

```

        IF delay LT 0.0 THEN BEGIN
            cursor, x, y, /down
        ENDIF ELSE WAIT, delay
    ENDFOR

```

```

ENDIF ELSE IF nchar=3 THEN BEGIN

```

```

    nx = s[0]
    nz = s[1]
    nt = s[2]

```

```

    val = data

```

```

    IF KEYWORD_SET(profile) THEN BEGIN

```

```

        FOR i=0, nx-1 DO BEGIN

```

```

            FOR j=0, nz-1 DO BEGIN

```

```

                FOR t=0, nt-1 DO BEGIN

```

```

                    val[i,j,t] = profile[i,j,t]

```

```

                ENDFOR

```

```

            ENDFOR

```

```

        ENDFOR

```

```

    ENDIF

```

```

    IF NOT KEYWORD_SET(yr) THEN yr = [MIN(val),MAX(val)]

```

```

    IF KEYWORD_SET(contour) THEN BEGIN

```

```

        IF KEYWORD_SET(bw) THEN BEGIN

```

```

            LOADCT, 0

```

```

        ENDIF ELSE BEGIN

```

```

            loadct, 39

```

```

        ENDELSE

```

```

        device, decomposed=0

```

```

        ;safe_colors, /first

```

```

    FOR i=0, nt-1 DO BEGIN

```

```

        contour, reform(val[*,*,i]), chars=chars, zr=yr, zstyle=1, $

```

```

        /fill, nlev=50, color=color, $

```

```

        title="time = "+strtrim(string(i),2), _extra=_extra

```

```

    IF delay LT 0.0 THEN BEGIN

```

```

        cursor, x, y, /down

```

```

    ENDIF ELSE WAIT, delay

```

Introduction to Programming

Dr Ben Dudson
University of York

Programming course (this term)

Aim

- Learn how to use Linux, IDL and either C or FORTRAN
- IDL needed in experimental labs (ICF and MCF)
- C or FORTRAN needed next term for computational lab

Programming course (this term)

Aim

- Learn how to use Linux, IDL and either C or FORTRAN
- IDL needed in experimental labs (ICF and MCF)
- C or FORTRAN needed next term for computational lab

The course

- Weekly problems
 - 10 credits total. Part of Fusion Lab
- Office hour to discuss problems
 - Wednesday 13:15 – 14:15, room A019
- Online forum to discuss issues:
<http://plasmaforum.york.ac.uk>

Why programming?

- Computers do not get bored, and don't make mistakes
- Perform calculations phenomenally quickly: A typical desktop performs nearly a billion additions/multiplications per second. Fastest supercomputers perform about a million times more.

Why write code? Why not use existing tools like spreadsheets?

Why programming?

- Computers do not get bored, and don't make mistakes
- Perform calculations phenomenally quickly: A typical desktop performs nearly a billion additions/multiplications per second. Fastest supercomputers perform about a million times more.

Why write code? Why not use existing tools like spreadsheets?

- Hard to extend: Try handling 4D arrays in Excel.
- For large problems these tools become too slow and/or cumbersome
- Research is often about doing something new. Often no program exists which will do exactly what you need.

Why programming?

- Programming is hard, and will take an effort to learn
- Requires attention to detail, creativity and abstract thought
- Can actually be very satisfying, even enjoyable!
 - Problem-solving (like crossword, sudoku etc.)
 - Get to see results of your work quickly
- A very “marketable” skill
 - Widens your choice of projects and careers
 - Shows general problem-solving ability

Programming Languages

- Way to specify computations and express algorithms precisely
- A human-readable language which can be automatically translated into processor operation codes (op-codes)

Programming Languages

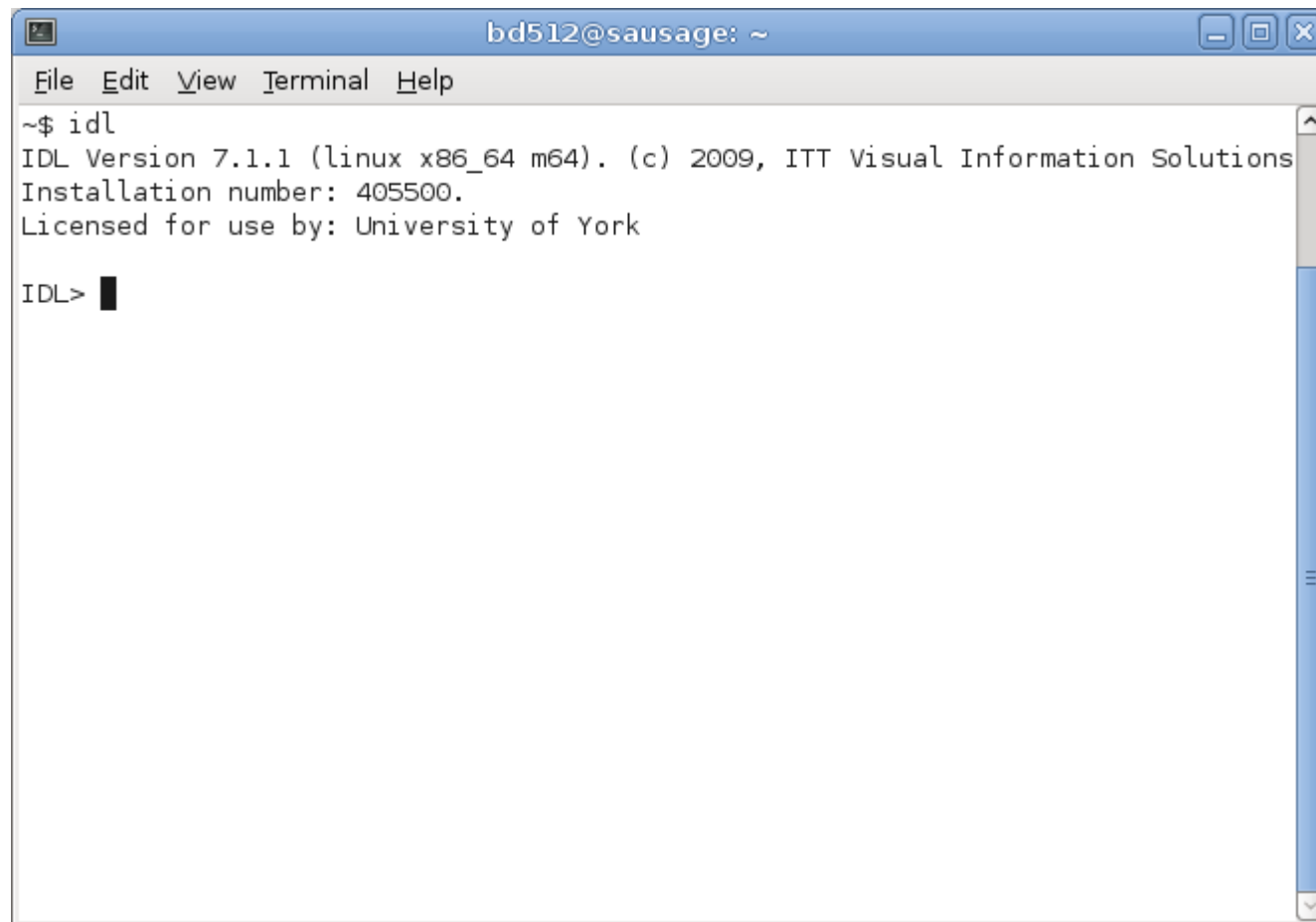
- Way to specify computations and express algorithms precisely
- A human-readable language which can be automatically translated into processor operation codes (op-codes)
- Many different languages
 - Approach problem solving in different ways so good to learn several
 - Have evolved as different approaches have been tried, and technology has improved
 - Each language has it's advantages and disadvantages for a particular application

Interactive Data Language

- Installed on your laptops (under Linux)
- A proprietary system created by Research Systems Inc
- Designed with scientists and engineers in mind, so very similar to FORTRAN. First version released 1979.
- Provides ways to visualise large amounts of data relatively easily, and create publication-quality plots.
- Used widely in space and plasma science: Culham, RAL, ESA, NASA, ...

