Introduction to

with Application to Bioinformatics

- Day 3

Review Day 2

- Give an example of a tuple
- What is the difference between a tuple and a list?
- How would you approach a complicated coding task?
- What is the different syntax between a function and a method?
- Calculate the average of the list [1,2,3.5,5,6.2] to one decimal
- Take the list ['i','know','python'] as input and output the string 'I KNOW PYTHON'
- What are the characteristics of a set?
- Create a set containing the integers 1,2,3, and 4, add 3,4,5, and 6 to the set. How long is the set?

Tuples

Give an example of a tuple:

In [ ]:
```
myTuple = (1,2,3,'a','b',[4,5,6])
myTuple
```

What is the difference between a tuple and a list?

A tuple is immutable while a list is mutable

How to structure code

- Decide on what output you want
- What input files do you have?
- How is the input structured, can you iterate over it?
- Where is the information you need located?
- Do you need to save a lot of information while iterating?
    - Lists are good for ordered data
    - Sets are good for non-duplicate single entry information
    - Dictionaries are good for a lot of structured information
- When you have collected the data needed, decide on how to process it
- Are you writing your results to a file?

Always start with writing pseudocode!

Functions and methods

What is the different syntax between a function and a method?

`functionName()` `<object>.methodName()`

Calculate the average of the list [1,2,3.5,5,6.2] to one decimal

In [ ]:

```
myList = [1,2,3,5,6]
round(sum(myList)/len(myList),1)
```

Take the list ['i','know','python'] as input and output the string 'I KNOW PYTHON'

In [ ]:
```
' '.join(['i','know','python']).upper()
```

Sets

What are the characteristics of a set?
A set contains an unordered collection of unique and immutable objects

Create a set containing the integers 1,2,3, and 4, add 3,4,5, and 6 to the set. How long is the set?

In [ ]:
```
mySet = {1,2,3,4}
mySet.add(3)
mySet.add(4)
mySet.add(5)
mySet.add(6)
len(mySet)
```

IMDb

How to find the number of movies per genre?

```
# Votes | Rating | Year | Runtime | URL | Genres | Title
   126807|   8.5|1957|5280|https://images-na.ssl-images-amazon.com/images/M_V1_.jpg|Drama,War|Paths of Glory
    71379|   8.2|1925|4320|https://images-na.ssl-images-amazon.com/images/N_V1_.jpg|Adventure,Comedy,Drama,Family|The Gold Rush
```

... Hm, starting to be difficult now...

New data type: `dictionary`

- A dictionary is a mapping of unique keys to values
- Dictionaries are mutable

Syntax:

`a = {}` (create empty dictionary)

`d = {'key1':1, 'key2':2, 'key3':3}`

In [ ]:

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

Operations on Dictionaries

Dictonary	
len(d)	Number of items
d[key]	Returns the item *value* for key *key*
d[key] = value	Updating the mapping for *key* with *value*
del d[key]	Delete key from d
key in d	Membership tests
d.keys( )	Returns an iterator on the keys
d.values( )	Returns an iterator on the values
d.items( )	Returns an iterator on the pair (key, value)

In [ ]:

```
myDict = {'drama': 4,
          'thriller': 2,
          'romance': 5}
len(myDict)
myDict['drama']
myDict['horror'] = 2
#myDict
#del myDict['horror']
#myDict
'drama' in myDict
myDict.keys()
myDict.items()
myDict.values()
```

Exercise

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

- How many entries are there in this dictionary?
- How do you find out how many movies are in the genre 'comedy'?
- You're not interested in biographies, delete this entry
- You are however interested in fantasy, add that we have 29 movies of the genre fantasy to the list
- What genres are listed in this dictionary?
- You remembered another comedy movie, increase the number of comedies by one

In [ ]:
```

```

Find the number of movies per genre

```
# Votes | Rating | Year | Runtime | URL | Genres | Title
   126807|    8.5|1957|5280|https://images-na.ssl-images-amazon.com/images/M_V1_.jpg|Drama,War|Paths of Glory
    71379|    8.2|1925|4320|https://images-na.ssl-images-amazon.com/images/N_V1_.jpg|Adventure,Comedy,Drama,Family|The Gold Rush
```

Hint! If the genre is not already in the dictionary, you have to add it first

Answer

```
drama:182          thriller:65
war:30             fantasy:29
adventure:55       romance:24
comedy:46          sci-fi:28
family:24          western:8
animation:17       musical:5
biography:25       music:3
history:18         historical:1
action:31          sport:7
crime:62           film-noir:7
mystery:41         horror:5
```

In [ ]:

```
fh        = open('../downloads/250.imdb', 'r', encoding = 'utf-8')
genreDict = {}     # create empty dictionary

for line in fh:
    if not line.startswith('#'):
        cols  = line.strip().split('|')
        genre = cols[5].strip()
        glist = genre.split(',')
        for entry in glist:
            if not entry.lower() in genreDict: # check if genre is not in dictionary, add 1
                genreDict[entry.lower()] = 1
            else:
                genreDict[entry.lower()] += 1   # if genre is in dictionary, increase count with 1
fh.close()
print(genreDict)
```

