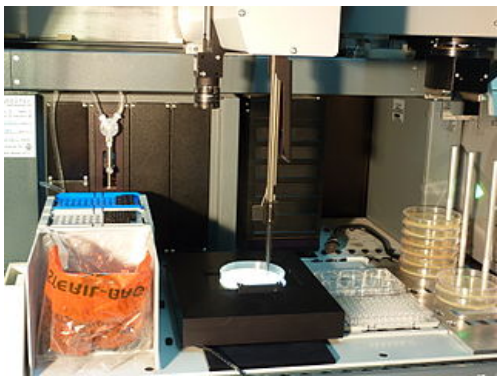
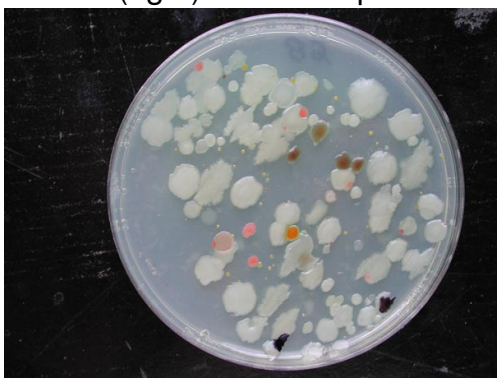
Colony picker

A *colony picker* is an instrument that is used to automatically localize, pick up or duplicate microbial colonies that grow on a fixed or fluid medium. Usually, a Petri dish is inserted on the light plate of the colony picker, after which algorithms for image analysis choose the right colonies and a operate a robot arm to sample those colonies. Such appliances are used in both research laboratories and industrial surroundings, e.g. to analyze food or blood samples.

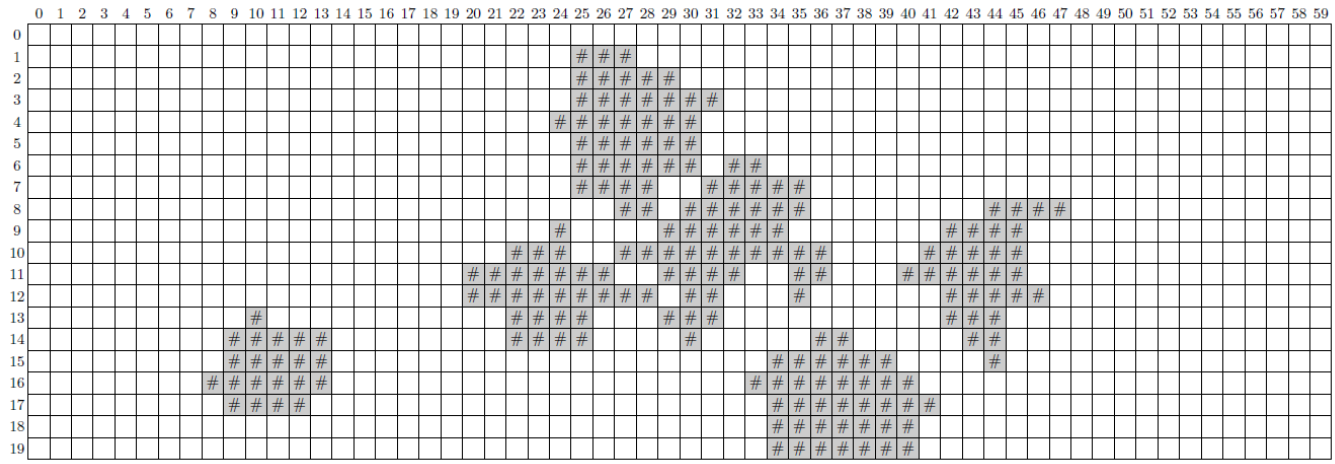


Pickolo — an automatic colony picker that was installed as an extension of a Tecan robot shows a typical laboratory setup. On the light plate of the robot (below) a Petri dish is placed. The camera (above) takes pictures of the dish and image processing software chooses the right colonies for further treatment. A disposable tip (from the rack on the right) is used to touch a colony and carry it over to a collecting dish (bottom right). In a completely automatized setup a stacker (right) is used to process multiple dishes one after another.



Every spot on the agar plate is a bacterial colony from the radius of a sponge. By the time a colony is visible for the human eye, it already consists of at least a million cells. Less than one percent of all bacteria can be grown in this manner in the laboratory, just to indicate how large the reservoir of unknown bacterial diversity still is. In order to map out the unknown bacteria, microbiologists use a combination of new growing techniques for molecular genetic characterization in order to research microbial communities.

