Introduction to python

with Application to Bioinformatics

- Day 3

Day 3

- **Session 1**
    - Quiz: Review of Day 2
    - Lecture: Go through questions, data type `set`
    - Ex1: IMDb exercise - Find the number of unique genres
- **Session 2**
    - Lecture: Data type `dict`
    - Ex2: IMDb exercise - Find the number of movies per genre
    - PyQuiz 3.1
- **Session 3**
    - Lecture: Write you own functions
    - Ex3: Day 3, Exercise 3, Functions
- **Session 4**
    - Lecture: Pass arguments from command line using `sys.argv` and string formatting
    - Ex4: IMDb exercise - functions and `sys.argv`
    - PyQuiz 3.2
- **Project time**

Quiz: Review Day 2

Go to Canvas, `Modules -> Day 3 -> Review Day 2`

~20 minutes

Tuples (Q 1&2)

1. Which of the following variables are of the type tuple?

```
a = (1, 2, 3, 4)
a = ([1, 2], 'a', 'b')
```

Tuples (Q 1&2)

1. Which of the following variables are of the type tuple?

`a = (1, 2, 3, 4)`
`a = ([1, 2], 'a', 'b')`

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?

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.

How to structure the code (Q 3)

3. What does pseudocode mean?
Writing down the steps you intend to include in your code in more general language

Things to Consider When Writing Pseudocode

- Decide on the desired output.
- Identify the input files you have.
- Examine the structure of the input – can it be iterated over?
- Determine where the necessary information is located.
- Assess if you need to store information while iterating:
    - Use lists for ordered data.
    - Use sets for unique, non-duplicate entries.
    - Use dictionaries for structured, key-value information.
- After collecting the required data, decide how to process it.
- Determine if you'll need to write your results to a file.

Writing pseudocode before actual coding is a good habit.

Functions and methods (Q 4&5)

4. What are the following examples of?

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

Functions

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

What are the differences between a `function` and a `method`?

Function	Method
Standalone block of code	Function associated with an object
Called independently, e.g. `functionName()`	Called on an instance of a class, e.g. `obj.methodName()`
Not tied to any object or class	Tied to the objects they are called on
Defined outside of a class	Defined within a class

6. Calculate the average of the list `[1,2,3.5,5,6.2]` to one decimal, using Python

In [ ]:

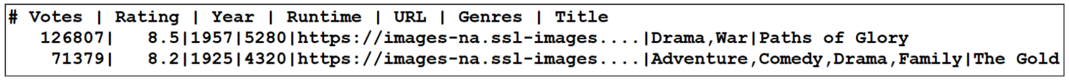
```
myList = [1, 2, 3.5, 5 ,6.2]
round(sum(myList)/len(myList),1)
```

7. Take the list `['I','know','Python']` as input and output the string 'I KNOW PYTHON'

In [ ]:
```
my_list   = ['I','know','Python']
my_string =' '.join(my_list).upper()
print(my_string)
```

Exerciese from yesterday

Find the movie with the highest rating in the file `250.imdb`



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

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

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 = -1.0

    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])
```

The `with` key word

- Use the `with` keyword to ensure the file handle be closed automatically

```
file = open("filename.txt", "r")
content = file.read()
file.close()
```

```
with open("filename.txt", "r") as file:
    content = file.read()
```

```
with open ("filename.txt", "r", encoding='utf-8') as file
    content = file.read()
```

However, for Python 3, the default encoding is usually 'utf-8', so it's not needed.

New data type: `set`

- A set contains an unordered collection of unique and hashable objects
    - **Unordered**: Items have no defined order
    - **Unique**: Duplicate items are not allowed
    - **Hashable**: Each item must be hashable

Syntax:

```
setName = set() # Create an empty set
```

```
setName = {1,2,3,4,5} # Create a populated set
setName = set([1,2,3,4,5]) # Alternative way
```

Set is unordered

In [ ]:

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

Set has unique elements

```
In [ ]: mySet = {"1", "1", "2", "2", "3"}
        print(mySet)
```

Set can only have hashable elements

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!

Basic operations on `set`

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

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

Basic operations on `set`

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

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

In [ ]:
```python
# membership checking
1 in myset
```

Basic operations on `set`

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

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

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

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

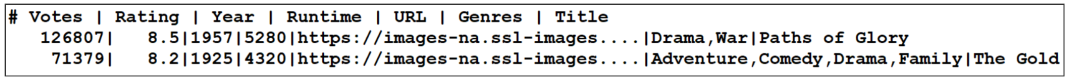
When the size of list is large, membership checking with `set` tends to be much faster than with `list`

In [ ]:

```
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.")
```

Day 3, Exercise 1 (~20 min)

Find the number of unique genres in the file `250.imdb`



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

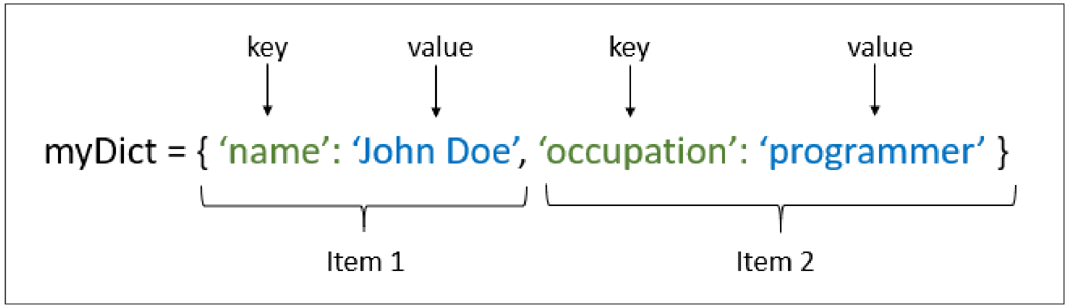
- Canvas -> Modules -> Day 3 -> IMDb exercise -> 1
- Take a break after the exercise (~10 min)

Session 2

- Lecture: Data type `dictionary`
- Ex2: IMDb exercise - Find the number of movies per genre + Extra
- PyQuiz 3.1

New data type: `dictionary`

- A dictionary is an unordered, mutable collection of key-value pairs.
- Dictionaries are mutable
- Each key in a dictionary must be unique and immutable, while the values associated with keys can be of any data type and can be duplicated



Syntax:

```
d = {} # Create an empty dictionary
```

```
d = {'key1':1, 'key2':2, 'key3':3} # create a populated dictionary
```

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

Basic 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}
myDict
```

In [ ]:
```
len(myDict)
```

Live Exercise

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

- 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.

Day 3, Exercise 2 (~50 min)

- #### Find the number of movies per genre
- #### (Extra) What is the average length of the movies (in hours and minutes) in each genre?
- Canvas -> Modules -> Day 3 -> IMDb exercise -> 2&3

Take a break after the exercise (~10 min)

PyQuiz 3.1 - set, list and dictionary (before lunch)

Lunch

Session 3

- Lecture: Write you own functions
- Exercise 3: Functions

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])
```

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?

Syntax of function

```
def function_name(arg1, arg2, ...):
    # Block of code
    return result
```

In [ ]:
```
def SayHi(name):
    print("Hi", name)

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

```
In [ ]: # Calculate the average duration of movies in the genre 'drama'
        genre = "drama"
        average = sum(genreDict[genre])/len(genreDict[genre])  # calculate aver
        hours   = int(average/3600)                                 # format se
        minutes = (average - (3600*hours))/60             # format seconds to m
        reformattedTime = str(hours)+'h'+str(round(minutes))+'min'
        print('The average length for movies in genre '+ genre +\
              ' is '+ reformattedTime)
```

In [ ]:

```python
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']:
            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))
```

Why use functions?

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

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

Scope

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

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
    return None
print(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
            return None
        print(WEIGHT)
```

We will talk more about the scope of variables tomorrow

Use external libraries in Python

```
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)
```

Why use libraries

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

How to define your own libraries

A simple library is just file with some python functions

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 [ ]: from myutils import formatSec, toSec

        formatSec(3601)
```

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

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
- Variables outside of functions are global variables
- Functions can be organized in separate files as libraries and be imported to the main code

Day 3, Exercise 3 (~30 min)

- Canvas -> Modules -> Exercise 3 - functions

Take a break after the exercise (~10 min)

Session 4

- Lecture: Pass arguments from command line using `sys.argv` and string formatting
- Ex4: IMDb exercise - functions and `sys.argv`
- PyQuiz 3.2

How to pass arguments to Python script from the command line?

Not just

```
python myscript.py
```

But also

```
python myscript.py arg1 arg2
```

`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
    - Position 2: the second argument
    - etc

How to use it

```
import sys

program_name = sys.argv[0]
arg1 = sys.argv[1] # index error if the first argument is not
provided in the command
arg2 = sys.argv[2] # index error if the second argument is not
provided in the command
```

Try out `sys.argv`

Python script is called `print_argv.py` and can be found in the downloads folder

Run the following commands in the terminal

```
python print_argv.py
python print_argv.py 1
python print_argv.py arg1 arg2 arg3
```

Naive code to copy a text file

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

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

```
In [ ]: # Code that can deal with command line arguments
        import sys

        usage = f"{sys.argv[0]} inputFile outputFile"

        if len(sys.argv) < 3:
            print(usage)
            sys.exit(1)

        input_file = sys.argv[1]
        output_file = sys.argv[2]

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

String formatting

Format text for printing or for writing to file.

What we have been doing so far:

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)
```

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)
```

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:
```

`format` method

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

`f-strings'' formatting is recommended

It works for other data types as well

```
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:.2f}")
```

It works for functions and operations as well

```
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")
```

The ancient way (for Python 2, but still working)

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

Summary

- Use `sys.argv` to deal with arguments passed to the python script from the command line
    - `sys.argv[0]` is the program name
    - `sys.argv[1]` is is the first argument and so on
- `f-strings` formatting is a convenient and recommended way to format the string
    - Extra reading about string formatting: https://www.w3schools.com/python/python_string_formatting.asp

Day 3, Exercise 4 (~30 min)

- #### Restructure and write the output to a new file
- Canvas -> Modules -> Day 3 -> IMDb exercise -> 4
- Work in pairs

PyQuiz 3.2

Project time