Appendix A

March 22, 2019

main script (main.py) :

```
### IMPORTATING STATION ###
from instruction_queue import * #for the timing table
from reservation_station import *
from load_station import *
from register import Registers

### IMPORTING MODULES ###
import numpy as np
import os

###################
### PARAMETERS ###
###################

memory_file_name = "memory.txt"
input_file_name = "custom_input.txt"

## Number of RS, Register entries ##
nb_add = 3
nb_mult = 2
nb_load = 6
nb_store = 6
nb_register = 11

RESVNUMCONFIG = {
    'Add': nb_add,
    'Mult': nb_mult,
    'Load': nb_load,
    'Store': nb_store
}

# clock cycle needed per type of operation ##
cpi_add = 2
cpi_sub = 2
cpi_mul = 6
cpi_div = 12
cpi_load = 3
cpi_store = 3

# Initial Register values ##
val_reg = np.zeros(nb_register)
reg_init = [6.0, 0., 3.5, 0., 10., 0., 0., 0., 7.8, 0.]

for i in range(len(reg_init)):
    val_reg[i] = reg_init[i]

## Initial pc and clock ##
clock = 0
pc = 0

## Creation of RS, Register ##
Add = Add_RS(RESVNUMCONFIG)
Mult = Mul_RS(RESVNUMCONFIG)
Load = Load_Station(RESVNUMCONFIG, memory_file_name)
Store = Store_Station(RESVNUMCONFIG, memory_file_name)
Register = Registers(nb_register, val_reg)

## MESSAGES ##

message_fp_issued = "NEXT FP INSTRUCTION TO BE ISSUED: "

######################
### MAIN FUNCTION ###
######################

def main():
    global pc
    global clock
```

```
    cdb_buffer = [] # treated as a priority queue for the commun data bus
    list_cdb=[["",-1,-1]]

    # Main iteration
    while not timing_table.check_everything_finished():
        #input("Press enter to simulate a clock")
        clock+=1 # clock cycle added one
        print("\n")

        print("#################################################################################################################")

        print("#################################################################################################################")
        print("\n")
        print("Clock cycle :", clock)
        print("PC :", pc, "\n")

        # Fetch instruction to reservations
        load_instruction(instructions)

        # Check for execution start of instruction in reservation
        started()

        # Load all finished instruction from the reservation station
        cdb_buffer = is_finished()
        timing_table_finished(cdb_buffer)

        # Broadcast Instruction of CDB to Register and RS
        if len(cdb_buffer)>0:
            # Broadcast instruction value of the smallest pc first
            tag_cdb, value_cdb, pc_cdb = cdb_buffer[0]
            list_cdb.append([tag_cdb, value_cdb, pc_cdb])
            # Reset the RS and Register that finished


        else:
            list_cdb.append(["",-1,-1])

        if list_cdb[-2][0] != "" and list_cdb[-2][0] !=list_cdb[-3][0]:
            # Update the timing table for Write back when the instruction is Broadcasted
            # (the one finished in the previous clock [-2])
            cdb_update(list_cdb[-2][0],list_cdb[-2][1])
            timing_table.timing_update_wb(list_cdb[-2][2], clock)
            print("Instruction '"+str(instructions[list_cdb[-2][2]])+"' BROADCASTED in CDB at clock
            ↪  "+str(clock)+" with value '"+str(list_cdb[-2][1])+"'" )
        # Update
        update()


        # Print the Tables
        timing_table.printList()
        Add.printList()
        Mult.printList()
        Load.printList()
        Register.printList()
        reset(list_cdb[-1][0])
    return("SIMULATION FINISHED")


### LOAD FUNCTIONS ###

# Load a single instrution of number pc to the corresponding RS if possible
def load_instruction(instructions):
    global pc
    global clock

    # If pc is bigger then the highest instruction pc no more instruction to issue
    if (pc >= len(instructions)):
        return True
```

```
    # Instruction has to be issued
    else:
        instruction = instructions[pc].split(" ")  # Spliting the instruction
        type_op = instruction[0]                   # First arg is the operation type
        print(message_fp_issued, "'"+str(instructions[pc])+"'","\n")
        if (type_op == "ADDD"):
            FetchInstruction(Add,instruction,cpi_add)
        if (type_op == "SUBD"):
            FetchInstruction(Add,instruction,cpi_sub)
        if (type_op == "MULTD"):
            FetchInstruction(Mult,instruction,cpi_mul)
        if (type_op == "DIVD"):
            FetchInstruction(Mult,instruction,cpi_div)
        if (type_op == "LD"):
            Fetchload(Load,instruction,cpi_load)

        # We move to the next instruction after successfully issuing one
        pc += 1

# Put the corresponding instruction to the RS entry
def FetchInstruction(station,instruction,cpi):
    global pc
    global clock

    type_op = instruction[0]    # Type of operation
    dest = instruction[1]       # Destination Register of the instruction
    reg_valueI = instruction[2] # First operand
    reg_valuek = instruction[3] # Second operand

    # Get a free Position from the corresponding station (Addition or Multiplication) in order
    # results[0] = -1 if no free position in RS
    result = station.getFreePosition()

    # Update the free RS entries when the destination Register is not Busy
    if result[0]>=0 and (not Register.isBusy(dest)):
        print("Instruction '"+type_op+" "+dest+" "+reg_valueI+" "+reg_valuek+"' ISSUED at clock "+str(clock)+"
        ↪  in '"+str(result[1])+"'")
        Register.updateRegisterTag(result[1],dest)
        operand1 = Register.getRegister(reg_valueI)
        operand2 = Register.getRegister(reg_valuek)
        station.loadInstruction(operand1[1], operand1[0], operand2[1], operand2[0], result[0], type_op, pc,
        ↪  cpi)

        # Update of the timing table entries after issuing
        timing_table.timing_update_issue(pc, clock)
