Python

* Why IT : python became powerful, tools, ...
* Variables - no keywords, spaces, can start with (_) / letter
  can have _ , letter, number.

None : special datatype (empty)

Strings:

1. slicing
2. Pets.index('t')
3. "Dragon" in pets
4. upper(), lower(), strip(), split().

Format string

print ("Hello {}, customer".format(name))

print ("Hello {name}, your number {num}".format(name=name, num=number))

1) {:.2f}
   ↳ Float with 2 decimal places

2) format function:

print ("{:>3} F | {:>6.2f} C".format(x, tempFunc(x)))

- {:>3} → align to right (3 spaces)
- {:>6.2f} → 6 spaces, 2 digits, align to right

3 styles (ways)

* "this is {}".format(name)
* "this {0} {2} {1}".format("is", "hello", "me")
  ↳ this is me hello
* "{var1} {var2}".format(var1=value1, var2=value2)
* "{:exp1} {:exp2}".format(val1, val2)

* '{:d}'.format(10.5) ⟶ 10

{:d} → int

{:.2f} → float & decimal

{:.2s} → (Python) → 'Py'

{:<6s} → (Python) → Python

Py → 'Py....'

↳ Align to left

{:>6s} → '....Py'

{:^6s} → '..Py..'

this way → formatted string – not common (3.6 introduced)

→ print (f'Hello {name}')

↳ formatted string

---

list:

lst1.append ('hello')

lst1.insert (0, 'First')

lst1.insert (25, 'End') → Not error (added at end)

lst1.remove ('End') → (1st occurence alone)

'element not in list: error'

lst1.pop (0) → 'First'

lst1 [1] = 'mod'

['mod']

Tuple: (immutable) : position have meaning

1) Function – return more than 1 value (use tuple)

Iterate

for num in lst1: | for index, num in enumerate (lst1):

index → value → Access elements along with index

[('mail1', 'name1'), ('mail2', 'name2')....]

for mail, name in address:

List Comprehension: (multiples of 7)

```
>> multiples = []
>> for x in range (1, 11):
       multiples.append(x * 7)
```

(or)

```
multiples = [x * 7 for x in range (1, 11)]
```

```
lst1 = ['Python', 'Pearl', 'Ruby', 'Go', 'Java', 'C']
print [len(x) for x in lst1]
[x for x in range (0, 101) if x % 3 == 0] ⟶ divisible by 3.
```

---

dictionaries: (Key, value)

```
filename = {'jpg': 10, ... }
filename ["jpg"] ⟶ 10
"html" in filename ⟶ False.
filename ['c'] = 12 ⟶ (added)
```

Already exist: updated!

del filename ["c"] ⟶ delete key & value

for key, value in filename.items(): | .keys() | .values()

⟶ Search fast. (not like list)

say: how many times each error occurs.

Immutable: Number, Booleans, Strings, Tuples : Keys of dictionary!

Set:

Store bunch of elements only once!

OOP – Everything is class in python (String class: eg)

powerful, flexible paradigm (class & objects)

*

dir("") ⟶ All attributes & methods of string

↓

__add__ ⟶ Special methods (inside (internal) Python)

* __len__ (called by len())
* __ge__ (greater than > operator uses)

help

* help(variable/value)
* help("str") → doc of that class

define class

* class Apple :
  pass → empty body (placeholder: empty block)

* class Apple :
  color = ""
  flavour = ""

* Jonagold = Apple()
  Jonagold.color = "red"
  Jonagold.flavor = "sweet"

[dot notation]

Instance methods

* class piglet :
  def speak (self):
  print ("oink")

→ Instance

* obj1 = piglet()
  obj1.speak()

With attribute

class piglet :
  name = "pig"
  def speak (Self):
  Print ("I'm {} ! oink".format
  (Self.name))

piglet().speak()

diff value for different instance.

