```
# ./01_EventExample.py
#----------------------------------------
import time;
import threading;


available = threading.Event();

myfile = open("sharedfile.txt", "w+");

if myfile is None:
    print ("Failed to open file 'sharedfile.txt'");
    exit();


def writer():
    global myfile;
    global available;
    while True:
        myfile.seek(0,0);
        print("writing to file");
        myfile.write("Hello all\n");
        available.set();
        time.sleep(3);

def reader():
    global myfile;
    global available;
    while True:
        available.wait();
        myfile.seek(0,0);
        l = myfile.readline();
        print(f"Line read from file is {l}");
        available.clear();


t1 = threading.Thread( target = reader);
t2 = threading.Thread( target = writer);

t1.start();
t2.start();

t1.join();
```

```
t2.join();
# ./01_Mutex.py
#----------------------------------------
import threading;
import time;

counter_lock = threading.Lock();
counter = 0;

def thr_func_nolock(arg):
    global counter;
    print(f'{arg} started.,');
    while True:
        counter += 1;
        print(f'{arg} : {counter}\n');
        time.sleep(0.1)

def thr_func(arg):
    global counter;
    global counter_lock;
    print(f'{arg} started.,');
    while True:
        counter_lock.acquire();
        counter += 1;
        print(f'{arg} : {counter}\n');
        counter_lock.release();
        time.sleep(0.1)

threadfunc = thr_func;

t1 = threading.Thread(target = thr_func, args = ('First',));
t2 = threading.Thread(target = thr_func, args = ('Second',));


t1.start();
t2.start();

t1.join();
t2.join();
# ./01_RLockMutex.py
#----------------------------------------
import threading;
```

```
import time;


import threading;
import time;

# counter_lock = threading.Lock();
counter_lock = threading.RLock();

counter = 0;

def thr_func_nolock(arg):
    global counter;
    print(f'{arg} started.,');
    while True:
        counter += 1;
        print(f'{arg} : {counter}\n');
        time.sleep(0.1)

def function3():
    global counter;
    global counter_lock;
    counter_lock.acquire();
    print(f"Function 3: {counter}");
    counter_lock.release();

def function2():
    print("Function 2");
    function3();

def function1():
    print("funciton 1");
    function2();

def thr_func(arg):
    global counter;
    global counter_lock;
    print(f'{arg} started.,');
    while True:
        counter_lock.acquire();
        function1();
        counter += 1;
```

```
        print(f'{arg} : {counter}\n');
        counter_lock.release();
        time.sleep(0.1)

threadfunc = thr_func;

t1 = threading.Thread(target = thr_func, args = ('First',));



t1.start();

t1.join();
# ./01_threadexample.py
#----------------------------------------
"""
Threads allow you to perform multiple tasks concurrently within a
single program.

Note that threads share the same memory space,
so you need to be careful when accessing shared
resources to avoid race conditions and synchronization issues.
You can use synchronization primitives like locks, semaphores,
or queues to coordinate access to shared resources.
"""
import threading
import time

# Function to be executed by a thread
def thread_function(name):
    print("Thread", name, "started" + '\n');
    time.sleep(3); # Simulate some work
    print("Thread", name, "finished" + '\n');

threads = [ 'Thread 1', 'Thread 2', 'Thread 3', 'Thread 4', 'Thread
5', \
            'Thread 6', 'Thread 7', 'Thread 8', 'Thread 9', 'Thread
10']
# Create threads
threadids = [];
for thread_name in threads:
```

```
    tid = threading.Thread(target=thread_function,
args=(thread_name,))
    # tid.start();
    threadids.append(tid);

for tid in threadids:
    tid.start();

# Wait for threads to finish
for tid in threadids:
    tid.join();

print("All threads finished")
# ./01_thread_Class.py
#----------------------------------------
import threading;
import time;

def setPin(pin):
    print(f"{pin} is set");

def clearPin(pin):
    print(f"{pin} is clear");

class Motor (threading.Thread):
    def __init__(self, pin, ontime, offtime):
        self.pin     =   pin;
        self.ontime  = ontime;
        self.offtime = offtime;
        super(Motor,self).__init__();

    def run(self):
        print("Motor contorl pin  {0}".format(self.pin))
        print("Motor will be on for  {0}
seconds".format(self.ontime))
        print("Motor will fbe off for {0}".format(self.offtime))

        while True:
            setPin(self.pin);
            time.sleep(self.ontime);
            clearPin(self.pin);
            time.sleep(self.offtime);
```

```
if __name__ == '__main__':
    rotor = Motor(10, 2, 5);
    fan   = Motor(21, 1, 3);
    print ("Starting rotor motor");
    rotor.start();
    print( "Staritng fan");
    fan.start();

# ./02_pythonmutex.py
#----------------------------------------
"""
In this example, we create a mutex (lock) using the threading.Lock()
class.

