faiza khurshid [faiza.khurshid@uni-bonn.de (mailto:faiza.khurshid@uni-bonn.de)](mailto:faiza.khurshid@uni-bonn.de)

Exercise 1 (Fixed Point Iteration, 6 Points)

• Implement a Python function that applies a fixed-point iteration to a given function g(x), starting at a given value x0, and returning the final value after N iterations. (2P)

In [151]:
```
import math
def fix_iter(x0,n):
    x=x0
    iterr=0
    while n>iterr:
        y = math.cos(x-1)+x
        x = y
        iterr+=1

    return y


print(fix_iter(0.8,10))
```

```
2.5707963267948966
```

• Implement an improved version of your function that accepts a boolean function stopon(x) as an additional optional parameter and stops the iteration early (still taking at most N steps) if stopon(x) returns True when evaluated on the current value of x. In addition to returning the final value of x, the function should now also indicate whether it used the maximum number of iterations. (2P)

In [156]:
```
from math import cos
def stopn(x,e):
    if abs(cos(x-1))<e:
        return True
    else:
        return False

def new_fix_iter(n,x,e,end=False):
    import math
    n=n
    x=x
    for itrr in range(n):
        updated_x=(math.cos(x-1))+x
        x=updated_x
        if end:
            if stopn(x,e):
                break
    if itrr<(n-1):
        print('not achived maximum iteration')
    return x
```

• Use your function to numerically solve Equation (1). Use stopon(x) to stop the iteration once | cos(x − 1)| < □ for a value of □ that is reasonably close to zero. Try your code for different values of x0 and □.

In [153]:
```
print(new_fix_iter(20,3,end=True,e=0.00001))
```

```
not achived maximum iteration
2.5838531634528574
```

In [157]:
```
print(new_fix_iter(100,0.01,end=True,e=0))
```

```
2.5707963267948966
```

Exercise 2 (Connected Components, 8 Points)

In [160]:
```
with open('binary-grid.txt') as f:
    grid = []
    for line in f:
        line = line.split() # to deal with blank
        if line:   # lines (ie skip them)
            line = [int(i)^1 for i in line]
            grid.append(line)



def neighbouring_labels(array, x, y):
    labels = set()

    # West neighbour
    if x > 0:  # Pixel is not on left edge of image
        west_neighbour = array[y] [x - 1]
        if west_neighbour > 0:  # It's a labelled pixel
            labels.add(west_neighbour)

        # North neighbour
    if y > 0:  # Pixel is not on top edge of image
        north_neighbour = array[y - 1][ x]
        if north_neighbour > 0:  # It's a labelled pixel
            labels.add(north_neighbour)

    return labels



labelled_image = grid


class UnionFind:

    # Constructor
    def __init__(self):
        self.__nodes_addressed_by_value = {}  # To keep track of nodes
    def MakeSet(self, value):
        if self.GetNode(value):
            return self.GetNode(value)
        node = Node(value)
        self.__nodes_addressed_by_value[value] = node

        return node

    def Find(self, x):
        if x.parent != x:
            x.parent = self.Find(x.parent)  # Flatten tree as you go (Path Compre
        return x.parent

    def Union(self, x, y):
        if x == y:
            return
        x_root = self.Find(x)
        y_root = self.Find(y)
        if x_root == y_root:
```

```
            return
        if x_root.rank > y_root.rank:
            y_root.parent = x_root

        elif x_root.rank < y_root.rank:
            x_root.parent = y_root
        else:
            x_root.parent = y_root
            y_root.rank = y_root.rank + 1
    def GetNode(self, value):  # Get node with value 'value' (O(1))
        if value in self.__nodes_addressed_by_value:
            return self.__nodes_addressed_by_value[value]
        else:
            return False
class Node(object):
    def __init__(self, value):
        self.value = value
        self.parent = self # Node is its own parent, therefore it's a root node
        self.rank = 0 # Tree of single node has rank 0


    uf = UnionFind()
    current_label = 1  # initialise label counter

# 1st Pass: label image and record label equivalences
for y, row in enumerate(labelled_image):
    for x, value in enumerate(row):

        if value == False:
            # Background pixel - leave output pixel value as 0
            pass
        else:
            # Foreground pixel - work out what its label should be

            # Get set of neighbour's labels
            labels = neighbouring_labels(labelled_image, x, y)

            if not labels:
                # If no neighbouring foreground pixels, new label -> use current_
                labelled_image[y][x] = current_label
                uf.MakeSet(current_label)
                current_label = current_label + 1  # increment for next time

            else:
                # Pixel is definitely part of a connected component: get smallest
                # neighbours
                smallest_label = min(labels)
                labelled_image[y][x] = smallest_label
                if len(labels) > 1:  # More than one type of label in component -
                    # equivalence class
                    for label in labels:
                        uf.Union(uf.GetNode(smallest_label), uf.GetNode(label))
final_labels = {}
new_label_number = 1
for y, row in enumerate(labelled_image):
    for x, pixel_value in enumerate(row):
```

```
        if pixel_value > 0:  # Foreground pixel
            # Get element's set's representative value and use as the pixel's new
            new_label = uf.Find(uf.GetNode(pixel_value)).value
            labelled_image[y][x] = new_label

            # Add label to list of labels used, for 3rd pass (flattening label li
            if new_label not in final_labels:
                final_labels[new_label] = new_label_number
                new_label_number = new_label_number + 1

    # 3rd Pass: flatten label list so labels are consecutive integers starting fr
    # of top to bottom, left to right)

for y, row in enumerate(labelled_image):
    for x, pixel_value in enumerate(row):

        if pixel_value > 0:  # Foreground pixel
            labelled_image[y][x] = final_labels[pixel_value]


def print_image(image):
    for y, row in enumerate(image):
        print(row)

print_image(labelled_image)
```

```
[1, 1, 0, 1, 1, 1, 0, 2]
[1, 1, 0, 1, 0, 1, 0, 2]
[1, 1, 1, 1, 0, 0, 0, 2]
[0, 0, 0, 0, 0, 0, 0, 2]
[3, 3, 3, 3, 0, 4, 0, 2]
[0, 0, 0, 3, 0, 4, 0, 2]
[3, 3, 3, 3, 0, 0, 0, 2]
[3, 3, 3, 3, 0, 2, 2, 2]
```

Exercise 3 (K-means, 7 Points)

In [129]:
```
import random

def k_mean(k,data):
    centers=[]
    k=3
    # find random center
    for i in range(k):
        rand=random.choice(data)
        centers.append(rand)
    c1x=[]
    c1y=[]
    c2x=[]
    c2y=[]
    c3x=[]
    c3y=[]
    mean_c1=[]
    mean_c2=[]
    mean_c3=[]
    new_center=[]
    k1=centers[0]
    k2=centers[1]
    k3=centers[2]
    #find the distance between centers and data
    while True:
        for m in data:
            dist1=((m[0]-k1[0])**2 +(m[1]-k1[1])**2) **0.5
            dist2=((m[0]-k2[0])**2 +(m[1]-k2[1])**2) **0.5
            dist3=((m[0]-k3[0])**2 +(m[1]-k3[1])**2) **0.5
    # check minimum distance
            if dist1<dist2 and dist1<dist3:
                c1x.append(m[0])
                c1y.append(m[1])
            elif dist2< dist1 and dist2<dist3:
                c2x.append(m[0])
                c2y.append(m[1])
            elif dist3<dist1 and dist3<dist2:
                c3x.append(m[0])
                c3y.append(m[1])
    # find the mean to get new centers
        mean_c1x=sum(c1x)/(len(c1x))
        mean_c1y=sum(c1y)/(len(c1y))
        mean_c2x=sum(c2x)/(len(c2x))
        mean_c2y=sum(c2y)/(len(c2y))
        mean_c3x=sum(c3x)/(len(c3x))
        mean_c3y=sum(c3y)/(len(c3y))

        mean_c1.extend((mean_c1x,mean_c1y))
        mean_c2.extend((mean_c2x,mean_c2y))
        mean_c3.extend((mean_c3x,mean_c3y))




        new_center.extend((mean_c1,mean_c2,mean_c3))
        if new_center==centers:
            break
```

```
        else:
            centers=new_center


    # data into txt file
    with open ('newdata.txt','w') as f1:
        clt1=[list(l) for l in zip(c1x,c1y)]
        clt2=[list(l) for l in zip(c2x,c2y)]
        clt3=[list(l) for l in zip(c3x,c3y)]
        mean={'mean_C1':mean_c1,'mean_C2':mean_c2,'mean_C3':mean_c3,}
        for p in clt1:
            line =" 1: {}, {}\n".format(p[0], p[1])
            f1.write(line)
        for li in clt2:
            line2=" 2: {}, {}\n".format(li[0], li[1])
            f1.write(line2)
        for lin in clt3:
            line3=" 3: {}, {}\n".format(lin[0], lin[1])
            f1.write(line3)
        f1.write(str(mean))


l=[]
with open('data.txt','r') as f:
    M=f.read().split('\n')
    for j in M:
        new=j.split(' ')
        l.append(new)
new_list=l[:-1]
data=[[float(k)for k in fi] for fi in new_list]

k_mean(3,data)
```

In [128]:
```
with open ('newdata.txt','r') as f1:
    print(f1.read())
```

```
3: -9.556752, -5.169512
3: -10.011646, -5.207454
3: -7.972646, -6.740516
3: -8.713239, -5.796295
3: -7.357077, -4.414195
3: -10.311904, -5.161901
3: -9.067364, -4.909175
3: -9.354613, -4.835358
3: -9.91323, -5.234074
3: -9.140521, -5.074881
3: -8.055294, -4.666674
3: -7.49677, -4.795778
3: -8.021506, -6.161022
3: -9.504835, -8.090491
3: -9.296454, -4.13462
3: -7.880349, -6.938514
3: -10.20108, -6.525425
3: -9.2138, -6.332983
3: -7.323691, -4.890603
3: -8.219401, -5.084889
```

Exercise 4 (Gangsters are Persons, 4 Points)

• Who is for sure a gangster? How can you tell?

Giovanni,carla,Natalia,Cosimo,Rose,paolo, they all are gangsters

the reason is in studying class diagram i noticed that gangster class has bribe association
with itself,its means that just gangster bribe the other gangsters. Also gangsters has steel from association with person, its means these those are gangsters who steal persons.
Natalia bribe the Cosimo, its means cosimo and natalia are gangsters.
giovani steal the claudio and we know steel from association is from gangster to person so giovani also a gangster.
similarly Carla has function to steel so he is a ganster.
Rosa bribes the paolo and both are gangsters.

• Who is for sure a person, but not a gangster? How can you tell? (1P)

cladio and jake are for sure persons because according to diagram jake and claudio do not have those
associations which just belong to gangster.

i noticed the incomming association of cladio from Giovanni is steel from and we know steel from association is only gangster to person and we also know Giovanni is a gangster.
cladio makes a deal with jake and i know makes a deal association is just from person to person. so iam sure they are person.