The image processing software sees the photo of the Petri dish in the format of a bitmap. This is nothing but a rectangle grid of which the boxes are called bits because they can only take two forms. Every bit of the grid is either empty (represented by a space) or covered by a colony (represented by a hash: #). As an example you can see a bitmap below on which a number of colonies can be seen. In a bitmap, a colony is formed by an area of neighbouring covered bits. Bits are here called neighbouring if they have a side in common. However, the software will never see an area of neighbouring covered bits as a colony, if it contains bits on the outer rim of the bitmap.



Bitmap made of a Petri dish. This bitmap is a rectangluar grid with 20 rows and 60 columns, where the bits that are covered by bacterial colonies are indicates with hashes (#). This bitmap has six areas of neighbouring covered bits, but only five of those are seen as colonies, because one of them has bits that are on the outer rim of the bitmap.

Assignment

Define a class Petridish which can be used to analyze colonies on a Petri dish. the objects of this class must contain at least the following methods:

- An initializing method __init__ to which the location of a text file must be given. This text file contains a bitmap of the Petri dish that needs to be analyzed.
- A method __str__ that prints the string representation of the Petri dish. This string representation gives the rectangle gird as it is read in the bitmap file. Apart from empty bits (represented by spaces) and bits that are covered by colonies (represented by hashes: #), the bits can also be processed (represented by full stops). This processing happens by calling the method colonies (see below).
- A method colony to which the row number and column number of a covered bit in the bitmap must be given. The rows of a bitmap are numbered from top to bottom, the columns from left to right, and the numbering always starts from zero. If the bit on the given position isn't covered, the method must raise an AssertionError with the text no colony found on position(*row*, *column*). Here, the fragments in italics must be filled out based on the arguments that are given to the method. Otherwise, all bits of the area of neighbouring covered bits to which the given bit belongs must be marked as processed (they are no longer seen as covered after the method was called), and the method must print the amount of bits this area consists of.
- A method undo that marks all processed bits of a bitmap as covered again. Note that the processed bits in a bitmap could possibly belong to multiple colonies, for example if the method colony was called multiple times before the method undo is called.
- Use the methods colony and undo to write a method count that prints the number of colonies that are not smaller than the given minimal size. The minimal size can be given to the optional parameter minimum (standard value: 1). The size of a colony is expressed as the number of bits in the corresponding area of neighbouring covered bits in the bitmap. The use of a minimal size is important for excluding smaller artifacts that can arise when converting of the photo of the Petri dish to the corresponding bitmap. We notice again that the areas of neighbouring covered bits that contain bits on the outer rim of the bitmap may not be see as colonies. These areas must be neglected when the method counts the

colonies. Make sure that the bitmap does not have any processed bits after the method was called.

- Use the methods colony and undo to write a method size that prints the size of all colonies that are not smaller than the given minimal size. The minimal size can be given to the optional parameter minimum (standard value: 1). As in the method above, this method may not see the areas of neighbouring covered bits that contain bits on the outer rim of the bitmap as colonies, and the bitmap may not show any processed bits after the method was called. The method must print the value None if there are no colonies that are larger than or equal to the given minimal size.

Example

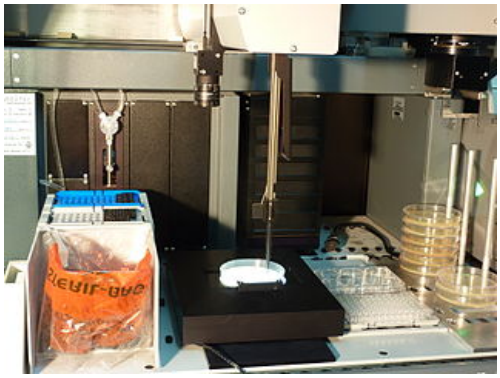
In the example session below we assume that the text file dish.txt is situated in the current directory. This file contains the bitmap of a Petri dish that was graphically shown above.

```
>>> dish = Petridish('dish.txt')
>>> print(dish)
<BLANKLINE>
                  ###
                  #####
                  #######
                 #######
                  ######
                  ###### ##
                  #### #####
                    ## ######     ####
                  #   ######    ####
                ### ##########   #####
               ####### #### ##  ######
               ######### ##  #    #####
        #       ####  ###      ###
       #####     ####   #   ##   ##
       #####            ######   #
      ######            ########
       ####             ########
                       #######
                       #######
>>> dish.colony(10, 35)
42
>>> print(dish)
<BLANKLINE>
                  ###
                  #####
                  #######
                 #######
                  ######
                  ###### ..
                  #### .....
                    ## ......      ####
                  #   ......     ####
                ### ..........   #####
               ####### ....  ..  ######
               ######### ..   .    #####
        #       ####  ...       ###
       #####     ####   .    ##   ##
       #####            ######   #
      ######            ########
```

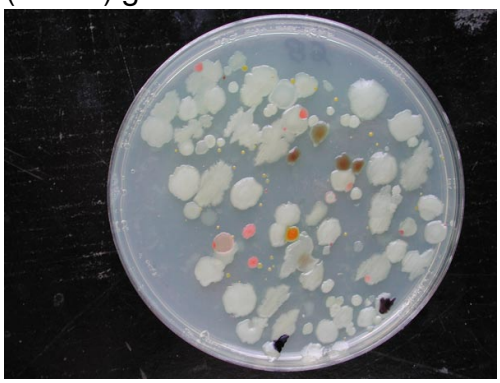
```
       ####                ########
                          #######
                          #######
>>> dish.colony(10, 40)
Traceback (most recent call last):
AssertionError: no colony was found on position (10, 40)
>>> dish.colony(10, 45)
30
>>> print(dish)
<BLANKLINE>
                    ###
                    #####
                    #######
                   #######
                    ######
                    ###### ..
                    ####  .....
                      ## ......        ....
                     #    ......       ....
                    ###  ..........    .....
                  ####### ....  ..   ......
                  ######### ..   .      .....
        #         ####   ...          ...
       #####       ####    .    ##     ..
       #####                ######    .
      ######                ########
       ####                 ########
                          #######
                          #######
>>> dish.undo()
>>> print(dish)
<BLANKLINE>
                    ###
                    #####
                    #######
                   #######
                    ######
                    ###### ##
                    ####  #####
                      ## ######      ####
                     #   ######     ####
                    ### ##########   #####
                  ####### #### ##  ######
                  ######### ##  #    #####
        #         ####  ###       ###
       #####       ####   #    ##    ##
       #####                ######   #
      ######                ########
       ####                 ########
                          #######
                          #######
>>> dish.colonies()
5
>>> dish.colonysize()
32.2
```

Een *koloniepikker* is een instrument waarmee microbiële kolonies die op een vaste voedingsbodem groeien automatisch kunnen worden gelocaliseerd, opgepikt en gedupliceerd op een vaste of vloeibare voedingsbodem. Doorgaans wordt een petrischaal aangebracht op de

lichtplaat van de koloniepikker, waarna algoritmen voor beeldanalyse de juiste kolonies uitkiezen en een robotarm aansturen om die kolonies te bemonsteren. Dergelijke toestellen worden zowel in onderzoekslaboratoria als in industriële omgevingen gebruikt, bijvoorbeeld om voedings- of bloedstalen te analyseren.

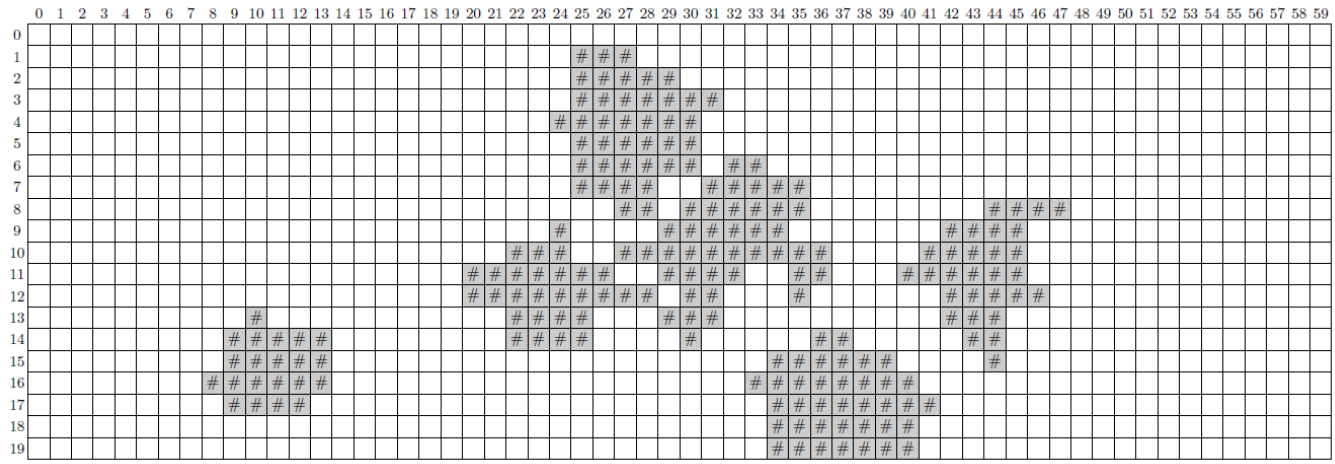


Pickolo — een automatische koloniepikker die als uitbreiding geïnstalleerd werd op een Tecan robot toont een typische laboratoriumopstelling. Op de lichtplaat van de robot (onder) wordt een petrischaal geplaatst. De camera (boven) neemt een foto van de schaal en beeldanalysesoftware kiest de juiste kolonies uit voor verdere verwerking. Een wegwerptip (uit het rek links) wordt gebruikt om een kolonie aan te raken en over te brengen naar een verzamelschaal (rechtsonder). In een volledig geautomatiseerde opstelling wordt een stapelaar (rechts) gebruikt om meerdere schalen na elkaar te kunnen verwerken.



Elk vlekje op deze agarplaat is een bacteriële kolonie afkomstig uit een staal van een zeespons. Tegen de tijd dat een kolonie zichtbaar wordt voor het menselijke oog, bestaat ze reeds uit minstens een miljoen cellen. Minder dan één procent van alle bacteriën kan op die manier in het laboratorium gekweekt worden, om maar aan te geven hoe groot het reservoir aan onbekende bacteriële diversiteit nog is. Om de onbekende bacteriën in kaart te brengen, gebruiken microbiologen een combinatie van nieuwe groeitechnieken en technieken voor moleculair genetische karakterisering om daarmee microbiële gemeenschappen te kunnen onderzoeken.

De beeldverwerkingssoftware krijgt de foto van een petriplaat te zien onder de vorm van een bitmap. Dit is niets anders dan een rechthoekig rooster waarvan de vakjes bits genoemd worden omdat ze slechts twee toestanden kunnen aannemen. Elke bit van het rooster is ofwel leeg (voorgesteld door een spatie) of wordt bedekt door een kolonie (voorgesteld door een hekje: #). Als voorbeeld zie je hieronder een bitmap waarop een aantal kolonies te zien zijn. In een bitmap wordt een kolonie gevormd door een gebied van aangrenzende bedekte bits. Bits worden hierbij aangrenzend genoemd als ze een gemeenschappelijke zijde hebben. De software zal een gebied van aangrenzende bedekte bits echter nooit als een kolonie beschouwen, als het bits op de buitenste rand van de bitmap bevat.



Bitmap gemaakt van een petriplaat. Deze bitmap is een rechthoekig rooster met 20 rijen en 60 kolommen, waarbij de bits die bedekt zijn door bacteriële kolonies worden aangeduid door hekjes (#). Deze bitmap telt zes gebieden van aangrenzende bezette bits, maar slechts vijf daarvan worden als kolonies beschouwd, omdat er één gebied bits heeft die op de buitenste rand van de bitmap liggen.

Opgave

Definieer een klasse Petrischaal waarmee kolonies op een petrischaal kunnen geanalyseerd worden. De objecten van deze klasse moeten minstens de volgende methoden hebben:

- Een initialisatiemethode __init__ waaraan de locatie van een tekstbestand moet doorgegeven worden. Dit tekstbestand bevat een bitmap van de petrischaal die moet geanalyseerd worden.
- Een methode __str__ die een stringvoorstelling van de petrischaal teruggeeft. Deze stringvoorstelling geeft het rechthoekig rooster weer zoals het uit het opgegeven bitmapbestand werd gelezen. Naast lege bits (voorgesteld door spaties) en bits die bedekt worden door kolonies (voorgesteld door hekjes: #), kunnen bits daarnaast ook verwerkt zijn (voorgesteld door punten). Dit verwerken gebeurt door het aanroepen van de methode kolonies (zie verder).
- Een methode kolonie waaraan het rij- en kolomnummer van een bedekte bit in de bitmap moeten doorgegeven worden. De rijen van een bitmap worden genummerd van boven naar onder, de kolommen van links naar rechts, en de nummering start telkens vanaf nul. Indien de bit op de opgegeven positie niet bedekt is, moet de methode een AssertionError opwerpen met de tekst geen kolonie gevonden op positie (*rij*, *kolom*). Hierbij moeten de cursieve fragmenten ingevuld worden op basis van de argumenten die aan de methode doorgegeven werden. Anders moeten alle bits van het gebied van aangrenzende bedekte bits waartoe de gegeven bit behoort als verwerkt gemarkeerd worden (ze worden dus niet langer als bedekt beschouwd nadat de methode werd aangeroepen), en moet de methode teruggeven uit hoeveel bits dit gebied bestaat.
- Een methode ongedaan_maken die alle verwerkte bits van de bitmap terug als bedekt markeert. Merk op dat de verwerkte bits in de bitmap mogelijks tot meerdere kolonies kunnen behoren, bijvoorbeeld als de methode kolonie verschillende keren werd aangeroepen voordat de methode ongedaan_maken wordt aangeroepen.
- Gebruik de methoden kolonie en ongedaan_maken om een methode aantal te schrijven die het aantal kolonies teruggeeft die niet kleiner zijn dan de opgegeven minimale grootte. De minimale grootte kan doorgegeven worden aan de optionele parameter minimum

(standaardwaarde: 1). De grootte van een kolonie wordt uitgedrukt als het aantal bits in het overeenkomstige gebied van aaneengesloten bedekte bits in de bitmap. Het gebruik van een minimale grootte is belangrijk om kleine artefacten uit te sluiten die kunnen ontstaan bij het omzetten van de foto van de petriplaat naar de overeenkomstige bitmap. We merken hier opnieuw op dat gebieden van aaneengesloten bedekte bits die bits op de buitenste rand van de bitmap bevatten niet als kolonie mogen beschouwd worden. Deze gebieden moeten dus genegeerd worden als de methode de kolonies telt. Zorg er voor dat de bitmap geen verwerkte bits heeft nadat de methode werd aangeroepen.

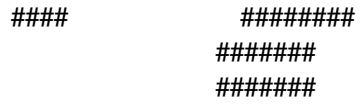
- Gebruik de methoden kolonie en ongedaan_maken om een methode grootte te schrijven die de gemiddelde grootte teruggeeft van alle kolonies die niet kleiner zijn dan de opgegeven minimale grootte. De minimale grootte kan doorgegeven worden aan de optionele parameter minimum (standaardwaarde: 1). Net zoals de vorige methode mag ook deze methode de gebieden van aaneengesloten bedekte bits die bits op de buitenste rand van de bitmap bevatten niet als kolonies beschouwen, en mag de bitmap ook geen verwerkte bits vertonen na het aanroepen van de methode. De methode moet de waarde None teruggeven indien er geen kolonies zijn die groter of gelijk zijn dan de opgegeven minimale grootte.

Voorbeeld

Bij onderstaande voorbeeldsessie gaan we ervan uit dat het tekstbestand schaal.txt zich in de huidige directory bevindt. Dit bestand bevat de bitmap van een petriplaat die hierboven grafisch werd weergegeven.

```
>>> schaal = Petrischaal('schaal.txt')
>>> print(schaal)
<BLANKLINE>
                ###
                #####
                #######
               #######
                ######
                ###### ##
                #### #####
                  ## ######     ####
                #   ######    ####
              ### ##########  #####
             ####### #### ##  ######
             ######### ##  #    #####
       #      ####  ###      ###
      #####     ####  #   ##   ##
      #####           ######  #
     ######           ########
      ####            ########
                     #######
                     #######
>>> schaal.kolonie(10, 35)
42
>>> print(schaal)
<BLANKLINE>
                ###
                #####
                #######
               #######
```

```
                ######
                ###### ..
                ####  .....
                  ## ......       ####
                #    ......      ####
               ### ..........    #####
              ######## ....  ..   ######
              ######### ..   .     #####
         #      ####   ...         ###
        #####     ####    .    ##    ##
        #####              ######    #
       ######              ########
        ####               ########
                          #######
                          #######
>>> schaal.kolonie(10, 40)
Traceback (most recent call last):
AssertionError: geen kolonie gevonden op positie (10, 40)
>>> schaal.kolonie(10, 45)
30
>>> print(schaal)
<BLANKLINE>
                ###
                #####
                #######
               #######
                ######
                ###### ..
                ####  .....
                  ## ......        ....
                #    ......       ....
               ###  ..........    .....
              ######## ....  ..   ......
              ######### ..   .     .....
         #      ####   ...         ...
        #####     ####    .    ##    ..
        #####              ######    .
       ######              ########
        ####               ########
                          #######
                          #######
>>> schaal.ongedaan_maken()
>>> print(schaal)
<BLANKLINE>
                ###
                #####
                #######
               #######
                ######
                ###### ##
                ####  #####
                  ## ######      ####
                #   ######     ####
               ### ##########   #####
              ######## ####  ##  ######
              ######### ##   #    #####
         #      ####  ###       ###
        #####     ####   #    ##    ##
        #####              ######   #
       ######              ########
```



```
>>> schaal.kolonies()
5
>>> schaal.koloniegrootte()
32.2
```