        return True

    # Otherwise Stall to wait for free RS entries
    pc-=1 # Reduce by one to maintain pc number in load_instruction(instructions)
    return False

# Put the corresponding instruction to the RS entry same as FetchInstruction(station,instruction,cpi)
# But less entries needed
def Fetchload(station,instruction,cpi):
    global pc
    global clock
    type_op = instruction[0]
    dest = instruction[1]
    operand = instruction[2]
    result = station.getFreePosition()
    if result[0]>=0 and (not Register.isBusy(dest)):
        Register.updateRegisterTag(result[1],dest)
        offset = extract_offset_reg(operand)[0]
        reg_value = extract_offset_reg(operand)[1]
        Load.loadInstruction(reg_value,offset, result[0], type_op, pc, cpi)
        timing_table.timing_update_issue(pc, clock)
        return True

    pc-=1
    return False
```

```
# Function to get value of the Memory Addresses
def extract_offset_reg(instruction_text):
    inst_split = instruction_text.replace(')', '(').split('(')
    offset = inst_split[0]
    #print("offset : ", offset)
    reg_value = inst_split[1][1]
    #print("reg_value : ", reg_value)
    return (offset, reg_value)


### UPDATE FUNCTIONS ###

# Update the Time left by instruction in the RS
def update():
    Add.update_clock()
    Mult.update_clock()
    Load.update_clock()

# Update timing table Execution start entries denpending on the time left of execution
# If Time left for execution = clock needed per instruction then we know the instruction started execution
def started():
    for item in Add.reservation :
        if item.op == "ADDD" and item.time == cpi_add-1 and item.fuState ==1:
            timing_table.timing_update_start(item.ins_pc, clock)
            print("Instruction '"+str(timing_table.instructionList[item.ins_pc].ins[0])+"' STARTED at clock
            ↪  "+str(clock)+" in '"+str(item.tag)+"'")
        if item.op == "SUBD" and item.time == cpi_sub-1 and item.fuState ==1:
            timing_table.timing_update_start(item.ins_pc, clock)
            print("Instruction '"+str(timing_table.instructionList[item.ins_pc].ins[0])+"' STARTED at clock
            ↪  "+str(clock)+" in '"+str(item.tag)+"'")
    for item in Mult.reservation :
        if item.op == "MULTD" and item.time == cpi_mul-1 and item.fuState ==1:
            timing_table.timing_update_start(item.ins_pc, clock)
            print("Instruction '"+str(timing_table.instructionList[item.ins_pc].ins[0])+"' STARTED at clock
            ↪  "+str(clock)+" in '"+str(item.tag)+"'")
        if item.op == "DIVD" and item.time == cpi_div-1 and item.fuState ==1:
            timing_table.timing_update_start(item.ins_pc, clock)
            print("Instruction '"+str(timing_table.instructionList[item.ins_pc].ins[0])+"' STARTED at clock
            ↪  "+str(clock)+" in '"+str(item.tag)+"'")
    for item in Load.reservation :
        if item.op == "LD" and item.time == cpi_load-1 and item.fuState ==1:
            timing_table.timing_update_start(item.ins_pc, clock)
            print("Instruction '"+str(timing_table.instructionList[item.ins_pc].ins[0])+"' STARTED at clock
            ↪  "+str(clock)+" in '"+str(item.tag)+"'")

# Check if any RS entries finished executing and are ready to be broadcast
def is_finished():
    list_add = Add.finish()
    list_mult = Mult.finish()
    list_load = Load.finish()
    list_finished = list_load+list_add+list_mult
    return list_finished

# Update timing table Execution finished entries depending of the is_finished() list
def timing_table_finished(list_finished):
    global clock
    for item in list_finished:
        timing_table.timing_update_finish(item[2], clock)
        print("Instruction '"+str(timing_table.instructionList[item[2]].ins[0])+"' FINISHED at clock
        ↪  "+str(clock)+" in '"+str(item[0])+"'")

# When a value is ready in RS => Broadcast
# Input is the tag of the operation and value
def cdb_update(tag, value):
    # Check and update RS
    Add.updateValueByTag(tag, value)
    Mult.updateValueByTag(tag, value)

    # Check and update Register
```

```
    Register.updateRegisterByTag(tag,value)

### RESET FUNCTIONS ###

# Reset the RS entries that finished execution and have been broadcasted
def reset(tag):
    if tag == "":
        return True
    tag_name = tag[:-1]
    tag_position = int(tag[-1])
    if tag_name == "Add":
        Add.reset(tag_position)
    if tag_name == "Mult":
        Mult.reset(tag_position)
    if tag_name == "Load":
        Load.reset(tag_position)

### PRINT FUNCTIONS ###

# Initial timing table with instructions inside
def initial_table(instructions):
    timing_table = Timing(instructions)

    ↪  print("=============================================================================================================================
    print("Clock cycle :", clock, "\n")
    timing_table.printList()
    Add.printList()
    Mult.printList()
    Load.printList()
    Register.printList()
    return timing_table

# Txt file decoder for instruction list
def input_file_decoder(in_file):
    input_file = open(in_file, 'r')
    instructions = []
    for line_not_split in input_file:
        if(line_not_split != ""):
            line_not_split = line_not_split.split("\n")[0]
            instructions.append(line_not_split.replace(",", " "))
    return instructions


#########################
### PROGRAM EXECUTION ###
#########################

if __name__ == '__main__':
    #input("Press Enter to Start")
    if len(input_file_name) > 1:
        instructions = input_file_decoder(input_file_name)
        print("INSTRUCTION LIST : ","\n")
        print("\n".join(instructions))
        timing_table = initial_table(instructions)
        main()
    else:
        print("Please specify input file!")
        exit(1)
```

