Introduction to

with Application to Bioinformatics

- Day 3

In [ ]:

```
myTuple    = (1, 2, 3)
myList     = [1, 2 ,3]
```

2. What is the difference between a tuple and a list?

A tuple is immutable while a list is mutable

In [ ]:
```
myTuple    = (1, 2, 3)
myList     = [1, 2 ,3]
```

In [ ]:
```
myList[2]  = 4
myList
```

2. What is the difference between a tuple and a list?

A tuple is immutable while a list is mutable

In [ ]:
```
myTuple    = (1, 2, 3)
myList     = [1, 2 ,3]
```

In [ ]:
```
myList[2]  = 4
myList
```

In [ ]:
```
myTuple[2] = 4
```

Is it true that we can never modify the content of a tuple?

In [ ]:
```
myTuple = (1, 2, [1,2,3])
print(myTuple)
myTuple[2][2] = 4
print(myTuple)
```

Is it true that we can never modify the content of a tuple?

In [ ]:
```
myTuple = (1, 2, [1,2,3])
print(myTuple)
myTuple[2][2] = 4
print(myTuple)
```

- The immutability of tuples in Python means that **the structure of the tuple itself cannot be changed**, you cannot add, remove, or replace elements in the tuple.
- However, if a tuple contains mutable objects like lists, dictionaries, or other objects, the contents of those mutable objects can still be changed.

Functions and methods (Q 4&5)

4. What are the following examples of?

```
len([1, 2, 3, 4])
print("my text")
```

Functions

5. What are the following examples of?

```
"my\ttext".split("\t")
[1, 2, 3].pop()
```

Methods

In [ ]:

```
myList = [1, 2, 3.5, 5 ,6.2]
round(sum(myList)/len(myList),1)
```

In [ ]:

```
my_list   = ['I','know','Python']

" ".join(my_list).upper()
```

In [ ]:

```
# Code Snippet for Finding the Movie with the Highest Rating
# Note that this is just one of the solutions
with open('../downloads/250.imdb', 'r') as fh:
    movieList = []
    highestRating = -100

    for line in fh:
        if not line.startswith('#'):
            cols = line.strip().split('|')
            rating = float(cols[1].strip())
            title = cols[6].strip()
            movieList.append((rating, title))
            if rating > highestRating:
                highestRating = rating
    print("Movie(s) with highest rating " + str(highestRating) + ":" )
    for i in range(len(movieList)):
        if movieList[i][0] == highestRating:
            print(movieList[i][1])
    print(sortedMovieList)
```

In [ ]:

```
mySet = {1,2,3,4,5}
for e in mySet:
    print(e)
```

In [ ]:

```
mySet = {"1", "1", "2", "2", "3"}
print(mySet)
```

In [ ]:

```
mySet = {1, "tga", (3, 4), 5.6, False}
print(mySet)
```

Set can only have hashable elements

In [ ]:
```
mySet = {1, "tga", (3, 4), 5.6, False}
print(mySet)
```

In [ ]:
```
mySet = {1, "tga", [3, 4], 5.6, False}
```

Set can only have hashable elements

In [ ]:
```
mySet = {1, "tga", (3, 4), 5.6, False}
print(mySet)
```

In [ ]:
```
mySet = {1, "tga", [3, 4], 5.6, False}
```

In [ ]:
```
mySet = {1, "tga", (3, 4, [1, 2]), 5.6, False}
```

Set can only have hashable elements

In [ ]:
```
mySet = {1, "tga", (3, 4), 5.6, False}
print(mySet)
```

In [ ]:
```
mySet = {1, "tga", [3, 4], 5.6, False}
```

In [ ]:
```
mySet = {1, "tga", (3, 4, [1, 2]), 5.6, False}
```

Although tuples are immutable, but when it contains mutable items, it becomes non hashable. Be careful!

