Homework 4 - Python control structures

BIOINF 575 - Fall 2021

Total 50 points

For each each problem part provide the solution in Code cells after the description of the problem part. Answers to questions should be written either as comments together with the code or in Markdown cell(s) for each part of the problem.

This homework will require the use of data types, control structures (with a focus on functions) and file read and write operations

FUNCTION - block of code that only runs when called and can be reused without having to copy the code

```
def function_name(arguments):
    <statements>
    return result

# Call the function:
function_name(values)
```

FILE - collection of data stored and identified as a unit by the operating system

```
# Open file and return a stream.  Raise OSError upon failure.
open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None,
closefd=True, opener=None)


with open(file_name, "r") as file
    for line in file:
        <statements>
```

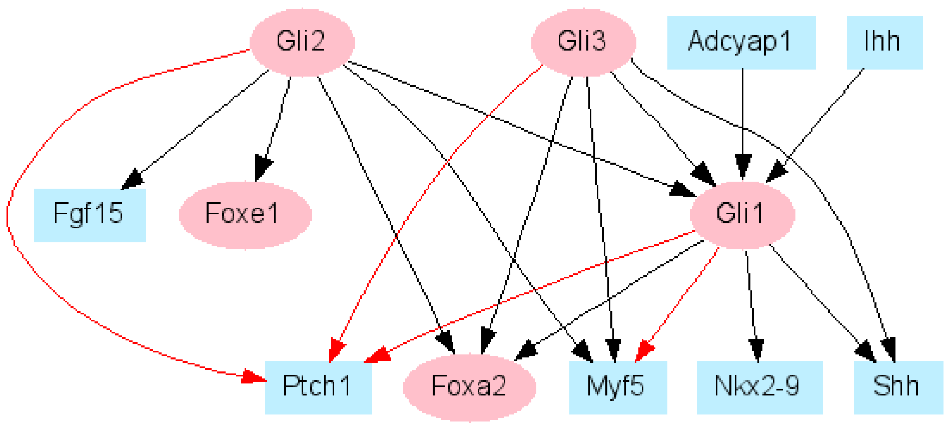
The result of the open function is an iterable object.
Iterable: An object capable of returning its members one at a time.
https://docs.python.org/3/glossary.html
https://docs.python.org/3/library/collections.abc.html?highlight=iterable#collections.abc.Iterable

In [1]:
```
#help(open)
```

Problem 1 - Exploring regulatory networks

To understand gene regulation, accurate and comprehensive knowledge of transcriptional regulatory elements is essential. Transcriptional Regulatory Element Database (TRED) has been built in response to increasing needs of an integrated repository for both cis- and trans- regulatory elements in mammals, and the lack of such resources at present.
http://rulai.cshl.edu/cgi-bin/TRED/tred.cgi?process=home

Gene Regulatory Networks for the 36 Transcription Factor families in human, mouse, and rat.
http://rulai.cshl.edu/TRED/GRN/grn.htm



Ellipses are transcription factors (TFs).

Boxes are genes.

Red lines indicate the protein-DNA binding is known.

http://rulai.cshl.edu/TRED/GRN/Gli.htm

Genes and TFs are network nodes.

The dictionary below is constructed based on the above network to store nodes that directly affect at least one node as keys and the nodes they directly affect in a set as values.

```
biological_network = {
    "Adcyap1": {"Gli1"},
    "Gli1": {"Ptch1", "Foxa2", "Myf5", "Nkx2-9", "Shh"},
    "Gli2": {"Ptch1", "Fgf15", "Foxe1", "Foxa2", "Myf5", "Gli1"},
    "Gli3": {"Ptch1", "Foxa2", "Myf5", "Gli1", "Shh"},
    "Ihh": {"Gli1"}
}
```

....

Part 1 (10 points) - Finding genes regulated by transcription factors.