Constructors

class Apple :
  def __init__ (Self, color, flavor):
  Self.color = color
  Self.flavor = flavor

Jonagold = Apple("red", "sweet")

Python uses default method : gives obj id

def __str__ (Self):
  return "Apple. {} {}".format
  (color, flavor)

↓

gives default display message!

doc Functions, classes, methods

help(Apple)

docstring: what something does

eg: def squa (hours, minutes, seconds):
"""
"""
→ docstring!
(Shows in help doc)

Note: write for methods & classes!

Jupyter notebooks:

Inheritance

>>> class Fruit:
def __init__(self, color, flavor):

>>> class Apple(Fruit): → Apple inherits from Fruit class
pass
Apple ("green", "tart")
(Acquires constructors, methods & Attributes)

>>> class Grape(Fruit):
pass

Composition (Inheritance creates ancestory)

* is-a : check ancestory
* repo has packages (attribute - not inheritance!)

eg:

```
class Repository:
    def __init__(self):
        self.packages = {}
    def add_package(self, package):
        self.packages[package.name] = package
    def total_size(self):
        result = 0
        for package in self.packages.values():
            result += package.size
        return result
```

Composition: allows using other classes as attributes & inside methods

Python modules

* organize func, class & other data together in a structured way.
* import random ⟶ random.randint(1,10)
* import datetime. datetime.now() ⟶ print(now)
* print (no + datetime.timedelta (days = 28))

PyPI repo : modules

Avoid reinventing the wheel

Task: daily report: tracks machine (which users connected which machine)

* dB: events info (login, logout)
* Input? o/p ?

1. Event : instance of event class
2. Event : — date
— ~~name~~
— user
— type
— machine

String

Login
Logout

o/p:

machine:
user:
date:
type:
⋮

(or)

machine name :
— user1
— user2

(or)

machine name :
user1, user2...

Approach:

Tools : check when Logged in, Logged out : clock on parameter

must Logged in to logged out

"chronological order"

* Sort() ⟶ modifies list/... doesn't !
* Sorted() ⟶ returns new list, won't modify!

Key (attributes) to Sort()

Print (Sorted (names, key = len))

↳ based on len.

Plan:

Login → add user

Logout → remove user

* dictionary, set
  inside!

Word Count:

1. punctuation marks (nil)
2. crop up (relevant to input - exclude uninteresting words)

   I/p : novel, ... (anything) : textfile.

goal:

dict with words & frequencies : → pass to generate_from_frequencies function of the wordcloud class

```
cloud = Wordcloud.Wordcloud()
cloud.generate_from_frequencies(frequencies)
cloud.tofile("myfile.jpg")
```

→ use this code after preparing dictionary.

Note: 1. Remove punctuation marks (line by line, char by char : isalpha())

2. split() [line in to words]

3. uninteresting set of words

Python - Interact with OS (Automation)

* OS : Software manages everything that goes on in a computer (memory, packets....)
* OS → Kernel (core - talks hardware, allocate resources)
  → User Space (sysprogram, UI - outside kernel)

Manage file & processes (goal):

* Windows (microsoft)
* MAC OS (Consumer space : Apple)
* Linux (open source) - heavily business space. → Ubuntu, → Redhat, → Android (Linux kernel)
  (Kernel name of Linus Torvis)
* Chrome OS (own way)

* Unix (Bell labs)

Linux uses Unix principles (tools of unix)

* MAC Kernel - Based on unix (BSD) family

Python - cross platform language! (same code!)

Python: » python --version

» python3 --version

* Generate pdfs, create compressed file, email, ...

pip : PyPI (repo for python modules)

Interpreter: dev cycle faster. Slower than Compiled ones.

» 'hello' * 10

magic line (tells OS, we want run using python3)

#! /usr/bin/env python3

run script directly: (make it executable : chmod : change permissions)

» chmod +x hello_world.py

» ./hello_world.py

Same for linux / mac os

why ./ :

* since file is not located in any directories located in path variable
* ./ (current directory) then execute.

own python module

* group code in logical block (small)
* module (large code)

eg: Script email ticket summary → list of tickets closed
→ send e-mail

parse sys log → event ? there
→ Trigger email.

