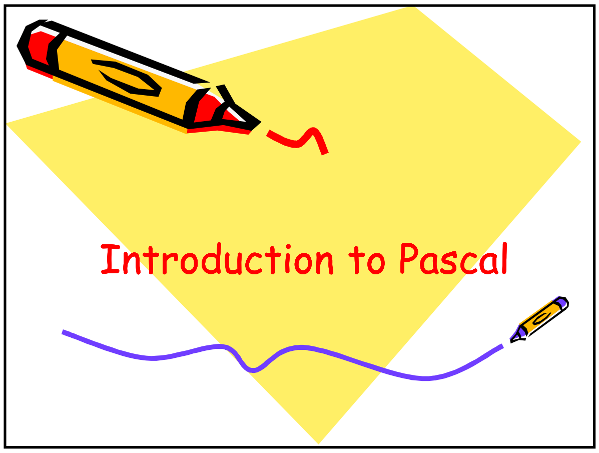
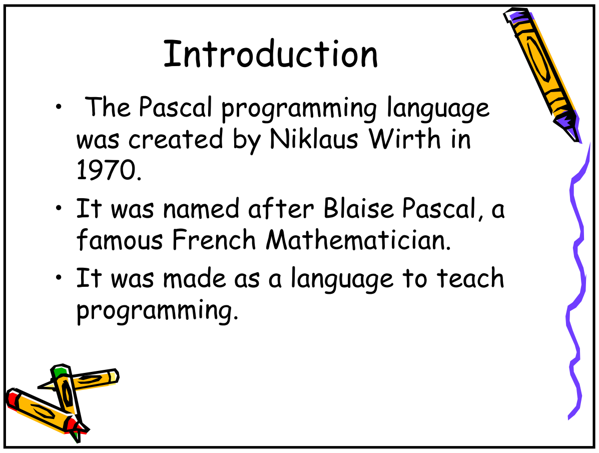
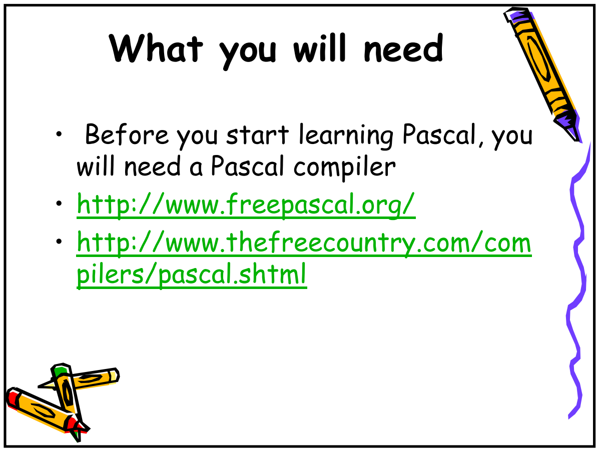


Introduction to Pascal



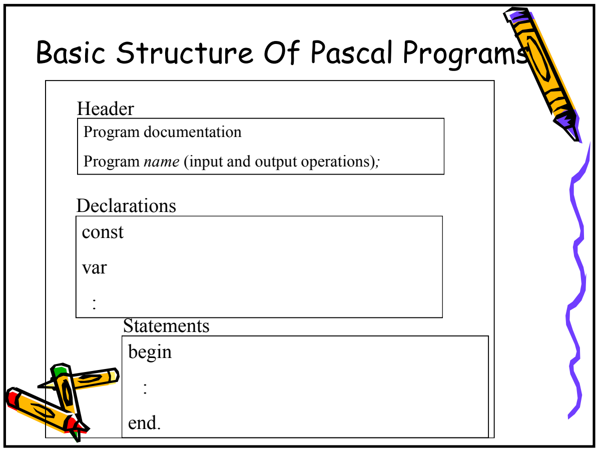
Introduction

- The Pascal programming language was created by Niklaus Wirth in 1970.
- It was named after Blaise Pascal, a famous French Mathematician.
- It was made as a language to teach programming.



