Cheat sheet - Advanced geomatics

```python
# Lists
mylist = ["merano", "Bolzano", "Trento"]
print(mylist)

# Access elements in certain positions
print("The elemnets start at position 0:", mylist[0])

# Add objects
mylist.append("Potsdam")
print(mylist)

# Remove, the string must be exactly the same
mylist.remove("Potsdam")
print(mylist)

# Remove items by position in list
mylist.pop(0) #remove first item
print(mylist)

# Show the popped item

# Item contained , NB this is case sensitive, so bolzano is different
form Bolzano
mylist = ["merano", "Bolzano", "Trento"]
doIHaveBolzano = "Bolzano" in mylist
print(doIHaveBolzano)

doIHavePotsdam = "Potsdam" in mylist
print(doIHavePotsdam)

# Looping, prints each item
for item in mylist:
    print(item)

# Put together items in the same position
colours = ["red", "green", "blue", "purple"]
ratios = [0.2, 0.3, 0.1, 0.4]
# Range function, range 10 goes from 0 to 9!
for index in range(len(colours)):#from 0 to length of list color
    colour = colours[index]
    ratio = ratios[index]

    print(f"{colour}->{ratio}")

# Have an exact copy of a list
colours2 = colours.copy()
colours.sort()
print(colours)
print(colours2)
```

# Break a loop when I is a certain value. "break" breaks the loop. E.g. if you loop through data and look for a particular item, you break when you find it

```
for i in range(10):
    if i == 5:
        break
    print(f"A){i}")
```

# Continue

```
for i in range(10):
    if i == 5: #whenever the condition is met, this is skipped and the function continues
        continue
    print(f"B){i}")
```

# Loop over a sequence of numbers by increasing the value by 2

```
for i in range(0, 10, 2):
    print(f"C){i}")
```

# The inverse, from 10 to 0

```
for i in range(10, 0, -2):
    print(f"D){i}")
#The range object does not allow to use floats
```

# Sorting lists

```
mylist = ["Merano", "Bolzano", "Trento"]
print(f"My original list:{mylist}")
mylist.sort()
print(f"My sorted list:{mylist}")
mylist.sort(reverse = True)
print(f"My rev-sorted list:{mylist}")

mylist = ["banana", "Orange", "Kiwi", "cherry"]
mylist.sort()
print(f"A mixed case list, sorted: {mylist}") #sorted according to ASCII, so before the upper and then the lower cases

mylist.sort(key = str.lower) # use as function the lower case of the string object
print(f"A mixed case list, properly sorted: {mylist}")
```

# Numerical sorting

```
numlist = ["002", "01", "3", "004"]
numlist.sort()
print(numlist)

numlist = ["002", "01", "3", "004"]
```

# Define a function, "string" is the name of the varibale. We convert the string into integer. Then we can use the function as a key.

```
def toInt(string):
    return int(string)
```

```
numlist.sort(key = toInt)
print(numlist)

abc = ["a", "b", "c"]
cde = ["c", "d", "e"]

newabcde = abc + cde
print(newabcde)

# Remove brackets, indicating the separator I want to use. Do not use
comma!
print(";".join(newabcde))

numlist = [1.0, 2.0, 3.5, 6, 11, 34, 12]
print(max(numlist))
print(min(numlist))
print(sum(numlist))

avg = sum(numlist)/len(numlist)
print(avg)

# Caluclate average using for loop
somma = 0
count = 0
for item in numlist:
    somma += item #This is like writing: somma = somma + item
    count += 1 #This is like writing: count = count + 1
average = somma/count
print(average)

# Calculate the variance
```

Write and Read text files

# Writing textfiles

```
filePath = r"C:\Users\Michele\OneDrive - Scientific Network South Tyrol\EMMA\Year 1\Advanced geomatics\data.txt"
data = """# stationid, datetime, temperature
1, 2023-01-01 00:00, 12.3

2, 2023-01-01 00:00, 11.3
3, 2023-01-01 00:00, 10.3"""
with open(filePath,"w") as file: #w as write = open a file in this path i writing mode
    file.write(data)
```

# Add a new line, open the file in append mode

```
with open(filePath,"a") as file:
    file.write("\n1, 2023-01-02 00:00, 9.3") # add \n to have a new line, otherwise it will add just a new single line without spacing
    file.write("\n2, 2023-01-02 00:00, 8.3")
```

# Read file

```
with open(filePath,"r")as file:
    lines = file.readlines()
print(lines)
```

# If this is a huge file, count how many stations, using dictionaries.

A code that considers that some stations appear twice

```
stationsCount = {}
for line in lines:
    line = line.strip()
    #if line is # or is empty
    if line.startswith("#") or len(line) == 0:
        continue
    lineSplit = line.split(",")
    #print(lineSplit)
    stationId = lineSplit[0]
    #print(stationId)

    counter = stationsCount.get(stationId, 0)
    counter += 1 # if station is already inserted the counter increases to 2
    #put it in the dictionary
    stationsCount[stationId] = counter #For station id 1 put 1

    print(stationsCount)
```

Dictionaries

```
# Map = dictionary
# A key needs to be unique
townsProvincesMap = {
    "merano": "BZ",
    "bolzano": "BZ",
    "trento":"TN"
}
print(townsProvincesMap)
print(townsProvincesMap["merano"]) #NB Keys are case sensitive, so Merano is different from merano

# Add element
townsProvincesMap ["potsdam"] = "BR"
print(townsProvincesMap)

# Remove element
townsProvincesMap.pop("potsdam")
print(townsProvincesMap)

# Chceck if key is available
if townsProvincesMap.get("Merano") is None:
    print("key doesn't exist")
else:
    print("key exists")

# We set a efault value in case our key is not present
print(townsProvincesMap.get("Merano", "unknown"))

# for town, province in townsProvincesMap
for key, value in townsProvincesMap.items():
    print(key, "is in the province of", value)

print(townsProvincesMap.keys())
print(townsProvincesMap.values())

# Sort dictionaries by keys
#1 transform in list
keys = list(townsProvincesMap.keys())
keys.sort()
print(keys)

# loop over a dictionary, to sort by key
for key in keys:
    print(key, "is in the province of",townsProvincesMap[key])
```

Exercise rain data

```
# Monthly sum of the rain data
dataPath = r"C:\Users\Michele\OneDrive - Scientific Network South Tyrol\EMMA\Year 1\Advanced geomatics\01_exe_rain_data_1year.txt"


# Read data into a lines list
with open(dataPath,"r")as file:
    lines = file.readlines()

# Print the first 5 lines
date2ValuesListMap = {}
for line in lines[:5]:
    line = line.strip()#remove spaces
    # clean lines with # or empty
    if line.startswith("#") or len(line) == 0:
        continue

    # parse each line to extract the date(string) and value(num)
    lineSplit = line.split(",")
    date = lineSplit[0]
    value = float(lineSplit[1])
    #print(date, ":", value)

    # extract the year-month from the date. The key mus be unique
    month = date[:-2]
    #print(month, ":", value)

    # aggregate the values by month (key), i.e. colllect all values
    # for each date in a list
    values = date2ValuesListMap.get(month, [])
    values.append(value)
    date2ValuesListMap[month] = values

    #print(date2ValuesListMap)

for month, values in date2ValuesListMap.items():
    print(month, values)#each year-month has its rain amount
    cumRain = sum(values)
    print(f"Cumulated rain for month {month} is {cumRain}")

# after this I can remove the limit of 5 or 50 data and get all the data. If I get negative data for rain, it is probably an error (negative rain is impossible!) so I would need to filter these out
```