In [ ]:

```
# Add elements to a set
myset = set()
myset.add(1)
myset.add(100)
myset.add(100)
myset.add(100)
print(myset)
```

Basic operations on `set`

In [ ]:
```
# Add elements to a set
myset = set()
myset.add(1)
myset.add(100)
myset.add(100)
myset.add(100)
print(myset)
```

In [ ]:
```
# get the number of elements of a set
len(1)
```

Basic operations on `set`

In [ ]:
```
# Add elements to a set
myset = set()
myset.add(1)
myset.add(100)
myset.add(100)
myset.add(100)
print(myset)
```

In [ ]:
```
# get the number of elements of a set
len(1)
```

In [ ]:
```
# membership checking
12 in myset
```

Basic operations on `set`

In [ ]:
```
# Add elements to a set
myset = set()
myset.add(1)
myset.add(100)
myset.add(100)
myset.add(100)
print(myset)
```

In [ ]:
```
# get the number of elements of a set
len(1)
```

In [ ]:
```
# membership checking
12 in myset
```

Learn more on https://www.w3schools.com/python/python_sets.asp

In [ ]:

```python
import time, random

# Create a large list and set
large_list = list(range(10000000))
large_set = set(large_list)
elements_to_find = random.sample(range(10000001), 10)

# Measure time for list membership check
list_time = time.time()
for e in elements_to_find:
    e in large_list
list_time = time.time() - list_time

# Measure time for set membership check
set_time = time.time()
for e in elements_to_find:
    e in large_set
set_time = time.time() - set_time

print(f"List check: {list_time:.6f} seconds")
print(f"Set check: {set_time:.6f} seconds")
print(f"Set is approximately {list_time / set_time:.2f} times faster.")
```

In [ ]:

```
myDict = {'drama': 4,
          'thriller': 2,
          'romance': 5}
myDict
```

In [ ]:
```
myDict = {'drama': 4,
          5: 2,
          'romance': 5}
myDict
```

In [ ]:
```
list(myDict.items())
```

In [ ]:
```
myDict = {'drama': 182,
          'war': 30,
          'adventure': 55,
          'comedy': 46,
          'family': 24,
          'animation': 17,
          'biography': 25}
```

In [ ]:
```
myDict['comedy']
```

- How many genres are in this dictionary?
- How many movies are in the `comedy` genre?
- You're not interested in biographies, delete this entry
- You're interested in fantasy; add that we have `29` movies in the `fantasy` genre to this dictionary.
- Which genres are listed in this dictionary after the change?
- You remembered another comedy movie; increase the number of movies in the `comedy` genre by one.

In [ ]:

```
myDict['newkey'] = 25
myDict
```

In [ ]:

```
genreList = ["drama", "action", "drama", "horror", "thriller", "comedy"
myDict = {}
for g in genreList:
    if not g in myDict:
        myDict[g] = 1
    else:
        myDict[g] += 1

myDict
```

In [ ]:
```
print("Hello Python")
```

In [ ]:
```
len("ACCCCTTGAACCCC")
```

In [ ]:
```
max([87, 131, 69, 112, 147, 55, 68, 130, 119, 50])
```

We have used many built-in functions

In [ ]:
```
print("Hello Python")
```

In [ ]:
```
len("ACCCCTTGAACCCC")
```

In [ ]:
```
max([87, 131, 69, 112, 147, 55, 68, 130, 119, 50])
```

How to write your own functions?

In [ ]:

```
def SayHi(name):
    print("Hi", name)

SayHi('Anna')
SayHi('Mike')
```

In [ ]:
```
# Copy the previous code here
```

In [ ]:
```
for genre in ['drama', 'horror', 'comedy']:
    average = sum(genreDict[genre])/len(genreDict[genre])  # calculate
    hours   = int(average/3600)                                  # forma
    minutes = (average - (3600*hours))/60                 # format seconds
    reformattedTime = str(hours)+'h'+str(round(minutes))+'min'
    print('The average length for movies in genre '+ genre +\
          ' is '+ reformattedTime)
```

In [ ]:
```
for genre in ['drama', 'horror', 'comedy']:
    print('The average length for movies in genre '+ genre +\
          ' is '+ formatSec(genre, genreDict))
```

Scope

- Local variables - Variables within functions
- Global variables - Variables outside of functions

In [ ]:
```
WEIGHT = 5
def addWeight(value):
    return value * WEIGHT
print(addWeight(4))
```

Scope

- Local variables - Variables within functions
- Global variables - Variables outside of functions

In [ ]:
```
WEIGHT = 5
def addWeight(value):
    return value * WEIGHT
print(addWeight(4))
```

In [ ]:
```
WEIGHT = 5
def changeWeight():
    WEIGHT = 10
    print("WEIGHT inside the function is", WEIGHT)
    return None
changeWeight()
print("WEIGHT outside the function is", WEIGHT)
```

Scope

- Local variables - Variables within functions
- Global variables - Variables outside of functions