What you will need

- Before you start learning Pascal, you will need a Pascal compiler
- http://www.freepascal.org/
- http://www.thefreecountry.com/compilers/pascal.shtml



Basic Structure Of Pascal Programs

Header

Program documentation

Program *name* (input and output operations);

Declarations

const

var

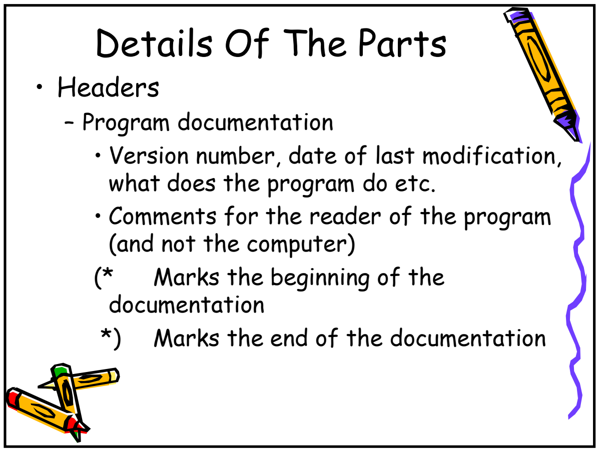
:

Statements

begin

:

end.

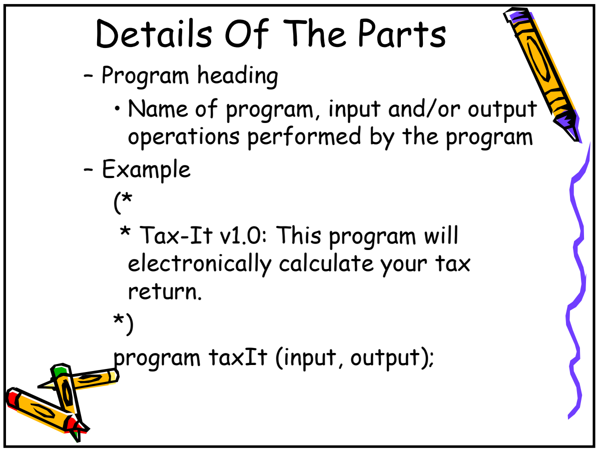


Details Of The Parts

- Headers
  - Program documentation
    - Version number, date of last modification, what does the program do etc.
    - Comments for the reader of the program (and not the computer)

    (* Marks the beginning of the documentation

    *) Marks the end of the documentation



Details Of The Parts

- Program heading
  - Name of program, input and/or output operations performed by the program
- Example

```
(*
 * Tax-It v1.0: This program will
  electronically calculate your tax
  return.
*)
program taxIt (input, output);
```

Details Of The Parts

- Declarations
  - List of constants and variables

Details Of The Parts

- Statements
  - The instructions in the program that actually gets stuff done
  - They tell the computer what to do as the program is running
  - Each statement is separated by a semicolon ";"

Variables

- Set aside a location in memory
- Used to store information (temporary)
- Types:
  - integer – whole numbers
  - real – whole numbers and fractions
  - char – alphabetic, numeric and miscellaneous symbols
  - boolean – true or false values

Variables

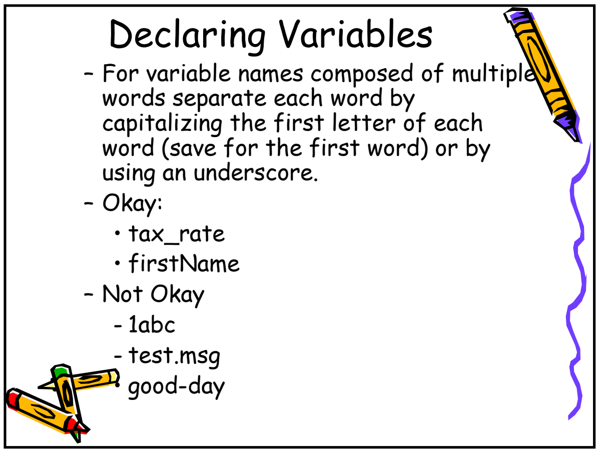
- Usage:
  - Declaration
  - Accessing or assigning values to the variables

Declaring Variables

- Sets aside memory
- Memory locations addressed through the name
- Naming conventions
  - Should be meaningful
  - Any combination of letters, numbers or underscore (can't begin with a number and shouldn't begin with an underscore)