Introduction exercises

```
folder = r"C:\Users\Michele\OneDrive - Scientific Network South
Tyrol\EMMA\Year 1\Advanced geomatics\LUISA_MENESTRINA"

#Exercise 1
age = 25
name = "Mario Rossi"
activity = "skating"
job = "engineer"
print(f"Hei, I am {name}\n I am {age} and I love to go {activity}\n
I work as an {job}")
print("---------------")

#Exercise 2
csvPath = f"{folder}/01_exe2_data.csv"

with open(csvPath, "r") as file:
    lines = file.readlines()
for line in lines:
    print(line)

for line in lines:
    line = line.strip()
    lineSplit = line.split(";")
    print(lineSplit)

    analogString = lineSplit[0]
    analogSplit = analogString.split(":")
    x1 = float(analogSplit[1])
    print(x1)

    maxvoltageString = lineSplit[1]
    y2 = float(maxvoltageString[11:])

    maxanalogString = lineSplit[2]
    x2 = float(maxanalogString.split(":")[1])

    # x2/x1 = y2/y1
    y1 = y2*x1/x2

    print(x1,x2,y1,y2)
print("---------------")

#Exercise 3
string = "a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s"
modified_string = string.replace(',', ';')
print(modified_string)
print("---------------")

#Exercise 4
list = [ 1, 2, 3, 4, 5]
for number in list:
    print(number)
print("---------------")

#Exercise 5
for number in list:
    print(f"Number {number}")
print("---------------")

#Exercise 6
list = [ 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 ]
print(list[:5])
shortList = list[:5]
for item in shortList:
    print(f"Number {item}")
print("---------------")

#Exercise 7
list1 = [1, 2, 3, 4, 5]
list2 = ["first","second","third","fourth","fifth"]
for item in range(len(list1)):
    print(f"{list2[item]} is {list1[item]}")
print("---------------")

#Exercise 8
string = """Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris
nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in
reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla
pariatur. Excepteur sint occaecat cupidatat non proident, sunt in
culpa qui officia deserunt mollit anim id est laborum."""

#Count characters
charactersCount= len(string)
print(charactersCount)

#Count words
wordCount = len(string.split())
print(wordCount)

#Count characters without spaces
wordWithoutSpace = len(string.replace(" ","").replace("\n",""))
print(wordWithoutSpace)
print("---------------")


# Exercise 9
ReadExe9 = f"{folder}/01_exe9_data.csv"

with open(ReadExe9,"r")as file:
  lines = file.readlines()
  for line in lines:
    line = line.strip()
    if line.startswith("#") or len(line) == 0:
        continue
    print(line)
print("---------------")

# Exercise 10
ReadExe9 = f"{folder}/01_exe9_data.csv"

with open(ReadExe9,"r")as file:
  lines = file.readlines()
  for line in lines:
    line = line.strip()
    if line.startswith("#") or len(line) == 0:
        continue
    columns = line.split(",")
    column1 = columns[0]
    column2 = columns[1]
    column2 = float(columns[1])
    if column2 <= 1000:
        print(f"{column1}, {column2}")

# Exercise 11
ReadExe11 = f"{folder}/01_exe11_data.csv"

with open(ReadExe11,"r")as file:
  lines = file.readlines()
  for line in lines:
      line = line.strip()
      lineSplit = line.split(";")
      # print(lineSplit)
      base = lineSplit[0].split("=")[1].replace("cm", "")
      base1 = float(base)
      height = lineSplit[1].split("=")[1]
      height1 = float(height)
      # print(base)
      # print(height)
      area = (base1 * height1*100) / 2
      print(f"base * height / 2 = {base1} * {height1*100} =
{area}cm2")
print("---------------")


# Exercise 12
who = {
    "Daisy": 11,
    "Joe": 201,
    "Will": 23,
    "Hanna": 44
}
what = {
    44: "runs",
    11: "dreams",
    201: "plays",
    23: "walks"
}
where = {
    44: "to town.",
    11: "in her bed.",
    201: "in the livingroom.",
    23: "up the mountain."
}
for key, value in who.items():
    activity = what[value]
    place = where[value]
    print(f"{key} {activity} {place}")

for key, value in who.items():
    activity = what.get(value)
    place = where.get(value)
    print(f"{key} {activity} {place}")
print("---------------")

# Exercise 12
who = {
    "Daisy": 11,
    "Joe": 201,
    "Will": 23,
    "Hanna": 44
}
what = {
    44: "runs",
    11: "dreams",
    201: "plays",
    23: "walks"
}
where = {
    "runs" : "to town.",
    "dreams" : "in her bed.",
    "plays" : "in the livingroom.",
    "walks": "up the mountain."
}
for key, value in who.items():
    activity = what[value]
    place = where[activity]
    print(f"{key} {activity} {place}")

for key, value in who.items():
    activity = what.get(value)
    place = where.get(activity)
    print(f"{key} {activity} {place}")
print("---------------")

# Exercise 13
list1 = ["a","b","c","d","e","f"]
list2 = ["c","d","e","f","g","h","a"]
list3 = ["c","d","e","f","g"]
```

```
list4 = ["c","d","e","h","a"]

lists = list1 + list2 + list3 + list4
letterCount = {}

for letter in lists:
    if letter in letterCount:
        letterCount[letter] += 1
    else:
        letterCount[letter] = 1
for letter, count in letterCount.items():
    print(f"count of {letter} = {count}")
print("---------------")

# Exercise 14
stationsPath = f"{folder}/stations.txt"

with open(stationsPath,"r")as file:
 lines = file.readlines()

for line in lines[:20]:
    print(line)
print("---------------")

# Exercise 15
stationsPath = f"{folder}/stations.txt"

with open(stationsPath,"r")as file:
 lines = file.readlines()

count = 0
for line in lines:
    if line.startswith("#") or len(line) == 0:
     continue
    count += 1
print(count)
print("---------------")

# Exercise 16
stationsPath = f"{folder}/stations.txt"

with open(stationsPath,"r")as file:
 lines = file.readlines()
 columns = line.split(",") #lineSplit = line.split(",")
 columnsNumber = len(columns)
 print(columnsNumber)
print("---------------")

# Exercise 17
stationsPath = f"{folder}/stations.txt"

with open(stationsPath,"r")as file:
 lines = file.readlines()
```

```
for line in lines[:20]:
    columns = line.split(",") #lineSplit = line.split(",")
    print(columns[0], columns[1])
print("---------------")

# Exercise 18
stationsPath = f"{folder}/stations.txt"

with open(stationsPath,"r")as file:
 lines = file.readlines()

somma = 0
count = 0

for line in lines:
    line = line.strip()
    if line.startswith("#") or len(line) == 0:
        continue
    columns = line.split(",")
    height = float(columns[5])
    somma += height
    count += 1

if count > 0:
    avg = somma / count
    print(f"The averae of the height of the stations is {int(avg)}")
print("---------------")

# Exercise 19
print(f"File info: stations.txt\n----------------")
print(f"Stations count: {count}")
print(f"Average value: {int(avg)}")
with open(stationsPath, "r") as file:
    line = file.readline()
    lineStrip = line.strip()
    columnNames = line.split(",")
    # print(columnNames)

print(f"Available fields:")
for item in columnNames:
    print(f"-> {item.strip()}")

print(f"First data lines:")
for line in lines[:6]:
    print(line)

# Exercise 21
n = 10
m = 5

for item in range(n):
    code = "*" * m
    print(code)
print("---------------")

# Exercise 22
n = 10
count = 0
for item in range(n):
    count += 1
    code2 = count * "*"
    print(code2)
print("---------------")

# Exercise 23
n = 10
count = 10
for item in range(n):
    count -= 1
    code3 = count * "*"
    print(code3)
print("---------------")

# Exercise 24
a = 10
somma = 0
for number in range(0,a):
    if number%2 == 0:
        print (number)
        somma += number
print(f"The sum of even numbers from 0 to a is {somma}")
print("---------------")

# Alternative exercise 24
a = 10
somma = 0
for number in range(0,a,2):
        print (number)
        somma += number
print(f"The sum of even numbers from 0 to a is {somma}")

print("---------------")

# Exercise 25
numbers = [123, 345, 5, 3, 8, 87, 64, 95, 9, 10, 24, 54, 66]
sumEven = 0
for number in numbers:
    if number%2 == 0:
        sumEven += number
print(f"The sum of even numbers is {sumEven}")
print("---------------")

# Exercise 26
print(f"Exercise 26\n")
ReadExe261 = f"{folder}/01_exe26_dataset1.csv"
ReadExe262 = f"{folder}/01_exe26_dataset2.csv"
```

```
dict261 = {}
with open(ReadExe261, "r") as file:
    for line in file:
        line = line.strip()
        id, x, y = line.split(",")
        dict261[id] = {"x": x, "y": y}

dict262 = {}
with open(ReadExe262, "r") as file:
    for line in file:
        line = line.strip()
        id, value = line.split(",")
        dict262[id] = value

for key, value in dict261.items():
    if key in dict262:
        row = dict261[key]
        row["value"] = dict262[key]
        print(f"ID: {key}, X: {row['x']}, Y: {row['y']}, Value:
{row.get('value')}")
```

