# Operating Systems Practicals

# Lab 1 – Windows cmd.exe

**Exploring the use of commands with options and arguments**

Taking a selection of Windows CLI commands from those given below, use the online help to examine the various options and arguments, and try them out. You can look up others at: <http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/ntcmds.mspx?mfr=true>

You're required carefully to write two A4 pages (Times 12 point or equivalent size) detailing your experiments with different options for between six and ten different commands.

To get the online help for a command, type command /?

e.g.

dir /?

*prompt*

*mkdir*

*color*

*title*

*tree*

*type*

*ver*

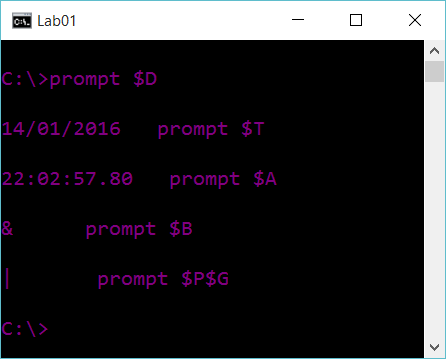
*print*

*xcopy*

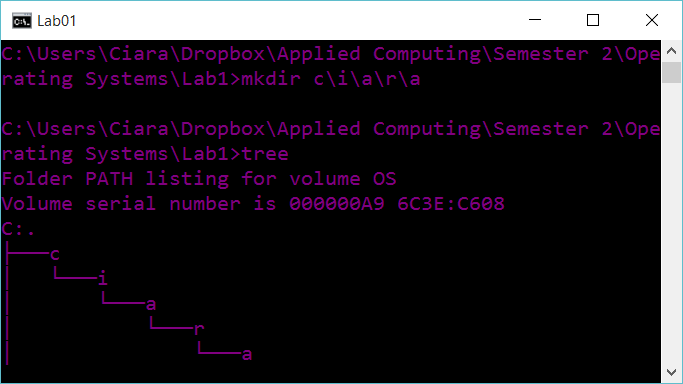
Type *help* at the windows command line prompt to see some more instructions.

**Prompt**

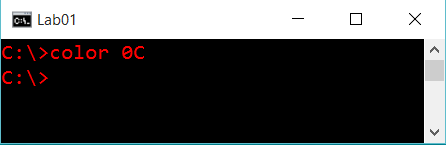
This command line argument is used as a prompt for a new command. The prompt is usually in the format “prompt [text] “, where the text element specifies the new command prompt. It can consist of normal characters and a few certain special codes, for example by entering “prompt $D”, the command line produced today’s date 14/01/2016. Another example would be “prompt $T”, which gave the current time in the format 18:22:33.89.These two examples are very useful as the exact time and date of the system is given. More examples of this command include prompts resulting in symbols such as ‘&’ and ‘|’, which are “prompt $A” and “prompt $B”, respectively. To get back to the default prompt showing the drive and path, the command is “prompt $P$G”.



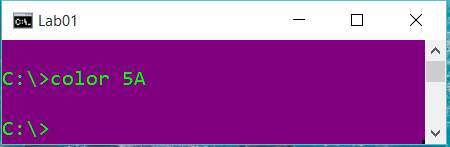
**Mkdir**

This command line input allows the user to create a path of directories that don’t previously exist. For example, I created a path of directories C:\c\i\a\r\a, by typing “mkdir c\i\a\r\a” while in the C: drive. This is a very easy and fast way to create many folders within folders, rather than going through the time consuming windows GUI methods of creating the same directory path.

**Color**

The color command allows the user to change the cmd background and foreground colour scheme, using two hex digits. This is written in the format “color [background] [foreground]”, with a hex digit being present for background and foreground. Certain colours are given a number/letter. For example, 0=Black and C=Light Red. By typing color 0C, I got the color output:

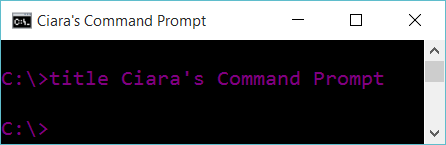
By typing colour 5A, the colours changed to:

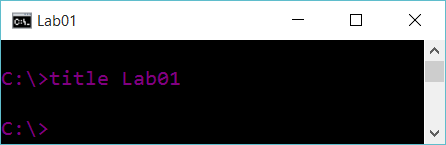


If no attributes are entered, the colours of the command prompt will revert back to the original colours it opened with. If equal attributes are entered, an error is flagged and the colour change doesn’t execute. This method of changing the colours is much faster than using the GUI and changing the colours through the properties option of the cmd.

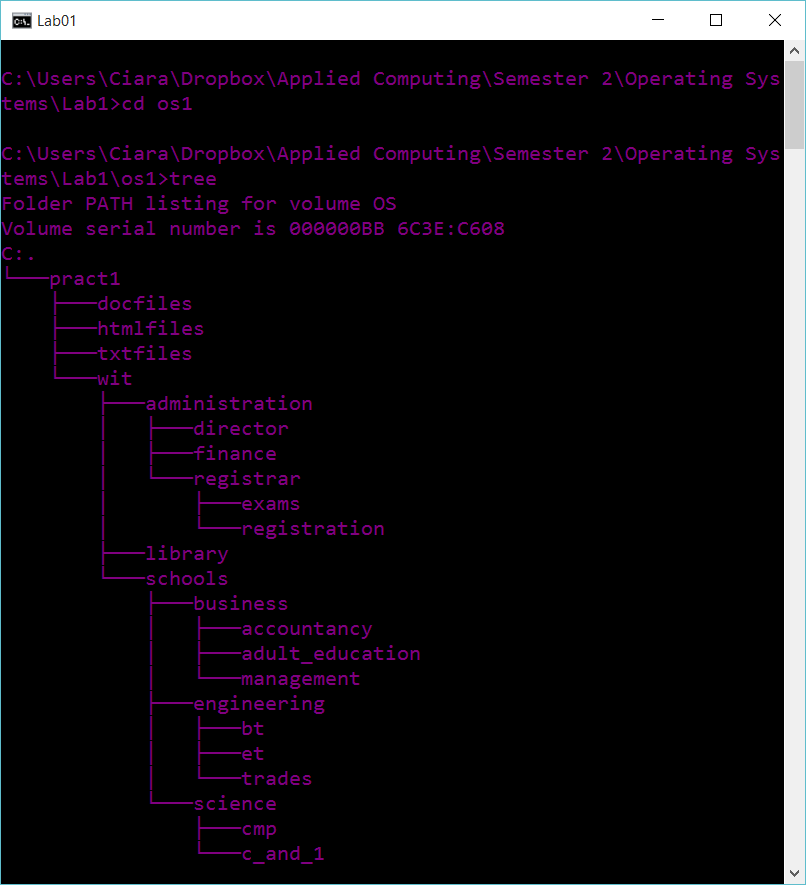
**Title**

This command line input allows the user to change the title of the cmd window. “Title” is a very useful tool for programmers that have more than one cmd window open at a time running different tasks, to distinguish between workspaces. This command is in the format “title [string]”, where the string is what the user wants to have on the cmd window. For example, I typed “title Ciara’s Command Prompt” and “title Lab01”, which produced the following outputs at the top of the window, respectively:

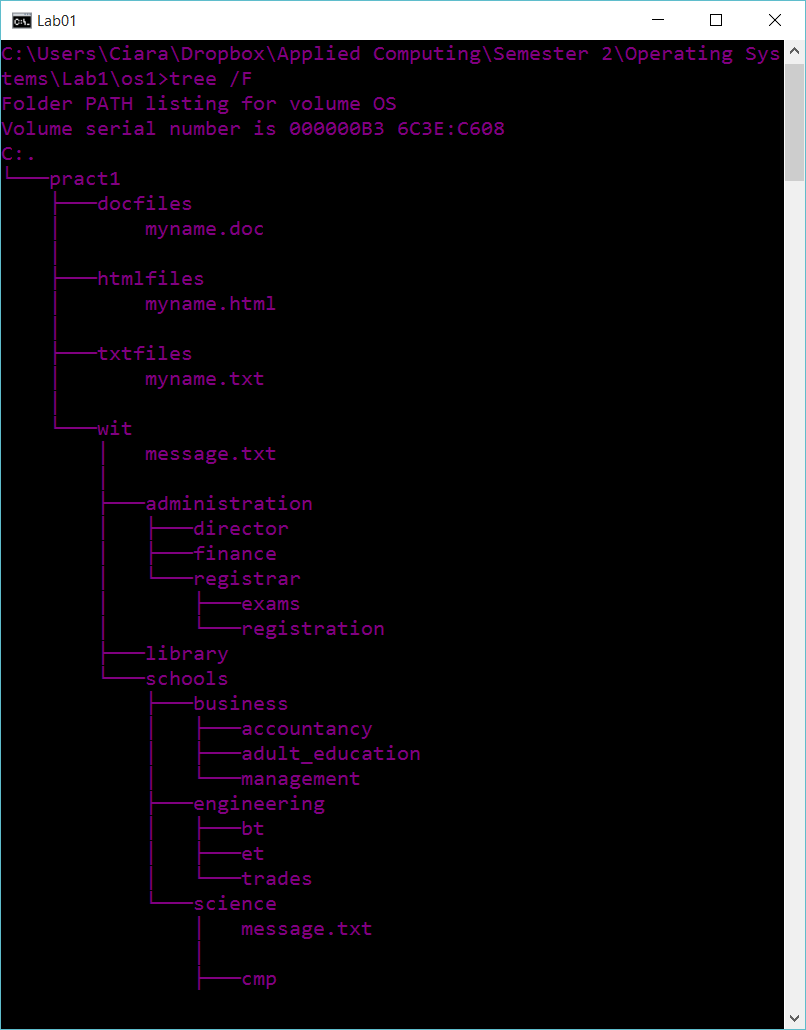




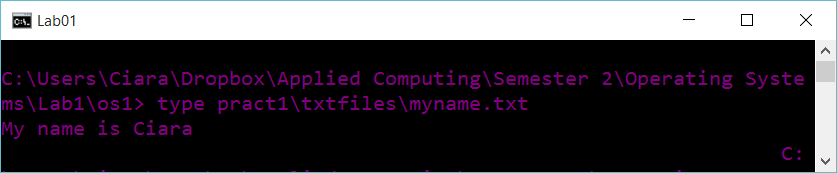
**Tree**

This command line input draws a tree diagram of every folder that lives within the root folder the cmd is working from. This is very useful as a tree like diagram is not easily available through the GUI windows explorer. For example, I wanted a tree diagram of the os1 folder below:

If the command “tree \A” is entered, ASCII values will be shown instead of extended characters. If the command “tree \F” is entered, each file is shown within the folders also e.g. myname.doc. I found this very useful as I could directly see where a file was kept, to navigate easily to it using the cmd window only.

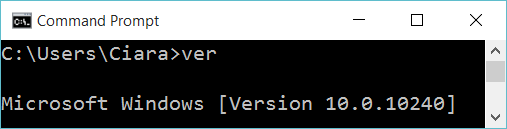


**Type**

The “type” command displays the contents of a file. It is used in the format “type file.txt”. For example I used this command to show the contents of the file myname.txt below:

This is very useful to quickly see the contents of a text file especially, without leaving the cmd window.

**Ver**

This command is a very quick and easy way to get the exact windows version that is running on your computer. It is much easier than using the properties tab of ‘My Computer’ in the GUI.

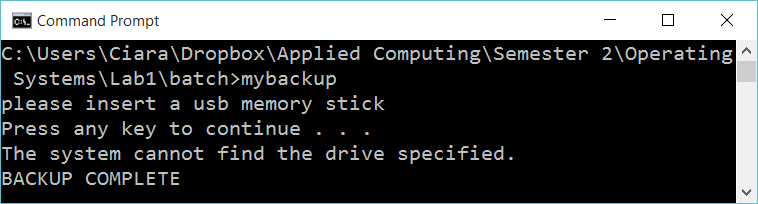
**Print**

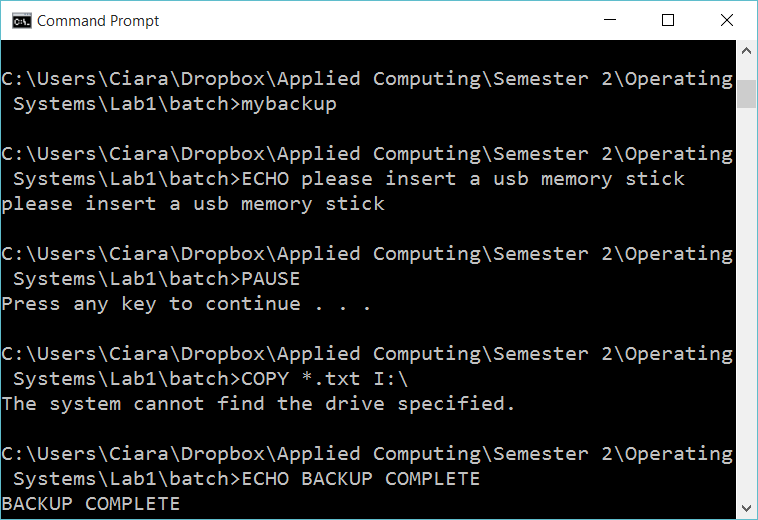
To print a document through cmd, this command is used. The format of this command is ”/D: device drive: path filename “ . For example, I used this command to print to Microsoft PDF. This is a very handy way to print a document in one quick step.