The increment_counter function represents a critical section of code
where a shared resource (in this case, shared_counter) is accessed.
Within this critical section, we acquire the mutex using
mutex.acquire(), perform the necessary operations on the shared
resource, and release the mutex using mutex.release().

Multiple threads are created, and each thread executes the
increment_counter function, trying to increment the shared counter.

By using the mutex, only one thread can acquire the lock (mutex) at
a time. This ensures that the critical section of code (incrementing
the shared counter) is executed atomically, preventing any race
conditions or data inconsistencies.

After all the threads complete, we print the final value of the
shared counter.

When you run this code, you'll see that the shared counter is
incremented correctly and the final value of the shared counter is
consistent due to the mutex ensuring mutual exclusion among the
threads.
"""
import threading
import time
```

```python
# Create a mutex
mutex = threading.Lock()

# Shared resource
shared_counter = 0

def increment_counter():
    global shared_counter

    # Acquire the mutex
    mutex.acquire()

    # Increment the shared counter
    shared_counter += 1

    # Release the mutex
    mutex.release()

# Create and start multiple threads
threads = []
for _ in range(5):
    thread = threading.Thread(target=increment_counter)
    thread.start()
    threads.append(thread)

# Wait for all threads to complete
for thread in threads:
    thread.join()

# Print the final value of the shared counter
print("Final value of shared counter:", shared_counter)
# ./02_pythonsemaphore.py
#----------------------------------------
import threading
import time

# Create a semaphore with an initial value of 2
semaphore = threading.Semaphore(2)

def perform_task(task_id):
    print(f"Task {task_id} is waiting to acquire the semaphore.")
```

```
    # Acquire the semaphore
    semaphore.acquire()
    print(f"Task {task_id} has acquired the semaphore.")

    # Perform some task
    print(f"Task {task_id} is performing the task...")
    time.sleep(2)

    # Release the semaphore
    semaphore.release()
    print(f"Task {task_id} has released the semaphore.")

# Create and start multiple threads
threads = []
for i in range(5):
    thread = threading.Thread(target=perform_task, args=(i,))
    thread.start()
    threads.append(thread)

# Wait for all threads to complete
for thread in threads:
    thread.join()
# ./02_subproceess.py
#----------------------------------------
import subprocess
try:
    p = subprocess.run(["python", "proc.py"]);
except:
    print("Subprocess thrown exception");

print(p.returncode)

print("*" * 40);

# Check is used to throw an exception when subprocess fails:
check=True
p = subprocess.run(["python", "proc.py"], check=True);
print(p.returncode)
print("*" * 40);

# Check is used to throw an exception when subprocess fails:
check=True
```

```
p = subprocess.run(["python", "failingporc.py"], check=False);
print(p.returncode)
print("*" * 40);

# Check is used to throw an exception when subprocess fails:
check=True
p = subprocess.run(["python", "failingporc.py"], check=True);
print(p.returncode)

print("*" * 40);

p = subprocess.run(["python", "proc.py"], check=True, timeout=2);
print(p.returncode)
print("*" * 40);

p = subprocess.run(["python", "proc.py"], check=True, timeout=2,
shell=True);
print(p.returncode)

print("*" * 40);

p = subprocess.run(["python", "proc.py"], check=True, timeout=2,
shell=False);
print(p.returncode)
# ./03_texttospeech.py
#----------------------------------------
import pyttsx3
engine = pyttsx3.init()
voices = engine.getProperty('voices')
