Northumbria University
NEWCASTLE

KD6041
Quantum Devices

Photonic Crystal Simulations:

MPB basics practical commands

Daniel Ho and Mike Taverne

Email: daniel.ho@northumbria.ac.uk ; mike.taverne@northumbria.ac.uk

Intended Learning Objectives

Northumbria University NEWCASTLE

- interactive mode and script mode
- print function
- variables
- functions
- conditional statements
- loops

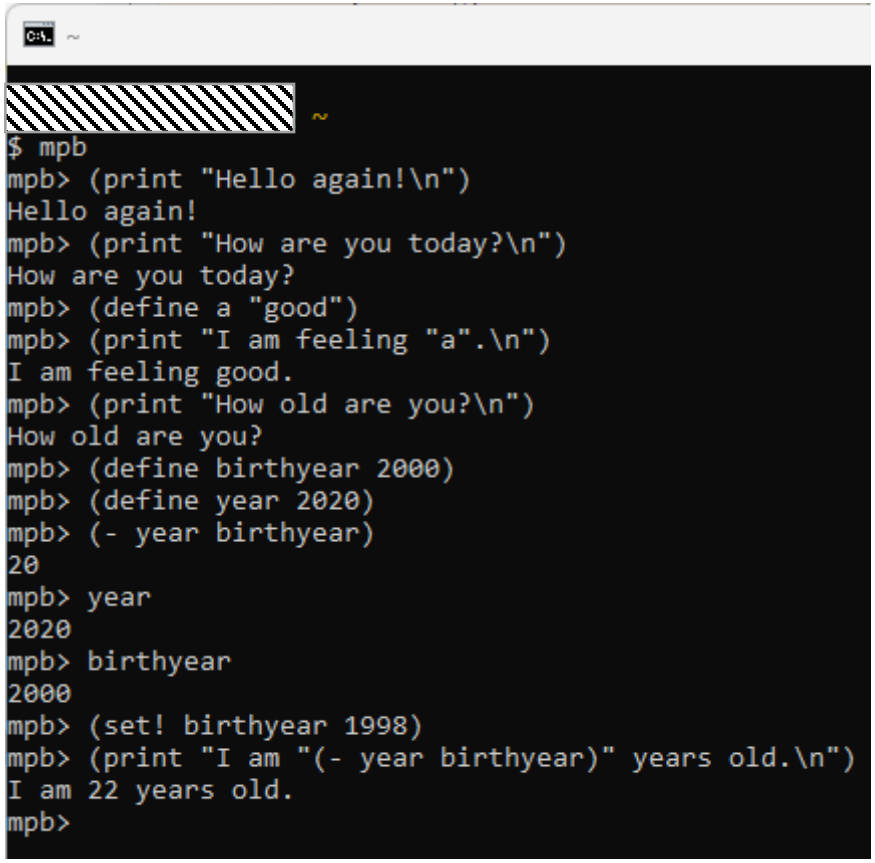
MPB basics

Northumbria University NEWCASTLE

- MPB input files are written using a programming language called **Scheme**.
- Scheme consists of a series of function calls of the form:
  (func arg1 arg2 arg3…)
- Comments in the code can be added by prefixing them with a semicolon (**;**). Example: **(func arg1 arg2 arg…) ; This is a comment.**
- The MPB script files are simple text files with a **.ctl extension** (ctl stands for control).
- To run an MPB script called **input.ctl**, type the following at a bash command prompt: **mpb input.ctl**
- Then press enter