**Xcopy**

This command copies files and directory trees. It can copy these taking into consideration certain guidelines set by the cmd user. It is used in the format “xcopy file destination guidelines”. For example, the guideline \P prompted me before creating each destination file. \H copied hidden files and system files also. \D:m-d-y copied files changed on or after the specified date, for example I used this to copy files created on the 18/01/2016, which was useful to efficiently copy the files I created that day.

1. What’s the purpose of the first line - @ECHO OFF? Remove it and see the effect.

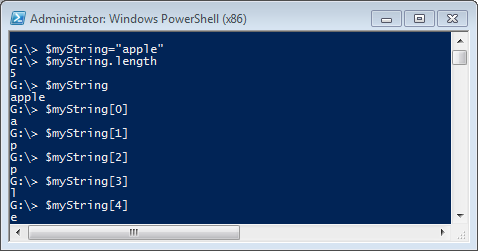
With the @ECHO OFF line, the commands all run together.

However when I removed this line of text from the batch file, each line ran separately, causing the batch file not to work efficiently.

# Lab 02 – PowerShell

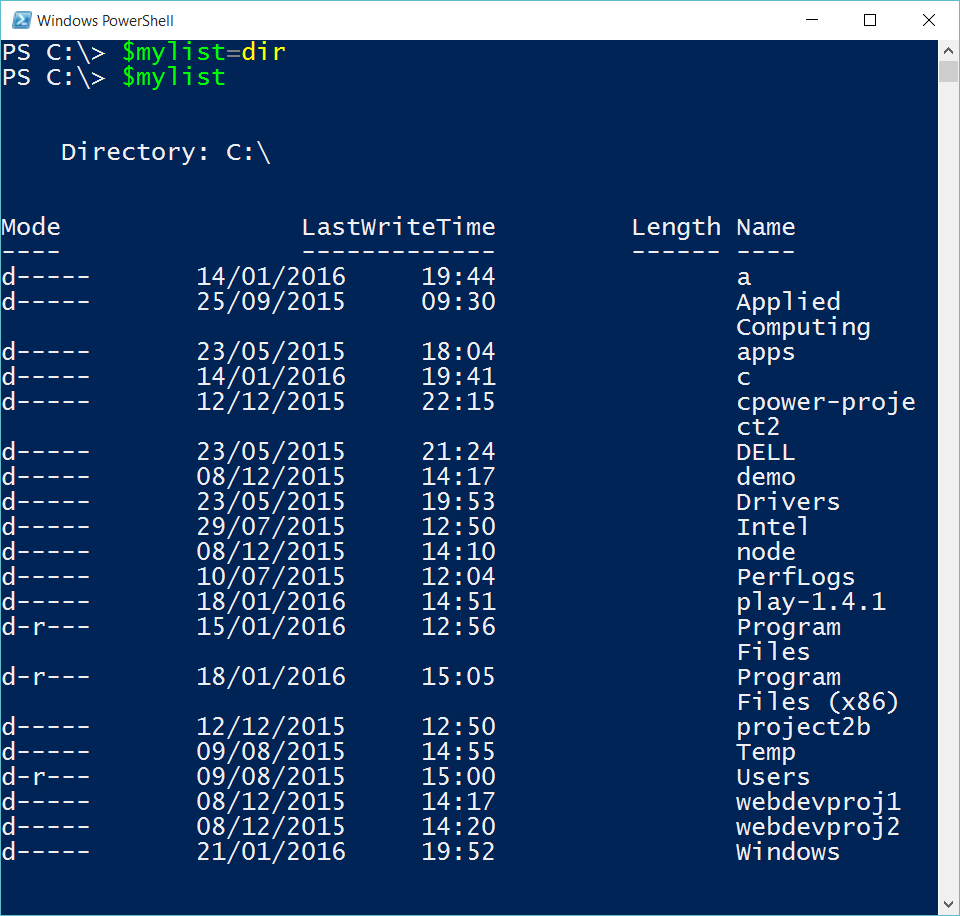
10. Arrays

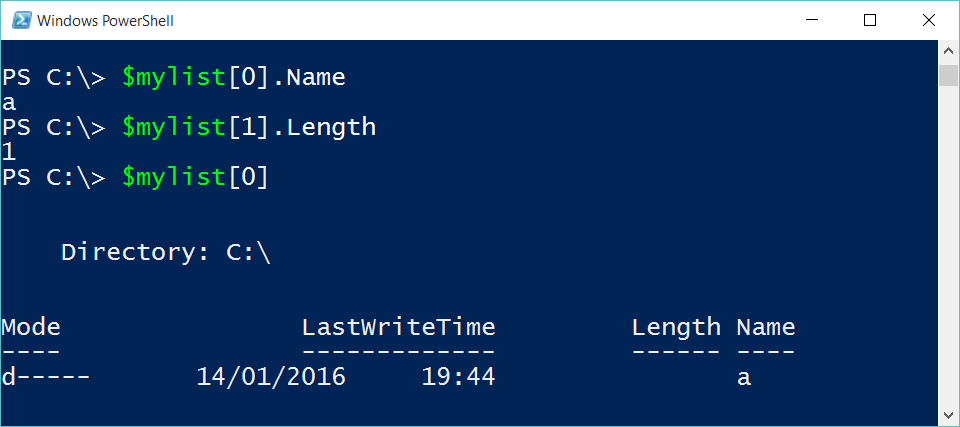
(10a) Test the above using your directory created earlier in part 4.



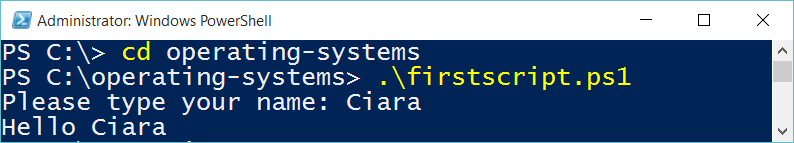
The variable $myString is a string. I used the command to single out each individual element of the array, as shown above.

Below, I created a variable $mylist to equal the working directory. This directory is shown below.



I then performed the required commands on this directory. The command $mylist[0].Name returns the Name element of the first entry of the array. When the command $mylist[1].Length is entered, the value for the Length element of the second entry of the array is returned. The command $mylist[0] simply returns the full first entry of the array, including all element categories.

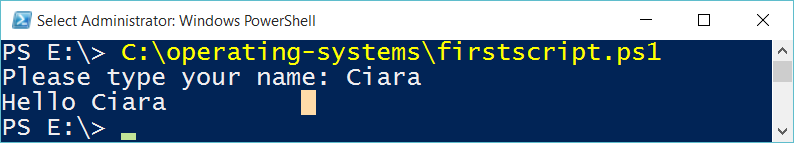
13. Creating a simple PowerShell script and getting it to run

13(a) Simply record what happens when you run this script.

The code that was saved in firstscript.ps1 was:

*$name = Read-Host "Please type your name" Write-Host "Hello" $name*

When this script was ran, all code in the file is executed. A variable $name is created to equal a string input by the user (using the Read-Host command). This is printed back out to the user by using the Write-Host command.

13(b) Find out how to run scripts if they’re in a directory other than the working directory.

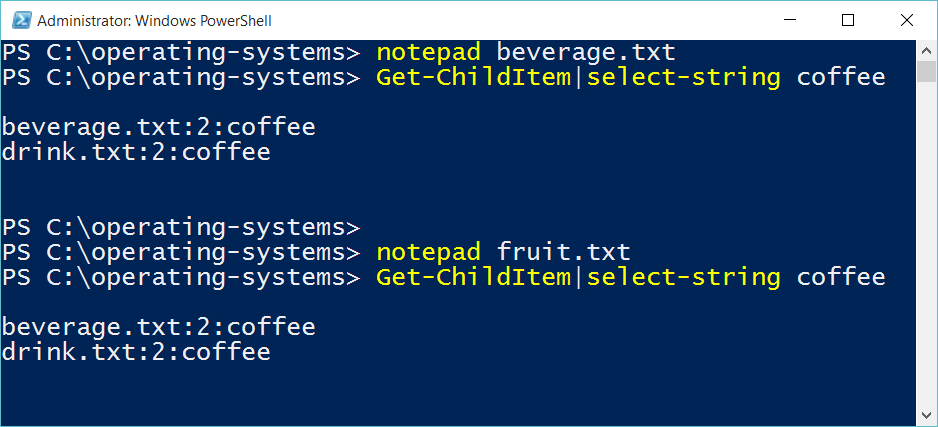
From the working directory E: drive, I ran the firstscript.ps1 (located in C:\operating-systems\ ) from this working directory by typing in the full path of the script file. This executed as normally.

14. Pipes

(14a) Create another file called beverage.txt identical to drink.txt.

(14b) Issue this command again “PS C:\Users\User1\green> Get-ChildItem | select-string coffee” and see what happens.

(14c) Create a file called fruit.txt with the list apple orange banana in it.

 (14d) Issue this command again “PS C:\Users\User1\green> Get-ChildItem | select-string coffee” and see what happens.

For (14a), I created the file beverage.txt, containing the text “tea coffee cocoa”.

For (14b), I issued the instructed command, which returned the files within which the string “coffee” is contained. For this example, it was present in beverage.txt and drink.txt. It also returns the position of the string in that file, in both cases here it is the second string.

For (14c), I created the text file fruit.txt, containing the text “apple orange banana”.

For (14d), I used the Get-ChildItem command again with the same instructed usage as before. This again returned all files that contain the string “coffee”. As shown, the file fruit.txt is not returned, as it does not contain the string “coffee”.

15. Concluding Things to Do

1. Explain why it’s found that people who know PowerShell well score the same sort of marks in tests of their knowledge regardless of whether they’re first rate experts or just people who know PowerShell well. What is it that gives the first rate experts the edge?

Powershell is quite easy to understand. People that know it well will know a great deal of information associated with Powershell, while first rate experts may just know of more useful and efficient ways to execute the same commands.

2. Find out and explain how to get help about any cmdlet.

The command “Get-Help” will bring up a list of all available help topics. All cmdlets will be shown in this list for the user to clearly see. To choose a particular cmdlet and get specific help on this topic, the user must type “Get-Help” followed by the required command, for example “Get-Help Add-Content”. Information about the use of the command will then be shown on screen.

3. Find out and explain what F7 does in PowerShell.

Pressing the F7 function key displays a menu of the previously executed commands. It displays a pop-up window with your command history and allows you to select a command. Use the arrow keys to scroll through the list. Press Enter to select a command to run, or press the Right arrow key to place the text on the command line.

4. What is the purposes of the following in PowerShell:

These two switches are used when the user is typing a command and they are unsure of the consequences. PowerShell can be very dangerous as all commands and resources are available to the administrator, which leads to the possible consequences of deleting or changing important computer system files.

1. The -whatif switch

The “-WhatIf” switch can be typed after a command, to show what will happen when the command is executed, without actually executing the command. It will list each action that would occur, all on separate lines. This is very useful to clearly see the consequences of the command in question.

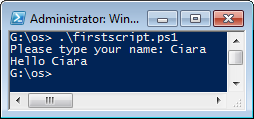
(b) The –confirm switch

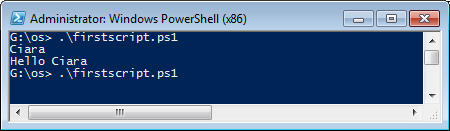
The “–confirm” switch is very similar to the “-WhatIf” switch, as it is also typed after a command to show what would happen when the command is executed. However, rather than listing the actions one by one as in the “-WhatIf” switch, this switch results in a dialog box pop-up to prompt the user to make a decision on the action that could occur if the command is executed. For example, for a “stop-process –confirm” command, a dialog would appear for each process to be stopped, with buttons for the user to make a choice –usually Yes/No/Yes to all/No to all. This is also very useful to clearly see the consequences of the command in question. This dialog box may also show up for a command even if the “-confirm” switch is not typed. This occurs when the command is seen as a high impact command (stopping a system process that the user did not start).

5. Write a note to explain how you can use tab to complete a command as soon as it’s unambiguous.

When typing in a command in PowerShell, we can use the tab button to autocomplete the command. For example, by typing “di” and then pressing the TAB button, PowerShell will try guess what command we are trying to type, by suggesting commands that begin with “di”. Many suggestions appear, the user presses TAB to scroll through the suggested commands, and ENTER to select. If what the user has typed in is unambiguous and there is only one command that corresponds, the TAB will automatically fill in this command.

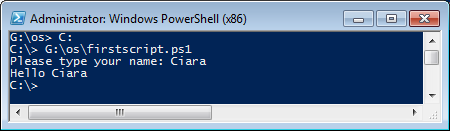
# Lab03 –PowerShell

Exercise 1 Simply record what happens when you run this script. What difference does it make if you leave out the text "Please type your name" from the first line of the script?