Use the dictionary and figure above to:

- **(1 point)** Create a tuple with the transcription factors in the network (pink elipses in the figure)
  - Write the transcription factor symbols (pink elipses in the figure) manually in a tuple
- **(1 point)** Create a tuple with the genes in the network (blue boxes in the figure)
  - Write the gene symbols (light blue boxes in the figure) manually in a tuple
- **(7 points)** Write a function that computes the set of GENES that a given TRANSCRIPTION FACTOR directly regulates in the network
  - Make proper validations:
    - e.g.: Make sure to check that what the function receives as a transcription factor is really a transcription factor, i.e. is in the transcription factors tuple
  - Make use of the functionality available for the data structures you are using

- Include function documentation. -**(1 point)** Test your function for the following 4 transcription factors: Gli1, Gli2, Gli3 and Foxa2.

In [174…

```
#Write your solution here, feel free to add new cells.
##main goal of this function is to determine the genes that are regulated
#by transcription factors.

biological_network = {
    "Adcyap1": {"Gli1"},
    "Gli1": {"Ptch1", "Foxa2", "Myf5", "Nkx2-9", "Shh"},
    "Gli2": {"Ptch1", "Fgf15", "Foxe1", "Foxa2", "Myf5", "Gli1"},
    "Gli3": {"Ptch1", "Foxa2", "Myf5", "Gli1", "Shh"},
  "Ihh": {"Gli1"}
}

#needed tf and gene tuples.
TF_tuple = ("Gli1", "Gli2", "Gli3", "Foxe1", "Foxa2")
genes_tuple = ("Fgf15", "Ptch1", "Adcyap1", "Ihh", "Myf5", "Nkx2-9", "Shh")

#defining the function with documentation.
def testing_genes(tf, network):
    '''
    Function computes set of genes that are regulated by a transcription factor.

    Parameters: tf (transcription factor) and network (your dictionary).

    Function returns a set of specific genes or prints statements if the transcription
    factor does not directly regulate genes in the dictionary or if the value for
    the tf parameter is not a transcription factor.
    '''
    gene_set = set()
    key_list = []
    for key in network.keys():
        key_list.append(key)

    if tf in TF_tuple:
        if tf in key_list:
            for values in (network[tf]):
                if values in genes_tuple:
                    gene_set.add(values)
        elif tf not in key_list:
            print("This transcription factor does not directly regulate any genes withi
    else:
        print("This is not a transcription factor.")

    return gene_set
```

In [175…

```
testing_genes("Gli1", biological_network)
```

Out[175…

```
{'Myf5', 'Nkx2-9', 'Ptch1', 'Shh'}
```

In [176…

```
testing_genes("Gli2", biological_network)
```

Out[176…

```
{'Fgf15', 'Myf5', 'Ptch1'}
```

In [177…
```
testing_genes("Gli3", biological_network)
```

Out[177…
```
{'Myf5', 'Ptch1', 'Shh'}
```

In [178…
```
testing_genes("Foxa2", biological_network)
```

```
This transcription factor does not directly regulate any genes within this dictionary.
```

Out[178…
```
set()
```

In [179…
```
#I tested a gene to make sure I got the correct statement
#if it is not a transcription factor, which I did!

testing_genes("Fgf15", biological_network)
```

```
This is not a transcription factor.
```

Out[179…
```
set()
```

In [ ]:
```

```

…………………………………………………………………………………………

Part 2 (10 points) - Finding upstream genes in a regulatory network

- **(7 points)** Write a function that computes the set of transcription factors that a given gene is directly regulated by in the network given by the biological_network dictionary from part 1.
  - Make proper validations:
    - e.g.: Make sure to check that what the function receives as a transcription factor is really a gene, i.e. is in the genes tuple you created in part 1
  - Make use of the functionality available for the data structures you are using
  - Include function documentation.
- **(3 points)** Test your function for the following 4 genes: Nkx2-9, Ptch1, Myf5 and Shh
  - Use a for loop to get the results for each of the tested genes
- **(4 BONUS POINTS)**
  - Identify the common regulators for these 4 genes by computing the intersection of the results.