MPB interactive mode

Northumbria University NEWCASTLE

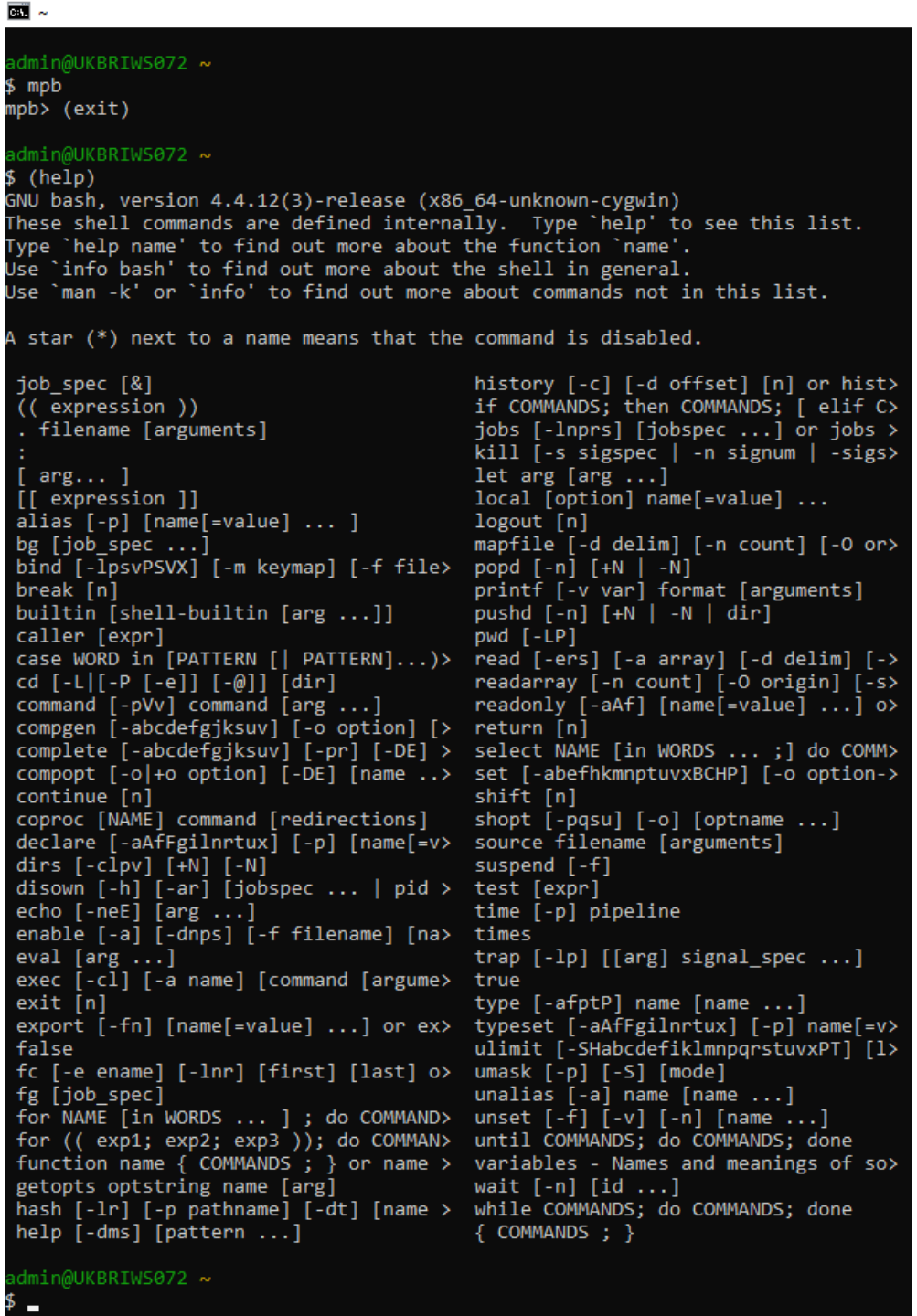


- The **MPB interactive mode** is very useful for debugging and trying out commands without writing scripts.
- It can be started by simply typing "*mpb*" and pressing *enter*.
- You can exit the interactive prompt by using the **(exit)** command or pressing **ctrl-D**.
- You can press the **tab key** once or twice to autocomplete partially typed function or variable names.

mpb

(exit)

(help)