# # print(voices);
# for voice in voices:
#         print(voice.languages[0]);
# for voice in voices:
#     print(voice, voice.id)
#     engine.setProperty('voice', voice.id)

#lang_choice = u'fr_CA'
lang_choice = u'hi_IN'
for voice in voices:
    if voice.languages[0] == lang_choice:
        engine.setProperty('voice', voice.id)
        break
```

```
tanu = "salut,ou j'habite dans swindon";
sonu = "Lets go";
sonutxt = "ટીપે ટીપે સરોવર બંધાય, કાંકરે કાંકરે પાડ બંધાય";
print(sonutxt);
engine.say(sonu)
engine.runAndWait()
# ./03_tts2.py
#-----------------------------------------
"""
en_US
it_IT
sv_SE
fr_CA
de_DE
he_IL
id_ID
en_GB
es_AR
nl_BE
en-scotland
en_US
ro_RO
pt_PT
es_ES
es_MX
th_TH
en_AU
ja_JP
sk_SK
hi_IN
it_IT
pt_BR
ar_SA
hu_HU
zh_TW
el_GR
ru_RU
en_IE
es_ES
nb_NO
es_MX
```

```
en_IN
en_US
da_DK
fi_FI
zh_HK
en_ZA
fr_FR
zh_CN
en_IN
en_US
nl_NL
tr_TR
ko_KR
ru_RU
pl_PL
cs_CZ
"""

import pyttsx3
engine = pyttsx3.init();
voices = engine.getProperty('voices')

lang_choice = u'en-scotland';
# for voice in voices:
#     if voice.languages[0] == lang_choice:
#         engine.setProperty('voice', voice.id)
#         break
# No error handling yet.,
with open('proverbs_en.txt','r') as pfile:
    proverbs = pfile.readlines();
    total_proverbs = len(proverbs)
    i = 0;
    while i < total_proverbs:
        print(proverbs[i]); # Print on console line in gujrati
        # engine.say(proverbs[i+1]); # say what is writeen in line
below in english
        engine.say(proverbs[i]); # say what is writeen in line below
in english

        engine.runAndWait()
        engine.stop();
        i = i + 1; # +2 if want to skip "Tweak around"
```

```
# ./04_chatgpt.py
#----------------------------------------
# https://www.geeksforgeeks.org/how-to-use-chatgpt-api-in-python/
# https://medium.com/geekculture/a-simple-guide-to-chatgpt-api-with-python-c147985ae28
import openai

openai.api_key = "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";

# messages = [
#  {"role": "system", "content" : "You are a kind helpful assistant"}
# ]

# content = input("User: ")
# messages.append({"role": "user", "content": content})

# completion = openai.ChatCompletion.create(
#   model="gpt-3.5-turbo",
#   messages=messages
# )

# chat_response = completion.choices[0].message.content;
# print("ChatGPT: {0}".format(chat_response));

def ask_chatgpt(prompt, model="gpt-3.5-turbo"):
    messages =     [ {"role": "system", "content" : "You are a kind helpful assistant"}]
    messages.append({"role": "user", "content": prompt})

    # messages = [{"role": "user", "content": prompt}]
    response = openai.ChatCompletion.create(
                                                model=model,
                                                messages=messages);

    return response.choices[0].message.content;

query = 'Link ot dynamic programming tutorial., ';
print(f'Asking chat gpt -> \'{query}\'');
answer = ask_chatgpt(query);
print(f'ChatGPT query is {query}\nRespons is\n{answer}');
```

```python
# ./05_Class_OperatorOverloading.py
#----------------------------------------
import time;


class MyData():
    def __init__(self, x = 0, y = 0, z = 0):
        self.x = x;
        self.y = y;
        self.z = z;

    def __del__(self):
        self.x = 0;
        self.y = 0;
        self.z = 0;

    # We can call print(given object) now to print
    def __str__(self):
        return (f">> x = {self.x}, y = {self.y}, z = {self.z}");

    # ######## Arithmatic Operator ########
    def __add__(self, other):
        res = MyData();
        res.x = self.x + other.x;
        res.y = self.y + other.y;
        res.z = self.z + other.z;
        return res;

    def __sub__(self, other):