* duplicating code (change twice)

↓ Alternate

module ! (import code)

areas.py — tri
— rect
— square

import areas
areas.triangle(3,5)

large module: many files (__init__.py)

__init__.py (Read by python when imported: check if directory with python files should be a module? (Recognize): leave it empty (but must)

why automate?

* Scale (keep up phase & demand)
* onboard (each time manual: time ↑) – hire, time waste (same work)

Improve process → (name, job desc): automate (when fail! – IT specalist takes)
read from HR system.
→ Fast & reliable (free HR to focus on other works)
centralize error! (Fix at one place: Code)

Pitfalls:

* careless design: serious problem
* Trade off: (worth automate? : how many times perform?)

time of automate < times * time to perform

Pareto principle:

20% of sys admin task: responsible for 80% of work.
Automate! save whole lot of time

* If change in flow (needs update)
* /dev/sda1 (new disc added): update program!

Bit rod: Software falling out of step with the environment.
handle errors. (Bad consequences)

1. Notification (fail →)
2. worst: do wrong action (periodic test (behaviour)) : automate (compare against master dB)
3. Sys log (investigate failure: forensic evidence)

---

health of Computer → disc space
→ ....

need? (check variables)

* Shutil module, disk_usage function: check available space

```
>> import shutil
>> du = shutil.disk_usage("/")
>> print(du)
usage(total = 125..., used = ..., free = ...)
```

CPU usage

* cpu_present from psutil (interval value: avg(realtime))

```
>> import psutil
>> psutil.cpu_percent(0.1)        → seconds
```

make a module (health_check.py)

```
#!/usr/bin/env python3

import shutil
import psutil

def check_disc_usage(disc):
    du = shutil.disk_usage(disc)
    free = du.free / du.total * 100
    return (free > 80)

def check_cpu_usage():
    usage = psutil.cpu_percent(1)
    return usage < 75
```

body:

```
if not check_disk_usage("/") or not check_cpu_
                usage():
    print("Error!")
else:
    Print("Fine!")
```

```
>> chmod +x health_check.py
>>
```

* Vm: Computer simulated through software (Simulate hardware)

* Comm using network card (emulated) ⟶ transmit to software runs Vm transmits packets!

* Reclaim resources (done!)

Config SSH using Linux OS

* Download PEM (give executable permission)
* SSh -i path username . . . .

Chmod 600 ~/Downloads/. . . .PEM

Ssh -i ~/Do . . .Pem username@External IPadd.

Files

Absolute path: Full path.
Relative path: Source path

Read file:

>> file = open ('spider.txt') [Same directory]

(File descriptor: Token, generated by OS (allows programs to do operations)

>> print (file.readline()) ⟶ single line (pointer moves to second line)

>> print (file.read()) ⟶ all lines

>> file.close()

* open — Locked!
* close - Release Lock!

```
with open ('spider.txt') as file:
    print ____
```
⟶ Block (Python automatically closes!)

Iterate

```
with open ('spider.txt') as file:
    for line in file:
        print (line.upper())
```

⟶ each iteration (next line)

each line has /n

Op: Hi

Hello

Avoid

Use Strip(), remove white spaces.

• Strip()

lines = file.readlines() → string array!

If super large files: takes lot of memory!