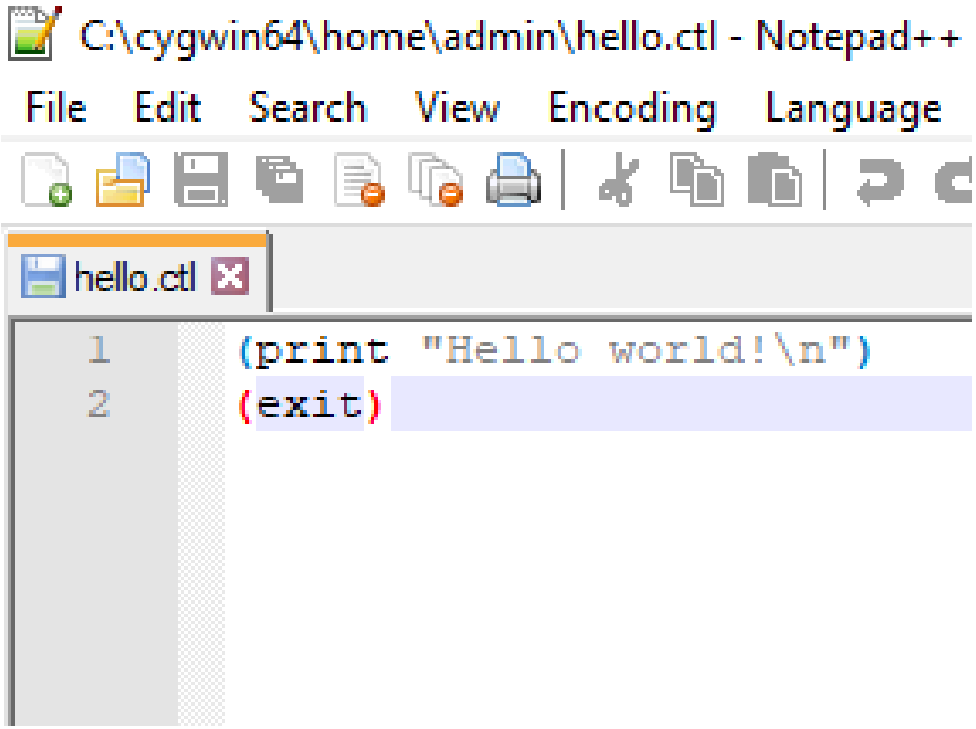
Northumbria University
NEWCASTLE

#TakeOnTomorrow

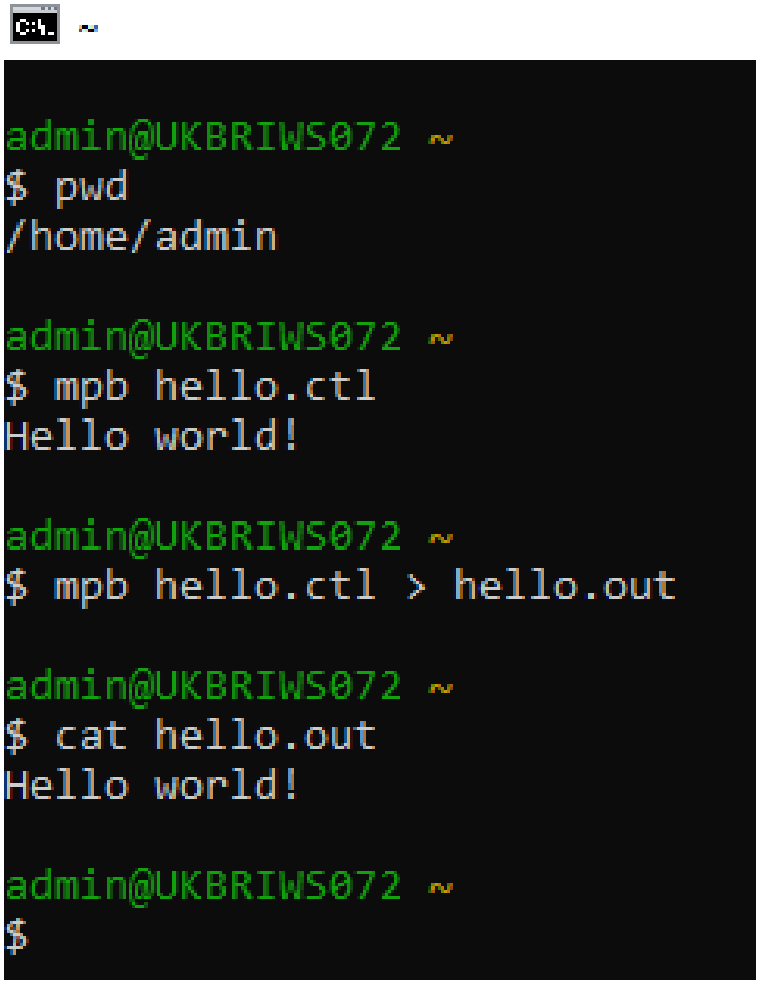
MPB “Hello world!” example

Northumbria University NEWCASTLE

Text editor with **hello.ctl** open

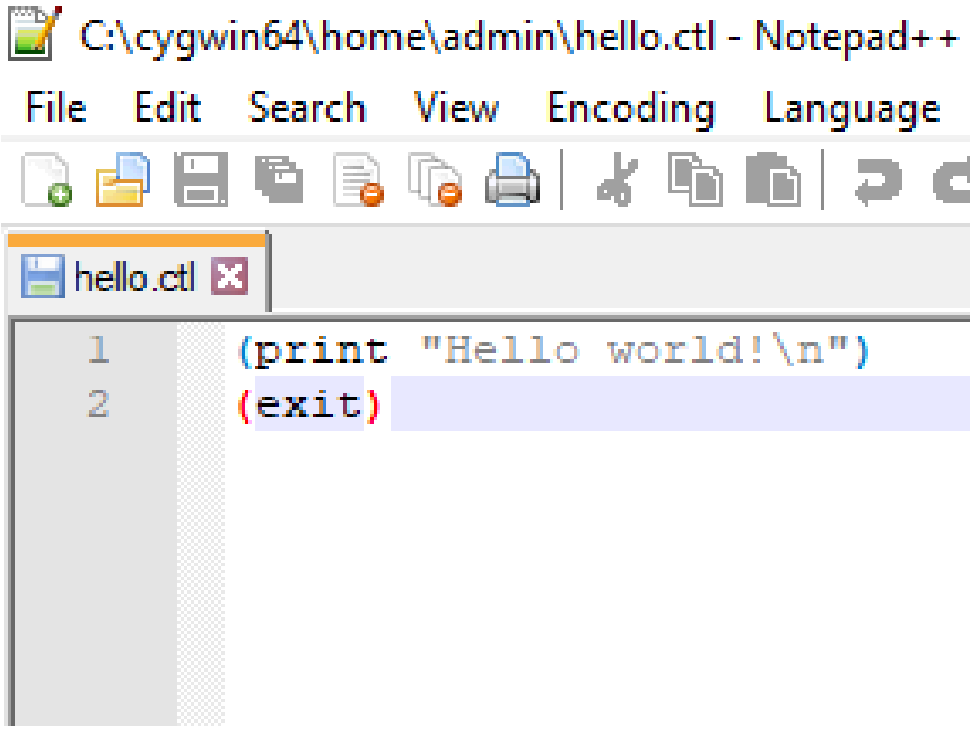


Console/Terminal/Shell



MPB “Hello world!” example
Code explanation

Northumbria University NEWCASTLE



- The **print** function prints a string to the standard output (stdout):

  Usage: ***(print STRING)***

- The **exit** function simply exits the script.

  Usage: ***(exit)***

- If exit is not called and no MPB run function is used, MPB drops to an **interactive command prompt**.

- “***\n***” is used in the string to **print a new line** (line break).

