COMP 204

Review and final exam preparation

Yue Li

CSUS review session and helpdesk hours

CSUS has planned a COMP 204 review session run by Helpdesk tutors for **Friday, April 26th from 6pm-9pm in MAASS 112**

If this time does not work for you, they are also welcome to drop by the Helpdesk (Trottier 3090) from 10am-5pm on any weekday until classes end with any questions you may have.

Final exam info

- Date: **April 30, 6:30-9:30 PM**; Location: TBA this week
- Weight: 35% of your final grade (or 55% if better than midterm grade for students who opted the second non-programming midterm assignment option)
- Closed book but 8.5 x 11 double-sided crib sheet allowed.
- Questions:
  - 9 multiple choice questions (total 27%). Answer on Scantron (**not on exam**). Follow instructions for each questions: For some questions you need to indicate the only ONE correct answer. For other questions you need to indicate ALL correct answers.
  - Answer the rest of the questions directly on exam
  - 8 short answer questions (7 questions each worth 4 points and 1 question worth 5 points) (total: 33%).
  - 1 bonus short answer question worth 5 point.
  - 4 long answer questions (10 point per question; total: 40%).

Final exam content

Main materials that are covered in the final exam include:

- Basics: functions, loops, variables, data types (string, list, tuple, dictionary, sets), difference between pass by copy and pass by memory addresses
- Algorithms: Searching (linear and binary search) and sorting (insertion and selection sort)
- Pattern searching by string indexing and regular expression (simple ones)
- Object oriented programming: class, attributes, class inheritance, class methods
- BioPython sequence handling covered in class (I will remind you what the methods are in the exam)
- Machine learning: know what supervised, unsupervised, reinforcement learning are, problems they can solve, TPR, FPR, overfitting, cross-validation, ROC, decision trees
- Image processing: basic understanding of going from a pixel in the image to numpy ndarray
- What to memorize? Nothing. Use cribsheet to note the

Preparing for the final exam

How best to prepare for the exam:

- Practice, practice, practice.
- Review all lecture notes, assignment solutions, midterm solutions
- Practice on the problems we've posted on MyCourses-Content
- Attend CSUS review session:
- Come to my office hours: Wednesday: 11:30-12:30
- Appointments outside of the office hours are also welcome

The following practice questions are display without the answer and the answer is shown the next slide.

Try to answer the question first before seeing the answer.

If applicable, try answering the same questions with different function arguments.

Functions

What prints out?

```
1  def myfun(x, y):
2      x = x + 1
3      y = y + 1
4      return x + y
5
6  x = 0
7  y = 1
8  z = myfun( myfun(x,y), x)
9  print(z)
```

Functions

What prints out?

```
1  def myfun(x, y):
2      x = x + 1
3      y = y + 1
4      return x + y
5
6  x = 0
7  y = 1
8  z = myfun( myfun(x,y), x)
9  print(z)
```

Answer: 5

Functions (pass by memory address)

What prints out?

```
1  def myfun(x, y):
2      x[0] = x[0] + 1
3      y[0] = y[0] + 1
4      return [x[0] + y[0]]
5
6  x = [0]
7  y = [1]
8  z = myfun( myfun(x,y), x)
9  print(z)
```

Functions (pass by memory address)

What prints out?

```
1  def myfun(x, y):
2      x[0] = x[0] + 1
3      y[0] = y[0] + 1
4      return [x[0] + y[0]]
5
6  x = [0]
7  y = [1]
8  z = myfun( myfun(x,y), x)
9  print(z)
```

Answer: [6]

Linear and binary search

How to search number 9 in this list by linear search and binary search? [2,5,7,9,10]

Linear and binary search

How to search number 9 in this list by linear search and binary search? [2,5,7,9,10]
Answer:

- linear search: search on number at a time starting from the first element in the list. Total takes 4 comparisons (i.e., $2 \neq 9$, $5 \neq 9$, $7 \neq 9$, $9 == 9$)
- binary search: start at the middle of the list, compare whether 9 is greater than that number which is 7; because 9 is greater than 7, continue the next search by comparing 9 with the number in the middle of the second half of the list (i.e., 9). 9 is found in index 3.

Selection and insertion sort

How to sort the following list by selection sort and insertion sort?
[2,10,5,9,7]

Selection and insertion sort

How to sort the following list by selection sort and insertion sort?
[2,10,5,9,7]
Answer:

Selection sort:
unsorted list: [2,10,5,9,7]
start with value in index 0: 2
find the index of the minimum value in [2,10,5,8,7] (4 comparisons): 2
insert the minimum value 2 in index 0: [2,10,5,9,7]
start with value in index 1: 10
find the index of the minimum value in [10,5,8,7] (3 comparisons): 5
insert the minimum value 5 in index 1: [2,5,10,9,7]
start with value in index 2: 10
find the index of the minimum value in [10,9,7] (2 comparisons): 7
insert the minimum value 7 in index 2: [2,5,7,9,10]
start with value in index 3: 9
find the index of the minimum value in [9,10] (1 comparisons): 9
insert the minimum value 9 in index 3: [2,5,7,9,10]
Selection sort takes 10 comparison

Insertion sort:

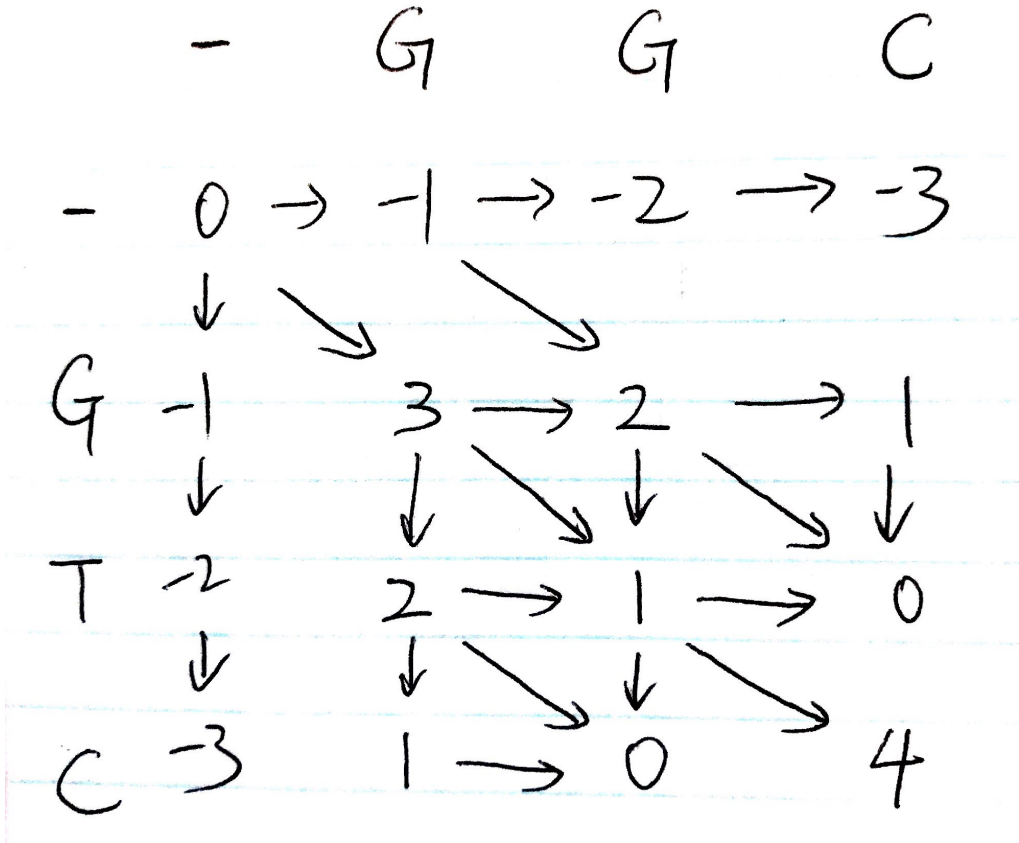
unsorted list: [2,10,5,9,7]
pick up 10
$[2, \_, 5, 9, 7]$ compare 10 with 2 ($10 > 2$), insert 10 at index 1
$[2, 10, 5, 9, 7]$
pick up 5
$[2, 10, \_, 9, 7]$ compare 5 with 10 ($5 < 10$), shift 10 to the right next index
$[2, \_, 10, 9, 7]$
compare 5 with 2 ($5 > 2$), insert 5 at index 1
$[2, 5, 10, 9, 7]$
pick up 10
$[2, 5, \_, 9, 7]$ compare 10 with 5 ($10 > 5$), insert 10 back to index 2
$[2, 5, 10, 9, 7]$
pick up 9
$[2, 5, 10, \_, 7]$ compare 9 with 10 ($9 < 10$), shift 10 to the right next index
$[2, 5, \_, 10, 7]$ compare 9 with 5 ($9 > 5$), insert 9 at index 2
$[2, 5, 9, 10, 7]$
pick up 10
$[2, 5, 9, \_, 7]$
compare 10 with 9 ($10 > 9$), insert 10 back to index 3
$[2, 5, 9, 10, 7]$
pick up 7
$[2, 5, 9, 10, \_]$ compare 7 with 10 ($7 < 10$), shift 10 to the right next index
$[2, 5, 9, \_, 10]$ compare 7 with 9 ($7 < 9$), shift 9 to the right next index
$[2, 5, \_, 9, 10]$
compare 7 with 5 ($7 > 5$), insert 7 at index 2
$[2, 5, 7, 9, 10]$
Insertion sort takes 10 comparisons

Sequence alignment (A2)

Given match score $+3$, mismatch score -2, gap score -1. What's the similarity score between sequence GGC with sequence GTC?

Sequence alignment (A2)

Given match score +3, mismatch score -2, gap score -1. What's the similarity score between sequence GGC with sequence GTC?
Answer:



List comprehension

Convert the following for loop into list comprehension with one line of code:

```
1 x = []
2 for i in range(5):
3     x.append(-2*i)
```

List comprehension

Convert the following for loop into list comprehension with one line of code:

```
1 x = []
2 for i in range(5):
3     x.append(-2*i)
```

Answer:

```
x=[-2*i for i in range(5)]
```

String pattern matching

Choose ALL of the correct boolean expression(s) that will match with a string s that starts with AUG and ends with stop codon UAG, UAA, and UGA

A
```
s[0:3]=="AUG" and s[-3:] in ["UAG", "UAA", "UGA"]
```

B
```
s[0:3]=="AUG" and s[len(s)-3:len(s)] in ["UAG",
"UAA", "UGA"]
```

C
```
re.research("^AUG.*(UAG|UAA|UGA)$", s)
```

D
```
re.research("AUG.*(UAG|UAA|UGA)", s)
```

E
```
re.research("^AUG.*[UAG|UAA|UGA]$", s)
```

F
```
s == "^AUG.*(UAG|UAA|UGA)$"
```

String pattern matching

Choose ALL of the correct boolean expression(s) that will match with a string s that starts with AUG and ends with stop codon UAG, UAA, and UGA Answer

A (correct) `s[0:3]=="AUG" and s[-3:] in ["UAG", "UAA", "UGA"]`

B (correct) `s[0:3]=="AUG" and s[len(s)-3:len(s)] in ["UAG", "UAA", "UGA"]`

C (correct) `re.search("^AUG.*(UAG|UAA|UGA)$", s)`

D `re.search("AUG.*(UAG|UAA|UGA)", s)`

E `re.search("^AUG.*[UAG|UAA|UGA]$", s)`

F `s == "^AUG.*(UAG|UAA|UGA)$"`

Object oriented programming: attributes

What are attributes in `Myclass`

```
1  class MyBus:
2      def __init__(self, stationID, passengers):
3          self.s = stationID
4          self.p = passengers
5          terminal = 0
```

Object oriented programming: attributes

What are attributes in `Myclass`

```
1  class MyBus:
2      def __init__(self, stationID, passengers):
3          self.s = stationID
4          self.p = passengers
5          terminal = 0
```

Answer:

s and p

Object oriented programming: methods

What prints out?

```
1  class Animal:
2      def __init__(self):
3          self.age = 0
4          self.claws=0
5      def grow(self):
6          self.age += 1
7          claws = self.claws + 1
8  animal = Animal()
9  animal.grow()
10 print(animal.age)
11 print(animal.claws)
```

Object oriented programming: methods

What prints out?

```
1  class Animal:
2      def __init__(self):
3          self.age = 0
4          self.claws=0
5      def grow(self):
6          self.age += 1
7          claws = self.claws + 1
8  animal = Animal()
9  animal.grow()
10 print(animal.age)
11 print(animal.claws)
```

Answer:

1

0

Object oriented programming: class inheritance

What prints out?

```
1  class Animal():
2      def __init__(self):
3          self.age = 0
4          self.claws=0
5      def grow(self):
6          self.age += 1
7          claws = self.claws + 1
8  class Predator(Animal):
9      def __init__(self):
10         Animal.__init__(self)
11         self.horns = 0
12         self.eyes = 0
13     def grow(self):
14         Animal.grow(self)
15         self.horns += 1
16         eyes = self.eyes + 1
17
18 pred = Predator()
19 pred.grow()
20 print(pred.claws, pred.age, pred.horns, pred.eyes)
```

Object oriented programming: class inheritance

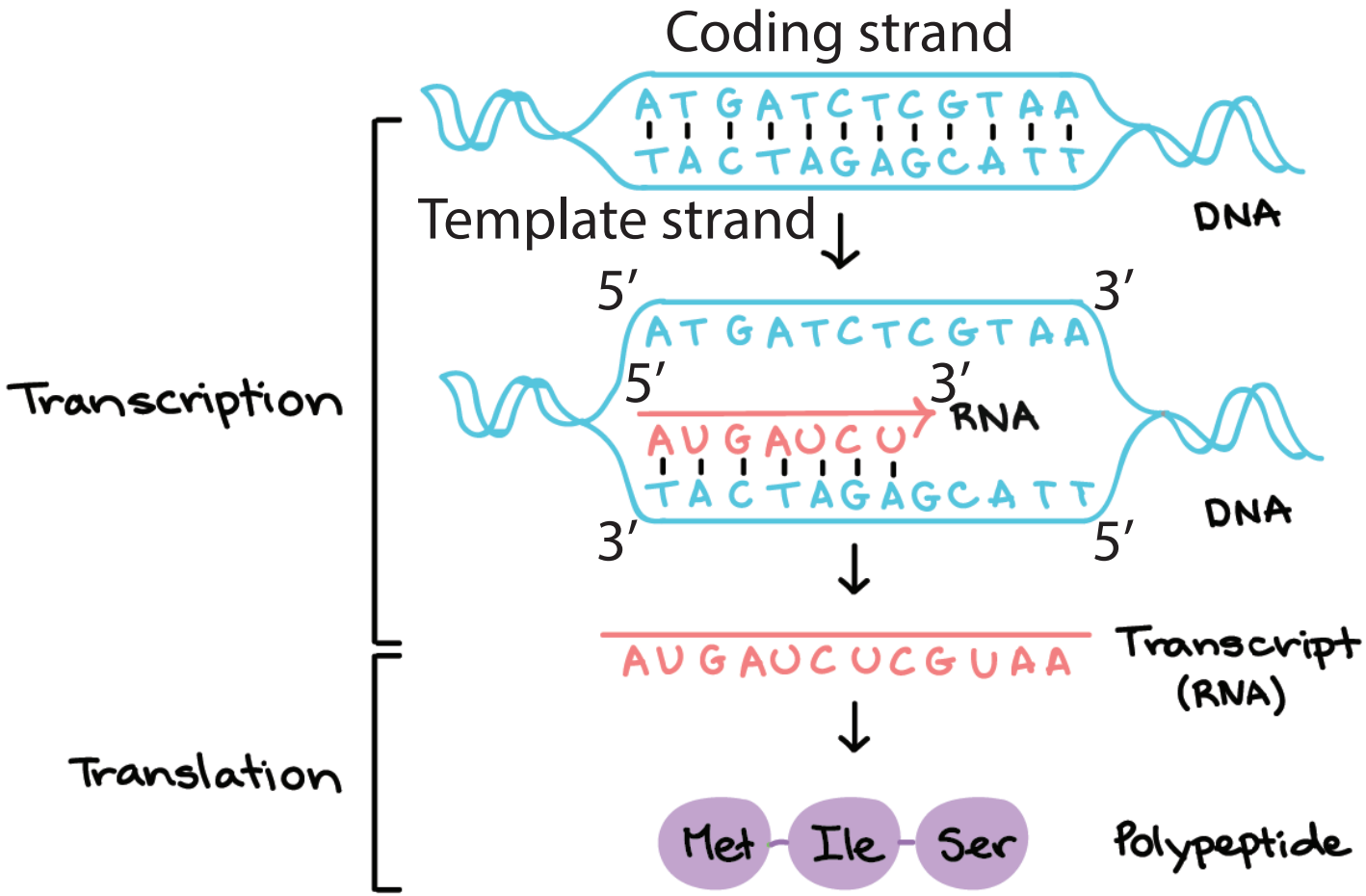
What prints out?

```
1  class Animal():
2      def __init__(self):
3          self.age = 0
4          self.claws=0
5      def grow(self):
6          self.age += 1
7          claws = self.claws + 1
8  class Predator(Animal):
9      def __init__(self):
10         Animal.__init__(self)
11         self.horns = 0
12         self.eyes = 0
13     def grow(self):
14         Animal.grow(self)
15         self.horns += 1
16         eyes = self.eyes + 1
17
18 pred = Predator()
19 pred.grow()
20 print(pred.claws, pred.age, pred.horns, pred.eyes)
```

Answer:

0 1 1 0

Central dogma



Every three DNA letters (i.e., codon) code for an amino acid

Transcription

Given a DNA string as the *template* strand say 5'-AGATCAT-3', write a function called `transcribe(dna)` that returns the transcribed RNA sequence (i.e., AUGAUCU)

```
1  def transribe(dna):
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19 return rna
```

Transcription

Given a DNA string as the template strand say 5'-AGATCAT-3', write a function called `transcribe(dna)` that returns the transcribed RNA sequence (i.e., AUGAUCU)

```
1  def transcribe(dna):
2      rna=""
3      dt = {"A":"U","T":"A","G":"C","C":"G"}
4      for i in dna[::-1]:
5          rna = rna + dt[i]
6      return rna
7
8  print(transcribe("AGCTAC")) # GUAGCU
```

Translation: codon table

1st base	2nd base								3rd base
	T		C		A		G		
T	TTT	(Phe/F) Phenylalanine	TCT	(Ser/S) Serine	TAT	(Tyr/Y) Tyrosine	TGT	(Cys/C) Cysteine	T
	TTC		TCC		TAC		TGC		C
	TTA	(Leu/L) Leucine	TCA		TAA	Stop (*Ochre*)[B]	TGA	Stop (*Opal*)[B]	A
	TTG[A]		TCG		TAG	Stop (*Amber*)[B]	TGG	(Trp/W) Tryptophan	G
C	CTT		CCT	(Pro/P) Proline	CAT	(His/H) Histidine	CGT	(Arg/R) Arginine	T
	CTC		CCC		CAC		CGC		C
	CTA		CCA		CAA	(Gln/Q) Glutamine	CGA		A
	CTG[A]		CCG		CAG		CGG		G
A	ATT	(Ile/I) Isoleucine	ACT	(Thr/T) Threonine	AAT	(Asn/N) Asparagine	AGT	(Ser/S) Serine	T
	ATC		ACC		AAC		AGC		C
	ATA		ACA		AAA	(Lys/K) Lysine	AGA	(Arg/R) Arginine	A
	ATG[A]	(Met/M) Methionine	ACG		AAG		AGG		G
G	GTT	(Val/V) Valine	GCT	(Ala/A) Alanine	GAT	(Asp/D) Aspartic acid	GGT	(Gly/G) Glycine	T
	GTC		GCC		GAC		GGC		C
	GTA		GCA		GAA	(Glu/E) Glutamic acid	GGA		A
	GTG		GCG		GAG		GGG		G

Not all mutations lead to a different animo acid

e.g., GCT and GCC both code for Alanine

Translation

Assume the codon table is provided to you as a dictionary `ct` with key as the 3-letter DNA string and value as the amino acid, write a function that translates an RNA into the amino acid sequence

```
1  def translate(rna, ct):
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19 return aa
```

Translation

Assume the codon table is provided to you as a dictionary `ct` with key as the 3-letter DNA string and value as the amino acid, write a function that translates an RNA into the amino acid sequence

```
1  def translate(rna, ct):
2      aa = ""
3      for i in range(0, len(rna)-3, 3):
4          aa = aa + ct[rna[i:i+3]]
5      return aa
```

Get candidate cancer cell

Suppose we obtain a collection of unknown cells from a patient. Each cell is a Cell object. We are provided with a function called `cancer_cell_score(cell)` that gives a cancer score to the unknown cell. Write a function that return the highest scoring cell.

```
1  def get_candidate_cancer_cell(unknown_cells):
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17 return ccc # candidate cancer cell
```

Get candidate cancer cell

Suppose we obtain a collection of unknown cells from a patient. Each cell is a Cell object. We are provided with a function called `cancer_cell_score(cell)` that gives a cancer score to the unknown cell. Write a function that return the highest scoring cell.

```
1  def get_candidate_cancer_cell(unknown_cells):
2      high_score = 0
3      for cell in unknown_cells:
4          myscore = cancer_cell_score(cell)
5          if myscore > high_score:
6              ccc=cell
7              high_score = myscore
8      return ccc
```

Average number of cell-type-specific cells per patient

Suppose we obtain a collection of single cells with known cell types from a set of cancer patient blood samples. We stored this as a dictionary with key as patient ID and values as cell types. For example (not real data):

```
singlecells = {
"patient0":["B-Cell", "B-Cell", "T-Cell", "Neutrophils"],
"patient1":["T-Cell", "T-Cell", "Neutrophils"],
"patient2":["B-Cell", "Neutrophils", "Cytokines"],
"patient3":["B-Cell", "B-Cell", "Cytokines"]}
```

Write a function to calculate the average number of cell-type-specific cells per patient sample.
Expected output:

```
>>> print(average_cell_types(singlecells))

{'B-Cell': 1.25, 'T-Cell': 0.75, 'Neutrophils': 0.75,
↪  'Cytokines': 0.5}
```

next page

```
1  def average_cell_types(singlecells):
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23 return avg
```

```python
def average_cell_types(singlecells):

    avg = {}
    for patId, cells in singlecells.items():
        for c in cells:
            if c not in avg:
                avg[c] = 0
            avg[c] += 1

    for ct in avg.keys():
        avg[ct] = avg[ct]/len(singlecells)

    return avg


singlecells = {
    "patient0":["B-Cell", "B-Cell", "T-Cell",
    ↪  "Neutrophils"],
    "patient1":["T-Cell", "T-Cell", "Neutrophils"],
    "patient2":["B-Cell", "Neutrophils", "Cytokines"],
    "patient3":["B-Cell", "B-Cell", "Cytokines"]}

print(average_cell_types(singlecells))
# {'B-Cell': 1.25, 'T-Cell': 0.75, 'Neutrophils': 0.75,
↪  'Cytokines': 0.5}
```

Find all palindromic DNA sequences

You are provided with `is_pal` function that determine whether a DNA sequence is palindromic. The reverse complement of a palindromic sequence is the same sequence. Write a function that returns a list of all palindromic sequences in a DNA sequence.

```
1  def find_all_palindromes(dna):
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17      return all_pals
```

Expected output:

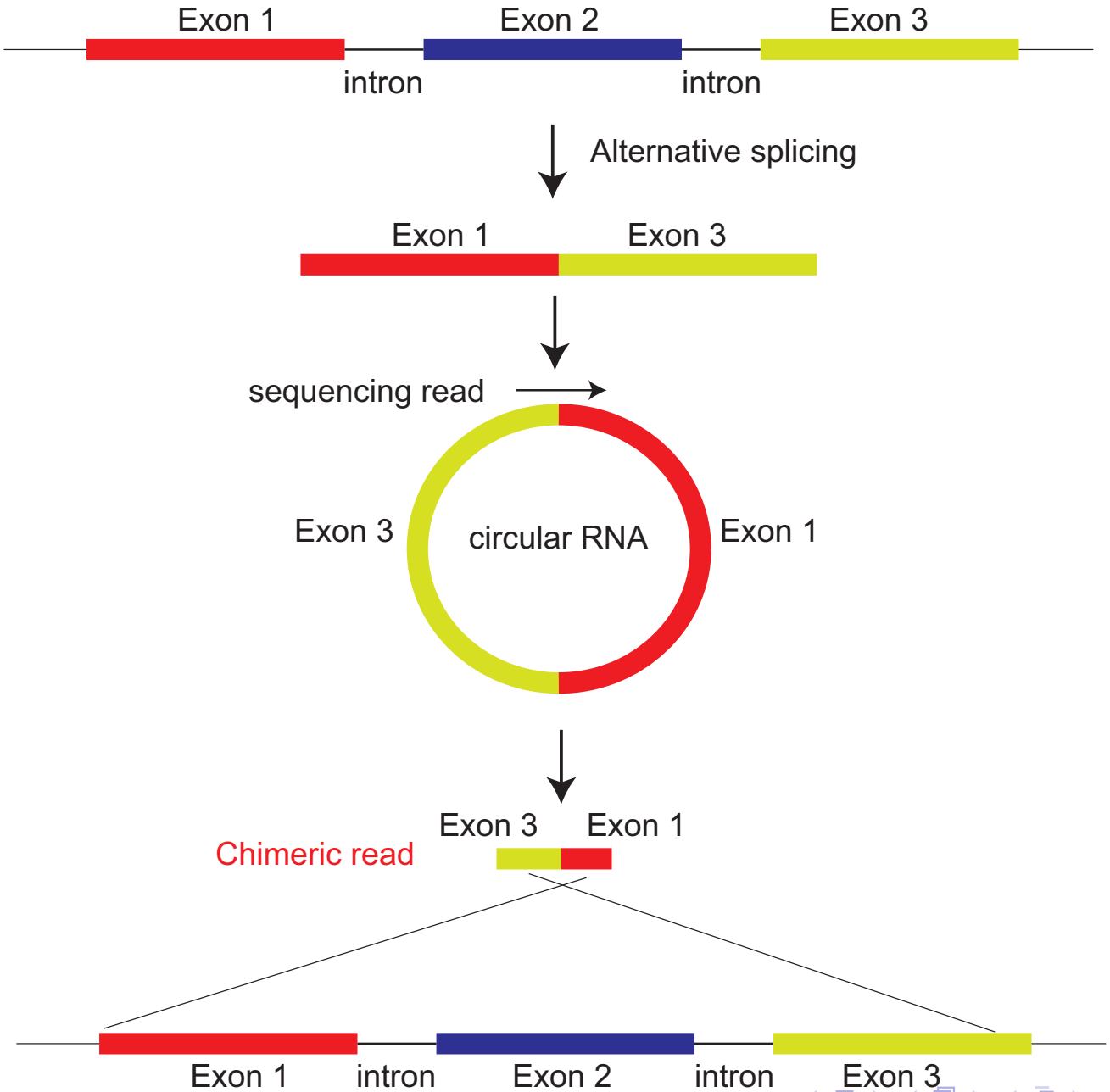
```
>>> print(find_all_palindromes("AGAATTCG"))
['GAATTC', 'AATT', 'AT', 'CG']
```

Find all palindromic DNA sequences

You are provided with `is_pal` function that determine whether a DNA sequence is palindromic. The reverse complement of a palindromic sequence is the same sequence. Write a function that returns a list of all palindromic sequences in a DNA sequence.

```
1  def is_pal(dna):
2      dt = {"A":"T","T":"A","G":"C","C":"G"}
3      dna_rc = ""
4      for i in dna[::-1]:
5          dna_rc = dna_rc + dt[i]
6      return dna == dna_rc
7
8  def find_all_palindromes(dna):
9      all_pals=[]
10     for i in range(len(dna)-1):
11         for j in range(i+1, len(dna)+1):
12             print(dna[i:j])
13             if is_pal(dna[i:j]):
14                 all_pals.append(dna[i:j])
15     return all_pals
16
17 print(find_all_palindromes("AGAATTCG"))
18 # ['GAATTC', 'AATT', 'AT', 'CG']
```

Circular RNA and chimeric reads



Determine chimeric read

Write a function that determines whether a read is a chimeric read. The function takes the candidate read sequence and an ordered list of exon sequences as they appear in the genome (i.e., exon1 is upstream of exon2, and exon2 is upstream of exon3, and so on). Assume exon with the longest match to the first half of the read is the exon that the read truly comes from.
Expected outputs:

```
>>> print(is_chimeric("AUGGCC", ["CGUAUG", "GCCUCA",
↪   "AUGCGU"]))
False
>>> print(is_chimeric("AUGGCC", ["GCCUCA", "CGUAUG",
↪   "AUGCGU"]))
True
```

```
def is_chimeric(read, exons):



return is_chim
```

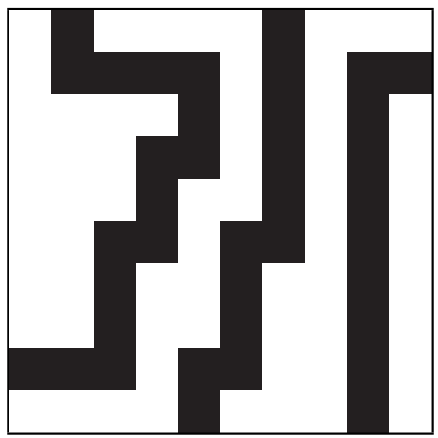
```python
def is_chimeric(read, exons):
    j = 0
    longest_match = 0
    for e in exons:
        for i in range(len(read)):
            if read[0:i] in e:
                if i + 1 > longest_match:
                    first_exonId = j
                    longest_match = i + 1
        j += 1

    j=0
    if longest_match < len(read):
        for e in exons:
            if read[i+1:len(read)] in e:
                second_exonId = j
                if first_exonId > second_exonId:
                    return True
    return False

print(is_chimeric("AUGGCC", ["CGUAUG", "GCCUCA", "AUGCGU"])) # False
print(is_chimeric("AUGGCC", ["GCCUCA", "CGUAUG", "AUGCGU"])) # True
```

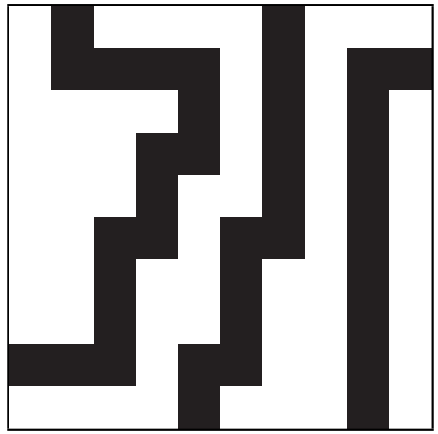
A simple maze

As shown below, we are given a 10×10 greyscale image showing "black tunnels" with entries and exits on the edges of the image.