---

write files

```
with open ('file.txt', 'w') as file
    file.write ('write file');
```

r - read only
w - write only (create) (no append)
a - append
r+ : read & overwritten
b - binary

when edited?
current size?
rename, delete.

os module (provides abstraction)

Note: make sure of paths

```
>> os.remove ('novel.txt')          (delete file)
>> os.rename ('old_name.txt', 'new_name.txt')
>> os.path.exists ('new_name.txt') → (file exists?)
>> os.path.getsize ('spider.txt') → 192 bytes (size?)
>> os.path.getmtime ( ) → long number (timestamp)
                          no. of seconds since 1970
```

why Jan 1, 1970: UNIX (before) : won't be any file! (dB, languages store time!)

```
>> import datetime
>> os.path.getmtime ('spider.txt') → time
>> datetime.datetime.fromtimestamp(time)
>> os.path.abspath ('spider.txt')
```

directories

getcwd → windows (python)
pwd → linux

```
>> print (os.getcwd ()) → current working directory
>> os.mkdir ('new_dir')
>> os.chdir ('new_dir') → change directory!
>> os.rmdir ('new_dir') → empty directory (remove)
                          else: remove all files!
```

>> os.listdir ('mydir') ⟶ list all files in a directory!

>> os.path.join (dir, file_name) [file / directory]

full path : Reference path (from your)

>> os.path.isdir ( ) ⟶ True?

os.path.join (dir, file_name) ⟶ Create a string by joing directory to filename with a valid full name!

↳ makes us indep from os (Reference path)!

Windows: \ (backslash) ⟶ join() handles this!

Linux, mac: /

Write csv files

Report : txt (line by line)

* parse: Analyze content of a file: correctly structure data!

* Html, JSON, CSV

o/p of a command: store (easy to parse: later)!

easy to use in scripts: csv

Read csv:

* import csv

>> f = open ('csv_file.txt')

>> csv_f = csv.reader (f) ⟶ open reader!

>> for row in csv_f :

row ⟶ list of all fields in a row

name, phone, role = row

print ('Name: {3. . . . '. format (name, . . .))

Unpacking into variables: easy to read

Instead of row[0], row[1], ... (follow: tough)

>> f.close()

Store & write in csv:

hosts = [['workstation.local', 'ip'], ...]

machines & ip' in our network!

with open ('hosts.csv', 'w') as hosts_csv :

writer = csv.writer (hosts_csv) ⟶ Instance of csv writer class

writer.writerows (hosts)

1. writerow
2. writerows

Read & write (using dictionaries)

* list: when we know what the fields are (same order!)

Common for csv: 1st row (column names)

DictReader() ⟶ each row into dictionary! (use keys to access data)

```
>> reader = csv.DictReader(Software)
>> for row in reader:
       Print (row ['name'])
```

write (list of dictionaries)

```
users = [{'name': 'Sol', ..... }, {...}]
Keys = ['name', ...]
writer = csv.DictWriter(by_department, field_names = keys)
writer.writeheader() ⟶ creates 1st row based on key!
write.writerows(users)
```

opened file

list of keys.

If omitted: first row used as keys.

Problem:

1. Dialect will be passed as a parameter
2. No well-defined pattern for csv files.
3. Flexible parser needed (Control how csv parser works?)

↓

Involves many parameter (can't be passed)

Solution: Dialect object (groups them)

* register dialect classes: user don't need to know parameter settings!

```
>> csv.register_dialect('empDialect', skipinitialspace = True,
                        strict = True)
```

```
>> employee_file = csv.DictReader(open('spider.csv'), dialect =
                                  'empDialect')
```

Iteration: → dictionary of each row!

for data in employee_file :

Regex

* regex / regexp (Search query : string pattern)

four-letter words?] Text processing!
diff error types?]

* grep, sed, awk (uses regex)

log = 'July 31 07:51:48 mycomputer bad_Process [12345]: ERROR Performing package upgrade'

→ Take.

* Find 1st square bracket (Index)

Problem: any length! (Error process)

regex:

```
import re
result = re.search (regex, log)
```

regex = r"\[(\d+)\]"

→ Any num (any leng)

Start & end with brackets

Basic matching

* grep → prints any line matches pattern

Spell checking (dictionary (.txt) – one word per line)

>> grep thon /usr/share/dict/words (highlights match) : case sensitive!

>> grep -i thon /usr/share/dict/words (case insensitive)

. [matches any character] – wildcard

^ , . , $

special char

beginning End.