In [ ]:
```
WEIGHT = 5
def addWeight(value):
    return value * WEIGHT
print(addWeight(4))
```

In [ ]:
```
WEIGHT = 5
def changeWeight():
    WEIGHT = 10
    print("WEIGHT inside the function is", WEIGHT)
    return None
changeWeight()
print("WEIGHT outside the function is", WEIGHT)
```

We will talk more about the scope of variables tomorrow

In [ ]:
```
math.sqrt(5)
```

Use external libraries in Python

In [ ]:
```
math.sqrt(5)
```

In [ ]:
```
import math
math.sqrt(5)
```

Use external libraries in Python

In [ ]:
```
math.sqrt(5)
```

In [ ]:
```
import math
math.sqrt(5)
```

In [ ]:
```
sqrt(5)
```

Use external libraries in Python

In [ ]:
```
math.sqrt(5)
```

In [ ]:
```
import math
math.sqrt(5)
```

In [ ]:
```
sqrt(5)
```

In [ ]:
```
from math import sqrt
sqrt(5)
```

In [ ]:

```
def formatSec(seconds):
    hours     = seconds/3600
    minutes   = (seconds - (3600*int(hours)))/60
    return str(int(hours))+'h'+str(round(minutes))+'min'


def toSec(days, hours, minutes, seconds):
    total = 0
    total += days*60*60*24
    total += hours*60*60
    total += minutes*60
    total += seconds

    return str(total)+'s'
```

In [ ]:
```
toSec(days=0, hours=1, minutes=0, seconds=1)
```

In [ ]:

```
Ref  Alternative
A    T,G                                    1/1
```

In [ ]:
```
input_file = "../downloads/250.imdb"
output_file = "newfile.txt"

with open(input_file, "r") as fi:
    with open(output_file, "w") as fo:
        for line in fi:
            fo.write(line)
```

In [ ]:
```
!ls -lah mynewseq.fa
```

String formatting

Format text for printing or for writing to file.

What we have been doing so far:

In [ ]:
```
chrom = "5"
pos = 1235651
ref = "C"
alt = "T"
geno = "1/1"
info = chrom + ":" + str(pos) + "_" + ref + "-" + alt + " has genotype:
print(info)
```

In [ ]:

```
chrom = "5"
pos = 1235651
ref = "C"
alt = "T"
geno = "1/1"
info = f"{chrom}:{pos}_{ref}-{alt} has genotype: {geno}"
print(info)
```

Other (better) ways of formatting strings:

f-strings (since python 3.6)

In [ ]:
```
chrom = "5"
pos = 1235651
ref = "C"
alt = "T"
geno = "1/1"
info = f"{chrom}:{pos}_{ref}-{alt} has genotype: {geno}"
print(info)
```

In [ ]:
```
info = chrom + ":" + str(pos) + "_" + ref + "-" + alt + " has genotype:
```

In [ ]:

```
chrom = "5"
pos = 1235651
ref = "C"
alt = "T"
geno = "1/1"
info = "{}:{}_{}-{} has genotype: {}".format(chrom, pos, ref, alt, geno
print(info)
```

In [ ]:

```
genes = ["TP53", "COX2"]
lengths = [355,  458]
print(f"Lengths of genes {genes} are {lengths}")
```

It works for other data types as well

In [ ]:
```
genes = ["TP53", "COX2"]
lengths = [355,  458]
print(f"Lengths of genes {genes} are {lengths}")
```

In [ ]:
```
gene = "COX1"
exp_level = 45.123253
print(f"Expression level of gene {gene} is {exp_level}")
```

It works for other data types as well

In [ ]:
```
genes = ["TP53", "COX2"]
lengths = [355,  458]
print(f"Lengths of genes {genes} are {lengths}")
```

In [ ]:
```
gene = "COX1"
exp_level = 45.123253
print(f"Expression level of gene {gene} is {exp_level}")
```

In [ ]:
```
print(f"Expression level of gene {gene} is {exp_level:.2e}")
```

In [ ]:
```
seq = "ATCGTAGCCCATAGC"
print(f"The length of sequence {seq} is {len(seq)}")
```

In [ ]:
```
text = "a string with many words  "
print(f"the text \"{text}\" is divided in to a list {text.split()}, "
      f"and it has {len(text.split())} elements")
```

In [ ]:

```
gene = "COX1"
exp_level = 45.123253
print("Expression level of gene %s is %f"%(gene, exp_level))
```