What is the average length of the movies (hours and minutes) in each genre?

```
# Votes | Rating | Year | Runtime | URL | Genres | Title
   126807|   8.5|1957|5280|https://images-na.ssl-images-amazon.com/images/M_V1_.jpg|Drama,War|Paths of Glory
    71379|   8.2|1925|4320|https://images-na.ssl-images-amazon.com/images/N_V1_.jpg|Adventure,Comedy,Drama,Family|The Gold Rush
```

Answer

drama	2h14min	thriller	2h11min
war	2h30min	fantasy	2h2min
adventure	2h13min	romance	2h2min
comedy	1h53min	sci-fi	2h6min
family	1h44min	western	2h11min
animation	1h40min	musical	1h57min
biography	2h30min	music	2h24min
history	2h47min	historical	2h38min
action	2h18min	sport	2h17min
crime	2h11min	film-noir	1h43min
mystery	2h3min	horror	1h59min

Tip!
Here you have to loop twice

In [ ]:

```
fh        = open('../downloads/250.imdb', 'r', encoding = 'utf-8')
genreDict = {}

for line in fh:
    if not line.startswith('#'):
        cols    = line.strip().split('|')
        genre   = cols[5].strip()
        glist   = genre.split(',')
        runtime = cols[3]      # length of movie in seconds
        for entry in glist:
            if not entry.lower() in genreDict:
                genreDict[entry.lower()] = [int(runtime)]   # add a list with the runtime
            else:
                genreDict[entry.lower()].append(int(runtime))   # append runtime to existing list
fh.close()

for genre in genreDict:      # loop over the genres in the dictionaries
    average = sum(genreDict[genre])/len(genreDict[genre])  # calculate average length per genre
    hours   = int(average/3600)                                  # format seconds to hours
    minutes = (average - (3600*hours))/60             # format seconds to minutes
    print('The average length for movies in genre '+genre\
          +' is '+str(hours)+'h'+str(round(minutes))+'min')
```

NEW TOPIC: Functions

```
fh        = open('../files/250.imdb', 'r', encoding = 'utf-8')
genreDict = {}

for line in fh:
    if not line.startswith('#'):
        cols    = line.strip().split('|')
        genre   = cols[5].strip()
        glist   = genre.split(',')
        runtime = cols[3]      # length of movie in seconds
        for entry in glist:
            if not entry.lower() in genreDict:
                genreDict[entry.lower()] = [int(runtime)]   # add a list with the runtime
            else:
                genreDict[entry.lower()].append(int(runtime))   # append runtime to existing list
fh.close()

for genre in genreDict:      # loop over the genres in the dictionaries
    average = sum(genreDict[genre])/len(genreDict[genre])  # calculate average length per genre
    hours   = average/3600                                 # format seconds to hours
    minutes = (average - (3600*int(hours)))/60             # format seconds to minutes
    print('The average length for movies in genre '+genre+' is '+str(int(hours))+'h'+str(round(minutes))+'min')
```

A lot of ugly formatting for calculating hours and minutes from seconds...

In [ ]:

```
def FormatSec(genre):   # input a list of seconds
    average   = sum(genreDict[genre])/len(genreDict[genre])
    hours     = int(average/3600)
    minutes   = (average - (3600*hours))/60
    return str(hours)+'h'+str(round(minutes))+'min'


fh        = open('../downloads/250.imdb', 'r', encoding = 'utf-8')
genreDict = {}

for line in fh:
    if not line.startswith('#'):
        cols    = line.strip().split('|')
        genre   = cols[5].strip()
        glist   = genre.split(',')
        runtime = cols[3]      # length of movie in seconds
        for entry in glist:
            if not entry.lower() in genreDict:
                genreDict[entry.lower()] = [int(runtime)]   # add a list with the runtime
            else:
                genreDict[entry.lower()].append(int(runtime))   # append runtime to existing list
fh.close()

for genre in genreDict:
    print('The average length for movies in genre '+genre\
          +' is '+FormatSec(genre))
```

Function structure

```
def functionName(arg1, arg2, arg3):

    finalValue = 0

    # Here is some code where you can do
    # calculations etc, on arg1, arg2, arg3
    # and update finalValue

    return FinalValue
```

Function structure

```
def functionName(arg1, arg2, arg3):

    finalValue = 0

    # Here is some code where you can do
    # calculations etc, on arg1, arg2, arg3
    # and update finalValue

    return FinalValue
```

In [ ]:

```
def addFive(number):
    final = number + 5
    return final

addFive(4)
```

In [ ]:

```
from datetime import datetime

def whatTimeIsIt():
    time = 'The time is: ' + str(datetime.now().time())
    return time

whatTimeIsIt()
```

In [ ]:

```
def addFive(number):
    final = number + 5
    return final

addFive(4)
#final

final = addFive(4)
final
```

Scope

- Variables within functions
- Global variables

In [ ]:

```
def someFunction():
#    s = 'a string'
    print(s)

s = 'another string'
someFunction()
print(s)
```

Why use functions?