        res = MyData();
        res.x = self.x + other.x;
        res.y = self.y + other.y;
        res.z = self.z + other.z;
        return res;

    def __mul__(self, other):
        res = MyData();
        res.x = self.x * other.x;
        res.y = self.y * other.y;
        res.z = self.z * other.z;
        return res;
```

```
    def __truediv__(self, other):
        res = MyData();
        res.x = self.x / other.x;
        res.y = self.y / other.y;
        res.z = self.z / other.z;
        return res;

    def __mod__(self, other):
        res = MyData();
        res.x = self.x % other.x;
        res.y = self.y % other.y;
        res.z = self.z % other.z;
        return res;

    def __pow__(self, other):
        res = MyData();
        res.x = self.x ** other.x;
        res.y = self.y ** other.y;
        res.z = self.z ** other.z;
        return res;
    # // floor division
    def __floordiv__(self, other):
        res = MyData();
        res.x = self.x // other.x;
        res.y = self.y // other.y;
        res.z = self.z // other.z;
        return res;

    # ######## Comparision Operator ########

    # <
    def __lt__(self, other):
        res = False;
        res =         self.x < other.x;
        res = res and self.y < other.y;
        res = res and self.z < other.z;
        return res;

    # >
    def __gt__(self, other):
        res = False;
```

```
        res =         self.x > other.x;
        res = res and self.y > other.y;
        res = res and self.z > other.z;
        return res;

    # <=
    def __le__(self, other):
        res = False;
        res =         self.x <= other.x;
        res = res and self.y <= other.y;
        res = res and self.z <= other.z;
        return res;

    # >=
    def __ge__(self, other):
        res = False;
        res =         self.x >= other.x;
        res = res and self.y >= other.y;
        res = res and self.z >= other.z;
        return res;

    # ==
    def __eq__(self, other):
        res = False;
        res =         self.x == other.x;
        res = res and self.y == other.y;
        res = res and self.z == other.z;
        return res;

    # !=
    def __ne__(self, other):
        res = False;
        res =         self.x != other.x;
        res = res and self.y != other.y;
        res = res and self.z != other.z;
        return res;

    # ######## Compound Assingment Operator ########

    # -=
    def __isub__(self, other):
        self.x = self.x - other.x;
```

```
        self.y = self.y - other.y;
        self.z = self.z - other.z;
        return self;

    # +=
    def __iadd__(self, other):
        self.x = self.x + other.x;
        self.y = self.y + other.y;
        self.z = self.z + other.z;
        return self;
    # *=
    def __imul__(self, other):
        self.x = self.x * other.x;
        self.y = self.y * other.y;
        self.z = self.z * other.z;
        return self;

    # /=
    def __idiv__(self, other):
        self.x = self.x / other.x;
        self.y = self.y / other.y;
        self.z = self.z / other.z;
        return self;

    # //=
    def __ifloordiv__(self, other):
        self.x = self.x // other.x;
        self.y = self.y // other.y;
        self.z = self.z // other.z;
        return self;

    # %=
    def __imod__(self, other):
        self.x = self.x % other.x;
        self.y = self.y % other.y;
        self.z = self.z % other.z;
        return self;

    # **=
    def __ipow__(self, other):
        self.x = self.x ** other.x;
        self.y = self.y ** other.y;
```

```
        self.z = self.z ** other.z;
        return self;

# Python does not support funtion overloading.
# Function name should be unique. If not, latest will take
precedence.