PY QGIS BASICS

```
from pyqgis_scripting_ext.core import *

CREATE A POINT
point = HPoint(30.0, 10.0)
print(point.asWkt())

# Create different types of geometry
# Longitude and latitude is the order to keep (as x and y, even though
usually is latitude, longitude).
coords = [[31, 11], [10, 30], [20,40], [40,40]]
line = HLineString.fromCoords(coords)
print(line.asWkt())

#Polygon; coordinates should close on the same starting coordinates
coords = [[32,12], [10,20], [20,39], [40,39], [32,12]]
polygon = HPolygon.fromCoords(coords)
print(polygon.asWkt())

# Create a polygon with a hole inside
exteriorPoints = [[35,10],[10,20],[15,40],[45,45],[35,10]]
holePoints = [[20,30],[35,35],[30,20],[20,30]]
polygonWithHole = HPolygon.fromCoords(exteriorPoints)

holeRing = HLineString.fromCoords(holePoints)
polygonWithHole.add_interior_ring(holeRing)
print(polygonWithHole) # the result will show a list of coordinates
separated by a comma

#Creating multi-geometries
```

```
coords = [[10, 40], [40, 30], [20, 20], [30, 10]]
multiPoints = HMultiPoint.fromCoords(coords)
print(multiPoints)

coords1 = [[10,10],[20,20],[10,40]]
coords2 = [[40,40],[30,30],[40,20],[30,10]]
multiLine = HMultiLineString.fromCoords([coords1, coords2])

# Multi-polygon
coords1 = [[30,20], [10,40], [45,40], [30,20]]
coords2 = [[15,5], [40,10], [10,20], [5,10], [15,5]]
multiPolygon = HMultiPolygon.fromCoords([coords1, coords2])

# Color the polygons with different colors
subGeometries = multiPolygon.geometries()
colorsList = ["red", "blue", "green"]

coordinates = polygon.coordinates()
for coord in coordinates:
    print(f"coord x = {coord[0]}/coord y = {coord[1]}")

# Take a string and convert into geometry
wkt = "POINT (156 404)"
pointGeom = HGeometry.fromWkt(wkt)
print(pointGeom)

wkt = """
MULTIPOLYGON (((130 510, 140 450, 200 480, 210 570, 150 630, 130 560,
130 510)),
((430 770, 370 820, 210 860, 20 760, 35 631, 100 370, 108 363, 154
284, 230 380,
140 400, 150 440, 130 450, 104 585, 410 670, 440 590, 450 590, 430
770)))
"""
polygonGeom = HGeometry.fromWkt(wkt)

CREATE A CANVAS
canvas = HMapCanvas.new()
for item in range(len(subGeometries)):
     geom = subGeometries[item]
     color = colorsList[item]
     canvas.add_geometry(geom, color, 2)
```

Visualize geometries

```
canvas = HMapCanvas.new()
canvas.add_geometry(point, "red", 2)
```

This does not show anything, we need to set it to an extent, which takes 4 coordinates → canvas set extent. The number represents the thickness.

- `canvas.add_geometry(multiPolygon, "magenta", 5)`
- `canvas.add_geometry(multiLine, "blue", 5)`
- `canvas.add_geometry(multiPoints, "red", 15)`
- `canvas.add_geometry(line, "blue", 2)`

- canvas.add_geometry(polygon, "green", 2)
- canvas.add_geometry(polygonWithHole, "magenta", 10)#
- canvas.set_extent([0, 0 , 50 ,50]) #x, y bottom left and x, y upper side right

```
canvas.show()
```

GEOMETRIES

```
from pyqgis_scripting_ext.core import *

#g1
coords = [[0, 0], [0, 5], [5,5], [5,0], [0,0]]
g1 = HPolygon.fromCoords(coords)
print(g1.asWkt())

# Bounding box
print("polygon boundingbox", g1.bbox())
```

- print("polygon length:", g1.length())
- print("polygon area:", g1.area())

```
# Distance from the nearest point between two geometries
print("distance between line and point:", g5.distance(g4))
```

PREDICATES → functions returning true or false

INTERSECTION

```
print(g1.intersects(g2)) # g2 is just touching, but it is considered anyways by the intersection
print(g1.intersects(g3))
```

TOUCHING - do not intersect their interior but have at least one point in common

```
print(g1.touches(g2))
```

CONTAINS

```
print(g1.contains(g2))
```

FUNCTIONS - functions return an extra geometry

INTERSECTION

```
print(g1.intersection(g2)) # touching polygons will give the line that they have in common
print(g1.intersection(g3)) # polygon and point will give the point itself
print(g1.intersection(g5)) # polygon and line → line
newGeom = g1.intersection(g6)
```

SYMDIFFERENCE

```
print(g1.intersection(g6))
newGeom = g1.symdifference(g6)
```

UNION

```
print(g1.union(g6))
newGeom = g1.union(g6)
```

```
DIFFERENCE
# - Mind the order when you use difference
print(g1.difference(g6))
newGeom = g1.difference(g6)

print(g6.difference(g1))
newGeom = g6.difference(g1)

BUFFERS
b1 = g3.buffer(1.0) # the buffer of a point
b2 = g3.buffer(1.0, 1) # the buffer of a point with few quandrant
segments, because a buffer point is a series of segments, if we reduce
the number of fragments we can se the segments. The default value
should be 8.
b3 = g5.buffer(1.0) # line buffer
b4 = g5.buffer(1.0, 2) # line buffer with few points

# square end cap style (flat, square, round)
b5 = g5.buffer(1.0, -1, JOINSTYLE_ROUND, ENDCAPSTYLE_SQUARE)

# CONVEX HULL, with many geometries to know the bounding box. The
"convex" will not mark the concavities of the shape, but creates a
shape that covers all shapes, however a convex one.

collection = HGeometryCollection([g1, g2, g3, g4, g5, g6])
hull = collection.convex_hull()

canvas = HMapCanvas.new()
canvas.add_geometry(g1, "black", 2)
canvas.add_geometry(hull, "orange", 2)

canvas.set_extent(hull.bbox())

# canvas.add_geometry(g2, "magenta", 2)

# canvas.add_geometry(b1, "orange", 3)

# show the intercection geometry/the dfference between geometries
canvas.add_geometry(newGeom, "magenta", 2)

canvas.set_extent([-1, -1 , 8 ,8]) #x, y bottom left and x, y upper
side right
canvas.show()
```

PROJECTIONS

```
from pyqgis_scripting_ext.core import *

crsHelper = HCrs()
crsHelper.from_srid(4326)
crsHelper.to_srid(32632)

point4326 = HPoint(11, 46)
point32632 = crsHelper.transform(point4326)

print(f"{point4326.asWkt()} -> {point32632.asWkt()}")

# transform backwards
backTo4326 = crsHelper.transform(point32632, inverse = True)
print(backTo4326.asWkt()) # instead of returing 46 it returns 45.999999 but it is ok
```

