Five-Step Problem Solving Process

- Computers only perform the steps we specify in detail.
- Computers can perform the tasks accurately with fantastic speed.
- Computers never get bored.
- Procedure for problem solving:
  1. State the problem clearly.
  2. Describe the input and output information.
  3. Work the problem by hand for a simple set of data.
  4. Develop a solution that is general in nature.
  5. Test the solution with a variety of data sets.

Computing an average

- **First step**: state the problem clearly, clear concise problem statement
  - Compute the average of a set of experimental data values.
- **Second step**: Describe the information needed to solve the problem and identify the values to be computed.
  - Input/output or I/O
  - Input=the list of experimental data values
  - Output=the average of the data values
- **Third step**: work the problem using a simple set of data.
  - 23.43+37.43+34.91+28.37+30.62=154.76 154.76/5=30.95
- **Fourth step**: Describe in general terms the operations you performed by hand.
  - Algorithm: the sequence of operations that solves the problem.
  - Top-down: Start at the **top** with the original problem and break it **down** into smaller problems that can be addressed separately.
  - Decomposition: identify the pieces of the problem that need to be solved sequentially
  - Stepwise Refinement: successively refine each smaller piece of the solution using more detail. Details are introduced as we begin the refinement of our algorithm.
  - Pseudocode: presents algorithm steps in a series of English-like statements.

- Decomposition:

Read the data values and sum them.
Divide the sum by the number of data values.
Print the average.

- Refinement
  - The refinement of the decomposition for this example uses pseudocode to specify the details or to outline the steps of the algorithm.
    1. Set the sum of the values to 0.
    2. Set a count of the values to 0.
    3. As long as there are more data values,
       1. Add the next data value to the sum
       2. Add 1 to the count.
    4. Divide the sum by the count to get the average.
    5. Print the average.

```
pro comput
;  This program computes and prints the average of a set of experimental data values.
sum=0.0
count=0
read,x,prompt='Enter data value='
read_data: print,x
if x ne 0.0 then begin
  sum=sum+x
  count=count+1
  read,x,prompt='Enter data value'
  goto, read_data
endif
averg=sum/float(count)
print,'average',averg
end
```

23.43+37.43+34.91+28.37+30.62=154.76 154.76/5=30.95

Constants and Variables

- Constants: value doesn't change. Pi
- Variables: A variable represents a memory location that is assigned a name. The memory location contains a value. We reference that value with the name assigned to the memory location.
  - Amount 36.84 Volume 183.0
- Each variable must have different name, which you provide in your program.
- First character of a name must be alphabetic.
- Uppercase/lowercase?

Data Types

Data Type	Examples
Integers	32 -7
Real values (floating point)	-15.45. 0.004
Double-precision values	3.141592653 1.000000006
Complex values	1-2i 5i
Character values	'velocity'. 'report_3'
Logical Values (Boolean)	True False

Integer values: no fractional portion and no decimal point.

Real values: contain a decimal point and may or may not have digits beyond the decimal point. 1.

- A memory location can contain only one type of value.
- The type of value stored in a variable is specified by two methods.
  - Implicit typing
    - X=1 x=1.0 x='a'
  - Explicit typing
    - Statements are used to specify the variable type. Make_array=(/float,/int,/double,/char)
- It is helpful to select a variable name that is descriptive of the value being stored.

Scientific Notation

- When a real number is very large or very small, decimal notation does not work satisfactorily.
- Number between 0 and 10 (mantissa) multiplied by a power of 10 (exponent)
- Limits on the number of significant positions in the mantissa and on the size of the exponent set by computer architecture.

Decimal	Scientific	Exponential
3,876,000,000	$3.876 \text{ X } 10^{9}$	0.3876E10
0.0000010053	$1,0053 \text{ X } 10^{-6}$	0.10053E-05
-8,030,000	$-8.03 \text{ X } 10^{6}$	-0.803E07
-0.000157	$-1.57 \text{ X } 10^{-4}$	-0.157E-03

Arithmetic Operations

- Assignment statement: variable name=expression
- Constant: PI=3.141593
- Variable must be on left hand side of equal sign.
- The term “initialized” is used to refer to the first value assigned to a variable in a program.
- Variables can only store one value at a time.
  - Width=36.7
  - Width=105.2
  - The second statement replaces the the value with the new value of 105.2 and the first value is lost.
  - Temp1=-52.6
  - Temp2=Temp1
  - The value in temp1 is not lost.