# class thisorthat():
#     def myname():
#         print("My name is good");
#     def myname():
#         print("My name is better");

#     def socool(self):
#         print("So cool");
#     def socool(self):
#         print("Not so cool")

# thisorthat.myname();

# t = thisorthat();
# t.socool();


a = MyData(1,1,1);
b = MyData(2,2,2);

print(a);
print(b);

c = a + b;
print(c);

d = b * c;
print(d);

e = MyData(b.x, b.y, b.z);
print('_'*20);
print(e);
print(d);
e *= d;
print(e);
```

```
print('+'*20);
print(d);
print(e);
e /= d;
print(e);


print("=="*40);
a = MyData(1,2,3);
b = MyData(1,2,6);

if( a == b):
    print("Both are equal");
else:
    print("Both are differing");


# TODO: Excercise.,
# Create example for all overloaded operators.# ./dataproc.py
#----------------------------------------
# print ("Hello World");

# f = open("testfile.txt", 'a')

# f.write('_' * 20)
# f.write("\nHello files\n")

# f.close();


# f = open('testfile.txt', 'r')
# data = f.read();
# f.seek(0, 0)
# # print (data);


# d = f.readlines();
# # print(type(d))
# # print(d);
# l = [];

# for li in d:
```

```
#     if not li.startswith('_'):
#         l.append(li);

# # print(type(l))
# # print(d)

# for el in l:
#     print(el)
# # print(l);
# f.close();

# f = open('testfile.txt', 'wb+')

# d = b'\x0A\x0B\x0C';
# f.write(d)

# f.seek(0, 0);

# data = f.read();

# a = data[0];
# b = data[1];
# c = data[2];
#
# print("a = {}, b = {}, c = {}".format(str(int(a)), str(int(b))
,str(int(c))));

# f.close();
# ./failingporc.py
#----------------------------------------
import time;

print("I am failing process");
time.sleep(1);
raise ("Exception :-)");

# ./fileio.py
#----------------------------------------

# message = 'Hello World 2';

# msgfile = open('message.txt', 'a');
```

```
# print(type(msgfile));

# msgfile.write(message);

# for i  in range(5):
#    msgfile.write('Msg id ' + str(i));
#    msgfile.write('\n');


# msgfile.close();


# msgfile = open('message.txt', "r")
# print(type(msgfile))

# readmsg = msgfile.read();

# lines = msgfile.readlines();
# print(type(lines));
# print(lines);
# for l in lines:
#       print('l items ' + l);
# # print ('Read from file -> ' + readmsg);

# msgfile.close();

# ########################## Different mode
# filename = 'appendfile.txt'
# f = open(filename,'a');
# f.write('some message\n');
# f.close()
# # what if I want to appned something., ?

# f = open(filename, 'a');
# debuglogs = ["OK\n", "OK\n", "NOT OK\n", "OK\n"];
# f.writelines(debuglogs);
# f.close();

# # r, w, a, r+, w+, a+, rb, wb, ab, rb+, wb+, ab+
# ########################## Binary
# f = open('dhruvit.txt', 'wb+');
# # bindata = b'\x65, \x41';
```

```
# # f.write(bindata);

# a = input('enter some value: ');
# print(type(a))
# a = int(a).to_bytes(1, 'big')
# print(type(a))
# f.write(a);

# # seek (offset, from where 0, 1, 2)
# f.seek(0,0);

# a = f.read(1);
# print(type(a))
# print(a);

# f.close();

"""
int i = 0xFFEE2341;
Big endian

&i -> 0xFF
& (i+1) -> 0xEE
& (i+2) -> 0x23
& (i+3) -> 0x41

Little endian

&i -> 0x41
& (i+1) -> 0x23
& (i+2) -> 0xEE
& (i+3) -> 0xFF

"""
# ########################## 'with' keyword

# """
# a) Close the file once it's usage is over to free system resources
# b) If we don't, it will be closed by GC later on.,
# c) If we use 'with' keyboard while opening the file, the file
#    gets closed as soon as it's usage is over.
# d) 'with' ensures that the file is closed even if an exception
```