Simple regex – Python

>> import re

>> result = re.search (r"aza", "plaza") → <re.Match object; span = (2,5), match = 'aza'>

Use raw strings for regex!

Span attribute - start & end index!

* doesn't match: None (not actual value)

Print (re.search (r"p.ng", "pangaea", re.IGNORECASE))

Wildcards

• (any character)

Strict matches (character objects):

[Pp]

[a-z]way

[^a-z]way

"cat|dog" ⟶ match substring cat (or) dog

Find all matches:

re.findall (___, ___) ⟶ list

Repitation qualifiers

1. longest word in a string
2. host name in a log file b/w brackets

Repeated matches (Greedy behaviour)

py.*n ⟶ py; any char (length), n (ending)

any substring!

[a-z]*

egrep ⟶ other implementations (two additional rep qualifiers). + ?

1) 0+ ⟶ 1/more characters
(1/more 0's)

2) ? ⟶ 0/1 occurence!

Escaping characters

use actual dot : '\.com'

\n → Python : new line.

\ → escape special regex char / special string char.

* Raw strings - avoids these confusions (special characters won't be interpreted when generating string)

* Interpreted during parsing only.

\w → Any alphanumeric + underscore
\b → word boundaries
\S → white space

www.regex101.com

Advanced regex - capture groups

capturing groups: portions of pattern enclosed in parenthesis.

>> result = re.search (r'^(\w*), (\w*)$', 'Lovelace, Ada')

match: 'Lovelace, Ada'

>> print (result.groups())

('Lovelace', 'Ada')

↓

index 0 : matched by match group 0

sub regex (inside parentheses)

result [0] → Lovelace, Ada (complete match)
result [1] → Lovelace (matched by (\w*) first one)
result [2] → Ada

\w → only alphanum and _

Pattern repeating specific no. of times

* hard to read & maintain : same pattern : write many times

Solution:

{1, 2} → min 1 time
max 2 times

"a scary ghost appeared"

re.findall (r"[a-zA-Z]{5}", ↓)

['scary', 'ghost', 'appea']

1 way

```
for word in str.split():
    if(re.search(r'^[a-zA-Z]{5}$', word)):
        lst.append(word)
```

['scary', 'ghost']

Other way:

\b → word limits at beg & end

```
Print(re.search(r'\b[a-zA-Z]{5}\b', str))
```

{5, } → any max!

Extract PID using regex

```
re.search(r'\[(\d+)\]')
```

↓

match: result[1] → PID

None: won't be result[1]!

Split and replace

*Similar with split of string class

* Takes regex as seperator!

Split Sentences

```
r'[.?!]'
```

↓

Inside square brackets

\. not needed

Inside [] no special meaning

no need for escape sequence!

re.split(r'([.?!])', 'One Sentence. Another Sentence? And!')

['One Sentence', '.', ' Another Sentence', '?', ' And', '!']

* Note: Both word & regex matched! (annotations)

sub: new string by substituting all /part

*like replace method (with regex)

Remove email address (anonymity):

_____@_____.com

(letter, num, underscore, dot, %, +, -) — part1 — part2 [alphanum, -, .]

[\w.%+-]+ @ [\w.-]+

Atleast 1! Atleast 1!

re.sub(r'[\w.+%-]+@[\w.-]+', '[Redacted]', 'Received an email from go_nuts95@myexample.com')

Received an email from [Redacted]

Safe than sorry (Redact anything looks like email)

* lot script regex (email check only!) — That's why \b not used!

r'\2 \1'

\1 → 1st captured group!

\2 → 2nd captured group!

re.sub(r'^([\w.-]*), ([\w.-]*)$', r'\2 \1', 'Lovelace, Ada')

group1 = Lovelace (group1 match.)

group2 = Ada (group2 match)

* Back references!