When this script was ran, all code in the file is executed. A variable $name is created to equal a string input by the user (using the Read-Host command). This is printed back out to the user by using the Write-Host command.

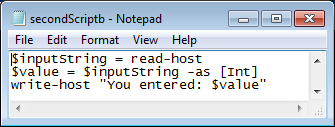
I edited the script so the text “Please type your name” wasn’t present. When I ran the script, PowerShell was waiting on input, but gave no text to prompt the user and let them know that it was waiting for input. Unless input was given, the script didn’t continue.

Exercise 2 Find out how to run scripts if they’re in a directory other than the working directory.

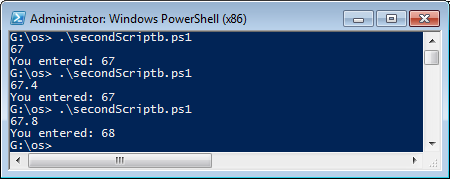
From the working directory C: drive, I ran the firstscript.ps1 (located in G:\os\ ) from this working directory by typing in the full path of the script file. This executed as normally.

Exercise 3

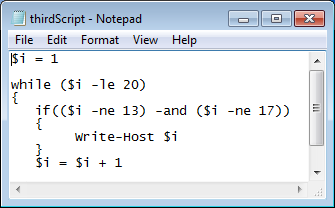
1. Find out how to do the same thing that the code above does except that it’ll only accept integers (such as 67).
2. Once you’re found the answer, find out what happens if you type a real number (such as 67.4 or 67.8)

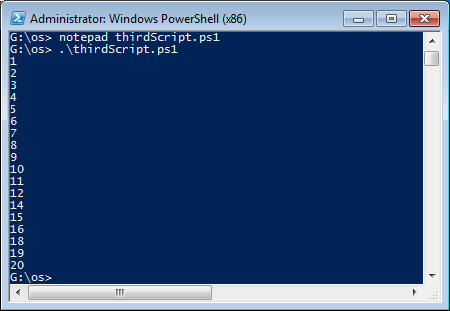
a)

Instead of having “ –as[Double] “ , I changed the script to include “-as [Int]”. This took a string as input and converted it to an Integer. If a string was entered that can’t be converted to an integer (eg h) , then no value was returned to the user.

b)

The above picture shows the script running three times. First, I gave “67” as input, and the value “67” was returned as it is an integer. Secondly, I gave “67.4” as input, and “67” was returned, as it is a real number , it was rounded to the nearest integer. Thirdly, I gave “67.8” as input, and “68” was returned, as it is a real number , it was also rounded to the nearest integer.

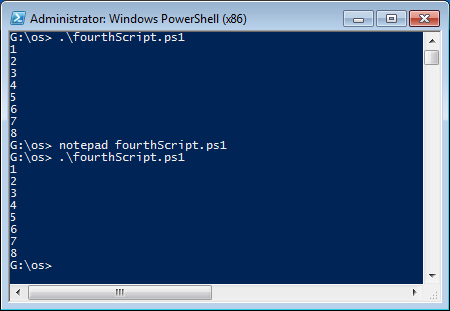
Exercise 3.c : What does the below script do?

This script sets a variable i . While i is less than 20, an if statement condition is tested. The script has conditions to only print out the variable if it is not equal to 13 and 17.The variable is incremented before the loop happens each time. This results in the output below (no 13 or 17 printed to user).

Exercise 4: We don’t need the brackets round ($i -ne 13) and ($i -ne 17) in the code above. Why do we not need them? (Hint: The answer is the same as for Java). Do you think that it’s a good idea to put them in even if they aren’t necessary? Explain your answer.

We don’t need the brackets because the equal, not equal, less than and more than conditions have greater precedence over the –and operator. I think we should keep the brackets to ensure easy readability, and to avoid any confusion.

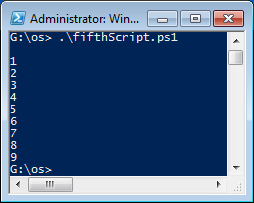
Exercise 5: We can replace $i = $i + 1 with something shorter, in the above two scripts. What do you think it is? Try it and see.

In the two previous scripts, we could replace “ $i=$i+1” with “$i++”. This does the same as the original statement, the variable $i still increases by 1 each time. Both give an identical output, shown below.

Original ---

Changed---

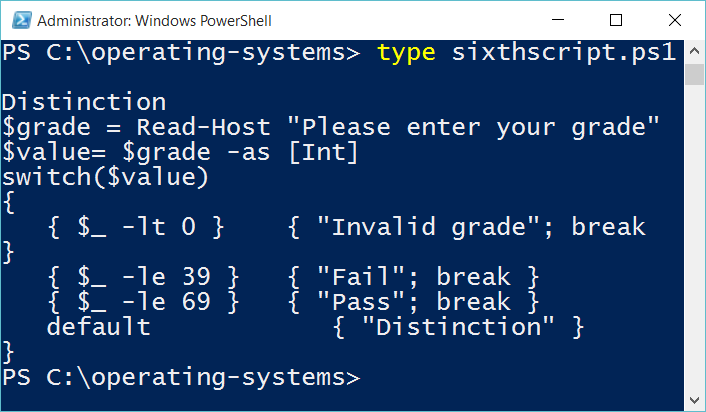
Exercise 6: Oops, I've put semicolons at the end of each of the lines, in the *do...until* loop above. I suppose it's because of my experience in writing programs in other languages that sometimes I put semicolons at the end of a line of PowerShell script, even when they're entirely unnecessary in PowerShell. Does PowerShell forgive me for doing this? Find out, and write your conclusion.

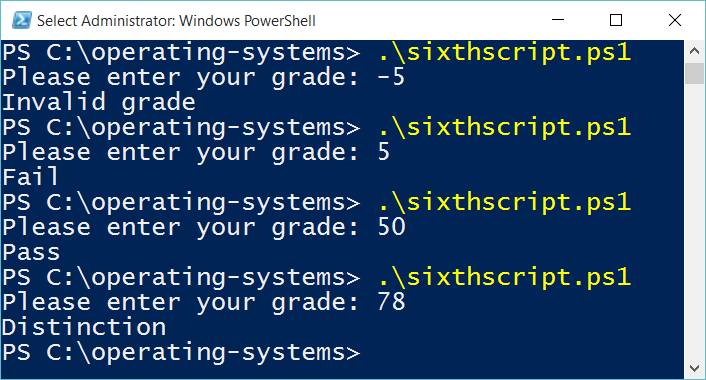


Powershell ignores the use of the semicolon. The script runs as expected, with or without the semicolon. This is useful as the user may be swapping between languages, so may incorrectly inout a semicolon in PowerShell.

Exercise 7: Alter this program to deal with grade categories (for example >70 is a distinction mark etc) in an examination and also allow the user to enter a grade.

I changed the script, to read in a grade from the user. I used a $value variable to convert this string to an integer. All possibilities are catered for here.

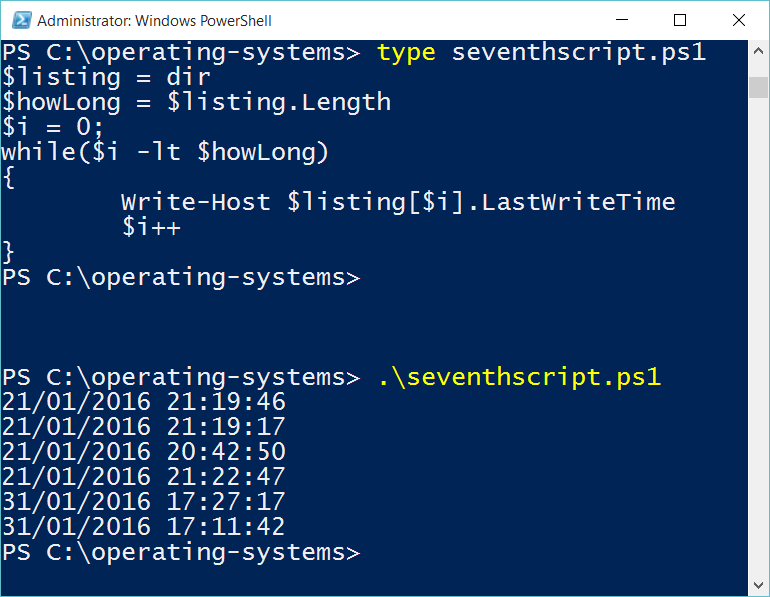




Exercise 8. Explain what the above example does. Modify it to show fields other than the *Name* field.

The code shown sets an array variable to include the contents of the directory. Another variable is set to equal the length of the array. A while loop is then used, with the counter variable $i. The Write-Host command is used to print out the name of each item in the array, which continues as the $i variable is incremented and does not equal to the length of the array.

To show more than the name field of each item in the array, I changed the code as shown below to show the LastWriteTime.



Exercise 9. Draw up a chart to show equivalent syntaxes for different control structures/data structures among Java, Unix script, and PowerShell. You'll have to revisit this question when you've learned some Unix/Linux.

|  |  |  |  |
| --- | --- | --- | --- |
| Data/Control Structure | Unix/Linux | Powershell | Java |
| Variable | variableName= variableValue | $variableName=variableValue RTN | variableType variableName = variableValue ; |
| Array | arrayName[index]=value | $arrayName = value1,value2,value3,value4 | dataType[] arrayRefVar = new dataType[arraySize];  OR  dataType[] arrayRefVar = {value0, value1, ..., valuek}; |
| Equal to? | == | -eq | == |
| Not Equal to? | != | -ne | != |
| Greater than or equal ? | -ge | -ge | >= |
| Greater than? | -gt | -gt | > |
| Less than or equal? | -le | -le | <= |
| Less than? | -lt | -lt | < |
| Logical AND | -a | -and | && |
| Logical OR | -o | -or | || |

Exercise 10. Compare the ways in which scripts are enabled to run in (a) Unix/Linux and (b) PowerShell. Again this is a question for review when you've done some Unix/Linux.

Unix/Linux: $ chmod u+x stats.sh

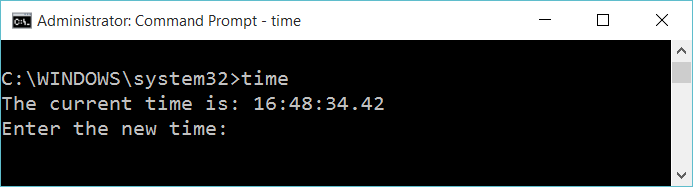
Powershell: C:\Users\User1\Set-ExecutionPolicy Unrestricted

The above are the two ways to enable a script to be run. Both change the permissions of the file.

# Lab04 – Unix/Linux

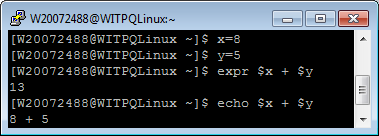
1. Find out what happens when you type time at the command line in (a) UNIX and (b) Windows CLI.
2. The “time” command in UNIX reports how long it took for a command to execute. It will display three statistics:
   * The elapsed real time between invocation and termination,
   * The user [CPU](http://www.computerhope.com/jargon/c/cpu.htm) time, and
   * The system CPU time.

It is entered with a command following it, this is the command that will be timed during execution.

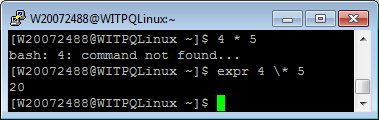
1.  The “time” command is used in Windows CLI to display the current time. It also prompts the user to enter a new time, allowing the user change the time held on the system.
2. If you have time left over from the above, please use that time to make your own summary of the last three practicals. You can also make your own table of Windows CLI, Windows PowerShell and UNIX commands, that will be less extensive than the one I’ve given on Moodle. If you haven’t got time during class, please do it at home, and put it in your portfolio.

|  |  |  |
| --- | --- | --- |
| Unix/Linux | Powershell | Windows DOS |
| Ls | Ls / dir | Dir |
| Cp | Cp / copy | Copy |
| Mv | Mv / move | Move |
| Rm | Rm / del | Del |
| Netstat | Netstat | Netstat |
| Man | Man / help | Help |
| Clear | Clear / cls | cls |
| Ps | Ps / tasklist | tasklist |
| Cat | Cat / type | type |
| History / h | History / h / F7 | F7 |
| Grep | Select-String |  |
| Edit | Edit | Edit |
| Kill | Kill / taskkill | Taskkill |

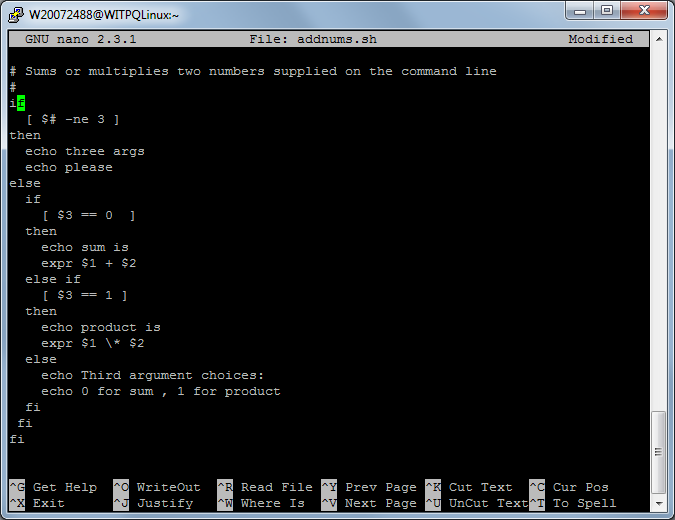
# Lab 05 – Unix/Linux

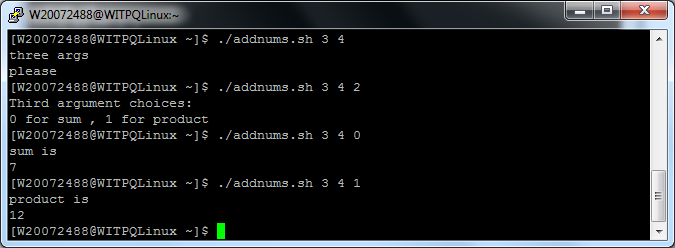
Exercise: 1. One might have supposed that echo would work on the last line above instead of expr. Try it with echo and see what happens. Later we’ll see a way of doing what this program does using echo.