```
#    occurs while processing it.
# """

# with open ('message.txt','r') as testfile:
#    data = testfile.readlines();
#    for d in data:
#       print("Data read from file : ", d, end="");

# print(type(testfile));

# # Moving withing file: seek function
# """
# Whenever we read file, internal file pointer move the the next
byte.,
# eg., if we read 10 bytes, next read will fetch from 11th bytes
onwards.,

# <file handle>.seek(offset, reference from where)
# here: offset is integer number in bytes to 'seek'
# reference from where: 0: Beginning of the file
#                       1: From current position
#                       2: From end of the file

# Note: We can move either from beginning(0) or end(2) in file
opened in 'text' mode
#       We can move from any position., 0, 1, or 2
# """
# # Example 1
# # seeking = open('seeking.txt', 'wb+');
# # data = b'\x45\x46\x47\x78\x2F\x20\x6A\x39';
# # seeking.write(data);
# # # seeking.close();

# # seeking.seek(0, 0);

# # dat = seeking.read();
# # print(dat);

# # seeking.close();

# # Example 2
# seeking = open('seeking.txt', 'w+');
```

```
# data = 'Hello Wolrd'
# # print(type(data))
# seeking.write(data);
# # seeking.close();

# seeking.seek(0, 0);
# seeking.seek(2, 2)

# dat = seeking.read();
# print(dat);

# seeking.close();



# ################## Serialization and deserialization
# # Example 1: Simple data serialization and deserialization
# f = open('hello.txt','w+');
# f.write(str(233) + '\n');
# f.write(str(3.14) + '\n');

# f.seek(0,0);
# a = int(f.readline());
# b = float(f.readline());

# print(a);
# print(b);

# # Exmple 2: Complex data
# """
# json module converts Python data into appropriate JSON types
before
# writeing data to a file -> Serialization.
# It also converts JSON dta types read from file into Python data. -
> Deserialization
# """
# #  Example 1
# import json;

# datafile = open ('london.bin', 'w+');
# newelem = [45, 46, 47]
# le = ['Hello', 45, 'World', 3.14, newelem]
```

```
# json.dump(le, datafile);

# datafile.seek(0,0);
# val = json.load(datafile);

# print(type(val));
# print(le);
# print(val);
# datafile.close();

# # Example 2
# import json
# f = open('mydata.bin', 'w+');
# userdata = { 'Anil' : 24, 'Ajay' : 29, 'Raj' : 30};
# print(userdata);

# json.dump(userdata, f);

# f.seek(0,0);
# readback = json.load(f);
# print('>>>>>>' + str(type(readback)));

# print(userdata);
# print(type(userdata['Anil']));
# print(readback);
# print(type(readback['Anil']));
# f.close();

# # Example 3: Into string instead of file
# import json;

# one = [1, 2, 3, 4, 5, 6];
# two = ['one', 'two', 'three', 'four'];

# os = json.dumps(one);
# print(one);
# print(os);

# print(type(one));
# print(type(os));

# ts = json.dumps(two);
```

```
# print(two);
# print(ts);

# print(type(two));
# print(type(ts));

# res = json.loads(os);
# print(type(os));
# print(type(res));
# print(res);
# print(os);

# ./ganttchart.py
#----------------------------------------
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import datetime
import gantt

# Change font default
gantt.define_font_attributes(fill='black', stroke='black',
stroke_width=0, font_family="Verdana")

# Add vacations for everyone
gantt.add_vacations(datetime.date(2014, 12, 25))
gantt.add_vacations(datetime.date(2015, 1, 1))
gantt.add_vacations(datetime.date(2015, 1, 13))

# Create two resources
rANO = gantt.Resource('ANO')
rJLS = gantt.Resource('JLS')

# Add vacations for one lucky resource
rANO.add_vacations(
    dfrom=datetime.date(2014, 12, 29),
    dto=datetime.date(2015, 1, 4)
    )
rANO.add_vacations(
    dfrom=datetime.date(2015, 1, 6),
    dto=datetime.date(2015, 1, 8)
    )
```

```
# Test if this resource is  avalaible for some dates
print(rANO.is_available(datetime.date(2015, 1, 5)))