IDL on your laptops

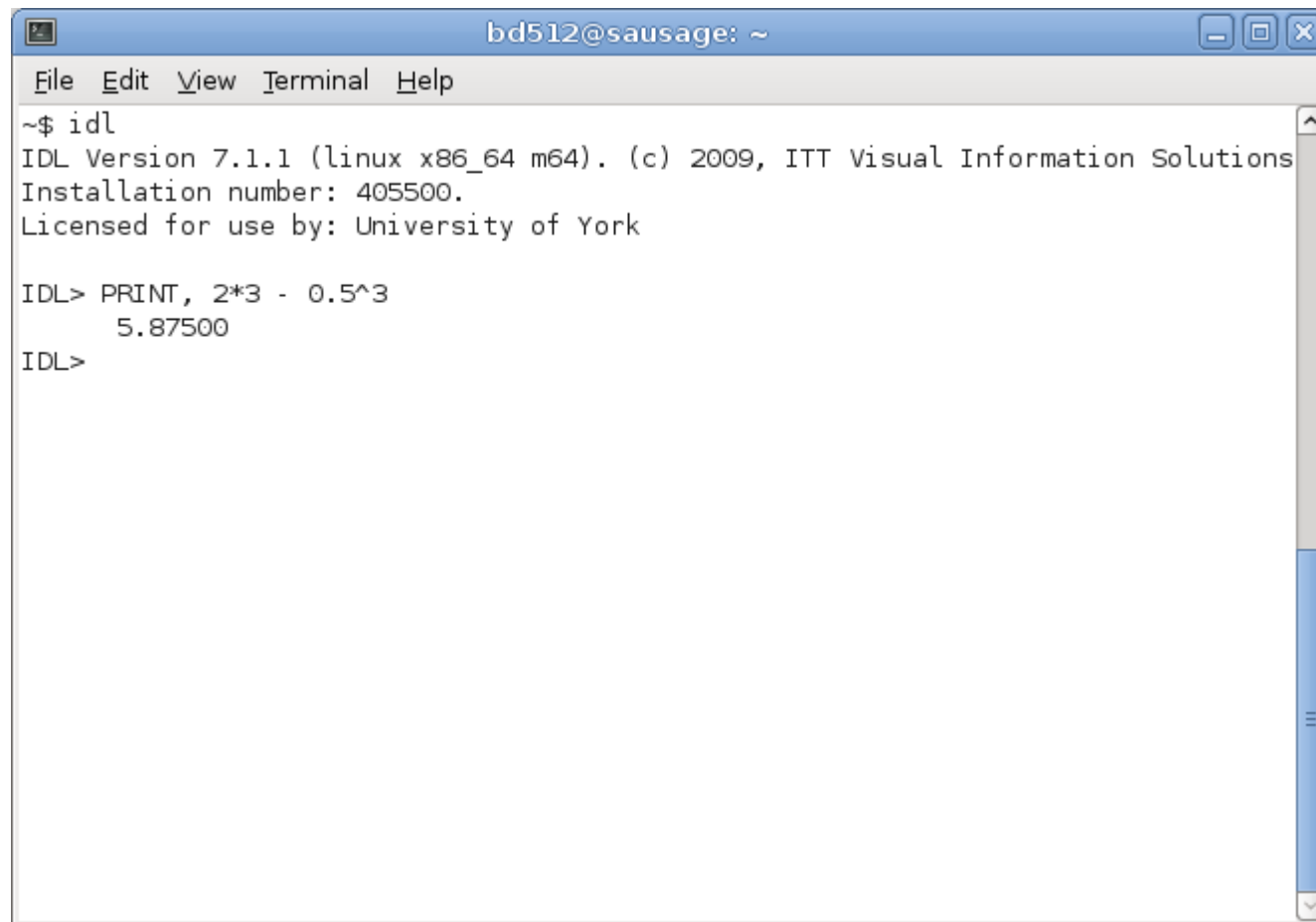
Start up IDL on your laptops. In a terminal window:

A screenshot of a terminal window titled "bd512@sausage: ~". The window has a menu bar with "File", "Edit", "View", "Terminal", and "Help". The terminal content shows the command "~\$ idl" being entered, followed by the output: "IDL Version 7.1.1 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions", "Installation number: 405500.", and "Licensed for use by: University of York". The prompt "IDL>" is visible with a black cursor. A vertical scrollbar is on the right side of the terminal area.

```
bd512@sausage: ~  
File Edit View Terminal Help  
~$ idl  
IDL Version 7.1.1 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions  
Installation number: 405500.  
Licensed for use by: University of York  
IDL> █
```

IDL expressions

- IDL can be used as a glorified calculator



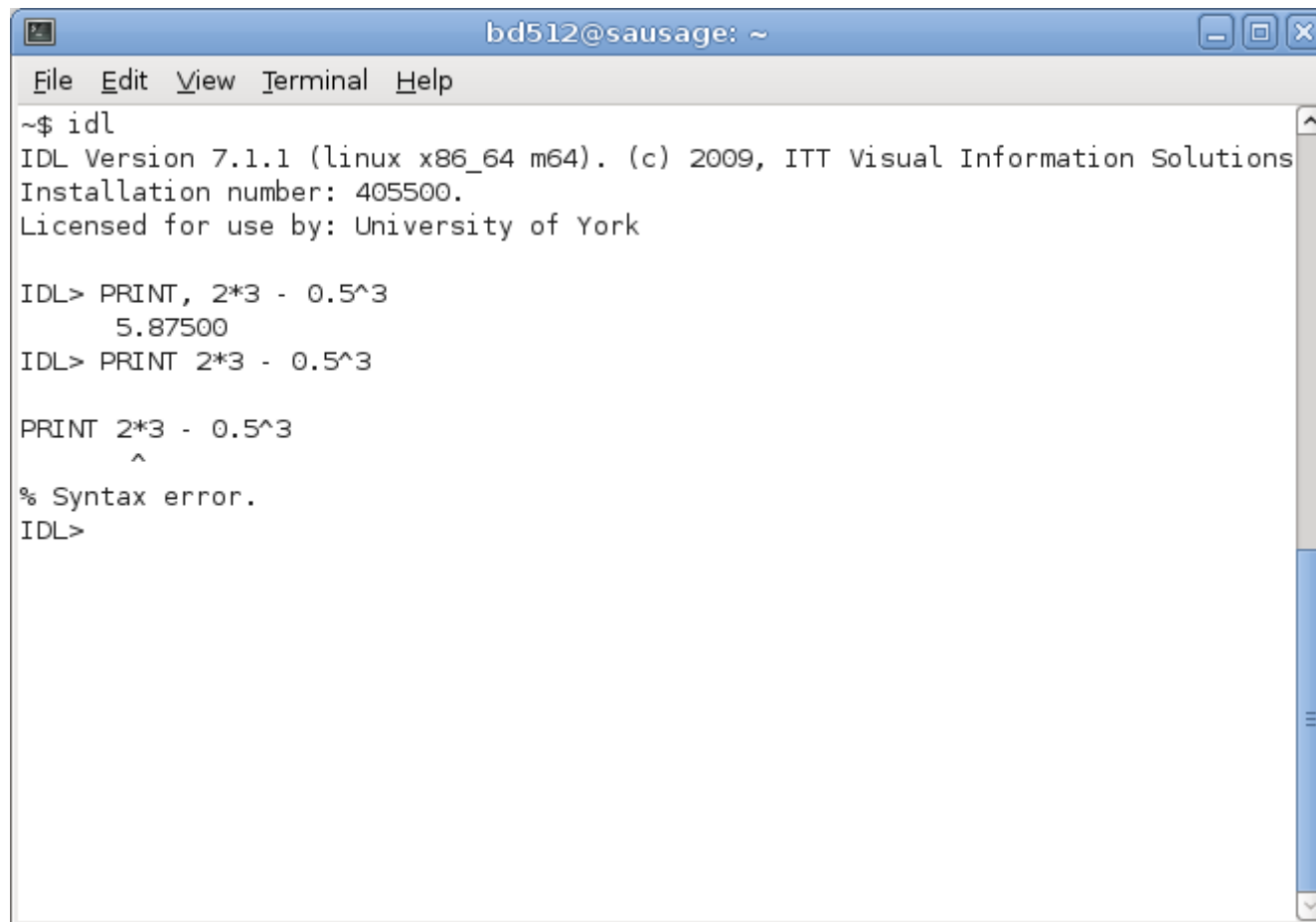
A screenshot of a terminal window titled "bd512@sausage: ~". The window has a menu bar with "File", "Edit", "View", "Terminal", and "Help". The terminal content shows the command `~$ idl` being executed, followed by the IDL version information: "IDL Version 7.1.1 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions", "Installation number: 405500.", and "Licensed for use by: University of York". Then, the command `IDL> PRINT, 2*3 - 0.5^3` is entered, and the output `5.87500` is displayed. The prompt `IDL>` is shown again on the next line.

```
bd512@sausage: ~
File Edit View Terminal Help
~$ idl
IDL Version 7.1.1 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions
Installation number: 405500.
Licensed for use by: University of York

IDL> PRINT, 2*3 - 0.5^3
      5.87500
IDL>
```

IDL expressions

- IDL can be used as a glorified calculator
- Watch out for missing commas!



```
bd512@sausage: ~  
File Edit View Terminal Help  
~$ idl  
IDL Version 7.1.1 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions  
Installation number: 405500.  
Licensed for use by: University of York  
  
IDL> PRINT, 2*3 - 0.5^3  
      5.87500  
IDL> PRINT 2*3 - 0.5^3  
  
PRINT 2*3 - 0.5^3  
      ^  
% Syntax error.  
IDL>
```