MPB “Hello world!” example
Command explanations

Northumbria University NEWCASTLE

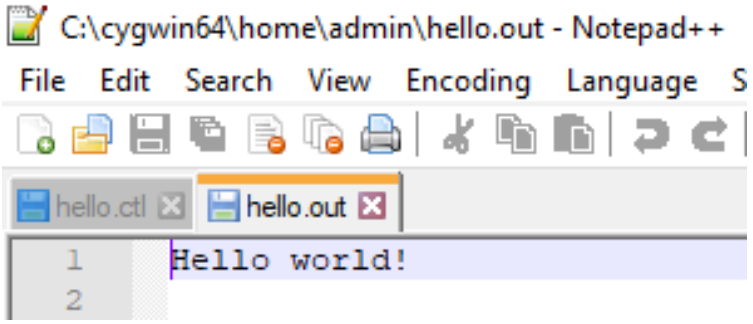
- When run with “***mpb hello.ctl***”, MPB outputs results on the screen.
- Most of the time, we want to **save** the **output** for analyzing the data.
- We can do this by redirecting the standard output to a file, with the “greater than” sign: **>**.
- “***mpb hello.ctl > hello.out***” will save the output in the file **hello.out**.
- “***cat*** *hello.out*” prints the contents of hello.out to the screen.
- You can also open “hello.out” with any text editor.

```
admin@UKBRIWS072 ~
$ mpb hello.ctl
Hello world!

admin@UKBRIWS072 ~
$ mpb hello.ctl > hello.out

admin@UKBRIWS072 ~
$ cat hello.out
Hello world!

admin@UKBRIWS072 ~
$
```