The expr command printed the result of the numerical calculation back to the user, however the echo command only printed the variables out as a string, rather than performing the calculation.

2. Something for you to find out: Find out how to do multiplication in Linux script. It is not as you might suppose, simply by replacing + with \*

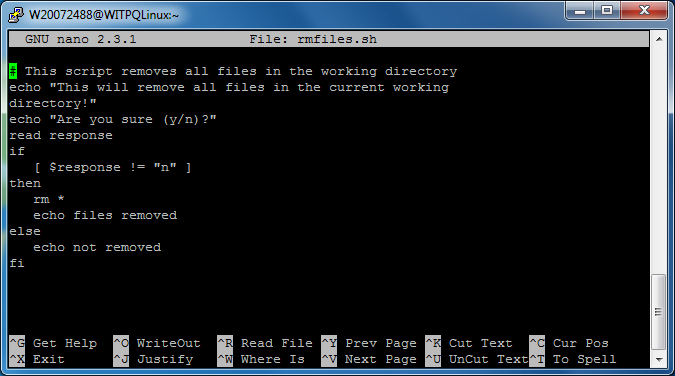
The usual multiplication sign \* does not work in Linux. Instead, the user must type \\* to multiply numbers. Both cases are shown above.

3. Modify the above code (addnums.sh) so that a third command line argument is used to specify whether the two numbers are to be added or multiplied.

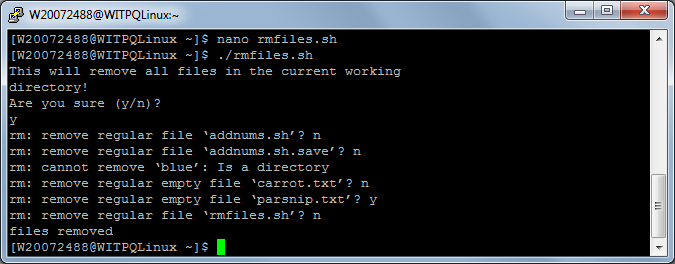
I used the third argument to be 0 for sum, or 1 for multiplication. I used an if statement to check if there are 3 arguments, and a nested if statement to check if the third argument is 0 or 1, which gives an error if the third argument is an invalid number. Multiplication is performed if the argument is 1; addition is performed if the argument is 0.

4. Rewrite the above program with using != rather than =.

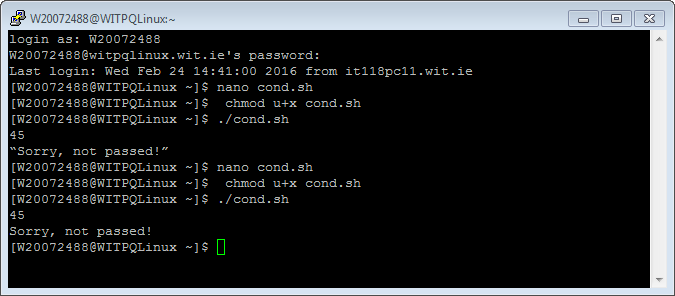
The if condition was changed to ‘ !=”n” ‘ instead of the original ‘ =”y” ‘.



5. Modify the above script to replace ‘rm \*’ with ‘rm –i \*’ and note the difference

When I changed rm\* to rm –i \* , I was prompted for a response each time as the console asked did I want to delete each file one by one, to allow me to decide which files to delete.

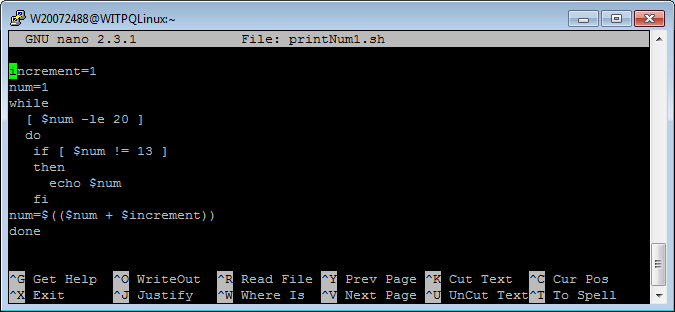
6. What difference does it make if the inverted commas around Sorry, not passed!! is omitted?



When the inverted commas were taken out, the string was printed out regardless. This is because the echo command was used, which prints out a string regardless of the inclusion of inverted commas.

8. Modify the program at the end of section 6 to print all the numbers between 1 and 20 except the supposedly unlucky 13

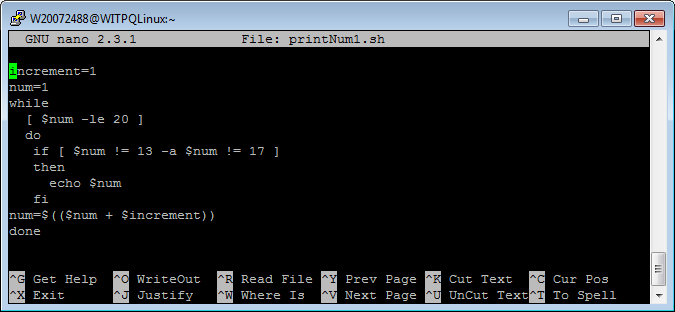
To print every number between 1 and 20, not including 13, I simply added an if statement to test whether the num variable was 13 or not. The num variable was printed out if it did not equal 13.

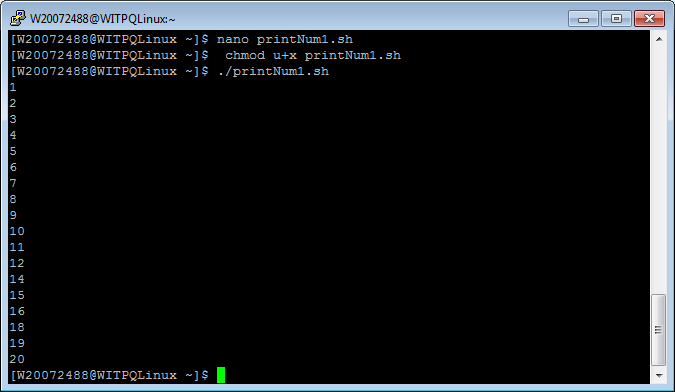


This gave this output:

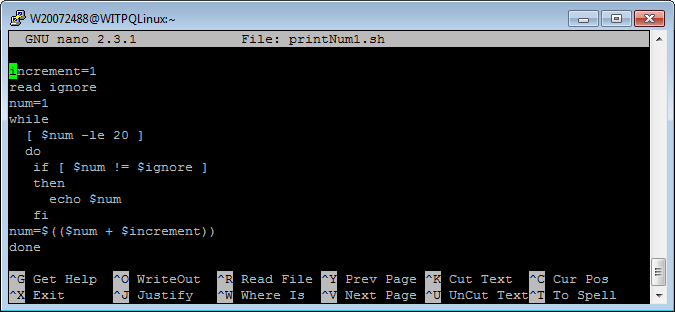
9. Modify your answer to the above problem to exclude 17 also.

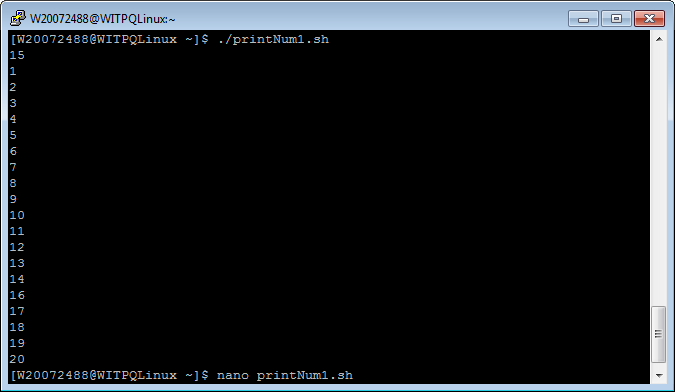
For this I simply added to the if statement condition, by testing if the num variable is not equal to 13 AND not equal to 17.



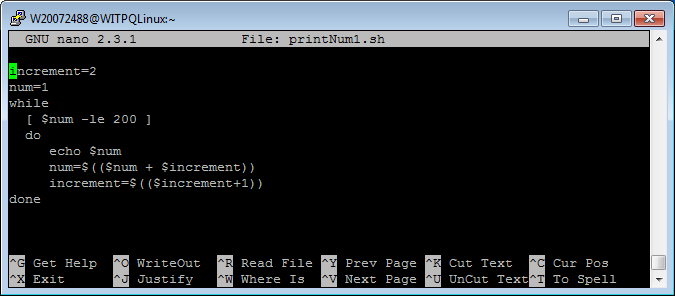
This gave this output :

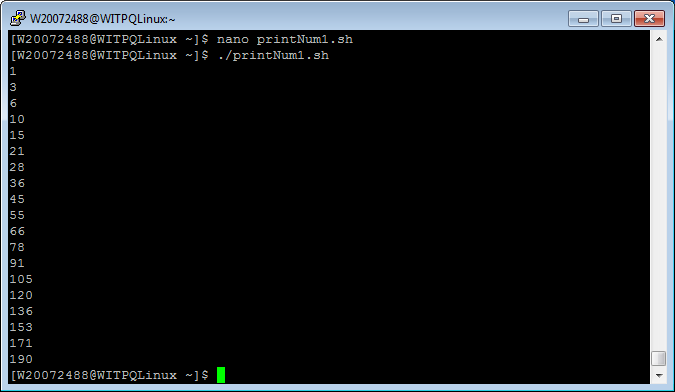
10. Modify the above program to print all the numbers between 1 and 20 except a number provided by the user after a request to do so by the program.

I added a variable to read the user input , and put this variable in the if statement condition, to test if the num variable is not equal to it.

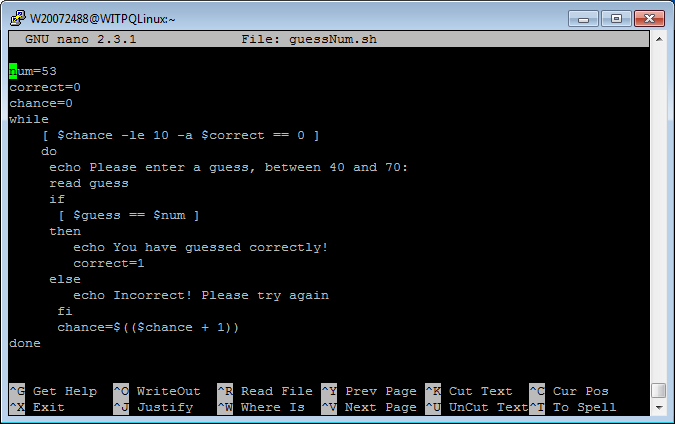
This gave this output :

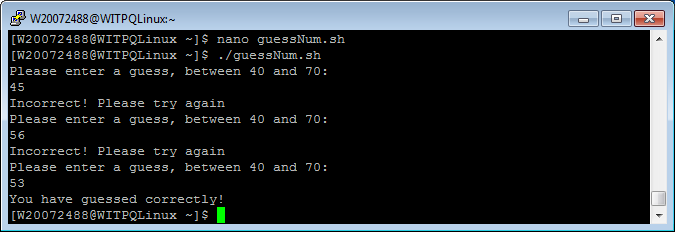
11. Write a program to print out the series 1, 3, 6, 10, 15, 21... until the number in the series below just below 200.

I added a line of code to increment the variable $increment by 1 each time, so that the difference between the numbers gets larger by 1 each time.

This gave this output:

12. Write a Linux script to give the user up to 10 chances to guess a number the computer is “thinking of”. For example, suppose the computer is thinking of the number 53, say, and the user has to guess that. Assume the user only knows that the number is between, 40 and 70, say; thus he/she has a sporting chance.