Variables

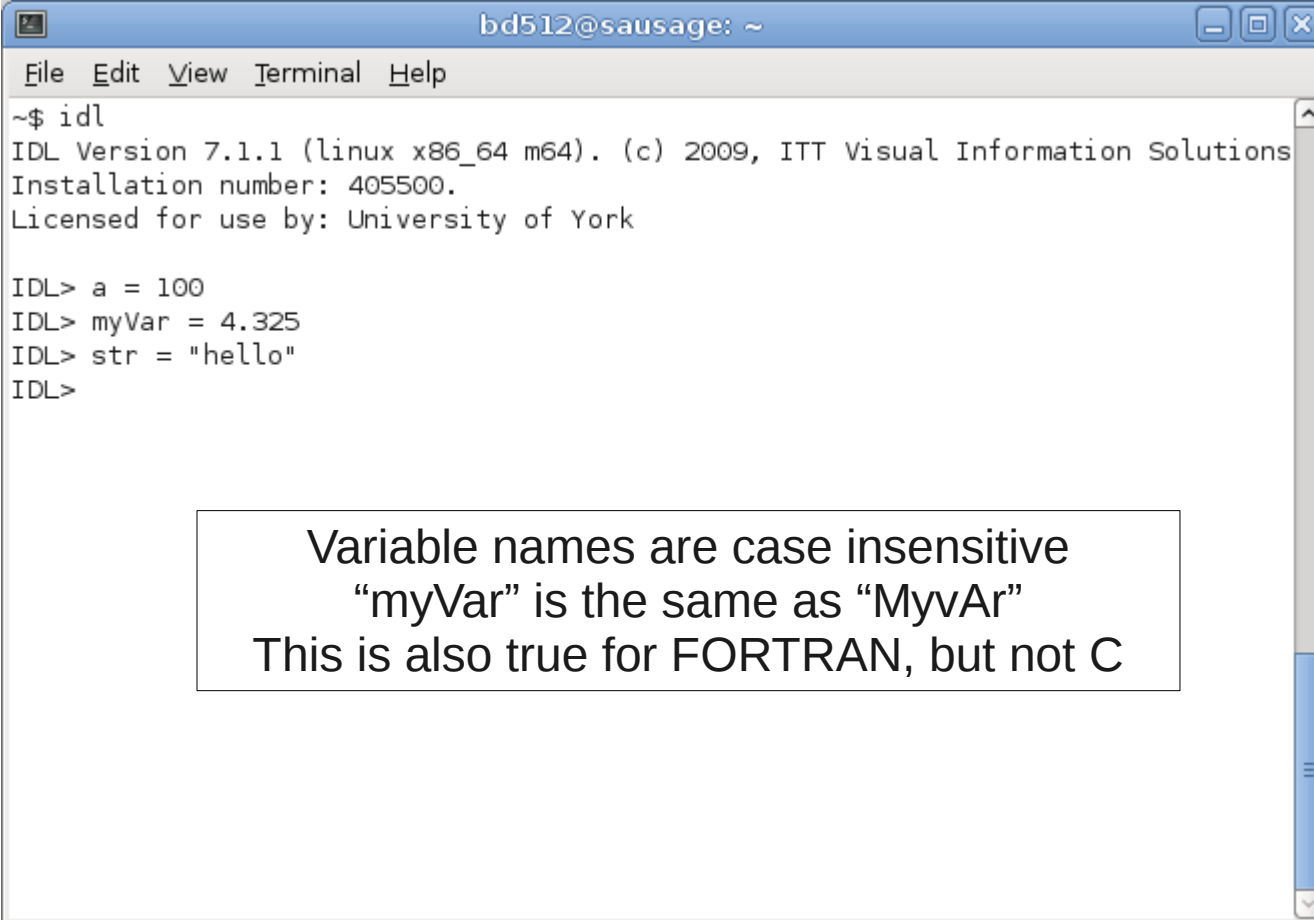
labels for values, similar to x, y, z in maths

- Have a “type”:

String	“Hello World!”
Integer	16-bit number (-32,767 to +32,767)
Long	32-bit number (+/- 2 billion)
Float	Single precision (about 7 digits)
Double	Double precision (about 16 digits)
Complex	Single precision complex number
Dcomplex	Double precision complex
- Names can be descriptive names
- Can be given (assigned) a value
- In most languages this value can be changed

Variables in IDL

- To create a variable in IDL, just give it a value



The screenshot shows a terminal window titled "bd512@sausage: ~". The window contains the following text:

```
File Edit View Terminal Help
~$ idl
IDL Version 7.1.1 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions
Installation number: 405500.
Licensed for use by: University of York

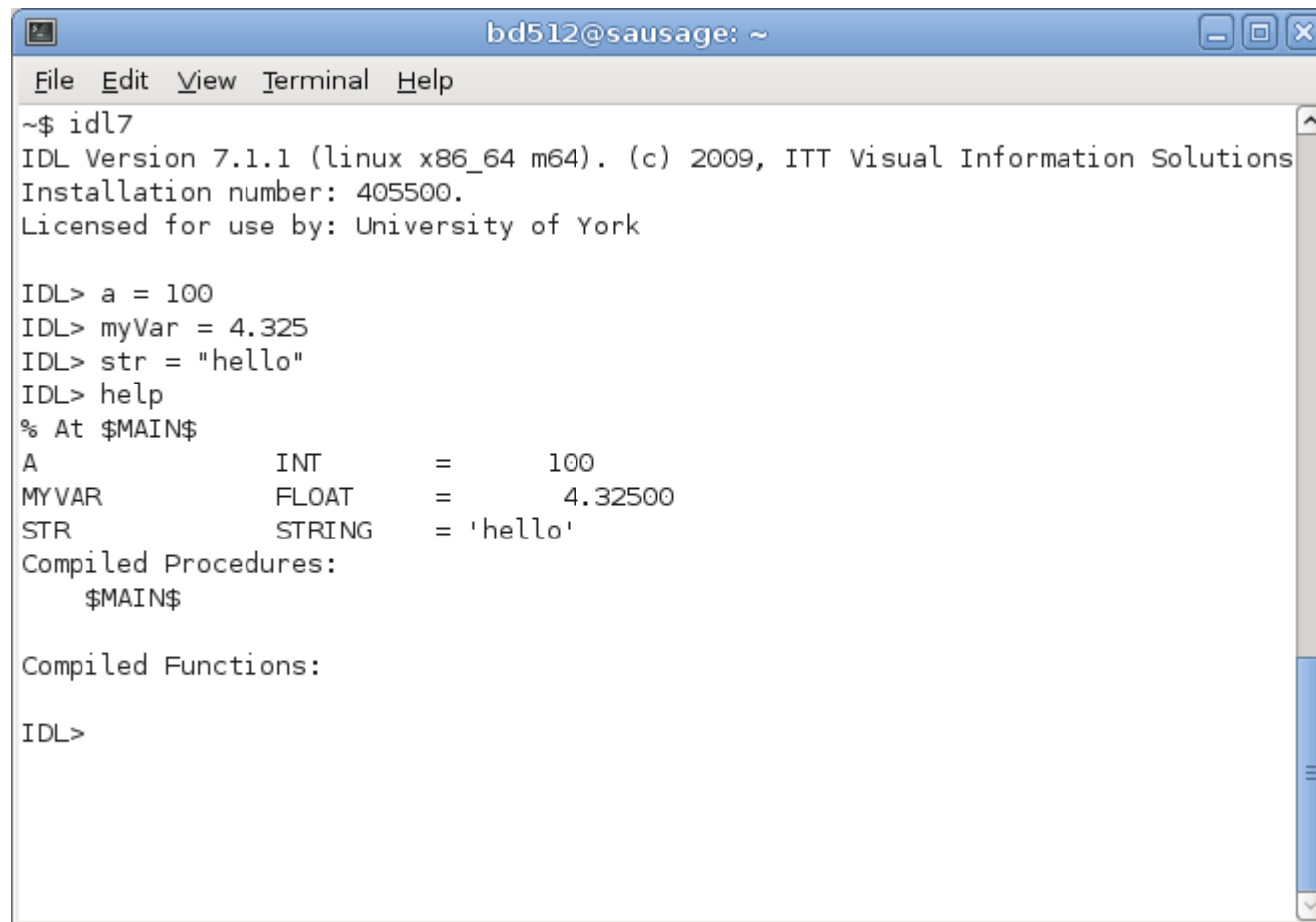
IDL> a = 100
IDL> myVar = 4.325
IDL> str = "hello"
IDL>
```

Below the terminal window, a text box contains the following text:

Variable names are case insensitive
"myVar" is the same as "MyvAr"
This is also true for FORTRAN, but not C