In [209…
```
##Write your solution here, feel free to add new cells.
##This function must compute the set of transcription factors
#that a given gene is directly regulated by in the network.

biological_network = {
    "Adcyap1": {"Gli1"},
    "Gli1": {"Ptch1", "Foxa2", "Myf5", "Nkx2-9", "Shh"},
    "Gli2": {"Ptch1", "Fgf15", "Foxe1", "Foxa2", "Myf5", "Gli1"},
    "Gli3": {"Ptch1", "Foxa2", "Myf5", "Gli1", "Shh"},
  "Ihh": {"Gli1"}
}

#needed tf and gene tuples.
TF_tuple = ("Gli1", "Gli2", "Gli3", "Foxe1", "Foxa2")
genes_tuple = ("Fgf15", "Ptch1", "Adcyap1", "Ihh", "Myf5", "Nkx2-9", "Shh")
```

```python
#defining the function with documentation.
def testing_tf(gene, network):
    '''
    Function computes the set of transcription factors that directly regulate
    a given gene.

    Parameters: gene (your choice of gene) and network (your dictionary).

    Function returns the set of specific transcription factors or prints statements
    if the gene is not regulated by transcription factors in the dictionary or if the
    value for the gene parameter is not a gene.
    '''
    tf_set = set()
    key_list = []
    for key in network.keys():
        key_list.append(key)


    for key in TF_tuple:
        if key in key_list:
            if gene in (network[key]):
                if gene in genes_tuple:
                    tf_set.add(key)

#This for loop below is hard code and not generalized. I attempted multiple times
#to make this part of the function general with the logic that if I called this
#function with a value for gene paramter that was in the genes_tuple but was not
#in any of the values for any of the keys in this dictionary, then I could tell
#the function to print "This gene is not regulated by transcription factors." I
#tried multiple ways to break these nested dictionaries apart to just get the values
#to then do either a for loop and/or an if statement but I kept getting my print
#statment printed multiple times instead of just once. So I remained with the
#hard code because it at least printed my statement once. Below is the general
#code I had but it was iterating.

#         test = set()
#         for value in biological_network.values():
#             test.update(value)
#             if gene not in test:
#                 print("This gene is not regulated by transcription factors.")

    for key in key_list:
        if gene in genes_tuple:
            if key == gene:
                print("This gene is not regulated by transcription factors.")

    if gene in TF_tuple:
        print("This is not a gene.")

    return tf_set
```

In [210…

```python
#testing the four genes (Nkx2-9, Ptch1, Myf5, and Shh) using a for loop.

test = ["Nkx2-9", "Ptch1", "Myf5", "Shh"]
for a in test:
    print(testing_tf(a, biological_network))
```

```
{'Gli1'}
```

```
{'Gli1', 'Gli2', 'Gli3'}
{'Gli1', 'Gli2', 'Gli3'}
{'Gli1', 'Gli3'}
```

In [211…

```
#Testing one gene that should not be regulated by any transcription factors.
testing_tf("Adcyap1", biological_network)
```

```
This gene is not regulated by transcription factors.
```

Out[211…

```
set()
```

In [212…

```
#Testing a TF to make sure we get the correct output.
testing_tf("Foxa2", biological_network)
```

```
This is not a gene.
```

Out[212…

```
set()
```

Problem 2 - Process a variant call format file to select variants of interest

20 points

We have variant data for a sample in the file `variant_data_file.vcf` , which contains information about approximatelly 1000 differences found in the genome (Y chromosome) of the specific sample when compared to the reference.