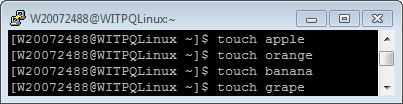


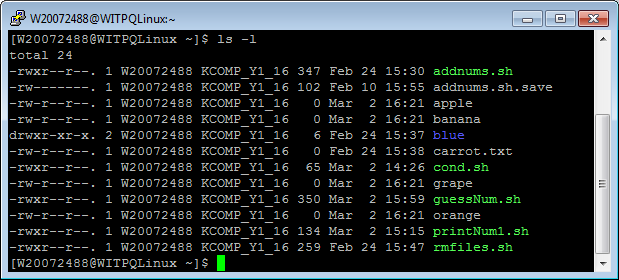


# Lab

# Lab06 – Unix/Linux

1. Create, using touch, four files called apple, orange, banana and grape.



1. Determine the initial permission values for these newly created files.

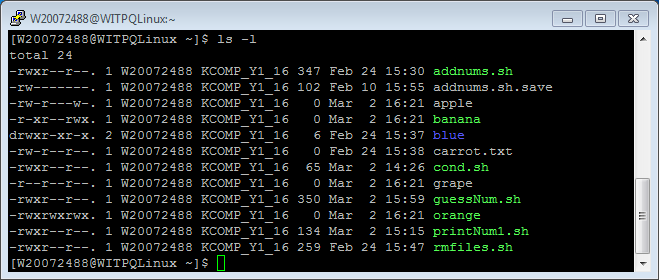
The permissions for each file apple, banana, orange, and grape shown above are: rw-r -- r—

1. State what the read, write and execute permission will become for the following chmod commands:
2. chmod 642 apple
3. chmod 777 orange
4. chmod 547 banana
5. chmod 444 grape

Now try it out on your computer, and, by using ls –l, check your work

Predictions:

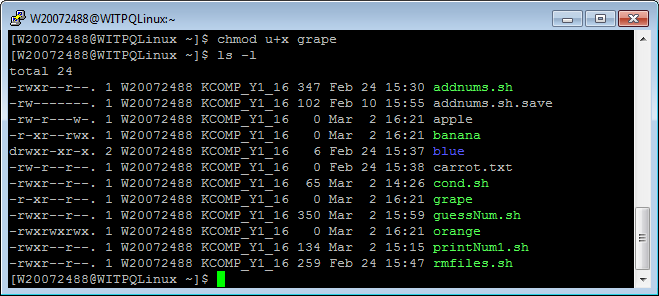
1. rw-r---w
2. rwx rwx rwx
3. r-xr--rwx
4. r—r—r

Outcome:

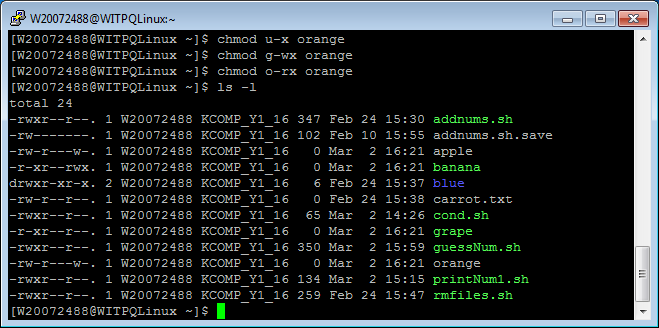
1. rw-r---w
2. rwx rwx rwx
3. r-xr--rwx
4. r--r--r
5. Continue now, by issuing the following command: chmod u+x grape. What do you predict will be be the read, write, execute permissions now? Check your answer by using ls –l

Predictions:

r-xr--r--

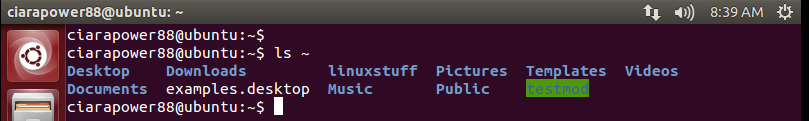
Output:

r-xr--r--

1. Make the file called orange have the same permissions as the file called apple.

# Lab 07 – Unix/Linux

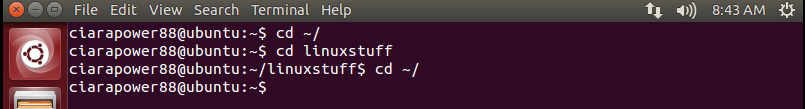
Exercise 1: Find out what is listed by: $ ls ~



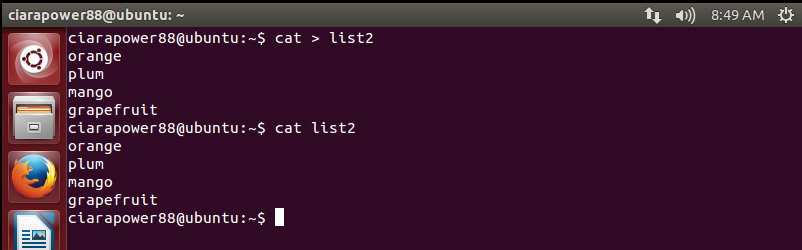
This command listed everything in the starting root directory.

Exercise 2: Find out what is listed by: $ ls ~/ ..

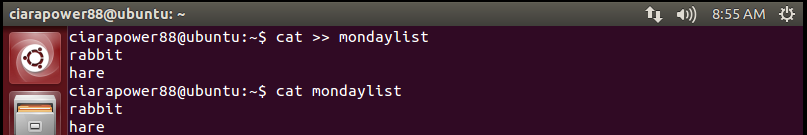
This command lists the parent directory.

Exercise 3: Find out what $ cd ~/ does.

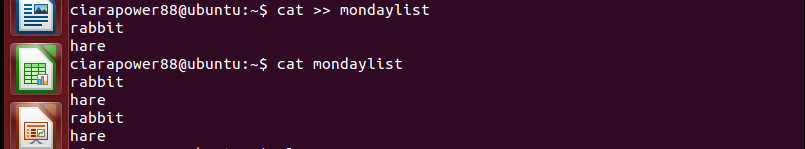
This command changes the working directory to the root directory folder. For example, nothing changed when I was already in the root directory, but when I was in a sub directory, the working directory changed to be at the root.

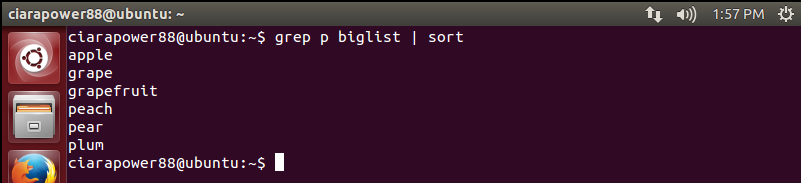
Exercise 4: Using the above method create another file called list 2 containing the following fruit: orange, plum, mango, grapefruit. List the contents of list 2.

Exercise 5: Find out what happens if you do cat >> mondaylist rabbit hare ^d and mondaylist doesn’t yet exist

Although the file didn’t exist and the command was supposed to add to an existing file, the terminal created the file and added the text to it.

Exercise 6: Find out what happens if you do: cat >> mondaylist rabbit hare ^d and mondaylist does already exist.

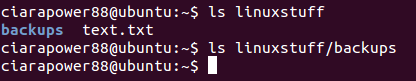
Once the file was created from the last exercise, I simply used the same command and added a duplication of the text into the file.

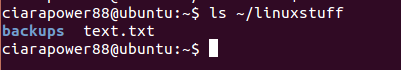
Exercise 7: Using pipes, display all lines of list1 and list2 containing the letter ‘p’ and sort the result.

Exercise 8: Using pipes and filters search all files ending in \*.txt in your linuxstuff directory that contain the names of ALL other users in your class currently logged in. Output the results in ascending order to Standard Output.

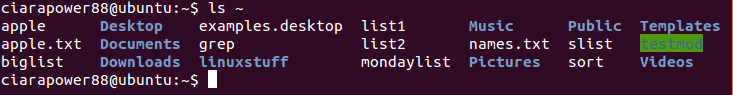
Exercises: Explain what the following do:

1. ls linuxstuff/backups

This command does not show anything, as it cannot list the files in an empty directory.

1. ls ~/linuxstuff

This command just lists the contents of the linuxstuff directory. It starts at the home directory ( use of ~ )

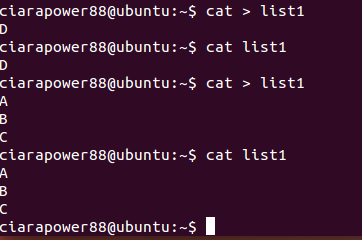
1. ls ~

This command lists the files from the home directory

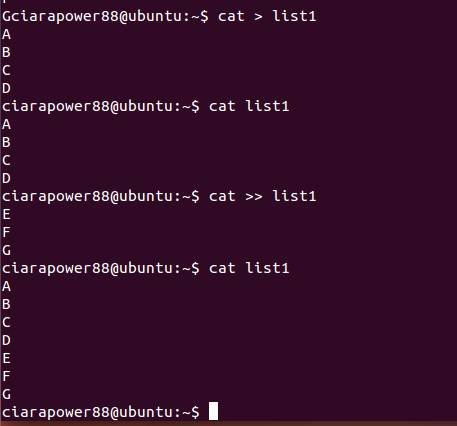
1. ls ~/..

This command lists the what’s in the directory above the home directory, which is the user in this case.

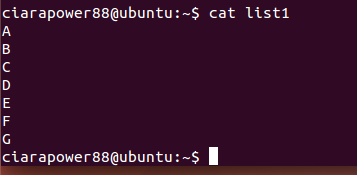
1. cat > list1 … ^D

This command takes standard input and puts it into the file list1. Any previous data in list1 is overwritten. The user can type the input then press ^D to finish.

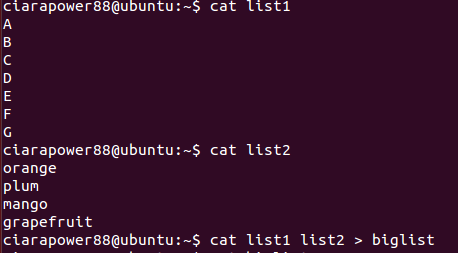
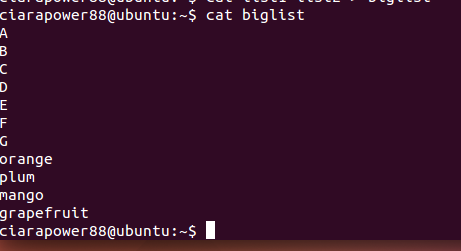
1. cat >> list1 … ^D

This command takes standard input and puts it into the file list1. This input is merged with the existing data in list1. The user can type the input then press ^D to finish.

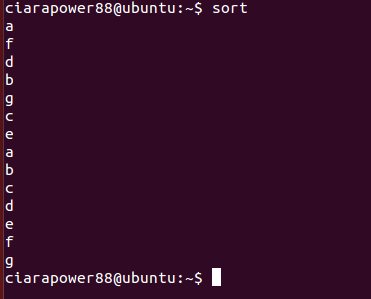
1. cat list1

This command lists the contents in list1.

1. cat list1 list2 > biglist

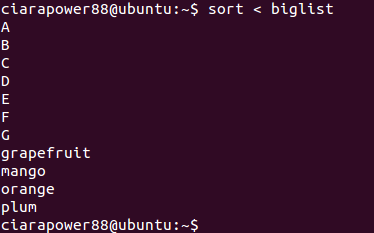
This command merges list 1 and list2 into the file biglist.

1. sort … ^D

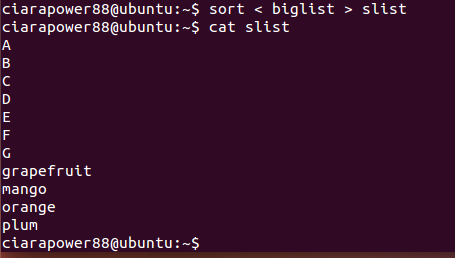
This command will alphabetically sort the input that is typed in after the command. I typed “ a, f,d,b,g,c,e” and the output was “a,b,c,d,e,f,g”

1. sort < biglist

This command means sorted list will be output to the screen.



1. sort < biglist > slist

This command means sorted list will be output to a file called slist.

1. ls list\*

This command lists everything in the directory beginning with the string “list”.

1. ls \*list

This command lists everything in the directory ending with the string “list”.

1. ls ?list

This command lists everything in the directory ending with the string “list”, with one missing letter “?”.

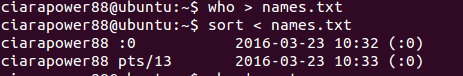
1. Who

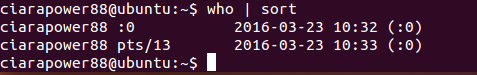
This command is to see who is on the system at the same time you are. In this case, it was only myself.

1. who > names.txt

sort < names.txt

who | sort

The first two commands put the names of who is on the system into a file names.txt, then sends that file to a sort command.