Variables in IDL

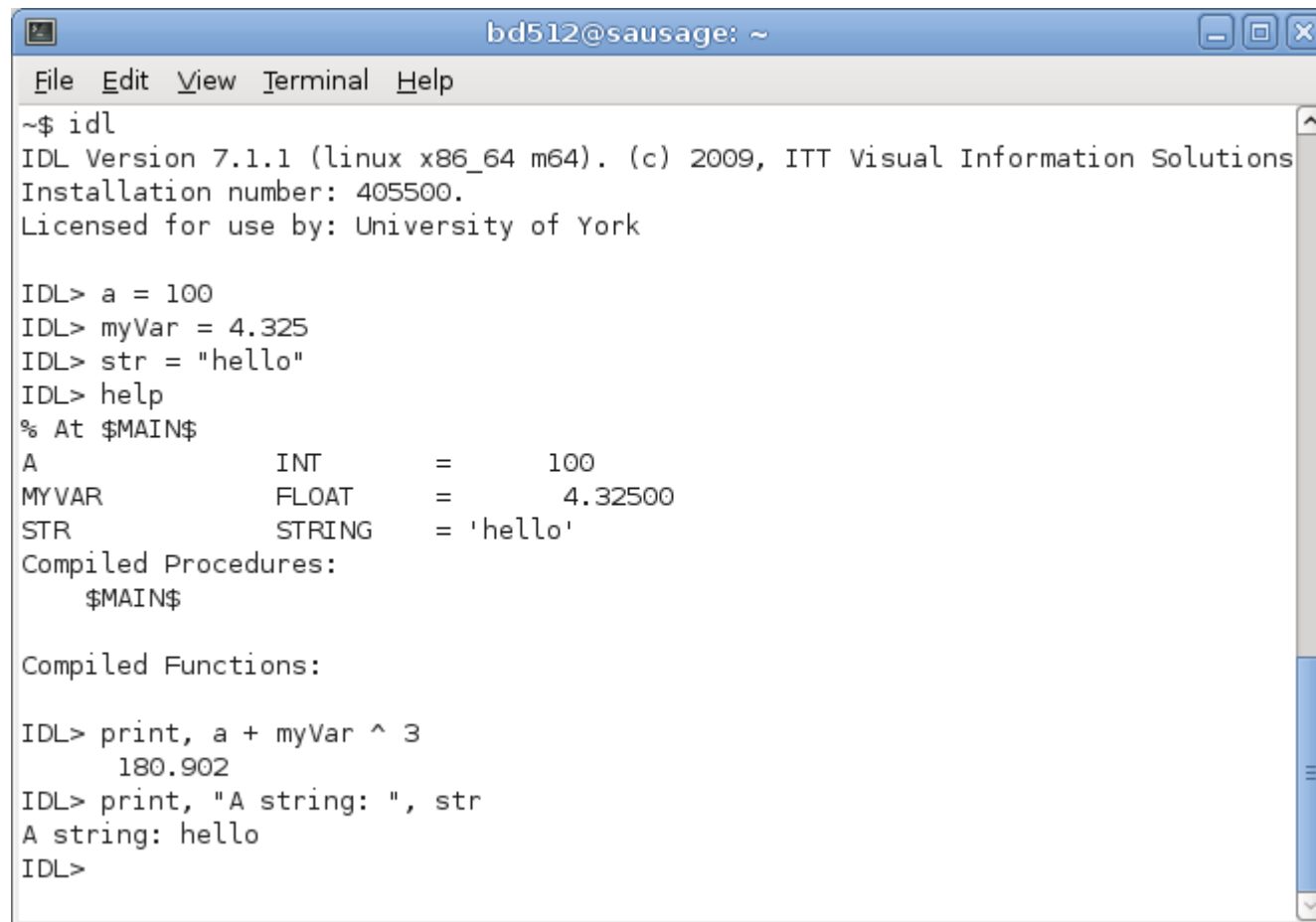
- To create a variable in IDL, just give it a value
- The “help” command tells you which variables are defined



```
bd512@sausage: ~  
File Edit View Terminal Help  
~$ idl7  
IDL Version 7.1.1 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions  
Installation number: 405500.  
Licensed for use by: University of York  
  
IDL> a = 100  
IDL> myVar = 4.325  
IDL> str = "hello"  
IDL> help  
% At $MAIN$  
A          INT      =      100  
MYVAR      FLOAT    =      4.32500  
STR        STRING   = 'hello'  
Compiled Procedures:  
    $MAIN$  
  
Compiled Functions:  
  
IDL>
```

Using variables

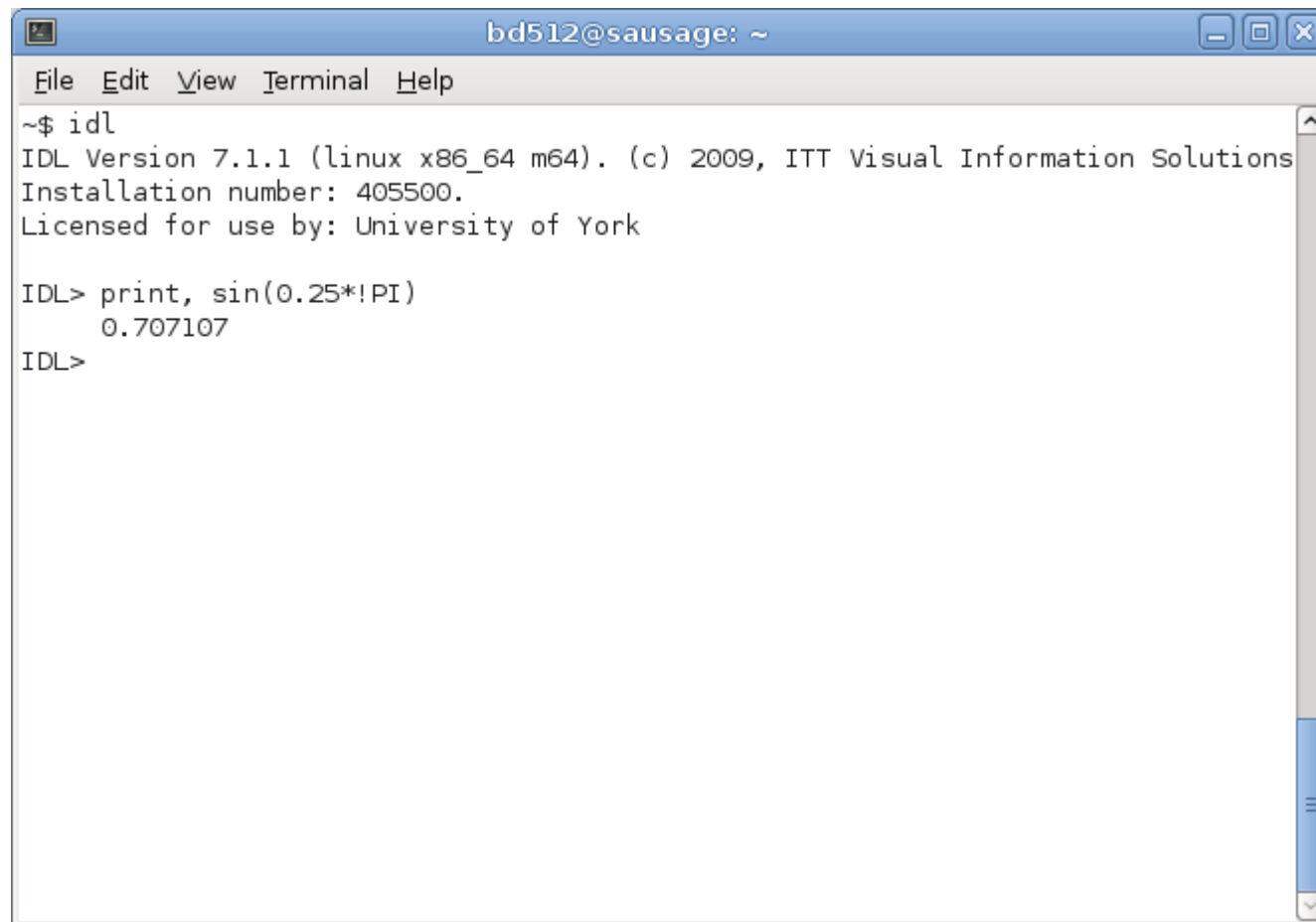
- Variables can be combined using operators (+ - * / ^)
- To print several quantities, separate with commas



```
bd512@sausage: ~  
File Edit View Terminal Help  
~$ idl  
IDL Version 7.1.1 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions  
Installation number: 405500.  
Licensed for use by: University of York  
  
IDL> a = 100  
IDL> myVar = 4.325  
IDL> str = "hello"  
IDL> help  
% At $MAIN$  
A          INT          =      100  
MYVAR      FLOAT       =      4.32500  
STR        STRING      = 'hello'  
Compiled Procedures:  
    $MAIN$  
  
Compiled Functions:  
  
IDL> print, a + myVar ^ 3  
      180.902  
IDL> print, "A string: ", str  
A string: hello  
IDL>
```


Built-in IDL functions

- IDL comes with lots of built-in functions for things like `sin()`, `cos()` and `tan()`



```
bd512@sausage: ~  
File Edit View Terminal Help  
~$ idl  
IDL Version 7.1.1 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions  
Installation number: 405500.  
Licensed for use by: University of York  
  
IDL> print, sin(0.25*!PI)  
      0.707107  
IDL>
```

IDL procedures and functions

- IDL makes a distinction between “procedures” like PRINT
`PRINT, var1, var2, ...`
These don't give (return) a result, so you couldn't have
`a = PRINT, var1, var2, ...`
- IDL functions do return a result, and need brackets
`a = SIN(x)`
If you don't do something with the result (e.g. store in a variable or print it) then IDL will complain
- If you're unsure, check the help pages....

Getting help

Typing '?' gets to the help system. Reference guide with all built-in commands

```
File Edit View Terminal
~$ idl
IDL Version 7.1.1 (linux
Installation number: 405
Licensed for use by: Uni

IDL> print, sin(0.25*PI)
0.707107
IDL> ?sin
% ONLINE_HELP: Starting
IDL>
```

The screenshot shows the 'Help - IDL Workbench' window. The left pane contains a 'Contents' list with items like 'About IDL', 'IDL Workbench Guide', 'IDL API Reference Guides', 'IDL Users' Guides', 'IDL Programmers' Guides', and 'Supplemental IDL Guides'. The right pane displays the documentation for the 'SIN' routine, including its syntax, return value, arguments, and keywords. The breadcrumb trail at the top of the right pane reads: 'IDL API Reference Guides > IDL Reference Guide > Part I: IDL Command Reference > Routines: S'.

Search: GO [Search scope:](#) All topics

Contents

- About IDL
- IDL Workbench Guide
- IDL API Reference Guides
- IDL Users' Guides
- IDL Programmers' Guides
- Supplemental IDL Guides

[IDL API Reference Guides](#) > [IDL Reference Guide](#) > [Part I: IDL Command Reference](#) > [Routines: S](#)

SIN

[Syntax](#) | [Return Value](#) | [Arguments](#) | [Keywords](#) | [Examples](#) | [Version History](#) | [See Also](#)

The periodic function SIN returns the trigonometric sine of X.