Reservation Station (reservation_station.py):

```
from tabulate import tabulate


class RS(object):
    def __init__(self, RESVNUMCONFIG, name):
        self.reservation = []
        self.size = RESVNUMCONFIG[name]
        for t in range(self.size):
            row = Row(name+str(t))
            self.reservation.append(row)

    def getFreePosition(self):
        for i in range(self.size):
            if(not self.reservation[i].isBusy()):
                return i, self.reservation[i].tag
        return -1, ""

    def loadInstruction(self, Qj, valueJ, Qk, valueK, position, type_op, ins_pc, cpi):
        row = self.reservation[position]
        row.Qj = Qj
        row.valueJ = valueJ
        row.Qk = Qk
        row.valueK = valueK
        row.op = type_op
        row.ins_pc = ins_pc
        row.time = cpi
        row.busy = True
        row.cpi_init = cpi

    def updateValueByTag(self, tag, value):
        for i in range(self.size):
            row = self.reservation[i]
            if(row.Qj == tag):
                row.Qj = ""
                row.valueJ = value
            if(row.Qk == tag):
                row.Qk = ""
                row.valueK = value

    def time_Left(self, position):
        return self.reservation[position].time

    def update_clock(self):
        for i in range(self.size):
            if (self.reservation[i].isBusy() and (self.reservation[i].Qj == "" or self.reservation[i].Qj ==
            ↪  self.reservation[i].tag) and (self.reservation[i].Qk == "" or self.reservation[i].Qk ==
            ↪  self.reservation[i].tag) and self.reservation[i].time >= 1 and self.reservation[i].time ==
            ↪  self.reservation[i].cpi_init):
                self.reservation[i].fuState = 1
                break
        for i in range(self.size):
            if(self.reservation[i].isBusy() and (self.reservation[i].Qj == "" or self.reservation[i].Qj ==
            ↪  self.reservation[i].tag) and (self.reservation[i].Qk == "" or self.reservation[i].Qk ==
            ↪  self.reservation[i].tag) and self.reservation[i].time >= 1 and self.reservation[i].fuState ==
            ↪  1):
                self.reservation[i].time -= 1

    def finish(self):
        finished_list = []
        for i in range(self.size):
            row = self.reservation[i]
            if row.time == 0:
                if row.op == "ADDD":
                    tag, value = row.tag, round(row.valueJ + row.valueK, 3)
                elif row.op == "SUBD":
                    tag, value = row.tag, round(row.valueJ - row.valueK,3)
                elif row.op == "MULTD":
                    tag, value = row.tag, round(row.valueJ * row.valueK,3)
```

```
            elif row.op == "DIVD":
                tag, value = row.tag, round(row.valueJ / row.valueK,3)
            finished_list.append([tag, value, row.ins_pc])
        return finished_list

    def reset(self, position):
        self.reservation[position].reset()


class Add_RS(RS):
    def __init__(self, RESVNUMCONFIG):
        super().__init__(RESVNUMCONFIG, "Add")

    def printList(self):

        ↪   print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        print("{:^120}".format("Reservation Station"))

        ↪   print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        column_names = ["Time left", "Tag", 'OP','Busy', 'valueJ', 'valueK', 'Qj', 'Qk']
        row_format = "{!s:^15}" * len(column_names)
        print(row_format.format(*column_names))
        for entry in self.reservation:
            if entry.time == -1:
                entry.time = ""
            entry_list = [entry.time, entry.tag, entry.op, entry.busy, entry.valueJ, entry.valueK, entry.Qj,
            ↪   entry.Qk]
            print(row_format.format(*entry_list))


class Mul_RS(RS):
    def __init__(self, RESVNUMCONFIG):
        super().__init__(RESVNUMCONFIG, "Mult")

    def printList(self):
        column_names = ["Time left", "Tag", 'OP','Busy', 'valueJ', 'valueK', 'Qj', 'Qk']
        row_format = "{!s:^15}" * len(column_names)
        #print(row_format.format(*column_names))
        for entry in self.reservation:
            if entry.time == -1:
                entry.time = ""
            entry_list = [entry.time, entry.tag, entry.op, entry.busy, entry.valueJ, entry.valueK, entry.Qj,
            ↪   entry.Qk]
            print(row_format.format(*entry_list))

        ↪   #print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        print("\n")



class Row(object):
    def __init__(self, tag):
        self.tag = tag
        self.reset()

    def isBusy(self):
        return self.busy

    def reset(self):
        self.op = ""
        self.Qj = ""
        self.valueJ = 0
        self.Qk = ""
        self.valueK = 0
        self.busy = False
        self.time = -1
        self.ins_pc = ""
        self.fuState = 0
        self.cpi_init = -1
```

```
def isFinished(self):
    if self.time == 0:
        return True
    else:
        return False
```