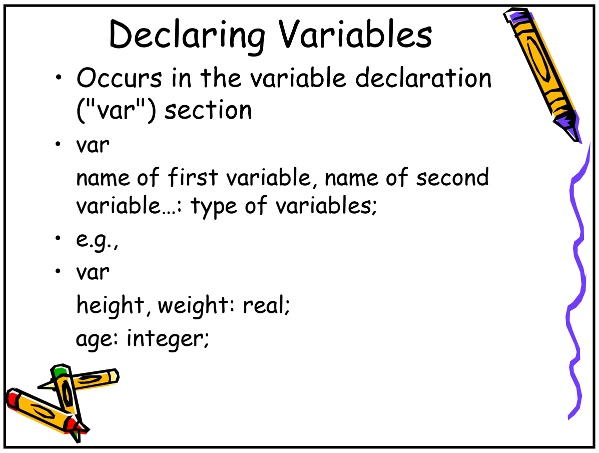
Declaring Variables

- Can't be a reserved word e.g., program, begin, end (see Appendix B)
- Avoid using words with an existing meaning e.g., integer, real, boolean, write, writeln, read, readln
- Avoid distinguishing variable names only by case



Declaring Variables

- For variable names composed of multiple words separate each word by capitalizing the first letter of each word (save for the first word) or by using an underscore.
- Okay:
  - tax_rate
  - firstName
- Not Okay
  - 1abc
  - test.msg
  - good-day



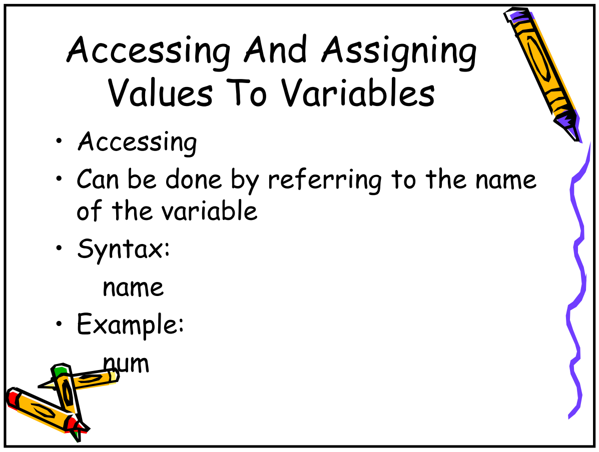
Declaring Variables

- Occurs in the variable declaration ("var") section
- var

  name of first variable, name of second variable…: type of variables;
- e.g.,
- var

  height, weight: real;

  age: integer;

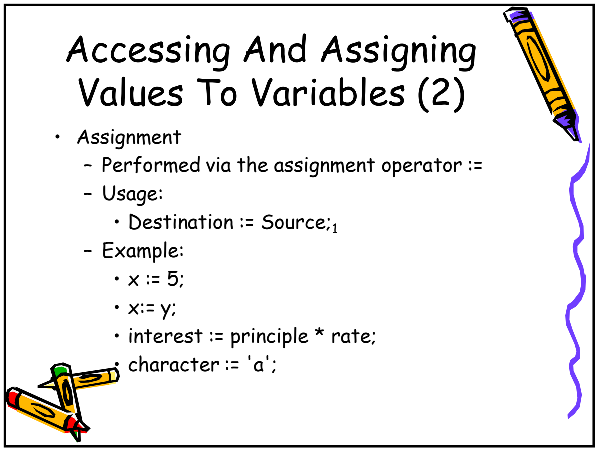


Accessing And Assigning Values To Variables

- Accessing
- Can be done by referring to the name of the variable
- Syntax:

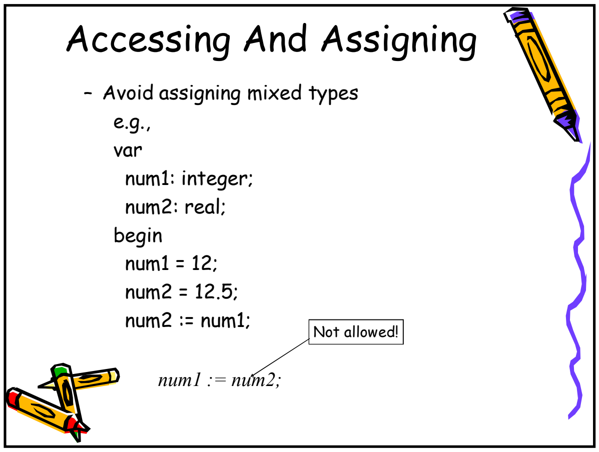
  name
- Example:

  num



Accessing And Assigning Values To Variables (2)

- Assignment
  - Performed via the assignment operator :=
  - Usage:
    - Destination := Source;[1]
  - Example:
    - x := 5;
    - x:= y;
    - interest := principle * rate;
    - character := 'a';



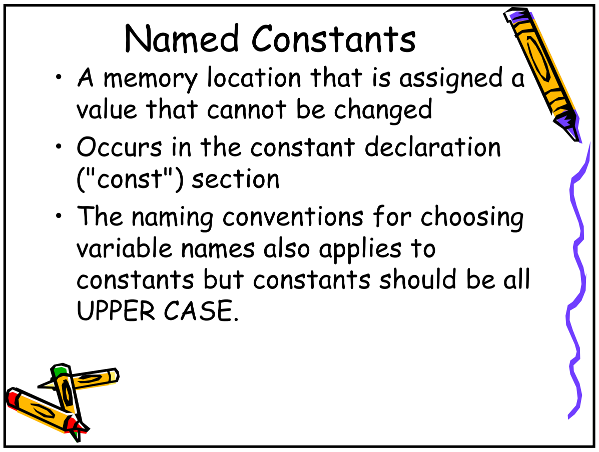
Accessing And Assigning

- Avoid assigning mixed types

```
e.g.,
var
  num1: integer;
  num2: real;
begin
  num1 = 12;
  num2 = 12.5;
  num2 := num1;
```

Not allowed!

num1 := num2;



Named Constants

- A memory location that is assigned a value that cannot be changed
- Occurs in the constant declaration ("const") section
- The naming conventions for choosing variable names also applies to constants but constants should be all UPPER CASE.

Named Constants

- Syntax:
- const

  NAME OF FIRST CONSTANT = value of first constant;

  NAME OF SECOND CONSTANT = value of second constant; etc.

Named Constants

- Examples:
- const

  TAXRATE = 0.25;

  SAMPLESIZE = 1000;

  YES = True;

  NO = False;

Purpose of Named Constants

- 1) Makes the program easier to understand
- e.g.,
- begin
- population_change := (0.1758 – 0.1257) * current_population;
- Vs.
- const
- BIRTHRATE = 0.1758;
- DEATHRATE = 0.1257;
- begin

  population_change := (BIRTHRATE - DEATHRATE) *
  current_population;

Purpose of Named Constants

- 2) Makes the program easier to maintain
- If the constant is referred to several times throughout the program.
- const
- BIRTHRATE = 0.1758;
- DEATHRATE = 0.1257;
- begin
-

BIRTHRATE

BIRTHRATE

DEATHRATE DEATHRATE BIRTHRATE

BIRTHRATE BIRTHRATE

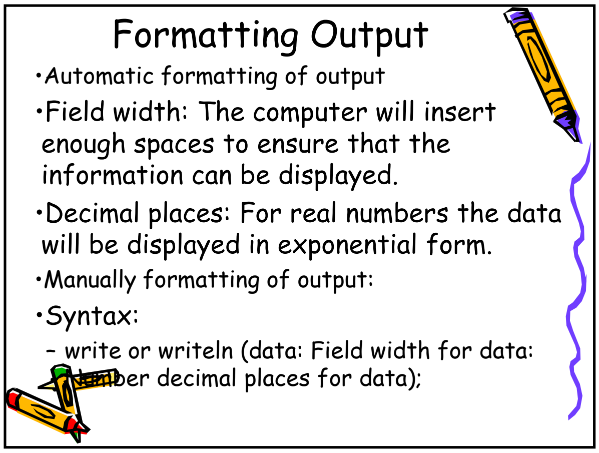
BIRTHRATE

Output

- Displaying information onscreen
- Done via the write and writeln statements
- Syntax (either write or writeln):
- write ('text message');
- or
- writeln('text message');
- write(name of variable or constant);
- or
- writeln (name of variable or constant);
- write('message', name of variable, 'message'…);
- or
- writeln('message', name of variable, 'message'…);