Syntax

Result = SIN(X)

Return Value

Returns the double-precision floating-point, complex or single-precision floating-point value.

Arguments

X

The angle for which the sine is desired, specified in radians. If X is double-precision floating or complex, the result is of the same type. All other types are converted to single-precision floating-point and yield floating-point results. When applied to complex numbers:

$$\text{SIN}(x) = (\text{EXP}(ix) + 1/\text{EXP}(ix))/(2i)$$

where i is defined as COMPLEX(0, 1).

If input argument X is an array, the result has the same structure, with each element containing the sine of the corresponding element of X.

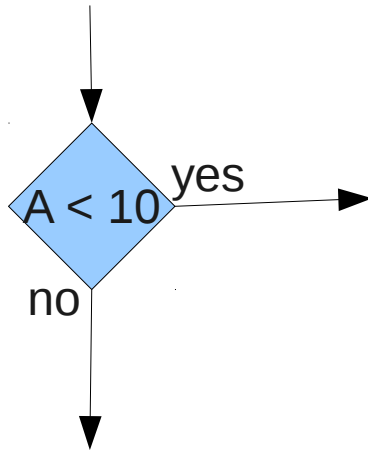
Keywords

Thread Pool Keywords

This routine is written to make use of IDL's *thread pool*, which can increase execution speed on systems with multiple CPUs. The values stored in the !CPU system variable control whether IDL uses the thread pool for a given computation. In addition, you can use the thread pool keywords TPOOL_MAXELTS, TPOOL and TPOOL_NOTHREAD to override the defaults established by !CPU for a single invocation of this routine. See

Making decisions

- Often want run different commands depending on some criterion. Statements to do this are called conditionals



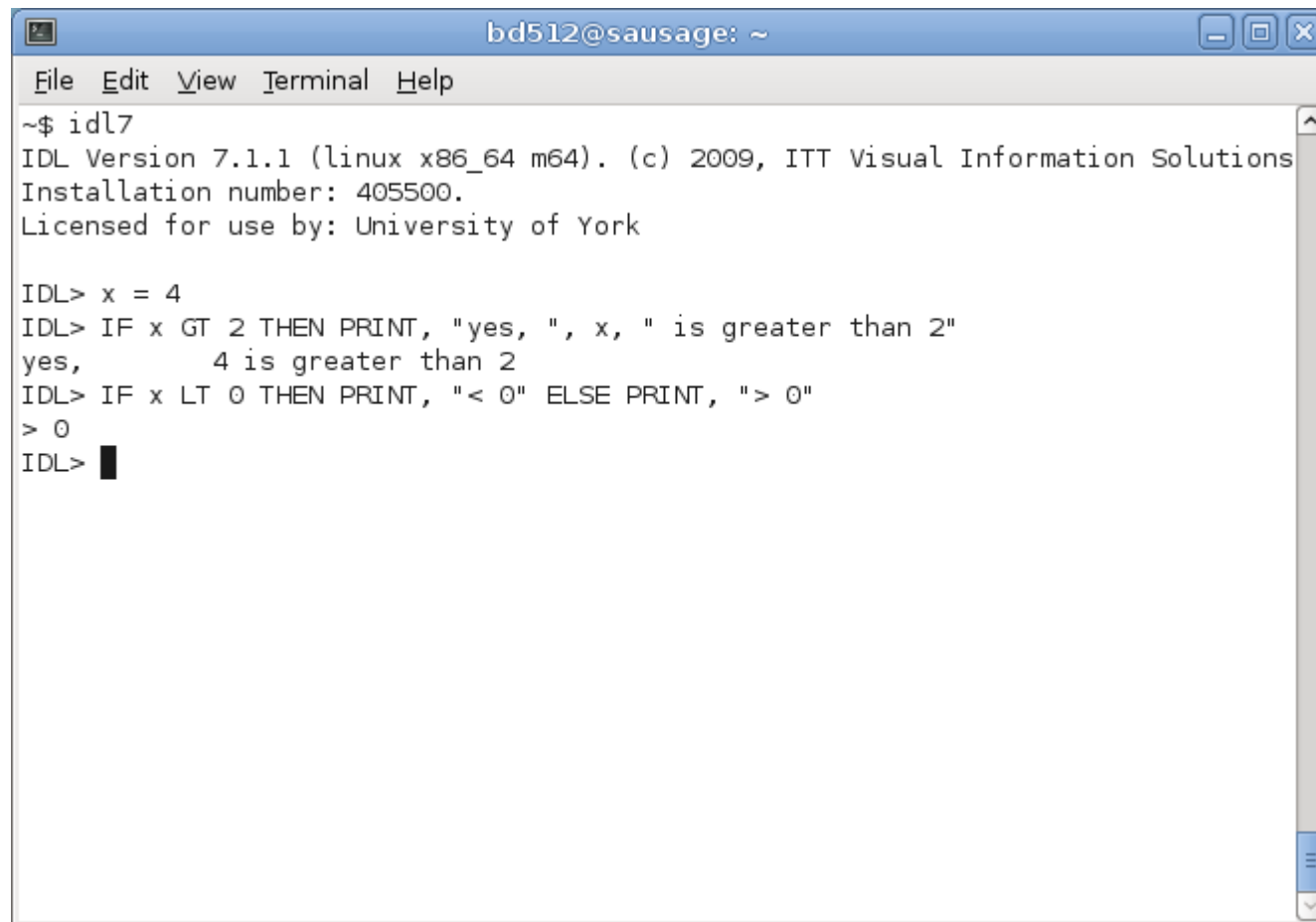
Note: Also called branching

Condition can be a combination of

- Numerical comparisons: Equal (EQ), greater than (GT), less than (LT), greater than or equal (GE) and less than or equal (LE)
- Boolean operators (AND, OR, NOT, XOR)

Conditionals in IDL

- Compare variables and values: EQ, GT, LT, GE, LE
- Used in IF statements to decide what code to run



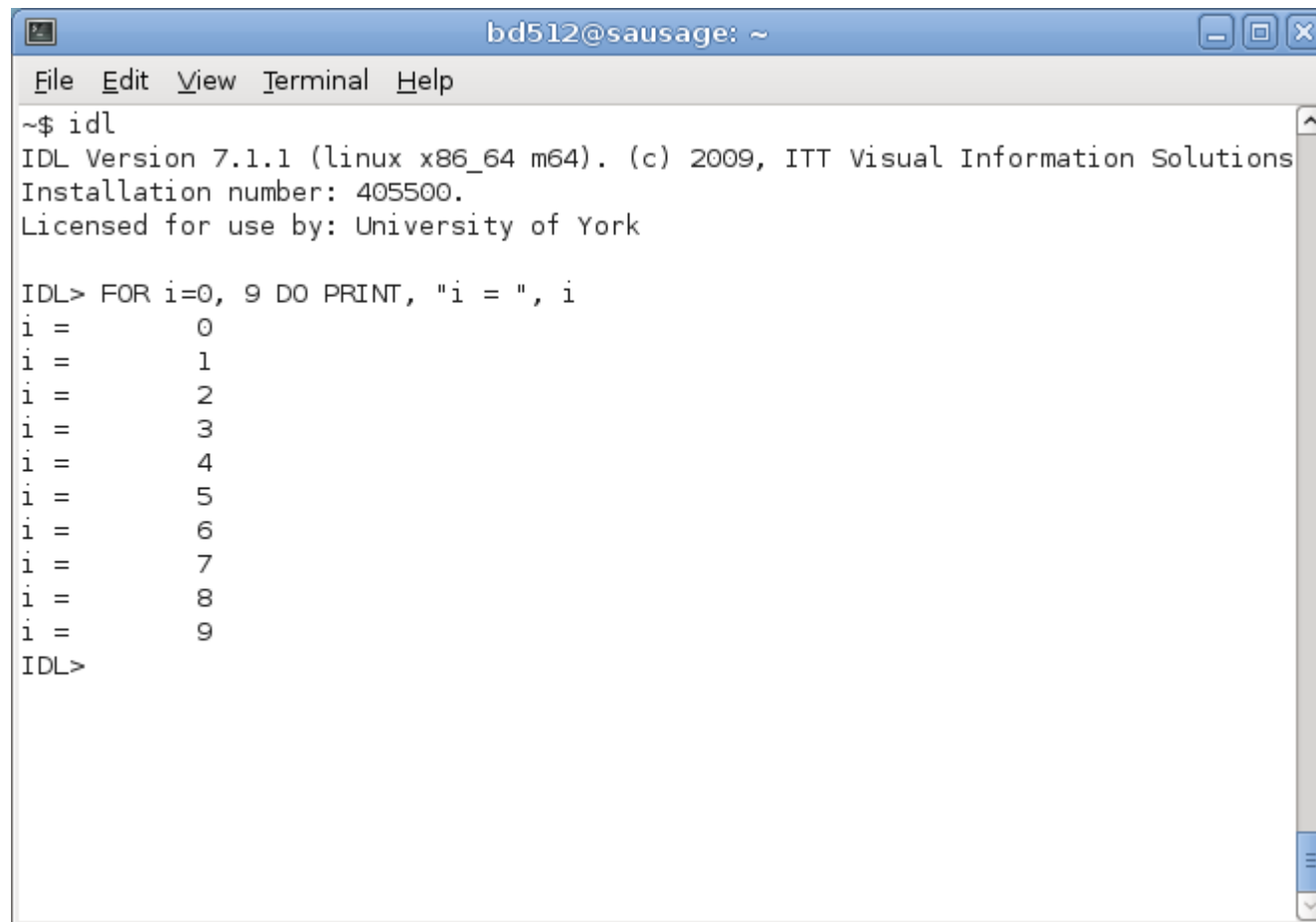
The screenshot shows a terminal window titled "bd512@sausage: ~". The window contains the following text:

```
File Edit View Terminal Help
~$ idl7
IDL Version 7.1.1 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions
Installation number: 405500.
Licensed for use by: University of York

IDL> x = 4
IDL> IF x GT 2 THEN PRINT, "yes, ", x, " is greater than 2"
yes,      4 is greater than 2
IDL> IF x LT 0 THEN PRINT, "< 0" ELSE PRINT, "> 0"
> 0
IDL> █
```