print(rANO.is_available(datetime.date(2015, 1, 8)))
print(rANO.is_available(datetime.date(2015, 1, 6)))
print(rANO.is_available(datetime.date(2015, 1, 2)))
print(rANO.is_available(datetime.date(2015, 1, 1)))


# Create some tasks
t1 = gantt.Task(name='tache1', start=datetime.date(2014, 12, 25),
duration=4, percent_done=44, resources=[rANO], color="#FF8080")
t2 = gantt.Task(name='tache2', start=datetime.date(2014, 12, 28),
duration=6, resources=[rJLS])
t7 = gantt.Task(name='tache7', start=datetime.date(2014, 12, 28),
duration=5, percent_done=50)
t3 = gantt.Task(name='tache3', start=datetime.date(2014, 12, 25),
duration=4, depends_of=[t1, t7, t2], resources=[rJLS])
t4 = gantt.Task(name='tache4', start=datetime.date(2015, 1, 1),
duration=4, depends_of=t1, resources=[rJLS])
t5 = gantt.Task(name='tache5', start=datetime.date(2014, 12, 23),
duration=3)
t6 = gantt.Task(name='tache6', start=datetime.date(2014, 12, 25),
duration=4, depends_of=t7, resources=[rANO])
t8 = gantt.Task(name='tache8', start=datetime.date(2014, 12, 25),
duration=4, depends_of=t7, resources=[rANO, rJLS])


# Create a project
p1 = gantt.Project(name='Projet 1')

# Add tasks to this project
p1.add_task(t1)
p1.add_task(t7)
p1.add_task(t2)
p1.add_task(t3)
p1.add_task(t5)
p1.add_task(t8)



# Create another project
p2 = gantt.Project(name='Projet 2', color='#FFFF40')
```

```
# Add tasks to this project
p2.add_task(t2)
p2.add_task(t4)


# Create another project
p = gantt.Project(name='Gantt')
# wich contains the first two projects
# and a single task
p.add_task(p1)
p.add_task(p2)
p.add_task(t6)


# Test cases for milestones
# Create another project
ptcm = gantt.Project(name='Test case for milestones')

tcm11 = gantt.Task(name='tcm11', start=datetime.date(2014, 12, 25),
duration=4)
tcm12 = gantt.Task(name='tcm12', start=datetime.date(2014, 12, 26),
duration=5)
ms1 = gantt.Milestone(name=' ', depends_of=[tcm11, tcm12])
tcm21 = gantt.Task(name='tcm21', start=datetime.date(2014, 12, 30),
duration=4, depends_of=[ms1])
tcm22 = gantt.Task(name='tcm22', start=datetime.date(2014, 12, 30),
duration=6, depends_of=[ms1])
ms2 = gantt.Milestone(name='MS2', depends_of=[ms1, tcm21, tcm22])
tcm31 = gantt.Task(name='tcm31', start=datetime.date(2014, 12, 30),
duration=6, depends_of=[ms2])
ms3 = gantt.Milestone(name='MS3', depends_of=[ms1])


ptcm.add_task(tcm11)
ptcm.add_task(tcm12)
ptcm.add_task(ms1)
ptcm.add_task(tcm21)
ptcm.add_task(tcm22)
ptcm.add_task(ms2)
ptcm.add_task(tcm31)
ptcm.add_task(ms3)
```

```
p.add_task(ptcm)


##########################$ MAKE DRAW ###############
p.make_svg_for_tasks(filename='test_full.svg',
today=datetime.date(2014, 12, 31), start=datetime.date(2014,8, 22),
end=datetime.date(2015, 1, 14))
p.make_svg_for_tasks(filename='test_full2.svg',
today=datetime.date(2014, 12, 31))
p.make_svg_for_tasks(filename='test.svg', today=datetime.date(2014,
12, 31), start=datetime.date(2015, 1, 3), end=datetime.date(2015, 1,
6))
p1.make_svg_for_tasks(filename='test_p1.svg',
today=datetime.date(2014, 12, 31))
p2.make_svg_for_tasks(filename='test_p2.svg',
today=datetime.date(2014, 12, 31))
p.make_svg_for_resources(filename='test_resources.svg',
today=datetime.date(2014, 12, 31), resources=[rANO, rJLS])
p.make_svg_for_tasks(filename='test_weekly.svg',
today=datetime.date(2014, 12, 31),
scale=gantt.DRAW_WITH_WEEKLY_SCALE)
##########################$ /MAKE DRAW ################
./intersubprocesscomm.py
#----------------------------------------
# Course Ref: https://realpython.com/python-subprocess/#processes-
and-subprocesses

# magic_number.py
#from random import randint
#print(randint(0, 1000))
import subprocess


"""
Passing a capture_output argument of True to run()
makes the output of the process available at the .stdout
attribute of the completed process object.