PYQGIS BASICS - GEOMETRIES

```
# Exercise 00
from pyqgis_scripting_ext.core import *

geomPath = r"C:\Users\Michele\OneDrive - Scientific Network South Tyrol\EMMA\Year 1\Advanced geomatics\02_exe0_geometries.csv"
with open (geomPath, "r") as file:
    lines = file.readlines()

mylines = []
mypolygons = []

for line in lines:
    line = line.strip()
    if line.startswith("#") or len(line) == 0:
        continue
    lineSplit = line.split(";")
    # print(lineSplit)

    shape = lineSplit[0]
    #print(shape)

    coordinates = lineSplit[1]
    # print(coordinates)

    # print(shape, ":", coordinates)
    if shape == "point":
        longitude = float(coordinates[0:4])
        latitude = float(coordinates[5:])
        point = HPoint(longitude,latitude)
        # print(point.asWkt())

    if shape == "line":
        lat_lon = coordinates.split(" ")
        for item in lat_lon:
            lat_lon1 = item.split(",")
            longitude = float(lat_lon1[0])
            latitude = float(lat_lon1[1])
            mylines.append([longitude,latitude])

        linea = HLineString.fromCoords(mylines)
        # print(linea.asWkt())

    if shape == "polygon":
        lat_lon = coordinates.split(" ")
        pointList = []
        for item in lat_lon:
            split = item.split(",")
            longitude = float(split[0])
            latitude = float(split[1])
            pointList.append((longitude, latitude))
        polygon = HPolygon.fromCoords(pointList)
        mypolygons.append(polygon)

print(mypolygons)

canvas = HMapCanvas.new()
canvas.add_geometry(point, "green", 15)
canvas.add_geometry(linea, "red", 2)
for polygon in mypolygons:
    canvas.add_geometry(polygon, "orange", 2)
canvas.set_extent([0, 0 , 50 ,50]) #x, y bottom left and x,y upper side right
canvas.show()
```

# Exercise 01

```
extent = 6

polygons = []

for lon in range(-180, 180, extent):
    minX = lon
    maxX = lon + extent
    minY = -84
    maxY = 84

    coords = [[minX, minY], [minX, maxY], [maxX, maxY], [maxX, minY], [minX, minY]]
    polygon = HPolygon.fromCoords(coords)
    polygons.append(polygon)

canvas = HMapCanvas.new()

# osm = HMap.get_osm_layer()
# canvas.set_layers([osm])
# #The problem is that the pojections are different from my drawing and the imported map. So we need to do a transformation.
```

```
for polygon in polygons:
    canvas.add_geometry(polygon)

canvas.set_extent([-180, -84, 180, 84])
canvas.show()

#Exercise02
from pyqgis_scripting_ext.core import *

stationsPath = r"C:\Users\Michele\OneDrive - Scientific Network South
Tyrol\EMMA\Year 1\Advanced geomatics\stations.txt"
with open (stationsPath, "r") as file:
    lines = file.readlines()

points = []
for line in lines[1:]:
    line = line.strip()
    lineSplit = line.split(",")
    if line.startswith("#") or len(line) == 0:
        continue
    # print(lineSplit)

    lat = lineSplit[3]
    lon = lineSplit[4]

    latSplit = lat.split(":")
    # print(latSplit)

    lat_deg = int(latSplit[0])
    lat_min = int(latSplit[1])/60
    lat_sec = int(latSplit[2])/3600

    newLat = float(lat_deg + lat_min + lat_sec)
    # print(newLat)

    lonSplit = lon.split(":")
    lon_deg = int(lonSplit[0])
    lon_min = int(lonSplit[1])/60
    lon_sec = int(lonSplit[2])/3600

    newLon = float(lon_deg + lon_min + lon_sec)
    # print(newLon)

    point = HPoint(newLon, newLat)
    points.append(point)
print(points[0])

crsHelper = HCrs()
crsHelper.from_srid(4326)
crsHelper.to_srid(3857)

Stations = []
```

```
for point in points:
    Point = crsHelper.transform(point)
    Stations.append(Point)

collection = HGeometryCollection(Stations)
hull = collection.convex_hull()

canvas = HMapCanvas.new()
for point in Stations:
    canvas.add_geometry(point, "red", 2)

osm = HMap.get_osm_layer()
canvas.set_layers([osm])
canvas.set_extent(hull.bbox())
canvas.show()

# Dictionary to store the count of stations per country
stations_by_country = {}

with open(stationsPath, "r") as file:
    for line in file:
        line = line.strip()
        if line.startswith("#") or len(line) == 0:
            continue

        lineSplit = line.split(",")
        country = lineSplit[2].strip()

        stations_by_country[country]                               =
stations_by_country.get(country, 0) + 1

for country, count in stations_by_country.items():
    print(f"{country}: {count}")
```

#Exercise03

```
from pyqgis_scripting_ext.core import *

# Necessary functions
def fromLatString(latString):
    sign = latString[0]
    latDegrees = float(latString[1:3]) # 3 excluded
    latMinutes = float(latString[4:6])
    latSeconds = float(latString[7:9])
    lat = latDegrees + latMinutes/60 + latSeconds/3600
    if sign == "-":
        lat = lat *-1
    return lat

def fromLonString(lonString):
    sign = lonString[0]
    lonDegrees = float(lonString[1:4])
    lonMinutes = float(lonString[5:7])
    lonSeconds = float(lonString[8:10])
    lon = lonDegrees + lonMinutes/60 + lonSeconds/3600
    if sign == "-":
        lon = lon *-1
    return lon

# Script start
lon = 11.34999
lat = 46.49809

stationsPath = r"C:\Users\Michele\OneDrive - Scientific Network South
Tyrol\EMMA\Year 1\Advanced geomatics\stations.txt"
centrePoint = HPoint(lon, lat)

with open (stationsPath, "r") as file:
    lines = file.readlines()

minDistance = 9999
nearestStationName = "none"
nearestDistancePoint = None

for line in lines[1:10]: #Skip the first line because are not numbers
so it will give error for float
    line = line.strip()

    lineSplit = line.split(",")
    name = lineSplit[1].strip()
    latString = lineSplit[3]
    lonString = lineSplit[4]

    latDec = fromLatString(latString)
    lonDec = fromLonString(lonString)
    # print(name, latDec, lonDec)
    point = HPoint(lonDec, latDec)

    distance = point.distance(centrePoint)
    if distance < minDistance:
        minDistance = distance
        nearestStationName = name
        nearestDistancePoint = point

print(nearestStationName, "->", nearestDistancePoint)
```

#Exercise 04

```
from pyqgis_scripting_ext.core import *

# Necessary functions
def fromLatString(latString):
    sign = latString[0]
    latDegrees = float(latString[1:3]) # 3 excluded
    latMinutes = float(latString[4:6])
    latSeconds = float(latString[7:9])
    lat = latDegrees + latMinutes/60 + latSeconds/3600
    if sign == "-":
        lat = lat *-1
    return lat

def fromLonString(lonString):
    sign = lonString[0]
    lonDegrees = float(lonString[1:4])
    lonMinutes = float(lonString[5:7])
    lonSeconds = float(lonString[8:10])
    lon = lonDegrees + lonMinutes/60 + lonSeconds/3600
    if sign == "-":
        lon = lon *-1
    return lon

# Script start
lon = 11.34999
lat = 46.49809
radiusKm = 20.0

stationsPath = r"C:\Users\Michele\OneDrive - Scientific Network South
Tyrol\EMMA\Year 1\Advanced geomatics\stations.txt"
centrePoint = HPoint(lon, lat)

with open (stationsPath, "r") as file:
    lines = file.readlines()

crsHelper = HCrs()
crsHelper.from_srid(4326)
crsHelper.to_srid(32632)

centerPoint32632 = crsHelper.transform(centrePoint)

buffer = centerPoint32632.buffer(radiusKm*1000)

for line in lines[1:10]:
    line = line.strip()

    lineSplit = line.split(",")
    name = lineSplit[1].strip()
    latString = lineSplit[3]
    lonString = lineSplit[4]

    latDec = fromLatString(latString)
    lonDec = fromLonString(lonString)

    point = HPoint(lonDec, latDec)
    point32632 = crsHelper.transform(point)

    if buffer.intersects(point32632):
        distance = point32632.distance(centerPoint32632)
        print(name, distance/1000, "Km", point)
```