The last command simply sends who is on the system straight to the sort command, using a pipe. There is no need to create a file.

1. who | wc –l

This command is to show how many users are logged on.

1. grep 'orange' list2

This command finds the string “orange” in list2.

1. grep '^grape' list2

This command finds the string “grape” with any suffix in list2.

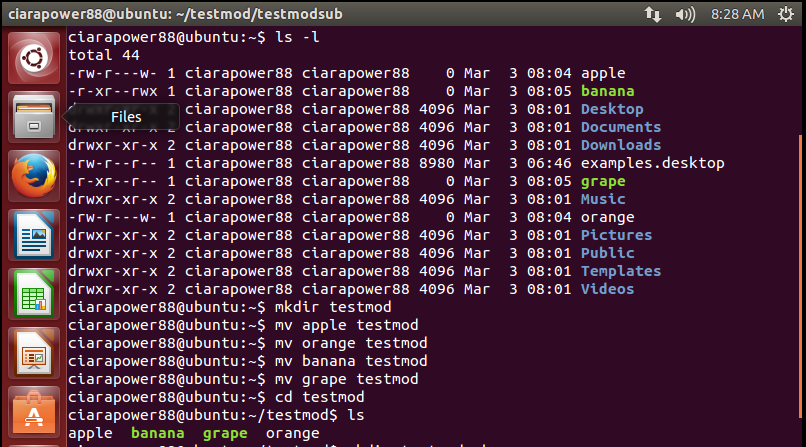
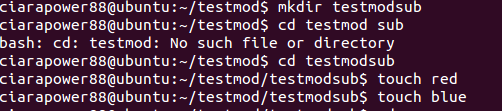
1. grep 'fruit$' list2

This command finds the string “fruit” with any prefix in list2.

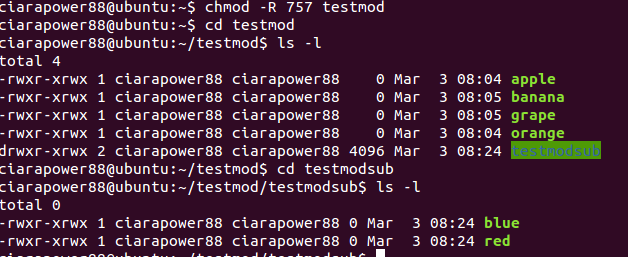
2. Write a note to explain the difference between ~ and / in describing directories.

The ~ symbol represents the path beginning from the very home starting directory. So regardless of where the user is in the directory tree, this can be used to locate a file or directory starting a path from the route directory.

In contrast to this, / is used to symbolize entering a child directory, when the user is in the parent directory. It can be used to create a path to a file or directory.

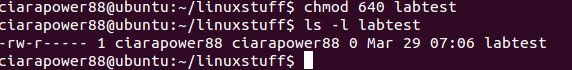
1. Put all the files, apple, orange, banana and grape, into a directory called testmod. In testmod create a directory called testmodsub and using touch create two files called red and blue. You now have a small directory tree.

Now go back to the directory containing testmod and change the permissions of all the files recursively in the directory tree starting at testmod, such that each file will now have the permission 757.

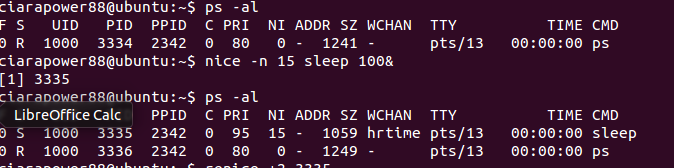


# Lab 08 – Unix/Linux

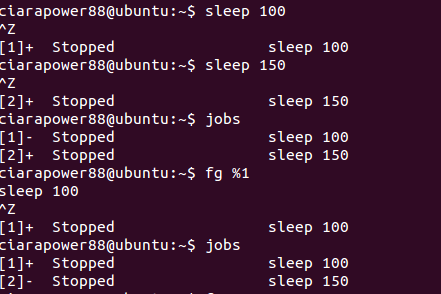
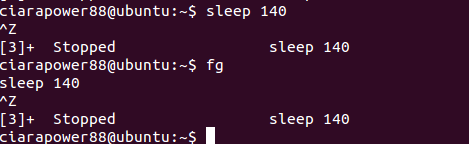
Exercise 1 Now try setting the permission to 640. Write down the 9 character rwx pattern.

The pattern is rw-r----- as shown below.

Exercise 2 Devise some experiments to see if you can test the effect of the nice command.

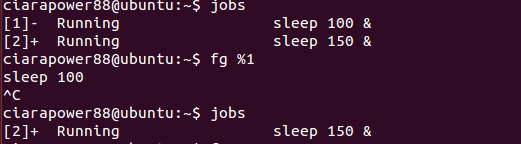
I used the nice command to set a process sleep 100 to run in the background, with a nice value of 15.

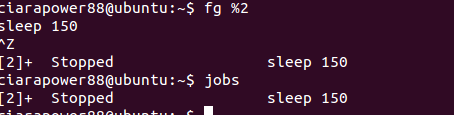
Exercise 3

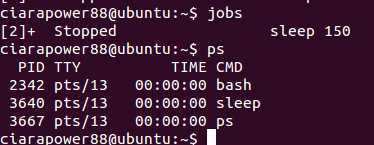
1. What is the difference between typing fg (i) with a job number specified and (ii) without a job number specified?
2. In this example, the number is specified as 1 which foregrounds the sleep 100 process.
3. In this example, the sleep 140 process is created and then suspended. The fg command is used without a number, which foregrounds the last suspended process (the sleep 140 process)

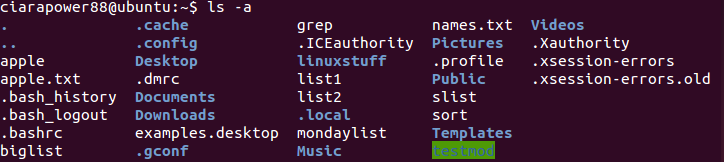
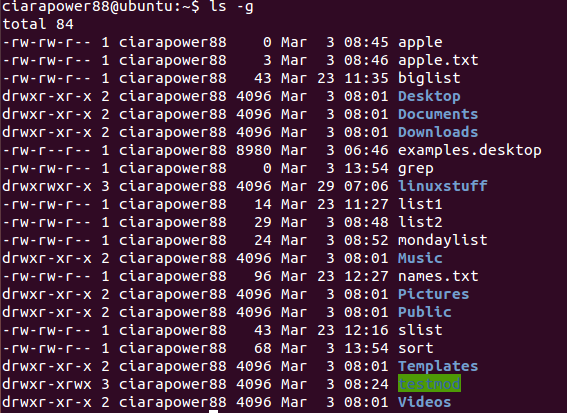
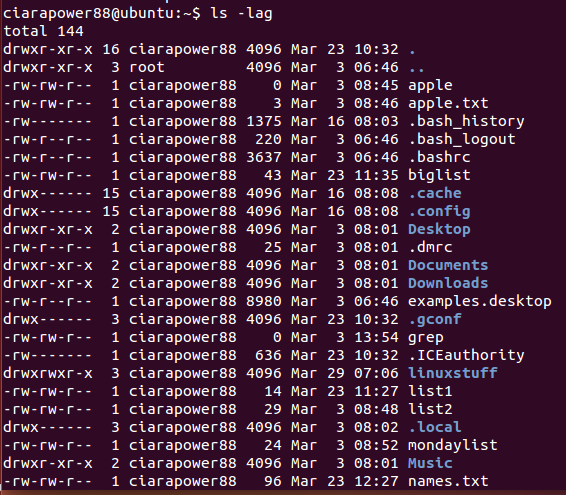
(b) What do you type to (i) cancel a process (ii) suspend a process.

1. To cancel a process, type ^C (CTRL C).



1. To suspend a process, type ^Z (CTRL Z).
2. Explain the difference between the ps command and the jobs command. (Hint: You may wish to consult the Internet to help find answers to this.)
3. jobs is a shell builtin. It tells you about the jobs that the current shell is managing. It can give you information that is internal to the shell, like the job numbers (which you can use in shortcuts like fg %2) and the original command line as it appeared before variable expansions.
4. ps is an external command which can tell you about all the processes running on the system. (By default it only shows a small subset, but there are options to select larger sets of processes to display.) It doesn't know about the shell-internal stuff.

The difference in output is shown below:

1. Write a tutorial style note, for first year students, showing the various switches (i.e. options) that are available for the ls command. (Hint: You may wish to consult the Internet to help find answers to this.)
2. **Ls – a :** This command lists all files in a directory, even hidden files that cannot usually be seen. An example is shown below.
3. **Ls –** **g :** This command lists all files and which group owns the files. An example is shown below.
4. **Ls –lag :** This command lists everything in the directory. An extract from an example is shown below.
5. **Ls \*.txt :** This command lists every file that has a .txt at the end of its name in the directory. An example is shown below.
6. **Ls list\* :** This command lists every file that has “list” at the beginning of its name in the directory. An example is shown below.

# Lab09- Threads

**PART 1**

***Things to try part 1.1:***

**1. Create an Eclipse Java Project to use this program and run the software 10 times, each time copying and pasting your results into a word processor file. Label this set of 10 results clearly with the heading *Threading Program 1.1***

**Do you see much variation on your results from one run to the next? Explain what conclusions you come to.**

**Threading Program 1.1**

**1:**

aaaaaaaaaaaaaaaa 1bbbbbbbbbbbbbbbbbbbbbb 2aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 3 4 5 6 7 8 9bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 10 11aaaaaaaaaaaaaaa 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**2:**

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69bbbbbbbbbbbbb 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**3:**

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62bbbbbbbbbbbbbbb 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**4:**

abbbbbbbbb 1aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabb 2bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 3bbbbbbbbbbbbbbb 4bbbbbbbb 5bbbbbbbbbbbbbb 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44bb 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**5:**

aaaaaaaaaaaaaaaaaabaaaaaaaaaaaabbbababbbbabbbaba 1bbb 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45aa 46bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 47 48 49 50 51 52 53 54 55 56 57 58 59aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100bbbbbbbbbbbaaaaaaaaaaaaaaa

**6:**

abaabaaabaaaabbbbbbbbbbbbbbbbbbbaabbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 1bb 2 3 4 5 6 7 8 9 10aaaaaa 11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 12a 13bbbbbbbbbbbbbbbbbbbbbbbbbbb 14aaaaaa 15bbbbbbbbbbbb 16 17aaa 18aa 19aaaaaaaaaaaaaaa 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**7:**

aaaaaaaaaaaaaabaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbb 1b 2bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 3bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 4b 5b 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**8:**

aaaaaaaaaaaaaa 1abaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 2aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22aa 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**9:**

aaabaaaaaaaaaaabbbbb 1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaa 2aa 3aaaaaaaaaaaaaaaaaa 4a 5aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 6aa 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**10:**

aaaaaaaaaaaaaaaaaaaaaaabbbbbbabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**Conclusion:**

There is a huge variation between the outputs each time I ran this program. There was no definite structure or pattern in the outputs. Output varied between integers and characters. This means the threads were randomly interchanged on the processor.

**2. If possible, repeat the exercise above with computers that are single core, dual**

**core and quad core. Is there a difference in how these machines perform?**

I was unfortunately unable to carry out this exercise on more than one computer, so could not compare single core, dual core, and quad core. However from my existing knowledge I can assume that quad core would be a lot more efficient than the others. There would be no need to swap between threads running on the processor, as each thread could run in parallel on a processor. There would be no randomly patterned output, as each thread can run to completion on its own processor.

**3. Replace the lines:**

**thread1.start();**

**thread2.start();**

**thread3.start();**

**with**

**thread1.run();**

**thread2.run();**

**thread3.run();**

**Does your software still run?**

**If it does, what difference do you see?**

Yes this software still ran.

There is a big difference, the code runs in order, and are not randomly interchanged.

This is because the code is ran as normal processes, as the code is calling methods, rather than creating threads.