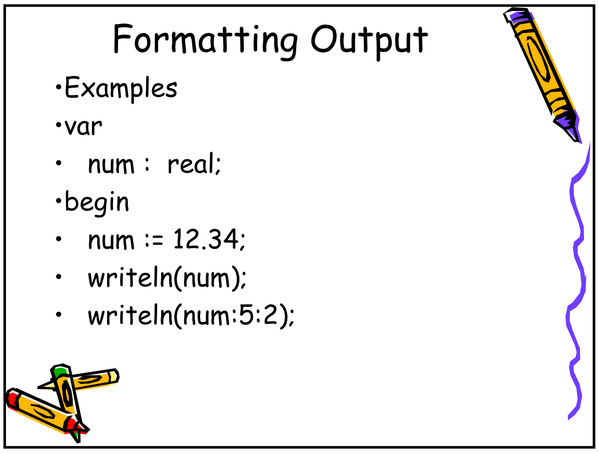
Output (2)

- Examples:
- var
- num : integer;
- begin
- num := 10;
- writeln('line1');
- write('line2A');
- writeln('line2B');
- writeln(num);
- writeln('num=',num);



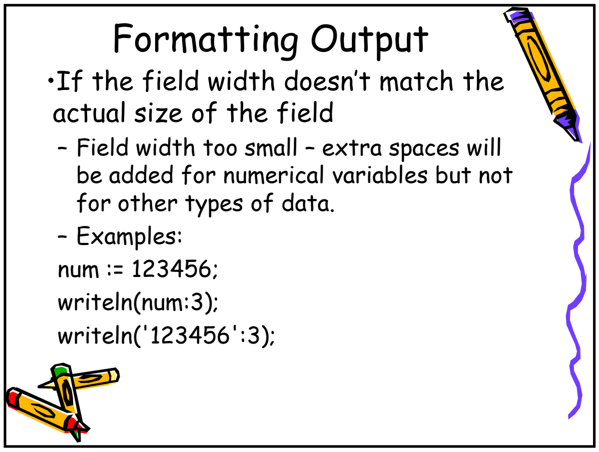
Formatting Output

- Automatic formatting of output
- Field width: The computer will insert enough spaces to ensure that the information can be displayed.
- Decimal places: For real numbers the data will be displayed in exponential form.
- Manually formatting of output:
- Syntax:
  - write or writeln (data: Field width for data: Number decimal places for data);



Formatting Output

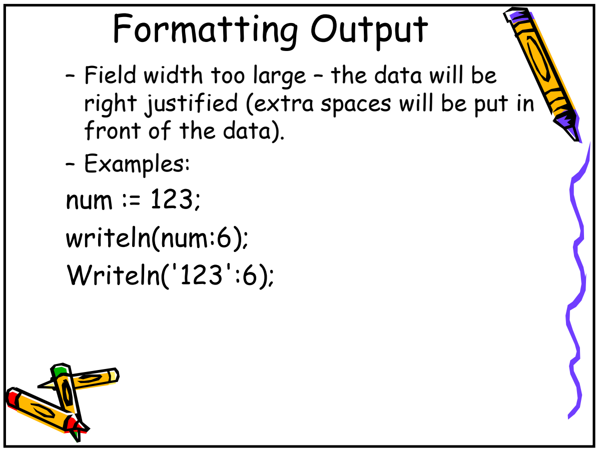
- Examples
- var
- num : real;
- begin
- num := 12.34;
- writeln(num);
- writeln(num:5:2);



Formatting Output

- If the field width doesn't match the actual size of the field
  - Field width too small – extra spaces will be added for numerical variables but not for other types of data.
  - Examples:

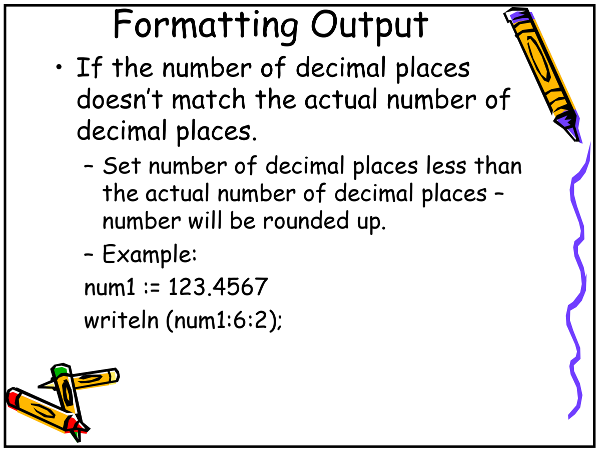
```
num := 123456;
writeln(num:3);
writeln('123456':3);
```



Formatting Output

- Field width too large – the data will be right justified (extra spaces will be put in front of the data).
- Examples:

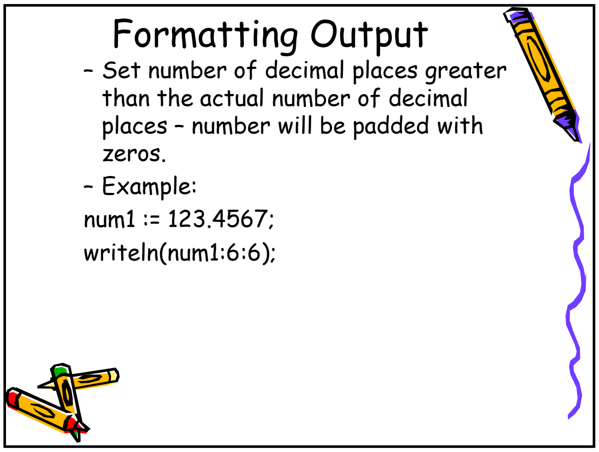
```
num := 123;
writeln(num:6);
Writeln('123':6);
```



Formatting Output

- If the number of decimal places doesn't match the actual number of decimal places.
  - Set number of decimal places less than the actual number of decimal places – number will be rounded up.
  - Example:

```
num1 := 123.4567
writeln (num1:6:2);
```



Formatting Output

- Set number of decimal places greater than the actual number of decimal places – number will be padded with zeros.
- Example:

```
num1 := 123.4567;
writeln(num1:6:6);
```

A Larger Example

```
program out1;
var
   num1 : integer;
   num2 : real;
begin
   num1 := 123;
   num2 := 123.456;
   writeln('Auto formatted by Pascal', num1, num2);
   writeln('Manual format':13, num1:3, num2:7:3);
   writeln('Manual not enough':13, num1:2, num2:6:3);
   writeln('Manual too much':16, num1:4, num2:8:4);
end.
```

Input

- The computer program getting information from the user
- Done via the read and readln statements
- Syntax:
- (single input)
- read (name of variable); or readln (name of variable);
- (multiple inputs)
- read (nv1, nv2…); or readln (nv2, nv3…);

Input

- Examples:

```
var
     num1, num2 : integer
begin
     read (num1);
     read (num2);
     read (num1, num2);
```

Input: Read Vs. Readln

- Both:
  - Reads each value inputted and matches it to the corresponding variable.
- Read
  - If the user inputs additional values they will remain
- Readln
  - Any additional values inputted will be discarded

Input: Read Vs. Readln (An example)

- e.g., read1.p

```
write('Input some integers making sure to separate each one with a space ');
write('or a new line: ');
read (num1, num2);
write('Input some integers making sure to separate each one with a space ');
write('or a newline: ');
read(num3, num4);
```

Extra Uses Of Readln

- To filter out extraneous input
- As an input prompt
- e.g.,

```
writeln('To continue press return');
readln;
```

Common Programming Errors

- Syntax/compile errors
- Runtime errors
- Logic errors

Some Useful Functions

Description	Input type	Type of result	Example
absolute value	integer	integer	abs(-2) = 2
	real	real	abs(-2.2) = 2.2
rounding	real	integer	round(2.6) = 3
truncation	real	integer	trunc(2.6) = 2
squaring	integer	integer	sqr(2) = 4
	real	real	sqr(1.1) = 1.21
square root	integer	real	sqrt(4) = 2.00

Questions