PROCESSING VECTOR DATA

```
from pyqgis_scripting_ext.core import *

# Cleanup, to remove maps from the projects before
HMap.remove_layers_by_name(["OpenStreetMap"])

geopackagePath = r"C:\Users\Michele\OneDrive - Scientific Network
South Tyrol\EMMA\Year 1\Advanced
geomatics\packages\natural_earth_vector.gpkg"
countriesName = "ne_50m_admin_0_countries"

# Load open street map layer
osm = HMap.get_osm_layer()
HMap.add_layer(osm)

# Load the countires layer
countriesLayer = HVectorLayer.open(geopackagePath, countriesName)

print("Schema (first 4 fields):")
counter = 0
for name, type in countriesLayer.fields.items():
    counter += 1
    if counter < 5:
        print("\t", name,"of type", type)

crs = countriesLayer.prjcode
print("Projection: ", crs)

Extent
print("Spatial extent:", countriesLayer.bbox())
print("Feature count:", countriesLayer.size())

print("Attributes for Italy:")
nameIndex = countriesLayer.field_index("NAME") # This is case
sensitive, if i write "name" this will show an error as -1
countriesFeatures = countriesLayer.features()

Feature = geometric + attribute table part
for feature in countriesFeatures:
    name = feature.attributes[nameIndex]
    if name == "Italy":
        geometry = feature.geometry
        print("Geom:", geometry.asWkt()[:50] + "...")

# QGIS represents features as lists of each record. That is why we
use the index, this gives us a name and then we can access the element.

# FILTERING, filter all items starting with an I and with a population
higher than 3 million

Alphanumeric expression filter
expressions = "NAME like 'I%' AND POP_EST > 3000000"
filterCountriesFeatures = countriesLayer.features(expressions)
```

```
count = 0
for feature in filterCountriesFeatures:
    print(feature.attributes[nameIndex])
    count += 1
print("Feature count with filter", count)
```

Bounding box filtering

```
lon = 11.119982
lat = 46.080428
point = HPoint(lon, lat)
```

`buffer = point.buffer(2)` → 2 is the number of degrees, which depending on the latitude is a certain amount of km.

`citiesLayer = HVectorLayer.open(geopackagePath, citiesName)` → instead of citiesName put None to open a shapefile

Geometry filters

```
count = 0
for feature in citiesLayer.features(geometryfilter = buffer):
    print(feature.attributes[citiesNameIndex])
    count += 1
print("Cities features listed with geometry filter:", count)
```

STYLE

- MARKER
- FILL
- STROKE

`HFill("0, 255, 0, 128")`

The first tree numbers are RGB, so green has 255 (scale goes from 0 to 255), and 4th value is the transparency (0 - 255)*check on websites for color combinations. In alternative to this numbers you can write the color in letters.

`field = "if(POP_MAX>1000000, concat(NAME, ' ('POP_MAX,')'),NAME)"`

if the population is >1000000 write name and population, otherwise write the name

concat(a,b) → put together strings

Rivers layer, no field bounded to the nation, so we use a spatial filtering

italyGeometry = countriesLayer.features()[0].geometry, create a geometry for Italy

`riversLayer.sub_layer(ItalyGeometry, "rivers_Italy", ['scalerank', 'name'])`

This creates a new layer and returns a new object

HMap.add_layer(riversLayerItaly)→ I use the new layer not the riversName one

```
from pyqgis_scripting_ext.core import *
```

```
geopackagePath = r"C:\Users\Michele\OneDrive - Scientific Network South Tyrol\EMMA\Year 1\Advanced geomatics\packages\natural_earth_vector.gpkg"
countriesName = "ne_50m_admin_0_countries"
citiesName = "ne_50m_populated_places"

# cleanup, to remove maps from the projects before
HMap.remove_layers_by_name(["OpenStreetMap", citiesName, "test"])

# load open street map layer
osm = HMap.get_osm_layer()
HMap.add_layer(osm)

# load the countires layer
countriesLayer = HVectorLayer.open(geopackagePath, countriesName)

print("Schema (first 4 fields):")
counter = 0
for name, type in countriesLayer.fields.items():
    counter += 1
    if counter < 5:
        print("\t", name,"of type", type)

crs = countriesLayer.prjcode
print("Projection: ", crs)
print("Spatial extent:", countriesLayer.bbox()) # To set Extent
print("Feature count:", countriesLayer.size())

print("Attributes for Italy:")
nameIndex = countriesLayer.field_index("NAME") # This is case sensitive, if i write "name" this will show an error as -1
countriesFeatures = countriesLayer.features()

for feature in countriesFeatures:
    name = feature.attributes[nameIndex]
    if name == "Italy":
        geometry = feature.geometry
        print("Geom:", geometry.asWkt()[:50] + "...")
```

# QGIS represents features as lists of each record. That is why we use the index, this gives us a name and then we can access the element.
# Feature = geometric + attribute table part.

```
# FILTERING
expressions = "NAME like 'I%' AND POP_EST > 3000000"
filterCountriesFeatures = countriesLayer.features(expressions)
count = 0
for feature in filterCountriesFeatures:
    print(feature.attributes[nameIndex])
    count += 1
print("Feature count with filter", count)

lon = 11.119982
```

```
lat = 46.080428
point = HPoint(lon, lat)
buffer = point.buffer(2)

citiesLayer = HVectorLayer.open(geopackagePath, citiesName)
HMap.add_layer(citiesLayer)

citiesNameIndex = citiesLayer.field_index("NAME")
aoi = buffer.bbox()# area of the buffer

count = 0
for feature in citiesLayer.features(bbox=aoi):
    print(feature.attributes[citiesNameIndex])
    count += 1
print("Cities features listed:", count)

count = 0
for feature in citiesLayer.features(geometryfilter = buffer):
    print(feature.attributes[citiesNameIndex])
    count += 1
print("Cities features listed with geometry filter:", count)

# Create temporary layer
# Create data
# Create a schema
fields = {
    "id": "Integer",
    "name": "String"
}
just2citiesLayer = HVectorLayer.new("test", "Point", "EPSG:4326",
fields)
just2citiesLayer.add_feature(HPoint(-122.42, 37.78), [1, "San
Francisco"])
just2citiesLayer.add_feature(HPoint(-73.98, 40.47), [2, "New York"])

folder = "C:/Users/Michele/OneDrive - Scientific Network South
Tyrol/EMMA/Year 1/Advanced geomatics/"
path = folder + "test.gpkg"
error = just2citiesLayer.dump_to_gpkg(path, overwrite=True)
if error:
    print(error)

# Create permanent layer
teatLayer = HVectorLayer.open(path, "test")
HMap.add_layer(just2citiesLayer)

fields = {
    "name": "String",
    "population": "Integer",
    "lat": "Double", #Double is float
    "lon": "Double"
}
```

```
oneCityMoreAttributes = HVectorLayer.new("test2", "Point",
"EPSG:4326", fields)
```

name of file = test2,
geometry = Point
coordinates = EPSG:4326
fields = name of the dictionary

```
oneCityMoreAttributes.add_feature(HPoint(-73.98, 40.47), \
                                    ["New York", 19040000, 40.47, -
73.98])
error = oneCityMoreAttributes.dump_to_gpkg(path, overwrite=False)
if(error):
    print(error)
```