The description of the format for the vcf file version 4.1 is available in the following document: https://samtools.github.io/hts-specs/VCFv4.1.pdf

Also, below is the header of the vcf file explaining each element on the line above.

```
##fileformat=VCFv4.0
##fileDate=20100610
##source=glfTools v3
##reference=1000GenomesPilot-NCBI36
##phasing=NA
##INFO=<ID=NS,Number=1,Type=Integer,Description="Number of Samples With
Mapped Reads">
##INFO=<ID=DP,Number=1,Type=Integer,Description="Total Depth">
##INFO=<ID=DB,Number=0,Type=Flag,Description="dbSNP membership, build
129">
##INFO=<ID=H2,Number=0,Type=Flag,Description="HapMap2 membership">
##FILTER=<ID=NUYR,Description="Variant in non-unique Y region">
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
##FORMAT=<ID=GQ,Number=1,Type=Integer,Description="Genotype Quality">
##FORMAT=<ID=DP,Number=1,Type=Integer,Description="Depth">
##INFO=<ID=AC,Number=.,Type=Integer,Description="Allele count in
genotypes">
##INFO=<ID=AN,Number=1,Type=Integer,Description="Total number of alleles
in called genotypes">
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO
Y   2728456 rs2058276   T   C   32  .   AC=2;AN=2;DB;DP=182;H2;NS=65
```

....

Part 1 (10 points)

- **(8 points)** Write a function that takes as input the data for a variant, which has the format of a line in the `variant_data_file.vcf` and **returns the rs-identifier** if:
  - the variant has an identifier (e.g.: rs2058276, missing identifier is denoted by "." in the file) AND
  - has a quality of at least 35 AND
  - a total read depth higher than 180.
    If the conditions are not met return `None`.

Note: The missing identifier value and the thresholds for read depth and quality can be arguments in your funtions if you want to allow for flexibility in checking other thresholds.

- **(2 points)** Test your function with at least the following two test variant:

```
test_variant1 = "Y  2728456 rs2058276   T   C   32  .
AC=2;AN=2;DB;DP=182;H2;NS=65"

test_variant2 = "Y  2728456 rs2058260   T   C   37  .
AC=2;AN=2;DB;DP=182;H2;NS=65"
```

Below is the header and a line that contains variant data:

CHROM POS ID REF ALT QUAL FILTER INFO

Y 2728456 rs2058276 T C 32 . AC=2;AN=2;DB;DP=182;H2;NS=65

```

- Score breakdown - these are not ordered steps of an algorithm but just elements of the solution to focus on:
  - **(1 points)** Correctly identifying the variant features you are looking to check the values for
  - **(3 points)** Correctly extracting the values for the variant features
  - **(3 points)** Correctly computing the result for different use cases (when all conditions are met, one of the conditions is not met, ...)
  - **(1 points)** Correctly returning the computed result
  - **(2 points)** Testing your function

In [189…

```
#Write your solution here, feel free to add new cells.
# One suggestion would be to break down the string into
#smaller parts and then look for the features of interest
# You could use the code from the solution of homework 3 to
#build your function

#variant features I need: ID, QUAL, DP
#These are global variables. These can be called into a local
#function but must be outside of the function.

header_list = "CHROM   POS   ID   REF   ALT   QUAL   FILTER INFO"
split_header = header_list.split()
```
```

```
index_ID = split_header.index("ID")
index_QUAL = split_header.index("QUAL")

def ID_extract(variant_data):
    '''
    This function returns the rs-identifier ONLY if the input from
    the data for the parameter variant_data meets ALL three criteria:
    1. the variant has an ID, 2. has a quality of at least 35,
    3. has a total read depth higher than 100.
    If the provided input for the paramter does not mean any one of these
    conditions, the function returns None (nonetype).
    '''
#this was odd for me. I could not use "\t" in the split method because
#it would give me an error message. So I tried four spaces and it worked.

    split_ID = variant_data.split("    ")[index_ID]

    split_QUAL = variant_data.split("    ")[index_QUAL]

    split_a = variant_data.split("    ")
    split_b = split_a[7].split(";")
    for x in split_b:
        split_c = x.split("=")
        if "DP" in split_c:
            split_DP = split_c[1]

    if split_ID != "." and int(split_QUAL) >= 35 and int(split_DP) > 180:
        return split_ID
    else:
        return None
```