- For each tunnel, there is exactly one path from the entry to the exit by going through the black pixels.
- Entry and exit are not the same pixel
- There is no diagonal turn in the tunnel.
- Each tunnel is perfectly separated by the white background pixels from another tunnel.

Write a function `get_tunnels_info(img)` to return a compound tuple of tuples, each tuple contains `(tunnel_size, (entry_i, entry_j), (exit_i, exit_j))`





In the above image, the expected output by running the function is:

```
import numpy as np
import skimage.io as io
from skimage.color import rgb2gray
import matplotlib.pyplot as plt

>>> img = rgb2gray(io.imread("maze.eps"))
>>> print(get_tunnels_info(img))
[(16, (0, 1), (8, 0)), (12, (0, 6), (9, 4)), (10, (9,
 ↪  8), (1, 9))]
```

For testing your code, you can download the original maze.eps image file by clicking on this link

- First, let's start implementing a helper function called `explore_tunnel(img, cur_i, cur_j)`
- It will fill a tunnel with white background based on one of the pixel at `cur_i,cur_j` location of the tunnel.
- If `cur_i,cur_j` is the pixel for the entry or the exit, we can start from that pixel and start moving along the tunnel.
- Otherwise, we will need to find out where the pixel for the entry or exit of the tunnel is before moving on. Otherwise, we will not get the correct answer.
- For example, if we go left to right and top to bottom pixel by pixel, the first pixel we find for the tunnel on the right is neither an entry nor an exit.

```
1  def explore_tunnel(img, cur_i, cur_j):
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25      return (tunnel_size, (entry_i, entry_j), (exit_i,
        ↪  exit_j))
```

Now complete the function `get_tunnels_info(img)` that uses `explore_tunnel`

```
1  def get_tunnels_info(img):
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22      return tunnels_info
```

```python
def explore_tunnel(img, cur_i, cur_j):

    i,j = cur_i, cur_j
    n_rows,n_cols = img.shape

    # determine the entry/exit
    if i in [0,n_rows-1] or j in [0,n_cols-1]:
        entry_i = i
        entry_j = j
    elif sum(img[i:n_rows,j]!=1) == n_rows-i:
        entry_i = n_rows - 1
        i = entry_i
        entry_j = j
    elif sum(img[i,j:n_cols]!=1) == n_cols-j:
        entry_i = i
        entry_j = n_cols - 1
        j = entry_j

    tunnel_size = 1
    img[i,j] = 1

    while (i==entry_i and j==entry_j) or (i not in [0, n_rows-1] and j
    ↪  not in [0, n_cols-1]):

        # try four choices
        if img[i, max(0, j-1)] != 1: # left
            j = max(0, j-1)
        elif img[i, min(j+1, n_cols-1)] != 1: # right
            j = min(j+1, n_cols-1)
        elif img[max(0, i-1),j] != 1: # top
            i = max(0, i - 1)
        elif img[min(i+1,n_rows-1), j] != 1: # bottom
            i = min(i + 1, n_rows-1)

        img[i,j] = 1
        tunnel_size += 1

    exit_i = i
    exit_j = j
    return (tunnel_size, (entry_i, entry_j), (exit_i, exit_j))
```

```
41  # for showing intermediate image only (not required in the exam)
42  def imshow(img):
43      if (img!=1).any():
44          plt.imshow(img, cmap="gray")
45          ax = plt.gca()
46          ax.set_xticks(np.arange(0,10))
47          ax.set_yticks(np.arange(0,10))
48          plt.grid(b=True, which='both', color='b')
49          plt.show()
50
51
52  def get_tunnels_info(img):
53      background=1
54      nb_tunnels = 0
55      n_rows,n_cols = img.shape
56
57      tunnels_info = []
58
59      imshow(img)
60
61      for i in range(n_rows):
62          for j in range(n_cols):
63
64              if img[i,j] != background:
65
66                  nb_tunnels += 1
67
68                  tunnel_size, (entry_i,entry_j), (exit_i,exit_j) =
                    ↪  explore_tunnel(img, i, j)
69
70                  tunnels_info.append((tunnel_size, (entry_i,entry_j),
                    ↪  (exit_i,exit_j)))
71
72                  imshow(img)
73
74      return tunnels_info
```