BASH

; are ONLY needed if you write multiple lines of code in same line

LOOPS
```
for (( initialization; condition; increment ))
do
   commands
done
```
 OR
```
for item in list
do
   commands
done
```

eg
```
for i in {1..5}
do
   echo "Number $i"
done
```

```
while condition
do
   commands
done
```

```
until condition
do
   commands
done
```

IF ELSE
```
if [ condition ]
then
   commands
else
   other_commands
Fi
```

READING FILES

```
if [ -f "$file" ]; then
```

while read line; do

#Reading each line

echo "Line No. $i : $line"

i=$((i+1))

done < $file

else

echo "File not found: $file"

Fi

—--

Multiple variables?

If data.txt looks like

John 25 India

Jane 30 USA

Kavya 22 Japan

THEN

while read -r name age country

do

    echo "Name: $name, Age: $age, Country: $country"

done < data.txt

Order matters: First word goes to `name`, second to `age`, third to `country`.

If extra words are there in the line, the last variable **absorbs the rest** unless you control it.
Set field splitter using IFS
Eg `while IFS=, read -r rollno quiz1 quiz2 midsem endsem total; do`

—-----

MISC
#${variable:offset} → Extracts a substring from variable, starting at offset

OPERATORS

Operator	Meaning	Example	Result
`=`	**Equal to**	`[ "$a" = "$b" ]`	True if a and b are the same
`!=`	**Not equal to**	`[ "$a" != "$b" ]`	True if a and b are different
`-z`	**Zero length** (empty string)	`[ -z "$a" ]`	True if string a is empty
`-n`	**Non-zero length** (non-empty)	`[ -n "$a" ]`	True if string a is *not* empty

Operator	Meaning	Example	Result
`-e`	**Exists** (file or directory)	`[ -e file.txt ]`	True if `file.txt` exists
`-f`	**Regular file** (not a dir)	`[ -f file.txt ]`	True if `file.txt` is a normal file
`-d`	**Directory**	`[ -d mydir ]`	True if `mydir` is a directory
`-r`	**Readable**	`[ -r file.txt ]`	True if you can read `file.txt`
`-w`	**Writable**	`[ -w file.txt ]`	True if you can write to `file.txt`
`-x`	**Executable**	`[ -x script.sh ]`	True if `script.sh` can be executed

Ignore header - `tail +2 gradessorted.csv`

REGEX

```
regex="^[A-Za-z_][A-Za-z0-9_]*\.cpp$"
filename="main.cpp"

if [[ "$filename" =~ $regex ]]; then
   echo "Valid .cpp filename"
else
   echo "Invalid"
fi
```

Most IMP commands of linux

SORT

sort -k2 file.txt
→ Sort based on the 2nd column.
`-s` means **stable sort**:
If two entries are *equal* based on the sort key, their **original order is preserved**.

-t',' sets delimiter as comma

`-k2.4` means:
Start sorting **from field 2, character 4** inside that field.

-k2,3 takes both filed 2 and 3 as one thing for sorting
You can **chain** sorts:
sort -t',' -k2,2 -k3n,3
First sort by field 2 alphabetically

If tie, sort by field 3 numerically (n)

-n is numeric -r is reverse

cut
cut -d',' -f2,4 data.csv
Outputs filed 2 to 4

Set outputdelimitter is --output-delimiter=' | '

-f1,3,5 gets 1 3 and 5

grep
Options

Option	What it does
`-i`	Ignore case (match Hello, HELLO, etc.)
`-v`	Invert match (print NON-matching lines)
`-n`	Show line numbers
`-c`	Count number of matching lines
`-o`	Only print the matching parts (not the full line)
`-w`	Match whole word only
`-r`	Recursively search in directories
`-l`	List file names with matches (not lines)
`-e`	Specify multiple patterns
`-E`	Use **extended regex** (allows `+`, `?`, `
`-f file`	Take patterns from a file instead of writing them in command

grep "hello" file.txt
Prints all lines in file.txt containing hello.

Regex Symbol	Meaning
`.`	Any one character
`*`	0 or more of previous character
`^`	Beginning of line
`$`	End of line
`[abc]`	Match any one of a, b, or c
`[^abc]`	Match anything except a, b, or c

`a|b`	Match either a **or** b (use `-E`)
`(pattern)`	Group things (use `-E`)