Manage data & processes

Read data interactively: (user I/p)

```
input('Enter age: ')
```

Standard Streams

* How python connects keyboard & screens ⟶ IOStream

IOstream:

* mechanism performing io operations
* Streams: as data keeps flowing!

IO Streams

1. Standard – STD IN
2. Standard o/p – STDOUT
3. Standard error – STDERR (eg: ls –ul) ↓ unsupported flag (random)

Environment Variables

* Shell: reads & executes command (application) – cmd interface with OS (Bash, Sash, Zsh)

`echo $PATH`

* python runs inside shell environment
* Variable set in that env – use in scripts!

* Env ⟶ shows environmental variable (PATH – very important)
* when we call python (check PATH for python in order: program to execute)

Print Paths

dictionary of os module

```
import os
print ('Home:' + os.environ.get ('HOME', ''))
      'SHELL:' + os.environ.get ('SHELL', ''))
      'FRUIT?' + os.environ.get ('FRUIT', ''))   → Not defined
```

environ ['HOME'] —not present→ Error

environ.get ('HOME', '') ⟶ '' (default value!)

define in current environment:

>> export FRUIT = pineapple (In terminal)

>> ./variables.py ⟶ os.environ.get ('FRUIT')

Command line arguments & exit status

* give arguments - when program started. (Sys admin tasks)
* access commandline arguments! (argv)

>> import sys

>> print (sys.argv)

No argument : ['./parameters.py']

>> ./parameters.py one two

['./parameters.py', 'one', 'two']

Exit status

1) Return code to shell
2) 0 - Success
3) other - failed. (info on error)

check exit status (last program) : $?

>> cat variables.py

>> echo $? ⟶ 0

>> cat not_found.py

>> echo $? ⟶ 1

Python (type error) ⟶) return other than 0

already exists ⟶

Subprocess

Run system program from python script.

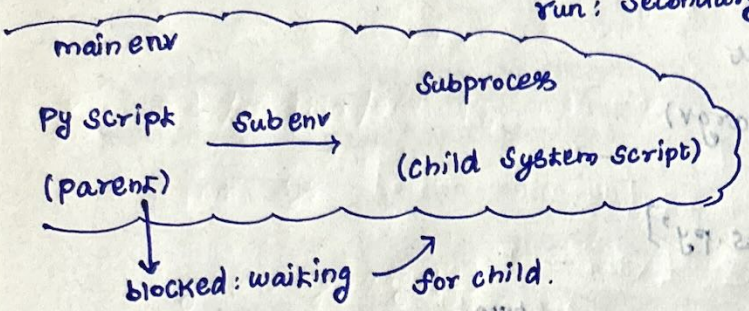
Subprocess module:

subprocess.run(['date'])

↓

returns object of Completed process type

run: Secondary env created for child process!



>> result = subprocess.run(['sleep', '2']) (returncode = 0)

>> print (result.returncode)

Using run: Just want to run (our .py script – no control over it)

Useful for: cp, chmod, sleep (no o/p) – no further use of o/p from functions.

But: If we care of o/p: different strategy!

Obtain o/p of Sys Command (capture)

eg: stats about logging of users!

capture_output = True

>> result = subprocess.run(['host', '8.8.8.8'], capture_output = True)

>> result.returncode ⟶ 0

>> result.stdout ⟶ b'8.8. . . . '

↓

Array of bytes (raw: not python's)

* UTF-8 (std)

Python doesn't know which encoding: So: byte of strings (array): UTF-8.

>> print (result.stdout.decode().split())

['8.8.8.8.in-addr.arpa', 'domain', 'name', 'pointer', 'dns..']

Stderr:

>> result.stderr ⟶ gives error message!

Advanced!

1. modify env variables (where process looks for values)

add entry to the path:

```
import os
import subprocess
>> my_env = os.environ.copy()  ⟶ all path variables (env)
                  ↗ dictionary!
>> my_env["PATH"] = os.pathsep.join(["/opt/myapp/", my_env["PATH"]])
                                  ↘ joins elements of list with path seperator (Based on OS)
>> result = subprocess.run(["myapp"], env=my_env)
```

* Give path: where to look for executable file! : add & give it to sub process!