Load Buffer (load_station.py):

```
from tabulate import tabulate


class Load_Store(object):
    def __init__(self, RESVNUMCONFIG, name,memory):
        self.reservation = []
        self.size = RESVNUMCONFIG[name]
        self.memory = memory
        for t in range(self.size):
            row = Row(name+str(t))
            self.reservation.append(row)


    def getFreePosition(self):
        for i in range(self.size):
            if(not self.reservation[i].isBusy()):
                return i, self.reservation[i].tag
        return -1, ""

    def loadInstruction(self, reg_value,offset, position, type_op, ins_pc, cpi):
        row = self.reservation[position]
        row.reg_value = reg_value
        row.offset = offset
        row.address = str(offset) +'+R'+str(reg_value)
        row.op = type_op
        row.ins_pc = ins_pc
        row.time = cpi
        row.busy = True
        row.cpi_init = cpi

    def updateValueByTag(self, tag, value):
        for i in range(self.size):
            row = self.reservation[i]
            if(row.value == tag):
                row.value = value

    def time_Left(self, position):
        return self.reservation[position].time

    def update_clock(self):
        for i in range(self.size):
            if (self.reservation[i].isBusy() and self.reservation[i].time == self.reservation[i].cpi_init):
                self.reservation[i].fuState = 1
                break

        for i in range(self.size):
            if self.reservation[i].isBusy() and self.reservation[i].fuState == 1 :
                self.reservation[i].time -= 1

    def finish(self):
        finished_list = []
        for i in range(self.size):
            row = self.reservation[i]
            if row.time == 0:

                if row.op == "LD":

                    file_object  = open(self.memory, "r")
                    count = 0
                    while count <= int(row.reg_value):
                            ret = file_object.readline()
                            count = count + 1
                    file_object.close()
                    row.value = float(ret) + float(row.offset)
                    tag, value = row.tag, row.value

                finished_list.append([tag, value, row.ins_pc])
        return finished_list
```

```
    def reset(self,position):
        self.reservation[position].reset()

class Load_Station(Load_Store,):
    def __init__(self, RESVNUMCONFIG,memory):
        super().__init__(RESVNUMCONFIG, "Load",memory)

    def printList(self):
        print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
        print("{:^65}".format("Load Station"))
        print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
        column_names = ["Time left", "Tag", "Busy", "Address"]
        row_format = "{!s:^15}" * len(column_names)
        print(row_format.format(*column_names))
        for entry in self.reservation:
            if entry.time == -1:
                entry.time = ""
            entry_list = [entry.time, entry.tag, entry.busy, entry.address]
            print(row_format.format(*entry_list))
        #print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")
        print("\n")


class Store_Station(Load_Store):
    def __init__(self, RESVNUMCONFIG,memory):
        super().__init__(RESVNUMCONFIG, "Store",memory)


class Row(object):
    def __init__(self, tag):
        self.tag = tag
        self.reset()

    def isBusy(self):
        return self.busy

    def reset(self):
        self.op = ""
        self.reg_value = -1
        self.offset = 0
        self.value = -1
        self.address = ""
        self.busy = False
        self.time = -1
        self.ins_pc = ""
        self.fuState = 0
        self.cpi_init = -1

    def isFinished(self):
        if self.time == 0:
            return True
        else:
            return False
```