Going round in circles

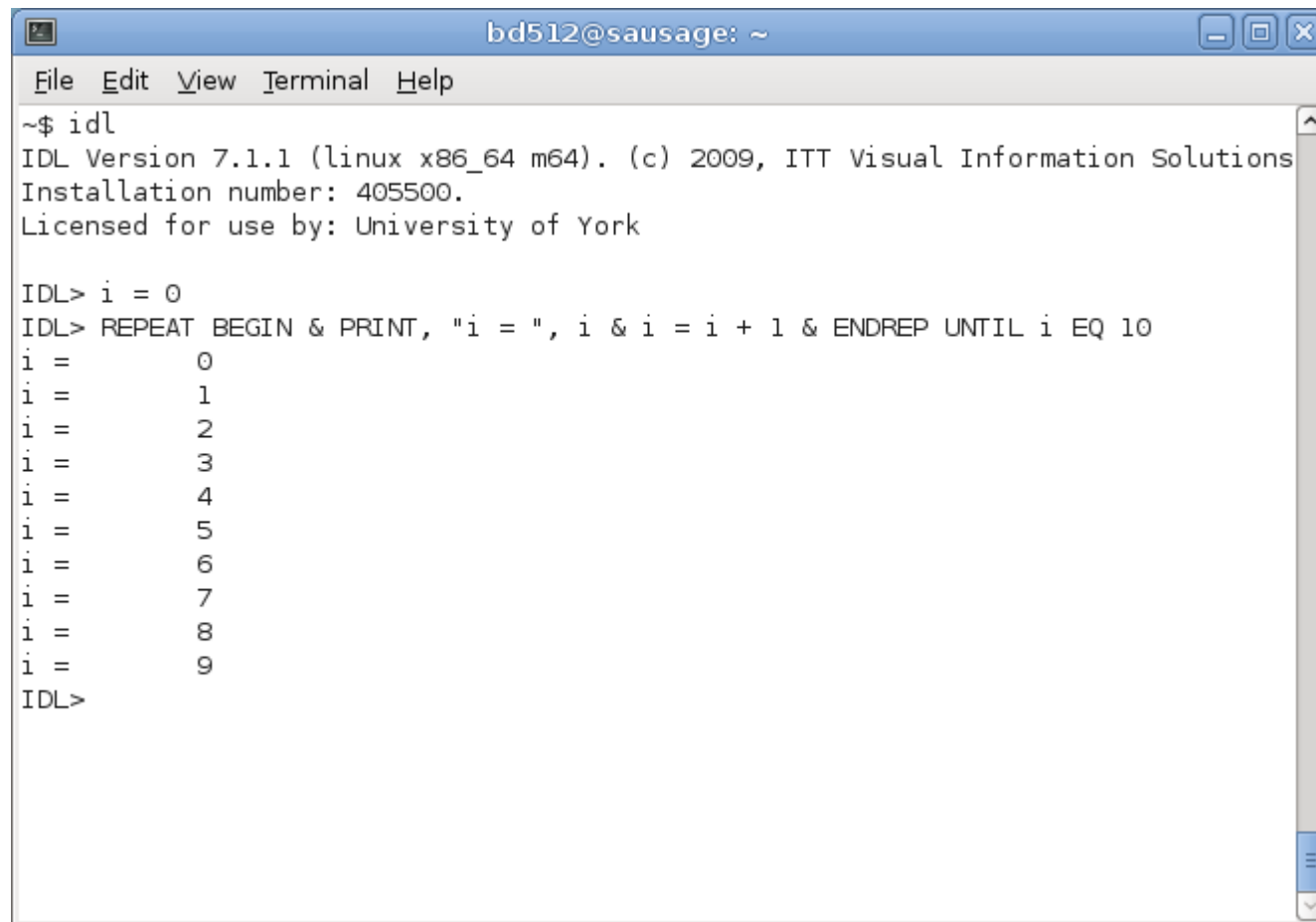
- Repeat operations, either a fixed number of times...



```
bd512@sausage: ~  
File Edit View Terminal Help  
~$ idl  
IDL Version 7.1.1 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions  
Installation number: 405500.  
Licensed for use by: University of York  
  
IDL> FOR i=0, 9 DO PRINT, "i = ", i  
i =      0  
i =      1  
i =      2  
i =      3  
i =      4  
i =      5  
i =      6  
i =      7  
i =      8  
i =      9  
IDL>
```

Going round in circles

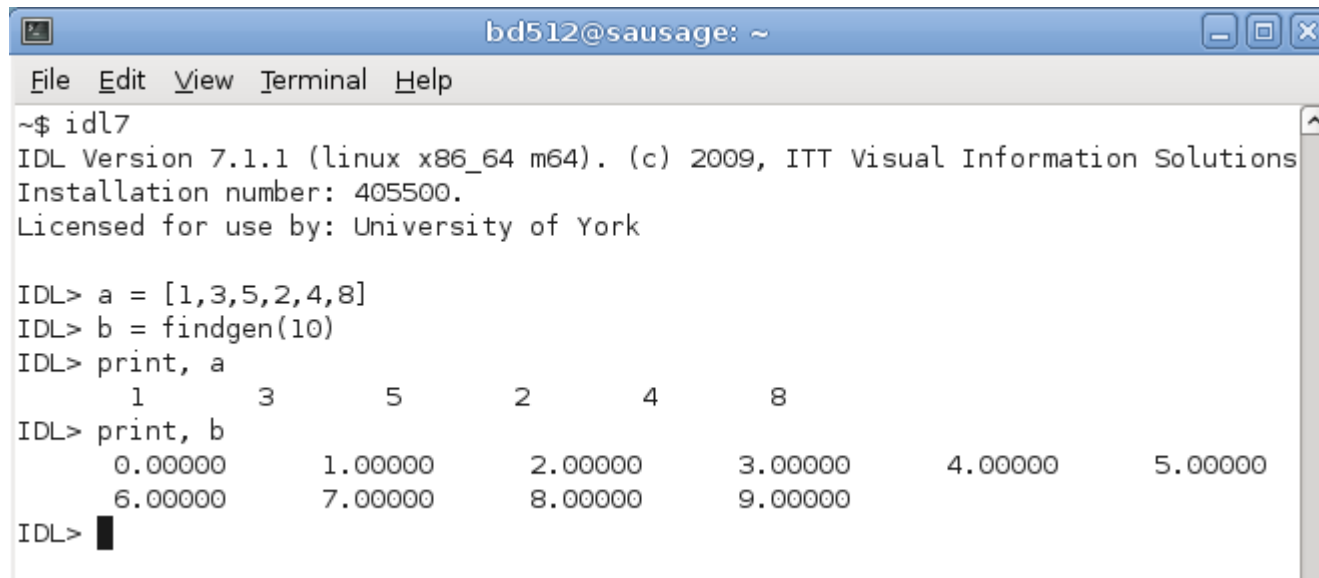
- Repeat operations, either a fixed number of times or until a condition is met



```
bd512@sausage: ~  
File Edit View Terminal Help  
~$ idl  
IDL Version 7.1.1 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions  
Installation number: 405500.  
Licensed for use by: University of York  
  
IDL> i = 0  
IDL> REPEAT BEGIN & PRINT, "i = ", i & i = i + 1 & ENDREP UNTIL i EQ 10  
i =      0  
i =      1  
i =      2  
i =      3  
i =      4  
i =      5  
i =      6  
i =      7  
i =      8  
i =      9  
IDL>
```