1. CWD — change current working directory
2. timer — longer, kill
3. shell — first execute in shell (This way: we can have shell expansions & use shell operations) ⟶ Security risk!
4. drawbacks: sys level commands (assumption wrong - unexpected failures)

* When Commands changed!
* Switch OS
* Any change to System!

* Automate quick: well defined task — use Sys commands
* Long run: use external modules! (PYPI repo)!

Log files (Filter with regex)

CRON Jobs: schedule jobs in UNIX

```
#! /usr/bin/env python3
import sys
logfile = sys.argv[1]
with open(logfile) as file:
    for line in file:
        if (line.search (r' CRON ')):
            print (line)
```

(or)

```
if "CRON" in line:
```

little more

Print each user who done a CRON job: (username)

re.search (pattern, line)

r"^USER \(([\w+])\)$"

Improve (how many times each user)

1. Dictionaries!

CRON ERROR Failed to start

July 31 04:11:32 myCompname CRON [__]: ERROR Failed to start CRON Job . . . . . !

Testing

* Evaluate: does expected!
* Manual, Automate! – Test cases

Unit testing

* Small, isolated parts of code : correct/not (production environment – No Failure)
* ip ⟶ o/p $\xrightarrow{\text{match}}$ ideal o/p

writing: (Run test, verify): Along side code (_test)

eg:

unittest module ⟶ Provides test case class!

reorange_test.py

```
#!/usr/bin/env python
from rearrange import re_arrange_name
import unittest

class TestReorange(unittest.TestCase):
    def test_basic(self):
        testcase = 'Lovelace, Ada'
        expected = "Ada Lovelace"
        self.assertEqual(re_arrange_name(
            testcase), expected)
```

equal (pass)
else (Failed)

Note: Any method with _test : automatically becomes test

→ unittest.main() ⟶ runs test for us

edge cases

1. choose test cases: may fail (empty string, null string): extreme ranges!

```
class TestRearrange(unittest.TestCase):
    def test_basic(self):
        testcase = 'Lovelace, Ada'
        expected = 'Ada Lovelace'
        self.assertEqual(rearrange_name(testcase), excepted)

    def test_empty(self):
        testcase = ""
        expected = ""
        self.assertEqual(rearrange_name(testcase), expected)

unittest.main()
```

debug

```
import re

def rearrange_name(name):
    result = re.search(r"^([\w .]*), ([\w .]*)$", name)
    if result is None:
        return ""
    return "{} {}".format(result[2], result[1])
```

> error is better than silent malfunction

Some test cases:

1. double name: 'Hopper, Grace M.' / 'Grace M. Hopper'
2. 'Voltaire' ⟶ one name (no comma)!
   'Voltaire'
   result[2] ⟶ won't be there!
   result ⟶ ""

```
if result is None:
    return name
```

assertTrue()
assertFalse()
assertIs()...

Blackbox (vs) whitebox

whitebox (clear box, transparent):

* Test creator's knowledge on Software (to build test case)
  - ↳ what Software for? works?

Black box (opaque box)

* Tester - doesn't know inside
* Know o/p & i/p : requirements.
  - ↳ what is supposed to do?

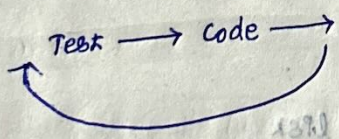
others:

1. Integration test (interaction b/w diff codes: works?) - API, client, ... (seperate test dB, env)
2. Regression tests (variant of unit test: verify fixed! bug)
   - ↳ same bug can't be introduced twice
3. Smoke test: (test hardware): Sanity test for major bugs!
4. Load test: traffic test! (performance)

Suite of tests

Most tests: (after code written)

Test driven development (while writing codes)



* Continuous integration (Test & deploy)

try-except

try:

except:

Raise errors

1) raise ValueError ('minlen must be positive')
2) assert → verifies & throws assertion error

* Testing expecting errors

assertRaises method!