LESSON 3 - EXERCISE 1

```
from pyqgis_scripting_ext.core import *
filePath = r"C:\Users\Michele\OneDrive - Scientific Network South
Tyrol\EMMA\Year 1\Advanced geomatics\stations.txt"

osm = HMap.get_osm_layer()
HMap.add_layer(osm)

def fromLatString(latString):
    sign = latString[0]
    latDegrees = float(latString[1:3]) # 3 excluded
    latMinutes = float(latString[4:6])
    latSeconds = float(latString[7:9])
    lat = latDegrees + latMinutes/60 + latSeconds/3600
    if sign == "-":
        lat = lat *-1
    return lat

def fromLonString(lonString):
    sign = lonString[0]
    lonDegrees = float(lonString[1:4])
    lonMinutes = float(lonString[5:7])
    lonSeconds = float(lonString[8:10])
    lon = lonDegrees + lonMinutes/60 + lonSeconds/3600
    if sign == "-":
        lon = lon *-1
    return lon

fields = {
    "Id": "Integer",
    "Name": "String",
    "Country": "String",
    "Height": "Integer"
}

stationsLayer = HVectorLayer.new("test", "Point", "EPSG:4326",
fields)

with open(filePath,"r")as file:
    lines = file.readlines()

for line in lines:
    line = line.strip()#remove spaces

    if line.startswith("#") or len(line) == 0:
        continue

    lineSplit = line.split(",")

    latString = lineSplit[3]
    lonString = lineSplit[4]

    latDec = fromLatString(latString)
    lonDec = fromLonString(lonString)

    stationsLayer.add_feature(HPoint(lonDec,                latDec),
[lineSplit[0].strip(),  lineSplit[1].strip(),  lineSplit[2].strip(),
lineSplit[5].strip()])

folder  =  "C:/Users/Michele/OneDrive  -  Scientific  Network  South
Tyrol/EMMA/Year 1/Advanced geomatics/"
path = folder + "Stations.gpkg"
error = stationsLayer.dump_to_gpkg(path, overwrite=True)
     overwrite=False → to add a layer
     overwrite=True → overwrite layer
if(error):
    print(error)
```

LESSON 3 - EXERCISE 2

```
from pyqgis_scripting_ext.core import *

geopackagePath = r"C:\Users\Michele\OneDrive - Scientific Network
South Tyrol\EMMA\Year 1\Advanced
geomatics\packages\natural_earth_vector.gpkg"
countriesName = "ne_50m_admin_0_countries"
HMap.remove_layers_by_name(["OpenStreetMap"])

osm = HMap.get_osm_layer()
HMap.add_layer(osm)

countriesLayer = HVectorLayer.open(geopackagePath, countriesName)

# print("Schema (first 4 fields):")
counter = 0
for name, type in countriesLayer.fields.items():
    counter += 1
    if counter < 5:
        print("\t", name,"of type", type)

# print("Attributes for France:")
nameIndex = countriesLayer.field_index("NAME") # This is case
sensitive, if i write "name" this will show an error as -1
```

```
countriesFeatures = countriesLayer.features()

for feature in countriesFeatures: # Feature = geometric + attribute
table part.
    name = feature.attributes[nameIndex]
    if name == "France":
        geomFrance = feature.geometry
        # print("Geom:", geometry.asWkt()[:50] + "...")

# print(geometry)

crsHelper = HCrs()
crsHelper.from_srid(4326) # Lat lon coordinates
crsHelper.to_srid(3857)
geometry3857 = crsHelper.transform(geomFrance)

#####################################
citiesName = "ne_50m_populated_places"
citiesLayer = HVectorLayer.open(geopackagePath, citiesName)

nameIndex = citiesLayer.field_index("NAME") # This is case
sensitive, if i write "name" this will show an error as -1
citiesFeatures = citiesLayer.features()

for feature in citiesFeatures: # Feature = geometric + attribute
table part.
    geometryCities = feature.geometry
    if geomFrance.contains(geometryCities):#these are all the cities
        print(geometryCities)

# define cities of france only and reproject

canvas = HMapCanvas.new()
osm = HMap.get_osm_layer()
canvas.set_layers([osm])

crsHelper = HCrs()
crsHelper.from_srid(4326) # Lat lon coordinates
crsHelper.to_srid(3857)
geometryCities3857 = crsHelper.transform(geometryCities)

canvas.add_geometry(geometry3857)
canvas.add_geometry(geometryCities3857, "green")
canvas.set_extent(geometry3857.bbox())
canvas.show()
```

LESSON 3 - EXERCISE 3

```
from pyqgis_scripting_ext.core import *

folder = "C:/Users/Michele/OneDrive - Scientific Network South
Tyrol/EMMA/Year 1/Advanced geomatics/"
```

```
geopackagePath = r"C:\Users\Michele\OneDrive - Scientific Network South Tyrol\EMMA\Year 1\Advanced geomatics\packages\natural_earth_vector.gpkg"
countriesName = "ne_50m_admin_0_countries"
HMap.remove_layers_by_name(["OpenStreetMap"], ["centroids"])

osm = HMap.get_osm_layer()
HMap.add_layer(osm)

schema = {
    "name": "string"
}

centroidsLayer = HVectorLayer.new("centroids", "Point", "EPSG:4326", schema)

countryLayer = HVectorLayer.open(geopackagePath, countriesName)

nonInCountryList = [] # Countries with centroids not inside the main polygon:
nameIndex = countryLayer.field_index("NAME")
for country in countryLayer.features():
    countryGeom = country.geometry
    name = country.attributes[nameIndex]

    centroid = countryGeom.centroid()
```

We have geom and name and can create a new feature. We can populate the centroid name

```
    centroidsLayer.add_feature(centroid,  [name])# We need a list so we create one and insert the name.

    if not centroid.intersects(countryGeom):
        nonInCountryList.append(name)

simpleStyle = HMarker("circle", 10) + HLabel("name") + HHalo()
centroidsLayer.set_style(simpleStyle)

HMap.add_layer(centroidsLayer)

print("Countries with centroids not inside the main polygon:")
for c in nonInCountryList:
    print(c)
```

LESSON 3 - EXERCISE 4

```
from pyqgis_scripting_ext.core import *

folder = "C:/Users/Michele/OneDrive - Scientific Network South Tyrol/EMMA/Year 1/Advanced geomatics/"
geopackagePath = r"C:\Users\Michele\OneDrive - Scientific Network South Tyrol\EMMA\Year 1\Advanced geomatics\packages\natural_earth_vector.gpkg"
```

```
countriesName = "ne_50m_admin_0_countries"
HMap.remove_layers_by_name(["OpenStreetMap"])

osm = HMap.get_osm_layer()
HMap.add_layer(osm)

countriesLayer = HVectorLayer.open(geopackagePath, countriesName)

ranges = [
    [80000000, float('inf')],
# from 80 million to infinite, or you can place a number with many zeros
    [1000000, 80000000],
    [float('-inf'), 1000000],
]

styles = [
    HFill("255, 0, 0, 70"), # we use the RGB way of styling because
we eant to establish also transparency
    HFill("0, 255, 0, 70"),
    HFill("0, 0, 255, 70"),
]

labelStyle = HLabel("POP_EST") + HHalo()
countriesLayer.set_graduated_style("POP_EST", ranges, styles,
labelStyle) # POP EST = field

HMap.add_layer(countriesLayer)
```

CLASS 6

```
from pyqgis_scripting_ext.core import *

folder = "/Users/hydrologis/Desktop/unibz/"

geopackagePath = folder + "natural_earth_vector.gpkg"
countriesName = "ne_50m_admin_0_countries"
citiesName = "ne_50m_populated_places"
testName = "test"

# cleanup
HMap.remove_layers_by_name(["OpenStreetMap", citiesName, testName])

# load openstreetmap tiles layer
osm = HMap.get_osm_layer()
HMap.add_layer(osm)

# load the countries layer
countriesLayer = HVectorLayer.open(geopackagePath, countriesName)

print("Schema (first 4 fields):")
counter = 0
for name, type in countriesLayer.fields.items():
    counter += 1
    if counter < 5:
        print("\t", name, "of type", type)

crs = countriesLayer.prjcode
print("Projection: ", crs)
print("Spatial extent:", countriesLayer.bbox())
print("Feature count:", countriesLayer.size())


print("Attributes for Italy:")
nameIndex = countriesLayer.field_index("NAME")
countriesFeatures = countriesLayer.features()
for feature in countriesFeatures:
    name = feature.attributes[nameIndex]
    if name == "Italy":
        geometry = feature.geometry
        print("Geom:", geometry.asWkt()[:50] + "...")


expressions = "NAME like 'I%' AND POP_EST > 3000000"
filteredCountriesFeatures = countriesLayer.features(expressions)
count = 0
for feature in filteredCountriesFeatures:
    print(feature.attributes[nameIndex])
    count += 1
print("Feature count with filter", count)

lon = 11.119982
lat = 46.080428
point = HPoint(lon, lat)
buffer = point.buffer(2)

citiesLayer = HVectorLayer.open(geopackagePath, citiesName)
HMap.add_layer(citiesLayer)

citiesNameIndex = citiesLayer.field_index("NAME")
aoi = buffer.bbox()

count = 0
for feature in citiesLayer.features(bbox=aoi):
    print(feature.attributes[citiesNameIndex])
    count+=1
print("Cities features listed with bbox filter:", count)

count = 0
for feature in citiesLayer.features(geometryfilter=buffer):
    print(feature.attributes[citiesNameIndex])
    count+=1
print("Cities features listed with geometry filter:", count)

# create data

# create a schema
```

```
fields = {
    "id": "Integer",
    "name": "String"
}
just2citiesLayer = HVectorLayer.new(testName, "Point", "EPSG:4326",
fields)
just2citiesLayer.add_feature(HPoint(-122.42, 37.78), [1, "San
Francisco"])
just2citiesLayer.add_feature(HPoint(-73.98, 40.47), [2, "New York"])

path = folder + "test.gpkg"
hopeNotError = just2citiesLayer.dump_to_gpkg(path, overwrite=True)
if hopeNotError:
    print(hopeNotError)

testLayer = HVectorLayer.open(path, testName)
HMap.add_layer(testLayer)

fields = {
    "name": "String",
    "population": "Integer",
    "lat": "Double",
    "lon": "Double"
}
oneCityMoreAttributes = HVectorLayer.new("test2",
"Point","EPSG:4326", fields)
oneCityMoreAttributes.add_feature(HPoint(-73.98, 40.47), \
                                    ["New York", 19040000, 40.47, -
73.98])
hopeNotError = oneCityMoreAttributes.dump_to_gpkg(path,
overwrite=False)
if hopeNotError:
    print(hopeNotError)
```

CLASS 7

```
from pyqgis_scripting_ext.core import *

folder = "/Users/hydrologis/Desktop/unibz/"

geopackagePath = folder + "natural_earth_vector.gpkg"
countriesName = "ne_50m_admin_0_countries"
citiesName = "ne_50m_populated_places"
riversName = "ne_10m_rivers_lake_centerlines_scale_rank"

# cleanup
HMap.remove_layers_by_name(["OpenStreetMap",             citiesName,
countriesName, "rivers_italy"])

# load openstreetmap tiles layer
osm = HMap.get_osm_layer()
HMap.add_layer(osm)
```

```
# load the layer
# cities layer
citiesLayer = HVectorLayer.open(geopackagePath, citiesName)
citiesLayer.subset_filter("SOV0NAME='Italy'")

pointStyle = HMarker("square", 6, 45) + \
                HFill("red") + HStroke("black", 1)
field = "NAME"
#pointStyle += HLabel(field, yoffset=-8) + HHalo("white", 1)
field = "if(POP_MAX>1000000, concat(NAME, ' (',
round(POP_MAX/1000000, 1), ')'), NAME)"

labelProperties = {
    "font": "Arial",
    "color": "black",
    "size": 20,
    "field": field,
    "xoffset": 0,
    "yoffset": -8
}
pointStyle += HLabel(**labelProperties) + HHalo("white", 2)
citiesLayer.set_style(pointStyle)

# polygon layer
countriesLayer = HVectorLayer.open(geopackagePath, countriesName)
countriesLayer.subset_filter("NAME='Italy'")
italyGeometry = countriesLayer.features()[0].geometry


print(italyGeometry.centroid())

polygonStyle = HFill("0,255,0,128") + HStroke("green", 2)
countriesLayer.set_style(polygonStyle)

# lines layer
riversLayer = HVectorLayer.open(geopackagePath, riversName)
riversLayerItaly = riversLayer.sub_layer(italyGeometry,
"rivers_italy", ['scalerank', 'name'])

# thematic styling
ranges = [
    [0, 0],
    [1, 5],
    [6, 7],
    [8, 9],
    [10, 11]
]
styles = [
    HStroke("blue", 7),
    HStroke("blue", 5),
    HStroke("blue", 3),
    HStroke("blue", 2),
    HStroke("blue", 1)
```

```
]
labelProperties = {
    "font": "Arial",
    "color": "blue",
    "size": 14,
    "field": 'name',
    "along_line": True,
    "bold": True,
    "italic": True
}
labelStyle = HLabel(**labelProperties) + HHalo("white", 1)
riversLayerItaly.set_graduated_style('scalerank',   ranges,   styles,
labelStyle)

# riversStyle = HStroke("blue", 2)

# riversStyle += labelStyle
# riversLayerItaly.set_style(riversStyle)

HMap.add_layer(countriesLayer)
HMap.add_layer(riversLayerItaly)
HMap.add_layer(citiesLayer)

printer = HPrinter(iface)
mapProperties = {
    "x": 5,
    "y": 25,
    "width": 285,
    "height": 180,
    "extent":     [10,44,12,46],     #     or     bounding     box:
countriesLayer.bbox(),
    "frame":True
}

labelProperties = {
    "x": 120,
    "y": 10,
    "text": "River and cities map",
    "font_size": 28,
    "bold": True,
    "italic": False
}

legendProperties = {
    "x": 215, # mm
    "y": 30,
    "width": 150,
    "height": 100,
    "max_symbol_size": 3 # not mm
}

scalebarProperties = {
    "x": 10,
    "y": 190,
    "units": "km",
    "segments": 4,
    "unit_per_segment": 10,
    "style": "Single Box",
    "font_size": 12
}

printer.add_map(**mapProperties)
printer.add_label(**labelProperties)
printer.add_legend(**legendProperties)
printer.add_scalebar(**scalebarProperties)

outputPdf = f"{tempfolder}/test.pdf"
printer.dump_to_pdf(outputPdf)

outputpng = f"{tempfolder}/test.png"
printer.dump_to_image(outputpng)
```

GROUPWORK - MOCK EXAM

```
from pyqgis_scripting_ext.core import *

folder = "C:/Users/Michele/OneDrive - Scientific Network South
Tyrol/EMMA/Year 1/Advanced geomatics/"

csvpath = folder + "/test/22yr_T10MN"
gpkgPATH = folder +
"/natural_earth_vector.gpkg/packages/natural_earth_vector.gpkg"
csvpath2 = folder + "/test/22yr_T10MX"

with open(csvpath2,'r') as file:
    lines = file.readlines()

canvas = HMapCanvas.new()
osm = HMap.get_osm_layer()
canvas.set_layers([osm])

grid = []
temps = []
TempPerCoord = {}

headerline = None
for i,line in enumerate(lines):
    if "Jan    Feb    Mar" in line:
        headerline = i
        if isinstance(headerline, int):
            break

for line in lines[(i+1):]:
    if not (line.startswith("#") or line.strip() == ""):
        line = line.strip()
        lineSplit = line.split(" ")
        Lat = float(lineSplit[0])
        Lon = float(lineSplit[1])
        AnnTemp = float(lineSplit[-1])

        temps.append(AnnTemp)
        # print(Lat, Lon)

        coords = [
            [Lon,Lat],
            [Lon,Lat+1],
            [Lon+1,Lat+1],
            [Lon+1,Lat],
            [Lon,Lat]]

        rectangle = HPolygon.fromCoords(coords)

        crsHelper = HCrs()
        crsHelper.from_srid(4326)
        crsHelper.to_srid(3857)

        convertedrec = crsHelper.transform(rectangle)

        grid.append(convertedrec)

countries = ["Italy", "Germany", "Austria"]

# GERMANY coords:
countriesName = "ne_50m_admin_0_countries"
countriesLayer = HVectorLayer.open(gpkgPATH, countriesName)

nameIndex = countriesLayer.field_index("NAME")
countriesFeatures = countriesLayer.features()

Countries_Geometries = []

for feature in countriesFeatures:
    name = feature.attributes[nameIndex]
    if name in countries:
        for country in countries:
            countryGeom = feature.geometry # get the geometry
            # print("GEOM:", germanGeom.asWkt()[:100] + "...")
            crsHelper = HCrs()
            crsHelper.from_srid(4326)
            crsHelper.to_srid(3857)

            COUNTRY = crsHelper.transform(countryGeom)
            Countries_Geometries.append(COUNTRY)

# print(min(temps))
# print(max(temps))

tempandcolor = {
    -10:'midnightblue',
    -5:'darkblue',
    0:'blue',
    5:'lightblue',
    6:'lightcyan',
    8:'gold',
    9:'yellow',
    15:'orange',
    20:'coral',
    30:'red'}

for geometries in Countries_Geometries:
    for rectangle,temp in zip(grid,temps):
        if rectangle.intersects(geometries):
            for limit,color in tempandcolor.items():

                if temp < limit:
                    cliped = rectangle.intersection(geometries)
                    canvas.add_geometry(cliped,color,1)
                    break

canvas.set_extent([-20037508.34,-20048966.1,20037508.34,20048966.1])
canvas.show()
```

EXAM - LAKES IN GERMANY AND ITALY

```
from pyqgis_scripting_ext.core import *

# ------------------------ FOLDERS -----------------------

#outputfolder = r"C:/Users/Michele/OneDrive - Scientific Network
South Tyrol/EMMA/Year 1/Advanced geomatics/output"
#geopackageFolder = folder2 = r"C:/Users/Michele/OneDrive -
Scientific Network South Tyrol/EMMA/Year 1/Advanced
geomatics/packages/natural_earth_vector.gpkg"

# ------------------------ CLEAN -------------------------

HMap.remove_layers_by_name(["OpenStreetMap","Lakes","ne_50m_admin_0_
countries"])

# ---------------- GET DATA FROM WIKIDATA ----------------

# import the http requests library to get stuff from the internet
import requests
# import the url parsing library to urlencode the query
import urllib.parse
# define the query to launch
endpointUrl = "https://query.wikidata.org/sparql?query=";
# define the query to launch
query = """
SELECT ?item ?itemLabel ?itemDescription ?area ?elev ?image ?coord
WHERE {
  ?item (wdt:P31/wdt:P279*) wd:Q23397.
  {?item wdt:P17 wd:Q38} UNION {?item wdt:P17 wd:Q183}.
  ?item wdt:P625 ?coord.
  ?item wdt:P2046 ?area.
  ?item wdt:P2044 ?elev
  OPTIONAL {?item wdt:P18 ?image.}
  SERVICE wikibase:label { bd:serviceParam wikibase:language
"[AUTO_LANGUAGE],en". }
}
"""
# URL encode the query string
encoded_query = urllib.parse.quote(query)

# prepare the final url
url = f"{endpointUrl}{encoded_query}&format=json"

# run the query online and get the produced result as a dictionary
r=requests.get(url)
result = r.json()
# print(result)


# -------------------- GET USEFUL DATA --------------------

coordinates = []
names = []
areas = []
elevations = []

for item in result['results']['bindings']:
    if "coord"in item:
        coord = item['coord']['value']
        coordinates.append(coord)
    if "itemLabel" in item:
        name = item['itemLabel']['value']
        names.append(name)
    if "area" in item:
        area = item['area']['value']
        areas.append(area)
    if "elev" in item:
        elevation = item['elev']['value']
        elevations.append(elevation)

# print(len(coordinates))
# print(len(names))
# print(len(areas))
# print(len(elevations))


# ------- CREATE LAKE GEOMETRIES AND TRANSFORM THEM -------

crsHelper = HCrs()
crsHelper.from_srid(4326) #spetial reference system ID
crsHelper.to_srid(3857)

newcoords = []
```

```
for coord in coordinates:
    pointGeom = HGeometry.fromWkt(coord)
    newcoord = crsHelper.transform(pointGeom)
    newcoords.append(newcoord)


# --------------------- CREATE LAYER -----------------------

fields = {
    "Name": "String",
    "Area": "Float",
    "Elevation": "Float",
}

LakesLayer = HVectorLayer.new("Lakes", "Point", "EPSG:3857", fields)

saved_names = []
for i, (coord, name, area, elevation) in enumerate(zip(newcoords,
names, areas, elevations)):
    if name not in saved_names:

LakesLayer.add_feature(newcoords[i],[names[i],areas[i],elevations[i]
])
        saved_names.append(name)

#print(len(saved_names))


# --------------- DUMP TO GEOPACKAGE -----------------------

path = outputfolder + "/Lakes.gpkg"
error = LakesLayer.dump_to_gpkg(path, overwrite=True)
if(error):
    print(error)

# -------------------- COUNTRY BORDERS ---------------------

countriesName = "ne_50m_admin_0_countries"
countriesLayer = HVectorLayer.open(geopackageFolder, countriesName)
countriesLayer.subset_filter("NAME='Italy' OR NAME='Germany'")

# ----------------------- STYLING --------------------------

# LAKES

ranges = [
    [float('-inf'),10],
    [11,100],
    [101,1000],
    [1001,float('inf')]
]

styles = [
    HMarker("circle",1) + HFill("skyblue") + HStroke("skyblue",0.5),
    HMarker("circle",1) + HFill("cornflowerblue") +
HStroke("cornflowerblue",0.5),
    HMarker("circle",1) + HFill("blue") + HStroke("blue",0.5),
    HMarker("circle",1) + HFill("black") + HStroke("black",0.5),
]

#labelstyle = HLabel("name") + HHalo("white",1)
#LakesLayer.set_graduated_style('Area', ranges, styles, labelstyle)
LakesLayer.set_graduated_style('Area', ranges, styles)


# COUNTRIES

style = HFill('rgba(0,0,0,0)') + HStroke('black',0.5)
countriesLayer.set_style(style)

# ----------------------- MAP SHOW -------------------------

osm = HMap.get_osm_layer()
HMap.add_layer(osm)
HMap.add_layer(countriesLayer)
HMap.add_layer(LakesLayer)

# -------------------- LAYOUT AS PDF ---------------------

printer=HPrinter(iface)
mapProperties ={
        "x":5,
        "y":25,
        "width": 285,
        "height":180,
        "frame": True,
        "extent": [-777542,7643212,2916896,4273906]
#LakesLayer.bbox()
    }
printer.add_map(**mapProperties)

legendProperties={
       "x":218,
        "y":30,
        "width": 150,
        "height":100,
       "frame":True
    }
printer.add_legend(**legendProperties)

labelProperties={
        "x":105,
        "y":8,
        "text":"LAKES in Germany and Italy",
        "bold":True,
        "font_size":20
    }
printer.add_label(**labelProperties)

subtitleProperties={
        "x":97,
        "y":17,
        "text":"by: Miriam Färber, Romina Lavarello, Luisa
Menestrina, Laura Morass",
        "bold":False,
        "italic":True,
        "font_size":10
    }
printer.add_label(**subtitleProperties)

scalebarProperties = {
    'x': 10,
    'y': 190,
    'units': "km",
    'segments': 4,
    'style': "Single Box",
}
printer.add_scalebar(**scalebarProperties)

imageName= f"LakesMapLayout.png"
imagePath = f"{outputfolder}/{imageName}"
pdfName= f"LakesMapLayout.pdf"
pdfPath = f"{outputfolder}/{pdfName}"

printer.dump_to_image(imagePath)
printer.dump_to_pdf(pdfPath)


# -------------------- PRINTING OUTPUTS --------------------

saved_names = []
lakesabove2000 =[]
lakes500_1000 = []
all_others =[]

for i, (coord, name, area, elevation) in enumerate(zip(newcoords,
names, areas, elevations)):
    if name not in saved_names:
        saved_names.append(name)

        if float(elevation) > 2000:
            lakesabove2000.append(name)
        elif float(elevation) > 500 and float(elevation) < 1000:
            lakes500_1000.append(name)
        else:
            all_others.append(name)

print("Total number of lakes:")
print(len(saved_names))
print("Lakes above 2000 masl:")
print(len(lakesabove2000))

print("Lakes between 500 and 1000 masl:")
print(len(lakes500_1000))

print("Other lakes:")
print(len(all_others))
```