***Things to try part 1.2:***

**1. Create a Java Eclipse Project to use this program and run the software 10  
 times. Each time copy and paste your results into a word processor file. Label  
 this set of 10 results clearly with the heading *Threading Program 1.2***

**Do you see much variation in your results from one run to the next? Explain  
 what conclusions you come to.**

**Threading Program 1.2**

**1:**

bbbbbbbbbbbbbbbab 1bab 2bbbbbbbbbbbbccaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaacb 3bcaaaaaaaaaaaaaaaaaaaaaaaaaaaaaacccccccbbbbbbbbbbbbbbb 4bbbbbbbbbbbbbbbbbbbbbbbbcccccccccccccccccccccccccccccbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**2:**

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbb 1bbbbbbbbbbbbbbbbbbbbbbbbbbb 2bbbbbbbbbbbbb 3bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 4b 5b 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50cccccccccccccccccccccccccccccccccccccccc 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**3:**

abbbbbbbbbbaaaaaaabaaaaaaabababaababbababbbababcbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbcbbbbaaaaaabbcbabcbabcbaaaaaacccccccccccccccccccccccccccccccccccbb 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**4:**

baaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbaaaabaaaaaaaaaaaabbbbba 1 2 3 4 5b 6b 7b 8b 9cccccc 10b 11ccccccccccc 12bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 13ccccccccccccccccccccccc 14 15 16 17 18 19 20 21 22b 23bb 24bb 25bbbbbbbbbbbbbbbbbbbbbbbb 26bbbbbb 27bbbbbbbb 28b 29b 30b 31b 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**5:**

bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaa 1babaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab 2ba 3ccaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 4 5 6 7 8 9 10ccccccccccccccc 11bbbbbbbbbbbbbbbbbbbbbbbbbb 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40ccccccccccccccccccccccc 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**6:**

aabababaaaaaaaaaaaaabbbbbbbaabbbbbbbbbbbbbbbbbbbbbbbbbbbcacccbcaacbcaacbcacbcacbcccccaacbcacccbbbbbbbbbbbbbbbbcacbb 1bccccccccccccccccaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbb 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100bbbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaa

**7:**

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 1a 2a 3 4 5 6 7 8 9 10 11 12a 13a 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 37 38 39 40 41 42aaaaaaaaaaaaaaa 43 44 45 46 47 48 49 50bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbcccccccccccccccccccccccccccccccccccccccc 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**8:**

babbbabaababbbbbbbbbbbbabababbbbaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbcbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 1 2 3abcbaaa 4abcbbbbbbaaaaaaa 5abcba 6abcbaaaaaaaaaaaa 7bcbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 8 9 10c 11b 12c 13bbbbbb 14c 15b 16c 17b 18 19 20 21 22 23 24ccccccc 25cccccccccccc 26c 27c 28c 29c 30ccccccc 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**9:**

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 1b 2b 3bbbbbbbbbbbb 4bbbbb 5bb 6bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 7 8 9 10 11 12 13 14 15 16 17bbbbbbbbbbbb 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44bb 45 46 47 48 49 50cccccccccccccccccccccccccccccccccccccccc 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**10:**

babbbaabbbbabbbbaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbb 1a 2b 3a 4b 5a 6b 7a 8b 9a 10b 11aaaaaaaaaa 12bbbbbbbbbbbbbbbbbb 13a 14b 15aaa 16bb 17aaaaaaaaaaaaaaaac 18 19bbbbbbbbbbbbccaccccbbbbbbbbbbbbbbbbbbbbb 20caccccccccc 21cccccccccccccccccccccccaaaaaaaaaaaaa 22a 23a 24a 25a 26a 27a 28a 29a 30a 31a 32a 33a 34a 35a 36a 37a 38aa 39aaaaaaa 40aaaaaaaaaa 41a 42aaaaaa 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**Conclusion:**

The thread that prints the char c, runs alongside the printed numbers thread, until the numbers printed reaches 50. If there are any c chars left to print, they are all printed before the rest of the numbers are printed.

**1. Create an Eclipse Java Project to use this program and run the software   
 10 times, each time copying and pasting your results into a word processor**

**Do you see much variation in your results from one run to the next? Explain  
 what conclusions you come to.**

**Threading Program 1.3**

**1:**

abaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbb 1bbbbbbbbbbbbbbb 2bbb 3bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 4bbbbbbbbbbbbbbb 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**2:**

aaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 1aaaaaaa 2aaaaaaaa 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**3:**

abaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**4:**

abbbbbbbbbbbbba 1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaab 2baaaaabb 3abaabaaaaaaaaaaaaaabbbbba 4ab 5ab 6aabaaaaaaaaaaaaaaaaaaaaa 7bbbbbbbbbbbbbbbbbbbbb 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**5:**

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**6:**

a 1ba 2bbabaa 3aaaaaaaaaaabaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 4ababaaaaaabbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaabaaaaaaababaaaaabbbbbbbbab 5abbbbbbbb 6bbbbbbbbbbbbbbbbbbbbb 7bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**7:**

abbaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbababbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**8:**

aaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbb 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**9:**

aaaaaaaaaaaaaaaaaaaaaaaaaaaabaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbb 1bbbbbbbbbbbbbbbbbbbb 2bbbb 3bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**10:**

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**Conclusion:**

The number thread pauses after each number printed, to give the other threads a chance to complete their actions. When all the other threads are complete eventually, we can see in the above outputs that the number thread just runs until completion.

***Things to try part 1.4:***

**1. Create a Java Eclipse Project to use this program and run the software 10 times,  
 each time copying and pasting your results into a word processor file. Label   
 this set of 10 results clearly with the heading *Threading Program 1.4***

**Do you see much variation in your results from one run to the next? Explain  
 what conclusions you come to.**

**Threading Program 1.4**

**1:**

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**2:**

ababaaaaaaaaaababaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**3:**

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**4:**

bababaaaaaaabbbbbbbbbbbabaababababbbbbbbbbbbbbbbbbbaabbbababababbabababaaaaaaaaaaaaaaaaaaaaaaaabbabbbbbb 1aaaaaaaaaaaaa 2b 3a 4b 5aaa 6b 7a 8b 9a 10b 11 12 13 14 15a 16b 17a 18b 19a 20bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 21aaaaaaaaaaaaaa 22bb 23 24 25 26 27a 28a 29a 30aaaa 31a 32aaaaa 33a 34a 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**5:**

abbbbbbbbbbbbbbbbaabababababaaaaaaaaaaaaaaaaaabbaaaaaa 1abaaaaaa 2aba 3aaaabaaaaaaaaaaaaaaaaaaaaaa 4aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbb 5 6 7b 8bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**6:**

aaaaaaaaaaa 1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 2aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 3 4 5 6 7 8 9 10 11bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 12aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 13aa 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**7:**

aaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbba 1aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbb 2bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**8:**

abbbbbababababbbbbbbbbbbaaaaaababaabbbbbbbbbbbbbbbbbbbbbbabbbbbbbbbb 1 2 3 4 5bab 6bab 7bab 8babbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 9babbbb 10bab 11bab 12 13 14 15 16 17 18a 19a 20a 21aa 22aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 23aaaaaaaaaaa 24aa 25aa 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**9:**

abbbabbbbbbbabbbbbabbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbba 1abaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 2aabaaaaaa 3aba 4aba 5aba 6ab 7 8 9 10 11 12 13bbbbbbbbbbbb 14bbbbbbbbb 15bbbbbbbbbbbbbbbbbbbbbb 16b 17b 18b 19b 20b 21bbbb 22b 23bbbbbbbbb 24b 25b 26b 27b 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**10:**

aaaaaaaa 1b 2a 3 4bbbbbbbbb 5aaaaaaaaaaaaaaaaaaaaaaaaaa 6 7 8 9 10 11 12b 13aaaaaaaaaaaaaaaaaaaaaaa 14bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 15aa 16bbbbbb 17aaaaaaa 18bbbbbbbbbbbbbbbbbb 19 20 21aaaaaaaaaaaaaaaaaa 22bbbbbbbbbbbbbbbbbbbbbbb 23 24aaaaaaaaaaaaa 25aa 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**Conclusion:**

The thread for printing numbers is set to sleep if it is printing a number greater than or equal to 50. In my tests of this program, the other threads appear to have finished before the number 50 had been reached. If this was not the case, the other threads would have had a one millisecond chance to complete between each printing of every number after 50.

**2. Repeat part one above several times, each time with a different sleep time.  
 Write out your conclusions.**

**a) Thread.Sleep(10)**

aaaaaaaaa 1bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 2aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 3bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 4a 5b 6a 7bbbb 8aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 9aaaaaaaaaaaaa 10aa 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**b) Thread.Sleep(50)**

baaaaaaaaaaaaaaaaaaaaaaaaaaababaaaaaaabaabbaabababaabaabaaaaaaaaaaaaabaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 1 2 3 4b 5bbbbbbbb 6b 7b 8b 9b 10bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 11bbbbbbbbbbbbbbbbbbbbbbbbbbbb 12b 13b 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**Conclusion:**

In my examples, using a sleep of 10, and a sleep of 50, there doesn’t seem to be much difference, as the print char threads are all complete before the number thread even reaches 50.

***Things to try part 1.5:***

1. **Consider now the program for part 1.1 above and, in particular, the following**

**lines:**

// Third code block: Start threads

thread1.start();

thread2.start();

thread3.start();

**Now change these lines to read:**

// Third code block: Start threads

thread1.start();

thread2.start();

thread3.setPriority(Thread.MAX\_PRIORITY);

thread3.start();

**Do the first of the *things to do* of part 1.1 (see page 3) again but using the modified program this time.**

**Compare your results with those that you got the first time.**

**Threading Program 1.5**

**1:**

1a 2a 3a 4a 5a 6a 7a 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22a 23a 24a 25a 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabaaaaaaaaaaaaaaaaaabaaaaabaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

**2:**

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabaaaaaaaaaaaabababbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

**3:**

a 1aaaaaaaabbbbb 2babbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 3bab 4babbbbbbbbbbbbbbbb 5babbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 6 7 8 9 10 11aa 12a 13a 14a 15a 16a 17aaaa 18a 19a 20a 21a 22aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 23aaaaaaaaaaaaaa 24 25 26 27 28a 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**4:**

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100ababaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbabbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaabababababbbbbaaabbbbbbbbbbbbbbaaabbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

**5:**

aaaa 1bbbbbbbb 2aaaaa 3 4 5 6 7 8 9 10 11 12b 13a 14b 15aa 16b 17a 18b 19aaaaaaaaaaaaa 20b 21aaa 22bbbb 23a 24b 25a 26b 27a 28b 29a 30b 31a 32b 33aaa 34b 35a 36b 37a 38b 39a 40b 41a 42b 43a 44b 45a 46bbbbbbbb 47a 48b 49a 50 51b 52a 53b 54a 55b 56a 57b 58a 59b 60a 61b 62a 63b 64a 65b 66a 67b 68a 69b 70aaaaaaaaaaaaaaaaaaaaaaaaa 71b 72aaaaaa 73bbb 74a 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93b 94a 95b 96a 97b 98a 99b 100aaaaaaaaaababbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb

**6:**

a 1 2 3 4 5 6bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbba 7aaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaa 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80bbbbbbbbbbbbbbbbbbbbbbb 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

**7:**

a 1a 2aaaaaaaaaaaaaaaaaaabbbbbbbbbbb 3bab 4bab 5bab 6bab 7bbbbbbab 8babbbbbbbbbbb 9bbbbabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 10bbab 11bab 12babbbbbbbb 13babbbb 14 15 16 17 18 19 20 21 22 23 24 25 26 27aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 28a 29a 30a 31a 32a 33a 34a 35a 36a 37aaaaaaaaa 38a 39a 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**8:**

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100aaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

**9:**

a 1aabbbbbbbbbbb 2a 3aaaaabbbbbbbbbbbbbbbbbbbbbbbbbbb 4b 5a 6bb 7a 8bbbbbbbbbbbbbbbbb 9a 10b 11a 12b 13a 14b 15a 16bbbbbbbbbbbbbbbbb 17aaa 18b 19a 20b 21aa 22b 23a 24b 25a 26b 27a 28b 29aaaaaaaaaaaaaaaaaa 30bbbbbbbbbbbbbbbb 31a 32aaaaaaaaaaaaaaaaaaaaaaaaaa 33a 34a 35a 36a 37a 38a 39a 40a 41a 42a 43a 44a 45a 46a 47a 48a 49a 50a 51a 52a 53a 54a 55a 56a 57aaaa 58a 59a 60a 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

**10:**

bbbbba 1bbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabb 2bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaaaab 3 4 5 6 7aaa 8 9bbbbbbbbbbbbbbbbbbbbbbbbbbbbbb 10 11 12 13 14 15aaaaaaaaaaaaaa 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100aaaaaaaa

**Conclusion:**

The above examples of this program show that the print number thread has max priority over the other threads. In most cases, the numbers 1-100 are nearly consecutively printed out to the console. The other threads printing the characters then complete, as they have only the default normal priority.

In the first part (1.1) of this lab, the character threads seem to have been mostly run before the print number thread. The max priority setting really differs the output of this program.

**PART 2**

***Things to try part 2:***

**When you’ve run the program above write a paragraph that explains what you see in terms of what you’ve learned in lectures about**

1. **buffers,**
2. **multiprocessing/multithreading**
3. **the producer-consumer problem.**

This program seems to first print out alternating strings, depending on the use of the buffer each time round. There is clear use of buffers in this program. The buffer size is set to 1, which is the max capacity of the buffer. A value is written to the buffer, and the next thread begins to execute, and in doing so it reads from the buffer. This determines the index number that is printed alongside each string, and also it shows the wait between each thread.

The Consumer thread must wait until the Producer thread completes before it runs, and vice versa. There are two conditions used to determine when the processor must wait until the next thread is ran. The use of the sleep function is used also. This demonstrates that multithreading is not in play here, as each thread runs separately and individually on the processor.

The producer-consumer problem seems to be solved here, as there are no parallel running threads, so in this analogy there is only the one consumer with the producer at any given time. The wait function and the conditions are used to determine when the next consumer/producer thread can run, as there should only be one at a time.