In [185…

```
#testing my function with test_variant1

test_variant1 = "Y    2728456    rs2058276    T    C    32    .    AC=2;AN=2;DB;DP=182;
print(ID_extract(test_variant1))
```

```
None
```

In [190…

```
#testing my function with test_variant2

test_variant2 = "Y    2728456    rs2058260    T    C    37    .    AC=2;AN=2;DB;DP=182;
print(ID_extract(test_variant2))
```

```
rs2058260
```

In [ ]:

```

```

....

Part 2 (10 points)

Select from the `variant_data_file.vcf` a set of rs-identifiers that denote the the variants that have a quality of at least 35 and a total read depth higher than 180.

```
Below is the header and a line that contains variant data:
#CHROM  POS ID  REF ALT QUAL    FILTER  INFO
```

```
Y	2728456 rs2058276	T	C	32	.	AC=2;AN=2;DB;DP=182;H2;NS=65
```

- Score breakdown - these are not ordered steps of an algorithm but just elements of the solution to focus on:
  - **(1 points)** Writing code that correctly opens the file
  - **(1 points)** Writing code that goes through the data in the file
  - **(4 points)** Applying the function created above to the lines that have variant data, or writing code that returns the correct result for each line
  - **(3 points)** Corectly building the resulting set of rs identifiers
  - **(1 points)** Writing code that ensures the file is closed at the end

In [213…

```python
#Write your solution here, feel free to add new cells.
#One suggestion would be to break down the string into smaller
#parts and then look for the element of interest I wnat to extract



#Important Note:I copied and paste my function from Problem 2,
#Part 1 below because the spaces in the vcf file provided for
#this homework had actualy tabs ("\t"), so my four spaces ("    ")
#would not work in when I used the split method in the function.
#So I renamed the function ID_extract_2.


header_list = "CHROM    POS     ID      REF     ALT     QUAL    FILTER  INFO"
split_header = header_list.split()
index_ID = split_header.index("ID")
index_QUAL = split_header.index("QUAL")

ID_set = set()

def ID_extract_2(variant_data):
    '''
    This function returns the rs-identifier ONLY if the input from
    the data for the parameter variant_data meets ALL three criteria:
    1. the variant has an ID, 2. has a quality of at least 35,
    3. has a total read depth higher than 100.
    If the provided input for the paramter does not mean any one of these
    conditions, the function returns None (nonetype).
    '''
    split_ID = variant_data.split("\t")[index_ID]

    split_QUAL = variant_data.split("\t")[index_QUAL]

    split_a = variant_data.split("\t")
    split_b = split_a[7].split(";")
    for x in split_b:
        split_c = x.split("=")
        if "DP" in split_c:
            split_DP = split_c[1]

    if split_ID != "." and int(split_QUAL) >= 35 and int(split_DP) > 180:
        return split_ID
    else:
        return None

#Below is my code regarding the vcf file.
```

```
with open("variant_data_file.vcf") as variant_file:
    for line in variant_file:
        if not line.startswith("#"):
            final_result = ID_extract_2(line)
            if final_result != None:
                ID_set.add(final_result)
print(ID_set)
```

```
{'rs34276300', 'rs28586912', 'rs2082033', 'rs2032658', 'rs17316597', 'rs9306841', 'rs231
9818', 'rs9786823', 'rs9792835', 'rs17250310', 'rs7893095', 'rs9785740', 'rs7067303', 'r
s2032631', 'rs1864469', 'rs17250177', 'rs7892876', 'rs4354487', 'rs17307245', 'rs978621
1', 'rs7892920', 'rs9786746', 'rs9786420', 'rs9786391', 'rs3907838', 'rs17222251', 'rs97
86781', 'rs7892893', 'rs9786485', 'rs7067412', 'rs9786223', 'rs9341301', 'rs17316778',
'rs9786433', 'rs9786810', 'rs7067460', 'rs9786516', 'rs9786062', 'rs57666145', 'rs978655
2', 'rs9785716', 'rs7067346', 'rs9786832', 'rs11096453', 'rs2874089', 'rs9786424', 'rs17
307670', 'rs2078144', 'rs7067469', 'rs17250163', 'rs17316372', 'rs9786191', 'rs9785959',
'rs9785739', 'rs7893070', 'rs7067440', 'rs17222167', 'rs9785743', 'rs3906451', 'rs978668
4', 'rs9785688', 'rs4141961', 'rs9786357', 'rs60115999', 'rs2072422', 'rs2032652', 'rs17
250992', 'rs9786502', 'rs9785784', 'rs1005041', 'rs16980396', 'rs7067278', 'rs35960273',
'rs9785824', 'rs34194308', 'rs9785839', 'rs891407', 'rs7893014', 'rs9786774', 'rs5603972
4', 'rs765557', 'rs17250226', 'rs9786450', 'rs9786741', 'rs9786335', 'rs9786856', 'rs978
6714', 'rs9785717', 'rs2075640', 'rs3853052', 'rs9786458', 'rs7893074', 'rs9786184', 'rs
9786097', 'rs1358368', 'rs9785945', 'rs957431', 'rs7067527', 'rs9785999', 'rs17842518',
'rs9786314', 'rs9786477', 'rs17307252', 'rs9785767', 'rs9786505', 'rs7067305'}
```