Defining variables

Northumbria University NEWCASTLE

- There are 2 ways of defining variables:
  - ✓If you do not need the ability to change the parameter outside the script:

    (define x 42)
  - ✓If you do need the ability to change the parameter outside the script:

    (define-param x 42)
- Once a variable is defined, its value can be changed in the following corresponding ways:

  (set! x 42)

  (set-param! x 42)

Defining variables

Northumbria University NEWCASTLE

- Variables are defined as follows:
  (define NAME VALUE)

- The value of an existing variable can be changed as follows:
  (set! NAME VALUE)

~

```
admin@UKBRIWS072 ~
$ mpb
mpb> (define x 42)
mpb> x
42
mpb> (set! x 2020)
mpb> x
2020
mpb> (define foo "hello world!")
mpb> foo
"hello world!"
mpb> (set! foo "goodbye world!")
mpb> foo
"goodbye world!"
mpb>
```

Defining variables that can be changed outside your script

Northumbria University NEWCASTLE

- It is often very convenient to run a script with slightly different parameters.
- ✓ Examples:
  - Change the refractive index or radius of something.
  - Enable/disable the geometry to normalize reflection/transmission measurements
  - Enable/disable a defect
  - Change the resolution of a simulation.
- To do so, you can declare variables with:

  define-param NAME VALUE
- This will create a variable called NAME with default value VALUE, but you can change it when running the script as follows:

  mpb NAME=NEWVALUE SCRIPT.ctl
- MPB's input variables such as resolution should be set using set-param! Instead of set!, so that they can be easily changed by command-line, while still allowing you to set your own default values.

Defining variables that can be changed outside your script

Northumbria University NEWCASTLE

- Create a new **example-1.ctl** file with the code on the left, then run it first like this:

  mpb example-1.ctl

- Then like this:

  mpb foo=5 foo_param=55 resolution=555 example-1.ctl

```
(print "=== Custom defined user variables without '-param': \n")
(define foo 42)
(print "after define: foo=" foo "\n")
(set! foo 45)
(print "after set!: foo=" foo "\n")
(print "=== Custom defined user variables with '-param': \n")
(define-param foo_param 42)
(print "after define-param: foo_param=" foo_param "\n")
(set-param! foo_param 45)
(print "after set-param!: foo_param=" foo_param "\n")
(print "=== MPB input variables: \n")
(print "before set-param!: resolution=" resolution "\n")
(set-param! resolution 256)
(print "after set-param!: resolution=" resolution "\n")
(exit)
```

Defining variables that can be changed outside your script

Northumbria University NEWCASTLE

```
admin@UKBRIWS072 ~
$ mpb example-1.ctl
=== Custom defined user variables without '-param':
after define: foo=42
after set!: foo=45
=== Custom defined user variables with '-param':
after define-param: foo_param=42
after set-param!: foo_param=45
=== MPB input variables:
before set-param!: resolution=10
after set-param!: resolution=256

admin@UKBRIWS072 ~
$ mpb foo=5 foo_param=55 resolution=555 example-1.ctl
command-line param: foo=5
command-line param: foo_param=55
command-line param: resolution=555
=== Custom defined user variables without
after define: foo=42
after set!: foo=45
=== Custom defined user variables with '-param':
after define-param: foo_param=55
after set-param!: foo_param=55
=== MPB input variables:
before set-param!: resolution=555
after set-param!: resolution=555

admin@UKBRIWS072 ~
$ _
```

Variables get defined and changed as usual:

Any parameter defined on the command-line will be listed at the start of the output.

Without **-param**, passing a value by command-line has no effect.

With **-param**, the command-line value is used instead of any default values specified in the code. *set-param!* has no effect if a value is passed by command line.