Register Station (register.py):

```
from tabulate import tabulate

class Register(object):
    def __init__(self, name, Qi, value):
        self.name = name
        self.Qi = Qi
        self.value = value

class Registers(object):
    def __init__(self, size, values):
        self.registerList = []
        self.size = size
        for i in range(size):
            register = Register("F"+str(i),"", values[i])
            self.registerList.append(register)

    def getRegister(self, name):
        return self.registerList[int(name[1:])].value,self.registerList[int(name[1:])].Qi

    def isBusy(self,name):
        if self.registerList[int(name[1:])].Qi == "":
            return False
        else :
            return True


    def editRegister(self, register, number):
        """
        edit Register F'n' for updating
        """
        self.registerList[number] = register

    def updateRegisterTag(self,tag, name):
        self.registerList[int(name[1:])].Qi = tag

    def updateRegisterByTag(self, tag, value):
        for i in range(self.size):
            if(self.registerList[i].Qi == tag):
                reg = Register(self.registerList[i].name,"", value)
                self.editRegister(reg, i)

    def printList(self):

        ↪  print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        print("{:^120}".format("Register"))

        ↪  print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        List = self.registerList.copy()
        List.insert(0,Register("Name","Qi","Value"))
        column_names = [List[i].name for i in range(self.size)]
        column_Qi = [List[i].Qi for i in range(self.size)]
        column_value = [List[i].value for i in range(self.size)]
        row_format = "{!s:^10}" * (len(column_names))
        print(row_format.format(*column_names))
        print(row_format.format(*column_Qi))
        print(row_format.format(*column_value))

        ↪  #print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Timing Table (instruction_queue.py):

```
from tabulate import tabulate

## Instruction queue object
class Time(object):
    def __init__(self, pc ,ins):
        self.pc = pc
        self.ins = ins,
        self.issue = "-"
        self.start = "-"
        self.finish = "-"
        self.isFinished = False
        self.wb = "-"

class Timing(object):
    def __init__(self, instructions):
        self.instructionList = []
        self.size = len(instructions)
        for i in range(self.size):
            time = Time(i,instructions[i])
            self.instructionList.append(time)

    def timing_update_issue(self, pc, clock):
        self.instructionList[pc].issue = clock

    def timing_update_start(self, pc, clock):
        self.instructionList[pc].start = clock

    def timing_update_finish(self, pc, clock):
        if not self.instructionList[pc].isFinished:
            self.instructionList[pc].finish = clock
            self.instructionList[pc].isFinished = True

    def timing_update_wb(self, pc, clock):
        if pc>=0:
            self.instructionList[pc].wb = clock

    def getList(self):
        return self.instructionList

    def check_everything_finished(self):
        for tt_entry in self.instructionList:
            if tt_entry.wb == "-":
                return False
        return True

    def printList(self):

        ↪  print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        print("{:^120}".format("TIMING TABLE"))

        ↪  print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        column_names = [ "PC", "INSTRUCTION", "ISSUED", "STARTED", "FINISHED", "Write CDB"]
        row_format = "{!s:^20}" * len(column_names)
        print (row_format.format(*column_names))
        for tt_entry in self.instructionList:
            tt_entry_list = [tt_entry.pc , tt_entry.ins[0], tt_entry.issue, tt_entry.start, tt_entry.finish,
            ↪  tt_entry.wb]
            print(row_format.format(*tt_entry_list))

        ↪  #print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        print("\n")
```