Arrays

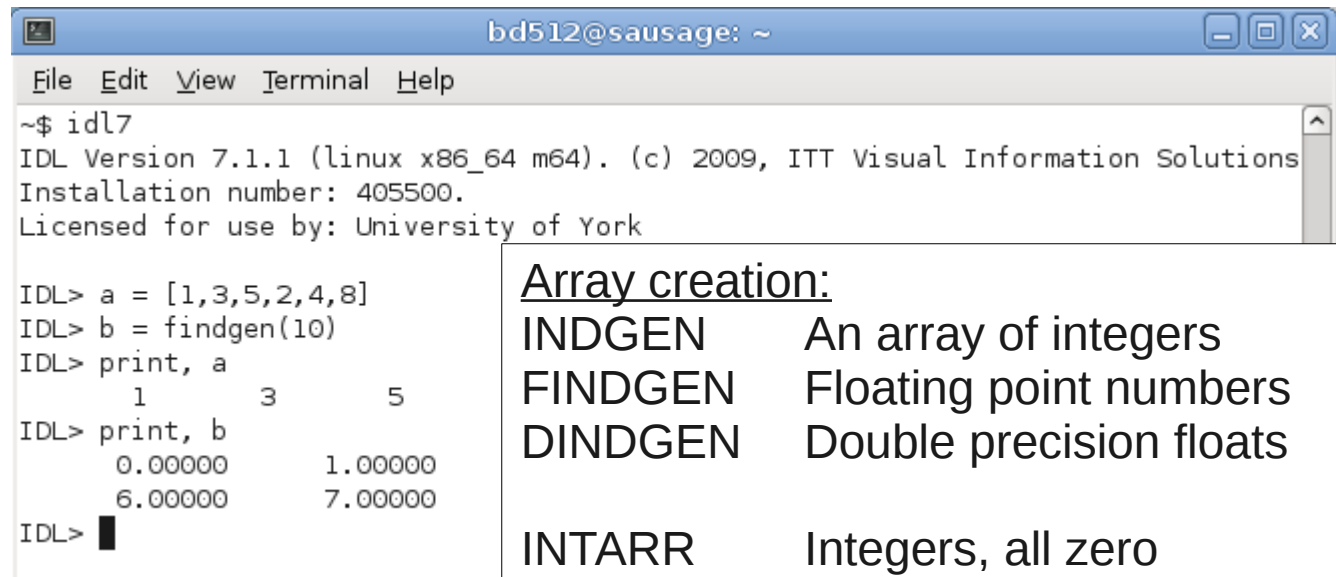
- When performing operations on lots of data, one way is to use loops. The (better) way is to use arrays
- Collection of variables of the same type
- Each component is labelled with a number (an index)



```
bd512@sausage: ~  
File Edit View Terminal Help  
~$ idl7  
IDL Version 7.1.1 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions  
Installation number: 405500.  
Licensed for use by: University of York  
  
IDL> a = [1,3,5,2,4,8]  
IDL> b = findgen(10)  
IDL> print, a  
      1      3      5      2      4      8  
IDL> print, b  
      0.00000      1.00000      2.00000      3.00000      4.00000      5.00000  
      6.00000      7.00000      8.00000      9.00000  
IDL> █
```


Arrays

- When performing operations on lots of data, one way is to use loops. The (better) way is to use arrays
- Collection of variables of the same type
- Each component is labelled with a number (an index)



```
bd512@sausage: ~  
File Edit View Terminal Help  
~$ idl7  
IDL Version 7.1.1 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions  
Installation number: 405500.  
Licensed for use by: University of York  
  
IDL> a = [1,3,5,2,4,8]  
IDL> b = findgen(10)  
IDL> print, a  
      1      3      5  
IDL> print, b  
      0.00000      1.00000  
      6.00000      7.00000  
IDL> █
```

Array creation:

INDGEN	An array of integers
FINDGEN	Floating point numbers
DINDGEN	Double precision floats
INTARR	Integers, all zero
LONARR	Long integers, all zero
FLTARR	Floats, all zero

Array operations

In IDL, operations on arrays apply to each element in the array:

```
IDL> a = findgen(5)
```

```
IDL> print, a
```

```
0.0  1.0  2.0  3.0  4.0
```

```
IDL> a = a + 1
```

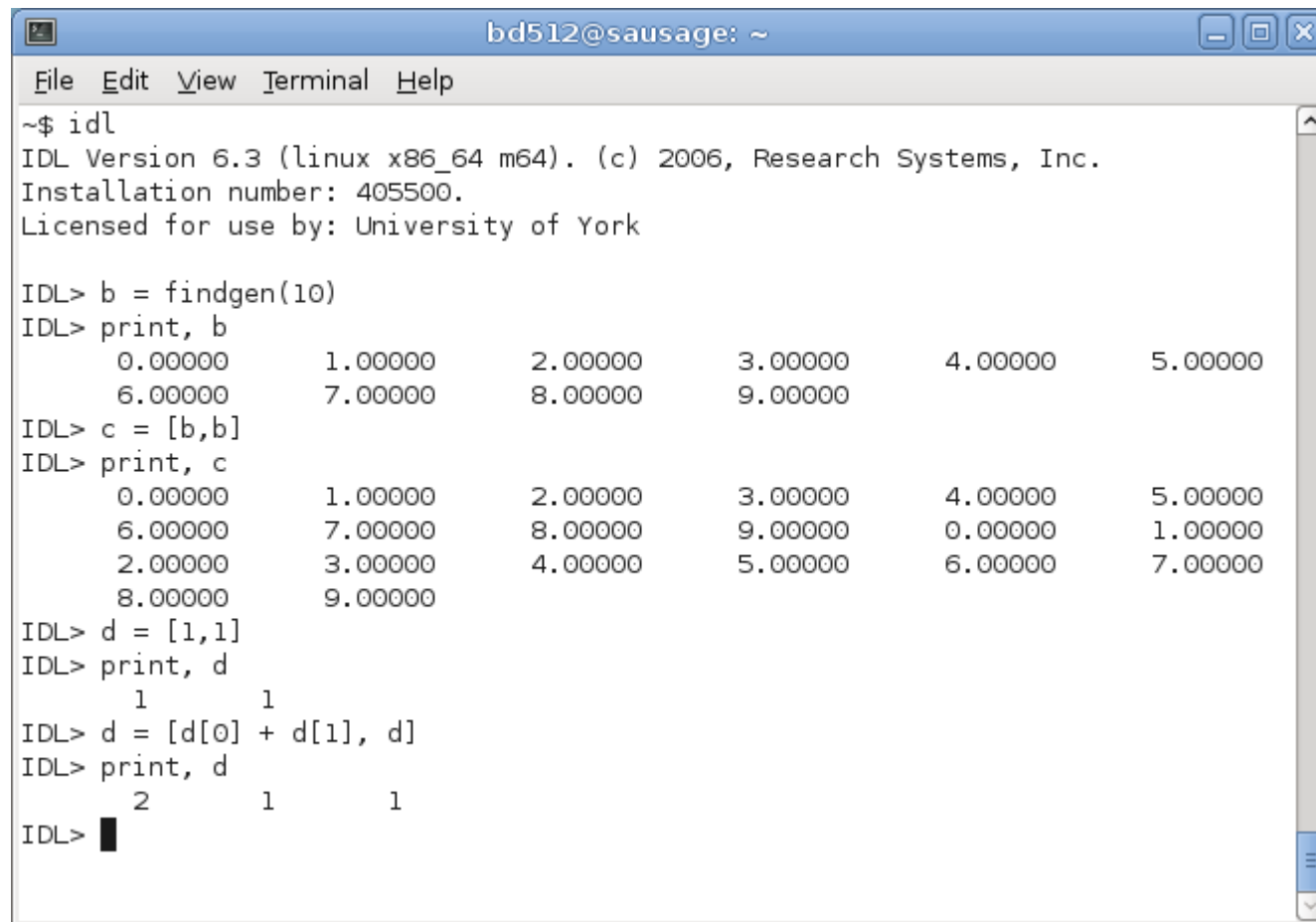
```
IDL> print, a
```

```
1.0  2.0  3.0  4.0  5.0
```

This is also true in FORTRAN, but not in C. There are ways to do this in C++.