IDL expressions for the basic arithmetic operations

Operation	Algebraic Form	IDL
Addition	A + B	A + B
Subtraction	A – B	A - B
Multiplication	A X B	A*B (x is a char)
Division	$\frac{A}{B}$	A/B (one line)
Exponentiation	A^3	A^3

Order of operations in Arithmetic Expressions: PEMDAS
Area=pi*radius^2
exponentiation or multiplication first give different answers

count=count+1
Looks invalid algebraically.

Priorities of Arithmetic Operations

Priority	Operation
1	Parentheses
2	Exponentiation
3	Multiplication and division
4	Addition and subtraction

When two operations are on the same priority level, as in addition and subtraction, operations are performed from left to right.

Parentheses placement is important.
Use extra parentheses if it helps.

Example: Real root of the quadratic equation

$X1 = \frac{-B+\sqrt{B^2-4AC}}{2A}$ where A,B and C are coefficients of the quadratic equation $A^2 + BX + C = 0$

X1=(-B+sqrt(B^2-(4.0*A*C)))/(2.0*A)

Omitting necessary parentheses results in incorrect calculations.

Using extra parentheses is permissible even though they may not be needed.

Break a long statement into several smaller statements

discr=B^2$-4.0 * A * C$

X1=(-B+sqrt(discr))/(2.0*A)

Do problems 2.6, 2.7,2.8,2.9 in Fortran Etter

Intrinsic Functions: Built in computer functions, called intrinsic functions, are available to handle routine computations.

The name of the function is followed by the input to the function called the argument.
result=ABS(X) result=ALOG10(X)

ABS	Returns absolute value of argument	CEIL	Returns the closest integer greater than or equal to its argument	FLTARR	creates a floating-point vector or array of the specified dimensions. Values are 0
ACOS	Returns arc cosine of X in radians	DOUBLE	Converts X into a double precision floating point value	INDGEN	returns an array with the specified dimensions. Each element of the array is set to the value of its one-dimensional subscript.
ALOG	Returns natural log of argument	FINDGEN	Creates a floating point array of the specified dimension X. Each element of the array is set to the value of its one dimensional subscript	INTARR	returns an integer vector or array. Each element set to 0
ALOG10	Base 10 logarithm of X	FINITE	Identifies whether or not the argument is finite	INTERPOL	performs linear, quadratic, or spline interpolation on vectors with a regular or irregular grid.
ASIN	Returns arc sine of X in radians	FIX	Converts the argument to an integer type	JULDAY	calculates the Julian Date (which begins at noon) for the specified date.
ATAN	Returns arc tangent of X in radians	FLOAT	Converst x to a single precision floating point value	LONG	returns a result equal to *x* converted to a longword integer type.
EXP	Natural exponential function of x	FLOOR	Returns the closest integer less than or equal to its argument	MAX	returns the value of the largest element of *Array*.

Result=MEAN(X) *Result* = STRMID(*Expression*, *First_Character* , *Length)* *Result* = STRSPLIT(*String* , *Pattern,/extract)*

MEAN	The MEAN function computes the mean of a numeric vector.	**ROUND**	rounds the argument to its closest integer.	**STRING**	returns its arguments converted to string type
MEDIAN	computes the median value of x	SIN	returns the trigonometric sine of X.	STRMID	The STRMID function extracts one or more substrings from a string expression. Each extracted string is the result of removing characters from the input string.
MIN	returns the value of the smallest element of *Array*.	SIZE	returns size and type information for its argument	STRSPLIT	splits its input *String* argument into separate substrings, according to the specified delimiter
MOMENT	computes the mean, variance, skewness, and kurtosis of a sample population contained in an *n*-element vector *X*.	SORT	returns a vector of subscripts that allow access to the elements of *Array* in ascending order.	STRLEN	returns an array, with specified dimensions, of double-precision floating-point values that represent times in terms of Julian dates.
N_ELEMENTS	returns the number of elements in x	SQRT	computes the square root of *X*.	STRTRIM	removes leading and/or trailing blank from the input *String*
REVERSE	reverses the order of one dimension of an array.	STDDEV	computes the standard deviation of an *n*-element vector.	TIMEGEN	returns the length of its string-type argument
ROTATE	returns a rotated and/or transposed copy of *Array*.	STRARR	returns a string array containing zero-length strings.	TOTAL	returns the sum of the elements of *Array*

Debugging Aids

- Print the values that you read in the program, immediately after reading them. Make sure that the values you want to give the variables are begin used.
- Double check the placement of parentheses. Be sure to have the same number of left parentheses as right parentheses.
- Make sure you have spelled variable names correctly.
- Be sure that arguments of functions are in the correct units.
- Be sure you do not have the characters I and 1 or the characters O and 0 interchanged.
- If you can’t find your error, start program over with a clean slate.

