Python (files)

file = open("filename") → would open the file,

print (file.readline()) → would read the first line of the opened file

print (file.read()) → would read the rest of the unread file in the terminal.

file.close() → closes the opened file.

with open("filename") as file → This opens the files and closes the file after the operation automatically.

variable.upper() → will change the variable to uppercase (one character)

variable.strip() → would ignore the new line character while operation

variable.sort() → would sort the data in the variable alphabetically.

open modes → r = read only, w = write only, a = append, r+ = read-write

If we open a file for writing and the file already exists, the old content will be deleted as soon as the file is opened

variable.write("content") → would write the content in the variable.

The modes of open are used as open("filename", "mode")

import os → This command imports the operating system module of python that has commands like remove, rename, path module etc

os.remove("filename") → would delete the particular file.

os.rename("first", "newname") → This would change the name first to new name of the file

os.path.exists("filename") → returns true value if file exists.

os.path.getsize("filename") → would return the size of the file

os.path.getmtime("filename") → would return a number timestamp that represents the number of seconds since January 1st 1970 representing when the file was used last

import datetime → would import datetime module of python. OS module class.

datetime.datetime.fromtimestamp(variable) → command to get time. class.

os.path.isfile("filename") → checks whether file exists.

os.path.abspath("filename") → returns the absolute path of the file.

os.getcwd() → would return the directory path of the existing

os.mkdir("directoryname") → would create a new directory.

os.chdir("directoryname") → would move to the directory with name in

os.rmdir("directoryname") → would delete the directory with name inside
This deletes only empty directory.

os.listdir("directory") → lists all files inside the directory.

os.path.isdir("directory") → checks whether the file is directory and
prints only it.

os.path.join(dir, name) → joins the filename and directory name into a
string.

---

csv file → Comma-separated values. → text file format

variable_f = csv.reader(f) → will read from the csv file.

print("x: {}".format(variable)) → will display the formatted variable

writerow → Writes one row at one time

writerows → writes multiple rows at one time seperated using comma

↳ The data must be packed in square brackets with row in the same

DictReader → This converts each row of the csv into dictionary.
and associates it with the

---

grep word-needed /usr/share/dict/words → would return all the words that
contain the word-needed (case sensitive)

grep -i word-needed /usr/share/dict/words → would return all the words that
contain the word-needed (case insensitive.

grep f.rts /usr/share/dict/words → would return all the words that
contain the word with . replacing anything (any letter).

^ → circumflex

grep ^word-needed /usr/share/dict/words → would return the words that
contain the word needed from the beginning of the word only.

grep word-needed$ /usr/share/dict/words → would return the words that
contain the wordneed from the end of the word only

---

variable = re.search(r" ", " ") indicates that the first string is a raw string.
The above line would return the result as <re.Match object; span=(,), match=' '> which means that the word given first is present in the second from the given span and the given word
If it doesnt match it returns : None.
If the string is present more than one time, it returns only first.

print(re.search(r"[ ]___", "___")) would return the matched object containing either one among the square bracket. even a-z can be used.

print(re.search(r"{first|second", " ")) would search for either of first & second. and returns valid output.

print(re.findall(r"___", " ")) returns all the matches unlike search.

re → RegEx library.

print(re.search(r"__.*n", " ")) would increase the span till n is found.

print(re.search(r"p?each", " ")) would search for peach in the string but would return positive even in case of each because of ?

To use the actual special characters in search we use \ before them.
Ex $com, .com. etc.

\w → searches for any alphanumeric characters (letter, upper-lower case)
\d → digits. \b → boundaries. www.regex101.com
\s → whitespaces

variable.groups() → returns the groups enclosed in ~~flowers~~ round brackets.
variable[0] → would contain the whole string
variable[1] → would contain the first group followed by others for 1, 2, 3, ...

\b inside the double quotes are used to indicate the beginning and end of the string.
print(re.Search(r"__{2}", " ")) would return the __ 2 times or finds for the result repeated 2 times in the second variable given

re.split(r" ", " ") → splits the second string w.r.t first.

re.sub(r" ", " ") → used to create new string by sub
if all or part of them in the string.

The input function always returns a string. We have to convert these
if we want → variable = input(" ");

Standard Input → STDIN
output → STDOUT
errors → STDERR
I/O streams are ways to take input, give output and display errors to the screen

Shell → A command line interface used to interact with your OS
Shells → example: bash, zsh, fish etc

Python programs get executed inside a shell command-line environment.
The variables sent in that environment are another source of information
that we can use in our scripts.

env → displays all the environmental variables
echo → prints texts and Linux shell and when we want to access th
value of the variable in the shell, we need a prefix and name of the
variable with a $.

os.environ.get(",") → checks for first string and returns empty
if it is not present.
export x= " " → would create a new env variable

command line arguments → parameters that are passed to a program
when its started
Exit status → value of the program to the shell.
wc → counts numbes of words, letters and characters in the python script.
echo $? → returns the exit status of the process.
When a python script runs successfully it returns 0.

export → command creates a new environment variable.
Subprocess.run(["date"]) → returns the object of the completed Process type
returncode=0.
Subprocess.run(["sleep", "x"]) → would block the interpreter for x time.
returncode=0

result = subprocess.run(["ls", "this_file_does_not_exist"])

(result.returncode) → will return a non zero value.

host command can convert the host name to ip address.

result = subprocess.run(["host", "8.8.8.8"], capture_output = True) → will store different values in the particular file holders.

UTF-8 coding is the default

(result.stdout) → would return the type in the screen.

result.stdout.decode().split() → would return the type in the array type.

(result.stderr) → would return the error of particular process

The failed process would return null when stdout is printed.

---

os.environ.copy() → would return the env variable which has to be stored in a variable

Path variable is a variable that looks where to execute the program.

cwd → change current working directory.

timeout → will kill the run function if the process is taking time more than of allocated.

shell → if set true will first run the shell program and then runs the program present inside it.

File path → The specific location of a file on a computer or web server.

Two types of file paths → relative and absolute

Relative → read and write file with file name alone

absolute → spell out the exact location of the file-drive name, then directory, and then file name.

File paths can also be used to access the env variables.

File paths are used to save and load information.

Unit tests are used to verify use isolated parts of a programs are correct

Unittest module can be imported.

Unittest also provides Testcase class

assertEqual → returns true if both the parameters given are true.

Edgecases → Invrts to our code that produce unexpected results and are found at the extreme ends of the ranges of input we imagine our programs

will typically work with.

White-box testing (clear box or transparent testing) Relies on the test creator's knowledge of the software being tested to construct the test cases.

Black-box tests are written with an awareness of what the program is supposed to do - its requirements or specifications - but not how it does it

loadtest, smoke test, Regression test, Integration test

testdriven development approach.

Bash

echo - print message to screen

cat - Show contents of files.

ls - list contents of a directory

chmod - change permissions of a file.

mkdir - make a directory

cd - change directory

pwd - print the current working directory

cp - copy directory/file.

.. → parent directory.

. → current directory.

touch → create an empty file

-l → provides extra info → permissions - number of inodes - owner - group - size - mod - name

-la → provides even the hidden files.

mv → move file/directory.

rm → remove

* → placeholder to ~~pt~~ ~~regex~~ all files.

rmdir → delete empty directory.

man → documentation print.

redirection → The process of sending a stream to a different destination.

> → transfered content or redirected the output

>> → two times redirection the output (append)

< → redirected the input.

~~Fil~~ ~~piping~~ Pipes → Connect the output of one program to the input of another in order to pass data between programs.