Extending arrays

- Arrays can also be joined together (concatenated) and extended



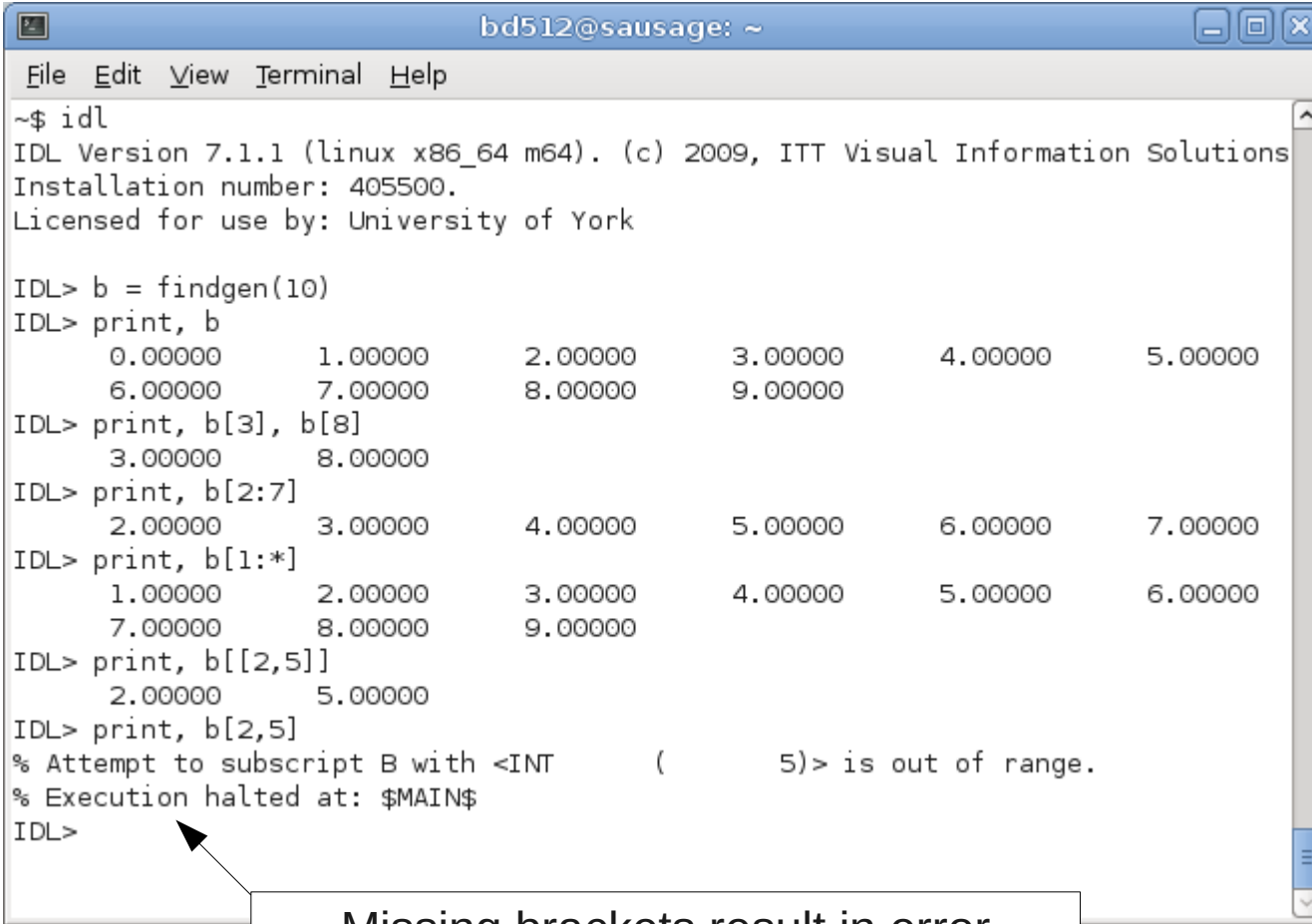
The screenshot shows a terminal window titled "bd512@sausage: ~" with a menu bar (File, Edit, View, Terminal, Help). The terminal displays the following commands and output:

```
~$ idl
IDL Version 6.3 (linux x86_64 m64). (c) 2006, Research Systems, Inc.
Installation number: 405500.
Licensed for use by: University of York

IDL> b = findgen(10)
IDL> print, b
0.00000 1.00000 2.00000 3.00000 4.00000 5.00000
6.00000 7.00000 8.00000 9.00000
IDL> c = [b,b]
IDL> print, c
0.00000 1.00000 2.00000 3.00000 4.00000 5.00000
6.00000 7.00000 8.00000 9.00000 0.00000 1.00000
2.00000 3.00000 4.00000 5.00000 6.00000 7.00000
8.00000 9.00000
IDL> d = [1,1]
IDL> print, d
1 1
IDL> d = [d[0] + d[1], d]
IDL> print, d
2 1 1
IDL>
```

Indexing and slicing arrays

- Array elements or ranges can be extracted
- Individual variables can also be changed



The screenshot shows a terminal window titled "bd512@sausage: ~" with a menu bar (File, Edit, View, Terminal, Help). The terminal output shows the following sequence of commands and results:

```
~$ idl
IDL Version 7.1.1 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions
Installation number: 405500.
Licensed for use by: University of York

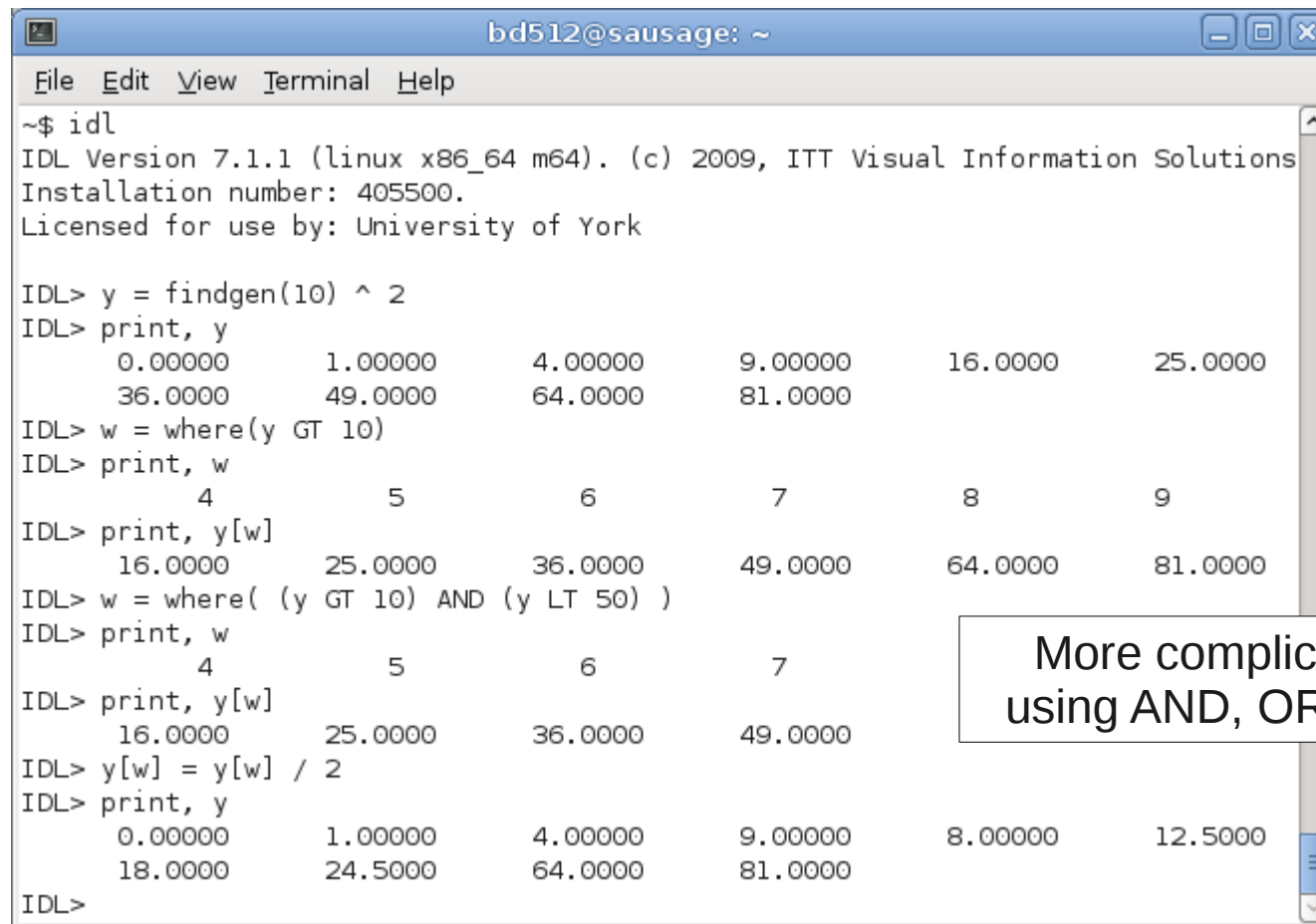
IDL> b = findgen(10)
IDL> print, b
  0.00000  1.00000  2.00000  3.00000  4.00000  5.00000
  6.00000  7.00000  8.00000  9.00000
IDL> print, b[3], b[8]
  3.00000  8.00000
IDL> print, b[2:7]
  2.00000  3.00000  4.00000  5.00000  6.00000  7.00000
IDL> print, b[1:*]
  1.00000  2.00000  3.00000  4.00000  5.00000  6.00000
  7.00000  8.00000  9.00000
IDL> print, b[[2,5]]
  2.00000  5.00000
IDL> print, b[2,5]
% Attempt to subscript B with <INT      (      5)> is out of range.
% Execution halted at: $MAIN$
IDL>
```

An arrow points from a text box at the bottom to the command `print, b[2,5]` in the terminal.

Missing brackets result in error

WHERE command

- Allows you to select and manipulate parts of an array, depending on some criterion. Like IF for arrays



```
bd512@sausage: ~  
File Edit View Terminal Help  
~$ idl  
IDL Version 7.1.1 (linux x86_64 m64). (c) 2009, ITT Visual Information Solutions  
Installation number: 405500.  
Licensed for use by: University of York  
  
IDL> y = findgen(10) ^ 2  
IDL> print, y  
0.00000 1.00000 4.00000 9.00000 16.0000 25.0000  
36.0000 49.0000 64.0000 81.0000  
IDL> w = where(y GT 10)  
IDL> print, w  
4 5 6 7 8 9  
IDL> print, y[w]  
16.0000 25.0000 36.0000 49.0000 64.0000 81.0000  
IDL> w = where( (y GT 10) AND (y LT 50) )  
IDL> print, w  
4 5 6 7  
IDL> print, y[w]  
16.0000 25.0000 36.0000 49.0000  
IDL> y[w] = y[w] / 2  
IDL> print, y  
0.00000 1.00000 4.00000 9.00000 8.00000 12.5000  
18.0000 24.5000 64.0000 81.0000  
IDL>
```

More complicated conditions
using AND, OR, XOR and NOT

Programming components

Several components are found in (almost) all languages

- Variables: Storage locations for values
- Expressions: Perform a calculation using variables
- Branching: Check if a condition is true, and if so perform some operation
- Loops: Repeat a set of operations until some condition is true
- Functions: A group of operations which can be applied to input variables and produce some result

Summary

- Programming requires thinking about algorithms, and is something best learnt through practice
 - This week you have lectures and classes on IDL
 - Problem sheets this term in IDL and C or FORTRAN
- IDL is a relatively easy language to learn, and will be vital for your experimental labs later this term and next.
- We have covered the basic building blocks of programming
- Go through programming handout
- Make a start at the IDL exercises

<http://www-users.york.ac.uk/~bd512/teaching.shtml>
for course information and links