- Cleaner code
- Better defined tasks in code
- Re-usability
- Better structure

Importing functions

- Collect all your functions in another file
- Keeps main code cleaner
- Easy to use across different code

Example:

1. Create a file called myFunctions.py, located in the same folder as your script
2. Put a function called `formatSec()` in the file
3. Start writing your code in a separate file and `import` the function

In [ ]:

```
from myFunctions import formatSec

seconds = 32154

formatSec(seconds)
```

In [ ]:

```
from myFunctions import  formatSec, toSec

seconds = 21154
print(formatSec(seconds))

days    = 0
hours   = 21
minutes = 56
seconds = 45

print(toSec(days, hours, minutes, seconds))
```

myFunctions.py

```
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 total
```

Summary

- A function is a block of organized, reusable code that is used to perform a single, related action
- Variables within a function are local variables
- Functions can be organized in separate files and imported to the main code

→ **Notebook Day_3_Exercise_1 (~30 minutes)**

NEW TOPIC AGAIN: `sys.argv`

- Avoid hardcoding the filename in the code
- Easier to re-use code for different input files
- Uses command-line arguments
- Input is list of strings:
    - Position 0: the program name
    - Position 1: the first argument

The `sys.argv` function

Python script called `print_argv.py` :

```
import sys

print(sys.argv)
```

Running the script with command line arguments as input:

```
nina@Nina-pc:~$ python3 print_argv.py input_file.txt output_file.txt
['print_argv.py', 'input_file.txt', 'output_file.txt']
```

Instead of:

```
fh  = open('../files/250.imdb', 'r', encoding = 'utf-8')
out = open('../files/imdb_copy.txt', 'w', encoding = 'utf-8')

for line in fh:
    out.write(line)

fh.close()
out.close()
```

do:

```
import sys

if len(sys.argv) == 3:
    fh  = open(sys.argv[1], 'r', encoding = 'utf-8')
    out = open(sys.argv[2], 'w', encoding = 'utf-8')

    for line in fh:
        out.write(line)

    fh.close()
    out.close()

else:
    print('Arguments should be input file name and output file name')
```

Run with:

```
$ python3 copy_file.py 250.imdb imdb_copy.txt
```

IMDb

Re-structure and write the output to a new file as below

```
> Western
8.3     For a Few Dollars More (1965) [2h12min]
8.3     Unforgiven (1992) [2h11min]
8.3     The Treasure of the Sierra Madre (1948) [2h6min]
8.6     Once Upon a Time in the West (1968) [2h25min]
8.9     The Good, the Bad and the Ugly (1966) [2h41min]
8.1     Butch Cassidy and the Sundance Kid (1969) [1h50min]
8.4     Django Unchained (2012) [2h45min]
8.2     The General (1926) [1h15min]
> Musical
8.6     La La Land (2016) [2h8min]
8.1     The Wizard of Oz (1939) [1h42min]
8.5     The Lion King (1994) [1h28min]
8.3     Singin' in the Rain (1952) [1h43min]
8.4     Sholay (1975) [2h42min]
> Music
8.5     Like Stars on Earth (2007) [2h45min]
8.5     Whiplash (2014) [1h47min]
8.3     Amadeus (1984) [2h40min]
> Historical
8.1     There Will Be Blood (2007) [2h38min]
```

Note:

- Use a text editor, not notebooks for this
- Use functions as much as possible
- Use `sys.argv` for input/output

Answer - Example

```
import sys

"""
Script that reformats an imdb file and writes it to another file
"""


def FormatSec(seconds):    # formats seconds to hours and minutes
    hours     = seconds/3600
    minutes   = (seconds - (3600*int(hours)))/60
    return str(int(hours))+'h'+str(round(minutes))+'min'

def FormatMovie(movie):    # returns a string with the correct format for writing to file
    formMovie = str(movie[0])+'\t'+movie[1]+' ('+str(movie[2])+') ['+movie[3]+']\n'
    return formMovie


if len(sys.argv) == 3:
    fh        = open(sys.argv[1], 'r', encoding = 'utf-8')
    genreDict = {}

    for line in fh:
        if not line.startswith('#'):
            cols   = line.strip().split('|')
            rating = float(cols[1].strip())
            year   = int(cols[2].strip())
            length = int(cols[3].strip())
            movie  = cols[6].strip()
            genre  = cols[5].strip()
            glist  = genre.split(',')
            for entry in glist:
                if not entry.lower() in genreDict:              # if genre in dictionary, add first movie
                    genreDict[entry.lower()] = []
                genreDict[entry.lower()].append([rating, movie, year, FormatSec(length)])
    fh.close()


    out = open(sys.argv[2], 'w', encoding = 'utf-8')
    for genre in genreDict:
        out.write('> '+genre.capitalize()+'\n')
        for movie in genreDict[genre]:
            out.write(FormatMovie(movie))
    out.close()

else:
    print('Number of arguments does not match')
```