Bash scripting

* os.listdir ⟶ Convert (going through all files) [complex - python]

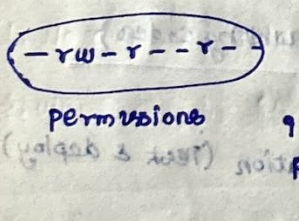
Bash - few lines - easy

* Bash - few lines automation
* eg: mount discs - archive & compress data!

echo, ls, chmod, cat → permissions.

Copy

* cp ../spider.txt .
  - ../ → parent directory
  - . → current directory



* ls -l ⟶ list
* ls -la ⟶ hidden

Rename

mv old new

→ rm * (all files)

→

Redirect streams

→ cat myfile.txt > file1 (create & put o/p) : overwritten!

→ cat myfile.txt >> file1 (append)

Send i/p

./streams_err.py < new_file.txt

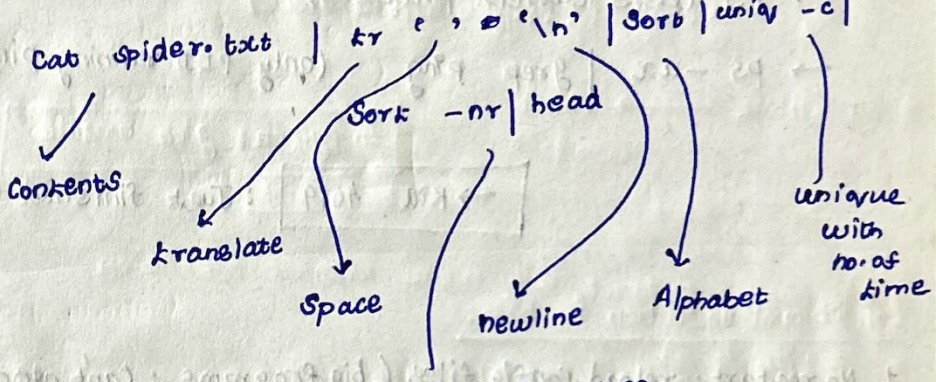
./streams_err.py < newfile.txt 2> error_file.txt

redirecting stderr to another file (file descriptor)! (stderr)

0 - stdin
1 - stdout
2 - stderr

pipelines & pipes (I/o stream redirection)

* connect 1 program o/p to o/p of another!
* '|'



head - 10 lines alones

→ cat spider.txt | tr ' ' '\n' | sort | uniq -c | sort -nr | head

→ cat hi.txt | ./capitalize.py [./capitalize.py < hi.txt]

Note: When sys command exist - use python script

Signals to running processes.

* Timeout — stop process
* Signals: Tokens delivered to running processes to indicate a desired action.

1. pause
2. Terminate
3. Reload config
4. close all open files

Send Signals → ctrl+c [prints summary + result] : Stopped (Finishes cleanly)

SIGINT

→ ctrl+z [Not properly finished] - Just stopped

SIGSTOP [Stop running without terminating] (pause)

>> fg (run again after ctrl+z)

Send other signals: (Kill command)

→ Kill → SIGTERM (need: PID (process id))
(program)

→ PS (lists current processes)

→ ps -ax | grep ping (only PID running ping) → 4619

→ Kill 4619 : Just finished (no clean termination)

1. No restart: reload config files (big programs: can't afford to rerun again)

2. webservices: end a connection! (terminate cleanly)

Command > file (overwrite)

Command >> file (append)

Command < file (Std in from file)

Command 2 > file (redirect error to file)

Command 1 | command 2 (redirect 1's o/p to 2)

ps → running processes

ps -ax (all users)

ps -e (environment)

kill PID

fg → run stopped process

bg → stopped process to background

jobs → jobs running/stopped