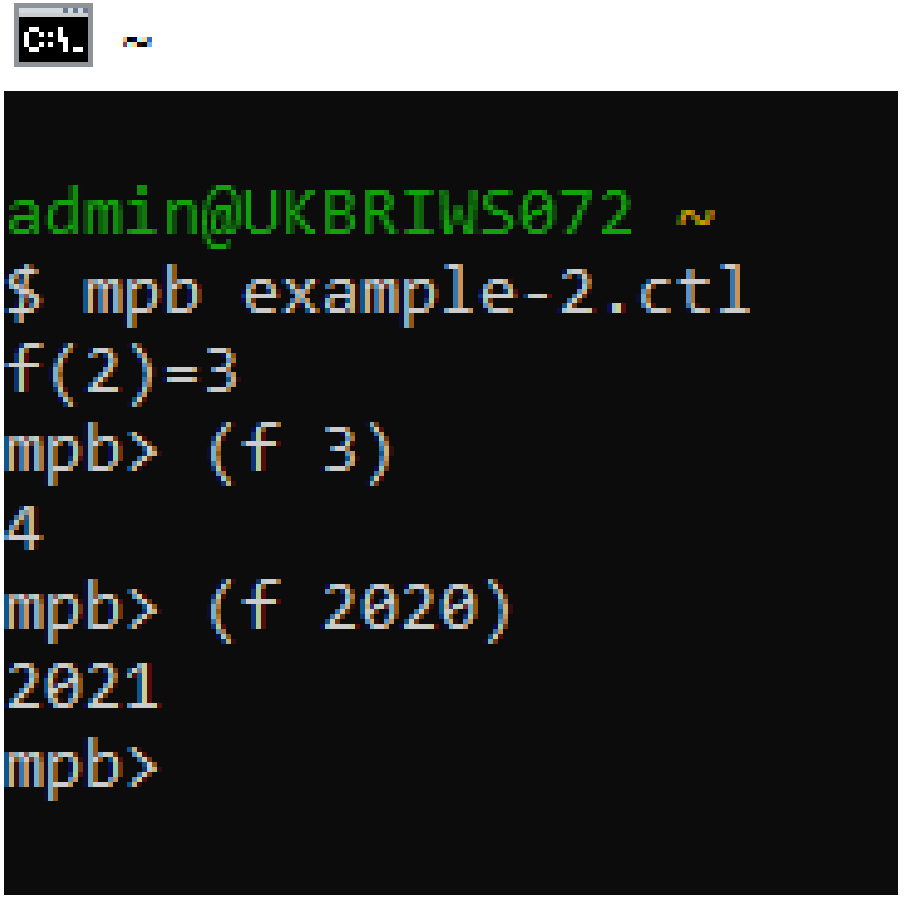
Functions

Northumbria University NEWCASTLE

example-2.ctl

```
; simple function returning
(define (f x)
  (define a 1) ; variable definition
  (+ a x) ; the last value gets returned
)

; test function
(print "f(2)=" (f 2) "\n")
```