In [ ]:

Problem 3 - Understand the code

10 points

....

Exploring a regulatory network (10 points)

Explain what the following code does and how it computes the result it displays.

- Explaining the line with the for keyword is 2 points and each of the other 8 statements is worth each 1 point.

In [195…

```
biological_network = {
    "AKT": {"BAD", "GSK3", "FOXO3A", "IKK"},
    "CALM": {"MLCK", "PHK", "CAMK", "ADCY", "PDLM", "BAD"},
    "RAS": {"PI3K", "BAD"}}
result = set()
extra = biological_network["AKT"]
for value in biological_network.values():
    result.update(value)
    extra = extra.intersection(value)
print(result)
print(extra)
extra.pop()
```

Write your solution here and then run the cell (Markdown)

Line 1: You are creating a dictionary called biological_network with strings as keys and sets as values.

Line 2: You are creating an empty set and assigning it to the variable called result.

Line 3: You are assigning the variable extra to the index key string "AKT" in the biological_network dictionary.

Line 4: You are beginning a for loop where your variable value iterates over all the values in the biolgoical_network dictionary.

Line 5: Within the for loop, you have your empty set which is the variable result updated with the variable called value, which will place all the values from the dictionary into this set. It is important to remember that you are adding the individual elements to the set and NOT the entire sets.

Line 6: We are finding what is common between the sets in the variable extra and setting all of that equal to the variable "extra".

Line 7: Print the variable result, which should be the completed set filled with all the values from the dictionary.

Line 8: Print the variable extra, which should be the one value among all the values in the dictionary that repeats itself more than once.

LIne 9. Pop out the value for the variable extra.

double click here to edit the cell _

In [196…

```python
biological_network = {
    "AKT": {"BAD", "GSK3", "FOXO3A", "IKK"},
    "CALM": {"MLCK", "PHK", "CAMK", "ADCY", "PDLM", "BAD"},
    "RAS": {"PI3K", "BAD"}}
result = set()
extra = biological_network["AKT"]
for value in biological_network.values():
    result.update(value)
    extra = extra.intersection(value)
print(result)
print(extra)
extra.pop()
```

```
{'PDLM', 'FOXO3A', 'ADCY', 'PHK', 'BAD', 'IKK', 'MLCK', 'GSK3', 'CAMK', 'PI3K'}
{'BAD'}
```

Out[196…

```
'BAD'
```

In [ ]: