how many cats each person has.

Compute and display the sum.

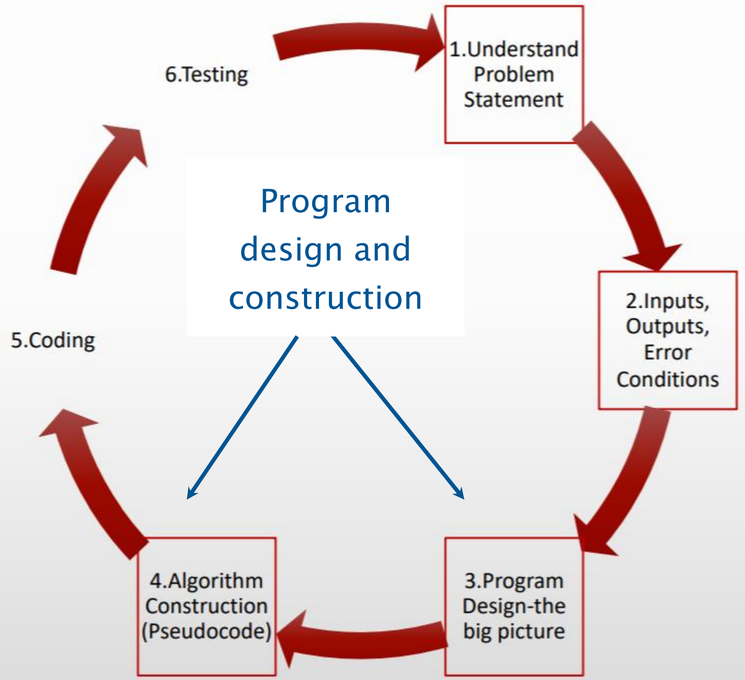
PSEUDOCODE:

READ EllensCats

READ AnasCats

COMPUTE sumOfCats **AS** EllensCats + AnasCats

DISPLAY sumOfCats



Example Program 1 (with error handling)

Problem. Ellen and Ana are moving together. Determine how many cats they will have in total?

Solution Design.

Find out how many cats each person has.

Handle error in input

Compute and display the sum.

ALGORITHM:

```
READ EllensCats
READ AnasCats
IF EllensCats < 0 OR AnasCats < 0 THEN
    DISPLAY ERROR
ELSE
    COMPUTE sumOfCats AS EllensCats + AnasCats
    DISPLAY sumOfCats
ENDIF
```

Test Cases

[input][output] pairs

[1, 3][4]

[0, 2][2]

[2, -5][ERROR]

[-3, 2][ERROR]

[-3, -4][ERROR]

Example Program 2

Problem. Calculate salary based on hours worked and hourly rate.

Solution Design.

Input: hoursWorked and ratePerHour

Output: totalPay

Error: negative inputs

ALGORITHM:

```
READ hoursWorked
READ ratePerHour
IF hoursWorked < 0 OR ratePerHour < 0 THEN
    DISPLAY ERROR
ELSE
    COMPUTE totalPay AS hoursWorked * ratePerHour
    DISPLAY totalPay
ENDIF
```

Test Cases

```
[input][output] pairs
[8, 10][ 80 ]
[0, 20][ 0 ]
[2, 0][ 0 ]
[-3, 20][ERROR]
[10, -20][ERROR]
```

Example Program 3

Problem. Determine if an integer is even or odd.

Solution Design.

Input: an integer

Output: even or odd

Error: none

Precondition: input is an integer

ALGORITHM:

READ number	← 1 operation
IF number % 2 is 0 **THEN**	← 1 operation
DISPLAY even	← 1 operation
ELSE	OR
DISPLAY odd	← 1 operation
ENDIF	Total of 3 operations

Test Cases

[input][output] pairs

[-2][even]

[0][even]

[2][even]

[3][odd]

Example Program 4

Problem. A cake up to 6” cost $10, 6”-12” cost $15, greater than 12” cost $20. Given the diameter of the cake, find the cost.

Solution Design.

Input: diameter of the cake (integer)

Output: the price

Error: diameters less than or equal to 0

Example Program 4


```
READ cakeSize
  IF cakeSize <= 0 THEN
      DISPLAY error
  ELSE
      IF cakeSize < 6 THEN
          SET price TO 10
      ELSE
          IF cakeSize >= 6 AND cakeSize <=12 THEN
              SET price TO 15
          ELSE
              SET price TO 20
          ENDIF
      ENDIF
      DISPLAY price
  ENDIF
```

Test Cases:

[input][output] pairs

[-3][error]

[0][error]

[5][10]

[6][15] *boundary condition*

[8][15]

[12][15] *boundary condition*

[13][20]

[15][20]

Example Program 5

Problem. Convert Fahrenheit to Celsius.

Solution Design.

Input: a temperature in Fahrenheit

Output: equivalent in Celsius

Error: temperature below absolute zero (-459.67F)

```
ALGORITHM:
    READ tempF
    IF tempF < -459.67 THEN
        DISPLAY ERROR: Temperature must be greater than -459.67
    ELSE
        COMPUTE tempC AS (tempF – 32) * 5/9
        DISPLAY tempC
    ENDIF
```

Test Cases

[input][output] pairs

[32][0]

[-459.68][error]

[-459.67][273.15]

[65][18.33]



INTRODUCTION TO
COMPUTER SCIENCE

Rutgers University

1. Algorithmic Thinking

- What is Algorithmic thinking?
- Algorithm development
- A Pseudo Language
- From Algorithm to pseudo code

INTRODUCTION TO COMPUTER SCIENCE

Rutgers University

1. Algorithmic Thinking and Design



https://introcs.cs.rutgers.edu



INTRODUCTION TO COMPUTER SCIENCE

Rutgers University

Extra slides

Quiz

Identify each of the following as variable, constant, arithmetic expression, Boolean expression or statement.

1. (x > 2)
2. 2*temperature
3. myGrade
4. AnaHouse
5. (temp < 0) or (temp > 10)
6. COMPUTE x AS x - 3

Quiz

Write an algorithm, using pseudocode, to solve the following problems. Identify input, output and error conditions.

1. Display “Hello world”
2. Add two numbers
3. Add two positive numbers
4. Find and display the larger of two numbers
5. Find and display the larger of three negative numbers
6. Compute and display the area of a circle given the radius
7. Display all even numbers from 1 to 100
8. Simulate an ATM machine that accepts deposits and withdrawals

Pseudocode

- **Display “hello world”**

  DISPLAY “hello world”

- **Add two numbers**

  READ num1

  READ num2

  COMPUTE sum AS num1 + num2

  DISPLAY sum

Pseudocode

- **Add two positive numbers**

```
READ num1
READ num2
IF (num1 <= 0) OR (num2 <= 0) THEN
    DISPLAY “ERROR”
ELSE
    COMPUTE sum AS num1 + num2
    DISPLAY sum
ENDIF
```

5 or 6 operations

Pseudocode

- **Find (and display) the larger of two numbers**

```
READ num1
READ num2
IF num1 > num2 THEN
   DISPLAY num1
ELSE
   DISPLAY num2
ENDIF
```

Pseudocode

- **Find (and display) the largest of three negative numbers**

```
READ num1
READ num2
READ num3
IF (num1 >= 0) OR (num2 >= 0) OR (num3 >= 0) THEN
        DISPLAY ERROR
ELSE
        SET largest TO num1
        IF (num2 > largest) THEN
            SET largest TO num2
        ENDIF
        IF (num3 > largest) THEN
            SET largest TO num3
        ENDIF
        DISPLAY largest
ENDIF
```

Pseudocode

- **Compute (and display) the area of a circle given the radius**

```
READ radius
IF (radius <= 0) THEN
    DISPLAY ERROR
ELSE
    COMPUTE area AS 3.1415 * radius * radius
    DISPLAY area
ENDIF
```

Pseudocode

Display all even numbers from 1 to 100

```
SET num TO 2
WHILE (num <= 100)
    DISPLAY num
    ADD 2 TO num
END WHILE
```

```
SET num TO 1
WHILE (num <= 100)
    IF num % 2 = 0 THEN
        DISPLAY num
    ENDIF
    ADD 1 TO num
END WHILE
```