Conditional statements

Northumbria University NEWCASTLE

example-3.ctl

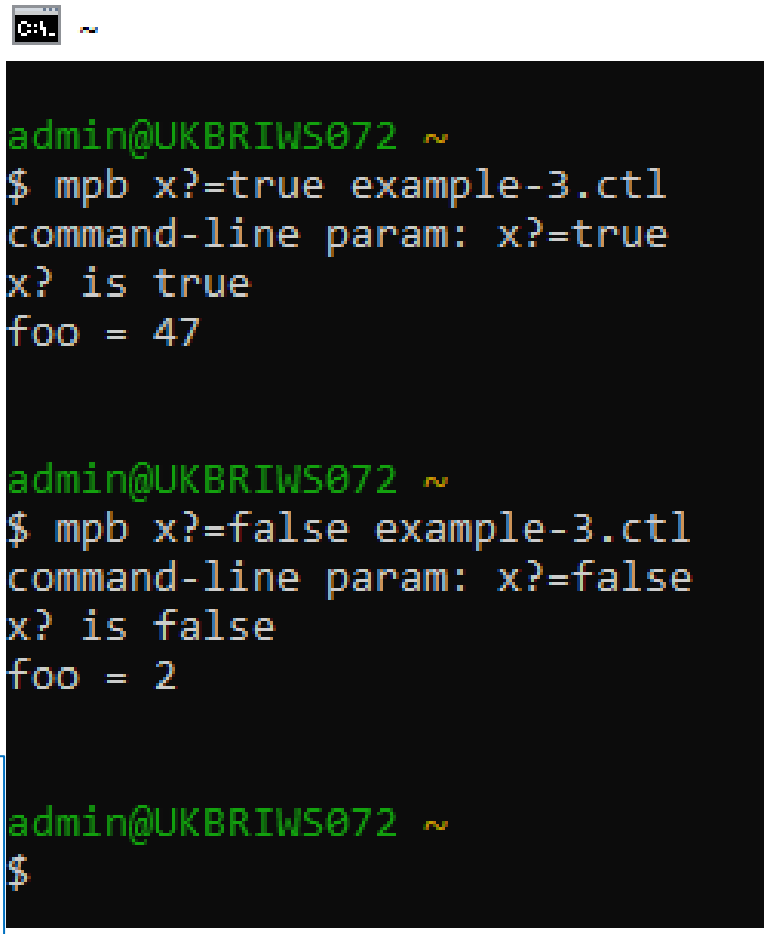
```
(define-param x? true)
(define foo 555)
(if x?
  (begin
    (print "x? is true\n")
    (set! foo 47)
  )
  (begin
    (print "x? is false\n")
    (set! foo 2)
  )
)
(print "foo = " foo "\n\n")
(exit)
```

It is common to name boolean variables with a question mark at the end.

The first statement is executed if the condition is true.

The second statement is executed if the condition is false.

(begin …) is a way of grouping multiple commands into one block. An if statement only allows up to two arguments, so you will need that to do more things in each case.



Conditional statements: inequalities

Northumbria University NEWCASTLE

example-4.ctl

```
(define-param x 42)
(print "x = 2: ") (if (= x 2) (print "TRUE\n") (print "FALSE\n") )
(print "x < 2: ") (if (< x 2) (print "TRUE\n") (print "FALSE\n") )
(print "x <= 2: ") (if (<= x 2) (print "TRUE\n") (print "FALSE\n") )
(print "x > 2: ") (if (> x 2) (print "TRUE\n") (print "FALSE\n") )
(print "x >= 2: ") (if (>= x 2) (print "TRUE\n") (print "FALSE\n") )
(exit)
```

```
admin@UKBRIWS072 ~
$ mpb x=1 example-4.ctl
command-line param: x=1
x = 2: FALSE
x < 2: TRUE
x <= 2: TRUE
x > 2: FALSE
x >= 2: FALSE

admin@UKBRIWS072 ~
$ mpb x=2 example-4.ctl
command-line param: x=2
x = 2: TRUE
x < 2: FALSE
x <= 2: TRUE
x > 2: FALSE
x >= 2: TRUE

admin@UKBRIWS072 ~
$ mpb x=3 example-4.ctl
command-line param: x=3
x = 2: FALSE
x < 2: FALSE
x <= 2: FALSE
x > 2: TRUE
x >= 2: TRUE

admin@UKBRIWS072 ~
$
```

Conditional statements: useful operators

Northumbria University NEWCASTLE

- **(and ARG1 ARG2)**: True if both ARG1 and ARG2 are true.
- **(or ARG1 ARG2)**: True if ARG1 or ARG2 is true.
- **(not ARG)**: Invert logical value.
- **(equal?** "hello world" "hello"**)**: Compare strings.

Loops

- The classic way, in Scheme, is to write a tail-recursive function:

```
(define (doit x x-max dx)
  (if (<= x x-max)
    (begin
      ...perform loop body with x...
      (doit (+ x dx) x-max dx)
    )
  )
)
(doit a b dx) ; execute loop from a to b in steps of dx
```

- There is also a do-loop construct in Scheme that you can use:

```
(do
  ( (x a (+ x dx)) ) ; <variable> <init> <step>
  (  (> x b) ) ; <test>
  ...perform loop body with x...
)
```

- If you have a list of values of x that you want to loop over, then you can use map:

```
(map (lambda (x) ...do stuff with x... ) list-of-x-values)
```