Control Structures

- Statements that allow us to control the sequence of the steps being executed.
- This control is achieved through statements that allow us to **select different paths** through our program and statements that allow us to **repeat certain parts** of our program.
- **Sequence**: a set of steps that are performed sequentially.
- **Selection**: a comparison is performed to determine which steps are to be performed next.
- A comparison tests a condition that can be **evaluated as true or false**. If the condition is true, then a step or group of steps is performed.
- An additional clause called an **else clause** that specifies an alternate set of steps to perform if the condition is false.
- **Else if clause** can test for multiple conditions.

Control Structures

- **Repetition**: Allows us to use loops, which are sets of steps that are repeated.
- *While loop*: repeats the steps as long as a certain condition is true.
- *Counting loop*: repeats the steps a specified number of times.
- The **counter** represents the number of times that the loop has been executed.
- Inside the loop the counter is **incremented**
- The steps within a loop can contain **sequential** steps, **selection** steps or other **repetition** steps.

Relational Operators

Relational Operator	Python	Interpretation
eq	==	Equal to
ne	!=	Not equal to
lt	<	Less than
le	<=	Less than or equal to
gt	>	Greater than
ge	>=	Greater than or equal to

Compound logical expression

Logical operators or and not

Logical Operators

A	B	.NOT. A	A.AND.B	A.OR.B
False	False	True	False	False
False	True	True	False	True
True	False	False	False	True
True	True	False	True	True

- Cloud base height, cloud thickness

- Clouds that are thin and high — cthick lt 3 and cbase gt 10
- Clouds that are thin or high — cthick lt 3 or cbase gt 10

IF (selection)

```
IF (logical expression) THEN
  statement 1
  .
  .
  .
  statement n
END IF
```

```
IF (logical expression 1) THEN
  statement 1
  …
  statement n
  IF (logical expression 2) THEN
    statement n+1
    …
    statement m
  END IF
  statement m+1
  …
  statement p
END IF
statement q
```

ELSE (selection)

```
IF (logical expression 1) THEN
  statement 1
  …
  statement n
ELSE
  statement n+1
  ...
  statment m
END IF
```

```
IF (logical expression 1) THEN
  statement 1
  …
  statement m
ELSE IF (logical expression 2) THEN
  statment m+1
  ...
  statment n
ELSE IF (logical expression 3) THEN
 statement n+1
 …
 statement p
ELSE
  statement p+1
  ...
  statement q
END IF
```

Self-Test

A = 2.2 B = -1.2 i=5 DONE=.TRUE.

1. A .LT. B	A – B .GE. 6.5
3. i .NE. 5	A + B .GE. B
5. .NOT. (A .EQ. 2*B)	i .LE. i - 5
7. (A .LT. 10.0) .AND. (B .GT. 5.0)	(ABS (i) .GT. 2) .OR. DONE

If time is greater than 5.0 increment time by 0.5 if time gt 5.0 then time=time+0.5

When the square root of poly is greater than or equal to 8.0, print the value of poly if sqrt(poly) ge 8.0 then print,poly

if dist is less than 50.0 or time is greater than 10.0,
increment time by 1.0. Otherwise, increment time by 0.5

```
if dist lt 50.0 or time gt 10.0 then begin
  time=time+1.0
endif else begin
  time=time+0.5
endelse
```

While Loop (repetition)

```
DO WHILE ( condition )
   statement 1
   …
   statement m
END DO
statement p
```

Iterative or Counting Loop (repetion)

```
DO index = initial, limit, increment
   statement 1
   …
   statement n
END DO
Statement n+1
```

Nested loops

```
DO i=1,5
    DO j=1,8
        DO k=2,10,2
          …..
        END DO
    END DO
END DO
```

```
DO j=1,5
    DO j=1,8
       …
    END DO
END DO
```

```
DO i=1,5
    DO k=1,8
      …
    END DO
    DO k=2,10,2
      …
    END DO
END DO
```

```
DO i=1,5
    DO j=1,8
      …
      i=i+1
    END DO
END DO
```

- The index of the for loop must be a variable, but it may be either real or integer.
- The increment can be either positive or negative, but it cannot be zero. Default is 1
- The value of the index should not be modified by other statements during the execution of the loop.

```
for i=0,10 do begin
  x=x+1
endfor
```

```
for i=0,10,2 do begin
   x=x+1
endfor
```

```
while x lt 10 do begin
  x=x+1
endwhile
```