You'll note that it's returned as a bytes object,
so you need to be mindful of encodings when reading it.
"""
```

```
res = subprocess.run( ["python", "magic_number.py"],
capture_output=True);
print("Subprocess returned : {0}".format(int(res.stdout)));


f = open('resultfile.txt', 'w+');
res = subprocess.run( ["python", "magic_number.py"], stdout=f);
f.seek(0,0);
print("Subprocess returned : {0}".format(f.read()));
f.close();

# ./magic_number.py
#----------------------------------------
from random import randint
print(randint(0, 1000))
# ./mapfilterreduce.py
#----------------------------------------
# l = [1, 2, 3, 4, 5, 6]
# nl = [];
# for i in range(len(l)):
#     nl.append (l[i] * 3);

# print(l);
# print (nl);

# def triple(a):
#     return a * 2;

# nl2 = map(triple, l);
# print(list(nl2))

# nl3 = map( lambda a: a * 3, l);
# print(list(nl3))

# """
# Map takes each element, supply to given function
# Create a new list with the result of funciton call with individual
elements.
# """


# ############# Filter
```

```
# s = list('Hello Wolrd');

# def isConsonant(c):
#     if c in list('aeiouAEIOU'):
#         return False;
#     else:
#         return True;

# ns = filter(isConsonant, s);

# print(s);
# print(list(ns));
# print(type(ns));


# a = [10, 1092, 192, 29, 38, 76, 45, 82, 200, 620, 720]

# b = filter(lambda x: x < 200, a);
# print(a);
# print(list(b));


"""
Reduce:
apply the equation,
use result as another input to the equation
"""
from functools import reduce;


l = [1, 2, 3, 4, 5, 6]

def add(x, y):
    return x + y;

def multiply(x, y):
    return x * y;

ll = reduce(add, l);
print(l);
print(ll);
```

```
p = reduce(multiply, l);
print(p);

import math;

# y = (lambda p : p * 3)(9);
# print(y);

i = [1, 2, 3, 4, 5]
q = reduce( lambda x, y : x + y, i);
print(q);




# y = x1 + x2 + x3 + ... + xn;
# y = x1 * x2 * x3 * ... * xn;

# ./practicalthread.py
#----------------------------------------
import threading
import requests

def download_file(url, filename):
    response = requests.get(url)
    with open(filename, 'wb') as file:
        file.write(response.content)
    print(f"Downloaded {filename}")

# URLs and corresponding filenames
file_urls = [
    ("https://example.com/file1.txt", "file1.txt"),
    ("https://example.com/file2.txt", "file2.txt"),
    ("https://example.com/file3.txt", "file3.txt")
]

# Create a list to hold the threads
threads = []

# Create and start a thread for each file download
for url, filename in file_urls:
```

```
    thread = threading.Thread(target=download_file, args=(url,
filename))
    thread.start()
    threads.append(thread)

# Wait for all threads to complete
for thread in threads:
    thread.join()
# ./proc.py
#----------------------------------------
# To be called form sub process
import time;

print("Hello Python");

delay = 1
time.sleep(delay);

# raise Exception("Something seriously failed");


# ./thr.py
#----------------------------------------
import threading;
import time;
from gantt import *;


# Function to be executed by a thread
def thread_function(name):
    print("Thread", name, "started")

    # Simulate some work
    for i in range(10):
        print("Thread", name, "working...")
        time.sleep(0.5)

    print("Thread", name, "finished")

# Create threads
thread1 = threading.Thread(target=thread_function, args=("Thread
1",))
```

```
thread2 = threading.Thread(target=thread_function, args=("Thread
2",))

# Start threads
thread1.start()
thread2.start()

# Wait for threads to finish
thread1.join()
thread2.join()
```