**PART 3**

***Things to try part 3:***

1. **Run the above program, experimenting with different sleep times, than 100ms. Try, for example, 1ms, 20ms, 50ms, 200 ms, 300ms.**
2. 1ms

When the sleep time was set to 1ms, the counter was too fast to even read the numbers printed in the dialog box. The stop button could not stop the counter as it was incrementing too fast.

1. 20ms

The counter slowed down, but still moved too fast to read the values, however the stop button was able to stop the counter at a value.

1. 50ms

The counter incremented slower again, but the values were more readable than previously. The stop button worked perfectly to stop the counter.

1. 200ms

The counter incremented quite slow in this test, so each value could be read easily, and the counter could be stopped on a chosen value with ease.

1. 300ms

The counter moved very slowly this time, as the sleep time is quite large.

1. **Now that you’ve run the program, can you see clearly (not from the code but simply from what you observe when running it) why this program, simple though it is, cannot be implemented with only one thread. Explain.**

This program needs more than one thread because as the counter is incrementing in one thread, there must also be another thread to hold the window and deal with the start and stop buttons. The go condition which is set in the window thread then determines if the counter thread should proceed or not.

1. **Now eliminate both the *try* block and the *catch* block so that the inside of the *while* block just looks like:**

counterWindow.setText("" + count);

count++;

**if**(count > 10)

{

count=1;

}

**After modification, run the program again and explain what you see.**

There is no sleep time for the counter anymore, so the counter increments very quickly. The program runs as before other than this.

**PART 4**

***Things to try part 4***

**Consider the following software:**

**RunnableJob.java**

package com.cakes;

public class RunnableJob implements Runnable {

@Override

public void run() {

Thread thread = Thread.currentThread();

System.out.println("RunnableJob is being run by " +

thread.getName() + " (" + thread.getId() + ")");

}

}

**ThreadExample.java**

package com.cakes;

public class ThreadExample {

public static void main(String[] args)

throws InterruptedException

{

RunnableJob runnableJob = new RunnableJob();

Thread thread1 = new Thread(runnableJob);

thread1.setName("thread1");

thread1.start();

Thread thread2 = new Thread(runnableJob, "thread2");

thread2.start();

Thread thread3 = new Thread(runnableJob);

thread3.start();

Thread currentThread = Thread.currentThread();

System.out.println("Main thread: " +

currentThread.getName() + "(" +

currentThread.getId() + ")");

}

}

**Edit the software above and instead of using *getName()* and *getID()* use:**

**thread = Thread.currentThread();  
System.out.println(thread);**

**The last line will make thread use the Thread class's toString method**

**Write down what you see.**

Before the change:

RunnableJob is being run by thread1 (10)

Main thread: main(1)

RunnableJob is being run by thread2 (11)

RunnableJob is being run by Thread-1 (12)

After change:

Main thread: main(1)

Thread[thread1,5,main]

Thread[thread2,5,main]

Thread[Thread-1,5,main]

The thread names are the same and in the same order as before the changes were made. However the ID has changed.

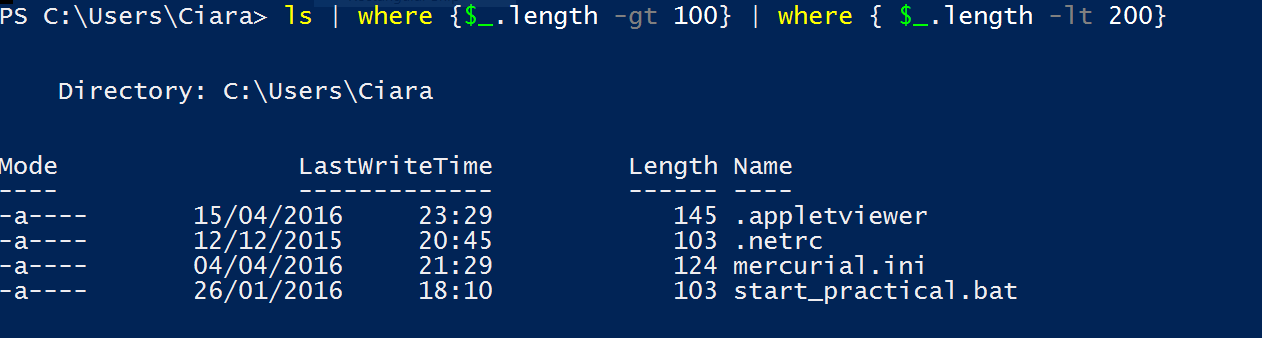
# Lab10 – PowerShell

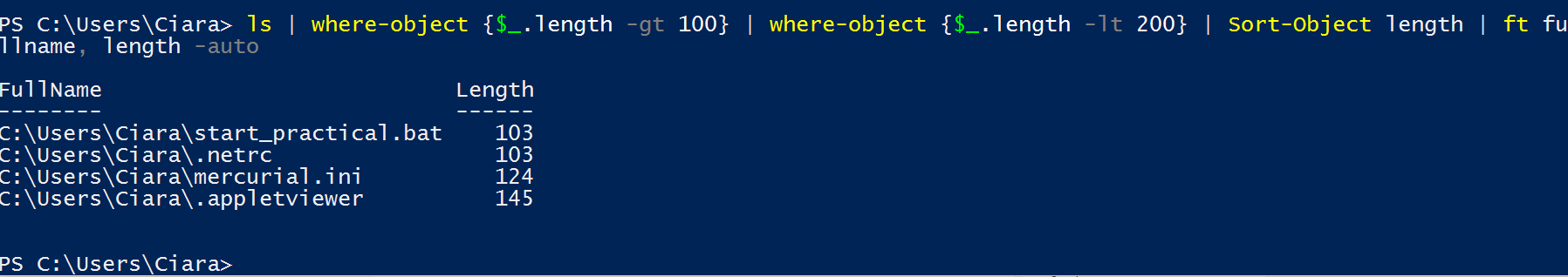
**Things to do:**

**1. In two distinctly different ways edit the line of code above,**

**i.e. dir | where-object {$\_.Length –gt 100}**

**so as to give a directory listing of those files whose length is greater than 100 bytes but less than 200 bytes.**

**a)**

**b)**

**2. Is dir the native command in PowerShell for getting a directory listing. If not what is the native command?**

No, the native command for getting a directory listing in Powershell is Get-ChildItem.

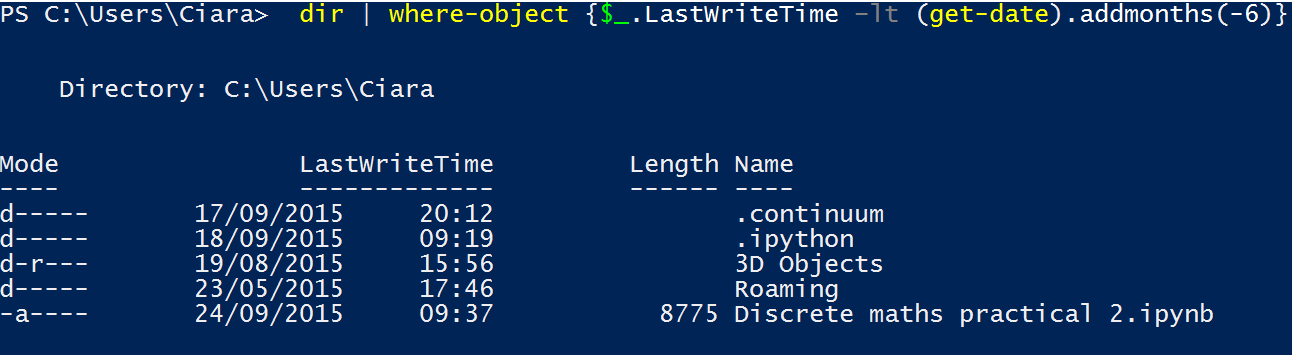
**3. Write out all the ways of getting directory listings in PowerShell, explaining by whom or in what context the different ways are likely to be used. (You may have to read over previous practicals to answer this.)**

**Get-ChildItem >>>>>Dir >>>>>Ls >>>>>Gci**

The Get-ChildItem is the actual command in PowerShell.

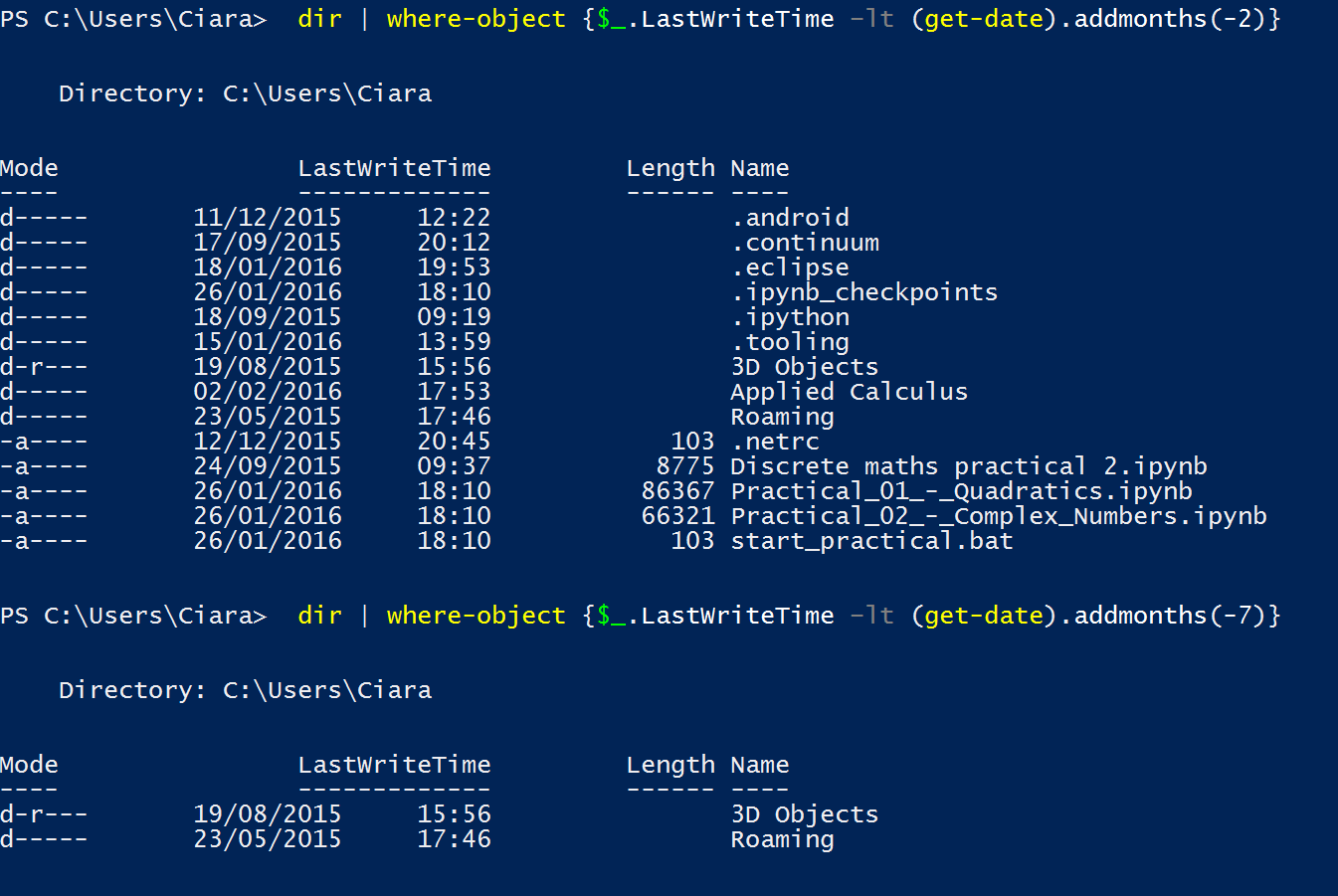
The other ways are simply alias’ for this command. They all perform the same function and list the working directory. Some of these are used in other terminals also, with dir being used in DOS, and ls being used in Linux.

**4. Find out, by using different values of the numerical parameter, what the following useful line of PowerShell does:**

 **dir | where-object {$\_.LastWriteTime –lt (get-date).addmonths(-6)}**

I tested with varying values:

dir | where-object {$\_.LastWriteTime –lt (get-date).addmonths(-2)}

dir | where-object {$\_.LastWriteTime –lt (get-date).addmonths(-7)}

This line of PowerShell code is very useful. It goes through the current working directory, and finds files that have a last write time less than a specified date. In the cases above, we find files that were changed earlier than 6/2/7 months before the current date.

**5. There is another way of single stepping through a script, using F10. Find out and explain what F10 does, comparing it with F11.**

**F11 (STEP INTO):**

Executes the current statement and then stops at the next statement. If the current statement is a function or script call, then the debugger steps into that function or script, otherwise it stops at the next statement.

**F10 (STEP OVER):**

Executes the current statement and then stops at the next statement. If the current statement is a function or script call then the debugger executes the whole function or script, and it stops at the next statement after the function call.