Introduction to

python

with Application to Bioinformatics

- Day 3

Review Day 2

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

~20 minutes

Tuples

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 [16]:
```
myTuple    = (1, 2, 3, 'a', 'b', [4,5,6])
myList     = [1, 2 ,3]
myList[2]  = 4
#myTuple[2] = 4
myList
```

Out[16]:
```
[1, 2, 4]
```

How to structure code

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

- 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

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

General syntax of Functions and Methods

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

A method always belongs to an object of a specific class, a function does not have to.

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

In [17]:

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

Out[17]:

```
3.5
```

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

In [18]:
```
my_list   = ['I','know','Python']
my_string =' '.join(my_list).upper()
print(my_string)
```

```
I KNOW PYTHON
```

Day 3

- Sets
- Dictionaries
- Functions
- sys.argv
- Formatting

IMDb

Find the number of unique genres

```
# 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
```

New data type: `set`

- A set contains an unordered collection of unique and immutable objects

Syntax:

For empty set:

```
setName = set()
```

For populated sets:

```
setName = {1,2,3,4,5}
```

Common operations on sets

`set.add(a)`

`len(set)`

`a in set`

In [20]:
```
x = set()
x.add(100)
x.add(25)
x.add(3)
x.add('3.0')
#for i in x:
#    print(type(i))
type(x)
##mySet = {2,5,1,3}
#mySet.add(5)
#mySet.add(4)
#print(mySet)
```

Out[20]:
```
set
```

Find the number of unique genres

```
# 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
```

Modify your code to use sets

In [21]:

```python
fh     = open('../downloads/250.imdb', 'r', encoding = 'utf-8')
genres = set()

for line in fh:
    if not line.startswith('#'):
        cols  = line.strip().split('|')
        genre = cols[5].strip()
        glist = genre.split(',')
        for entry in glist:
            genres.add(entry.lower())    # set only adds entry if not already in
fh.close()
print(len(genres))
sorted(list(genres))
```

```
22
```

Out[21]:

```
['action',
 'adventure',
 'animation',
 'biography',
 'comedy',
 'crime',
 'drama',
 'family',
 'fantasy',
 'film-noir',
 'historical',
 'history',
 'horror',
 'music',
 'musical',
 'mystery',
 'romance',
 'sci-fi',
 'sport',
 'thriller',
 'war',
 'western']
```

IMDb

Find the number of movies per genre

```
# 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
```

... 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 [22]:
```
myDict = {'drama': 4,
          'thriller': 2,
          'romance': 5}
myDict
```

Out[22]:
```
{'drama': 4, 'thriller': 2, 'romance': 5}
```

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 [23]:

```
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()
```

Out[23]:

```
dict_values([4, 2, 5, 2])
```

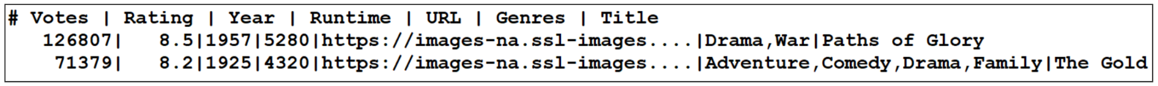
Exercise

In [24]:
```
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 [ ]:
```

```

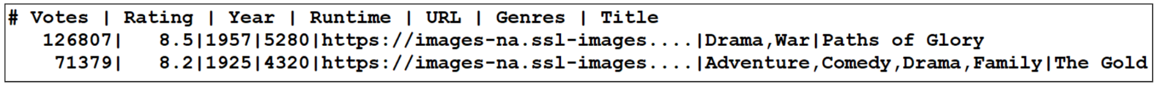
Find the number of movies per genre



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

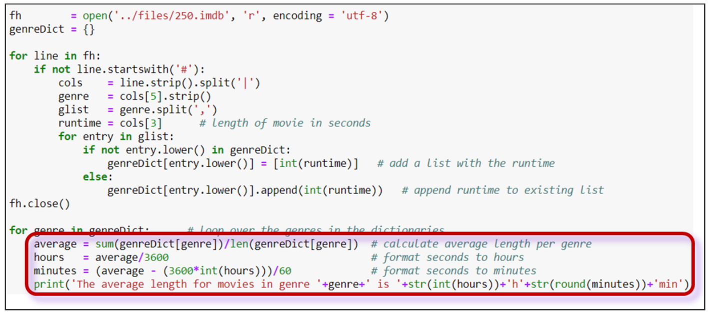
Hint! If the genre is not already in the dictionary, you have to add it first

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....|Drama,War|Paths of Glory
    71379|   8.2|1925|4320|https://images-na.ssl-images....|Adventure,Comedy,Drama,Family|The Gold
```

NEW TOPIC: Functions



```python
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 [27]:

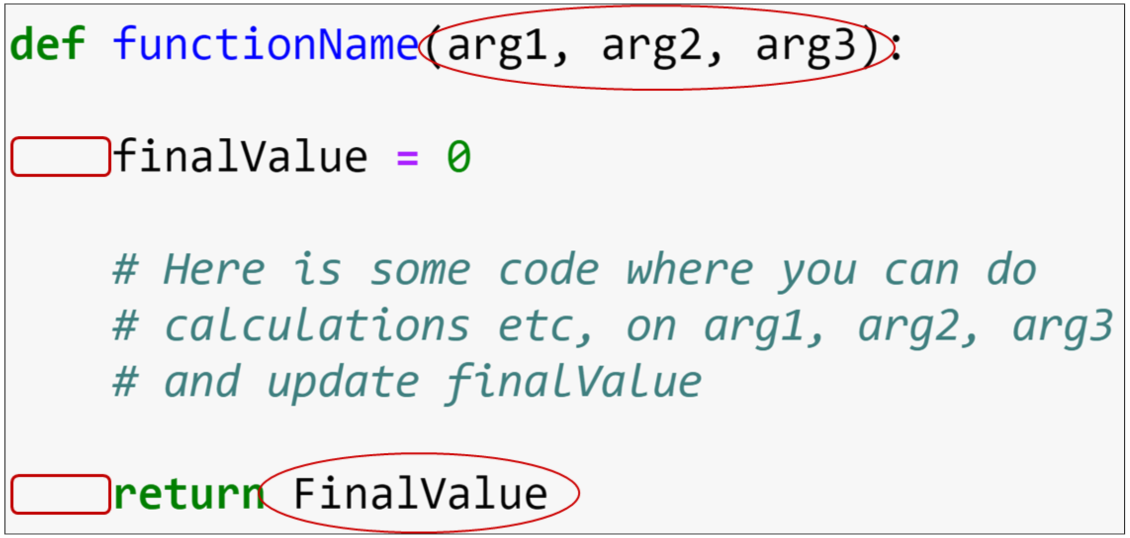
```python
def FormatSec(genre):   # input a list of seconds
    average   = sum(genreDict[genre])/len(genreDict[genre])
    hours     = int(average/3600)
    minutes   = (average - (3600*hours))/60
    return str(hours)+'h'+str(round(minutes))+'min'


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))
```

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

Function structure

```python
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 [28]:
```
def addFive(number):
    final = number + 5
    return final

addFive(4)
```
Out[28]:
```
9
```
In [29]:
```
from datetime import datetime

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

whatTimeIsIt()
```
Out[29]:
```
'The time is: 12:53:24.875493'
```
In [30]:
```
def addFive(number):
    final = number + 5
    return final

addFive(4)
#final

final = addFive(4)
final
```
Out[30]:
```
9
```

Scope

- Variables within functions
- Global variables

In [31]:

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

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

```
another string
another string
```

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:

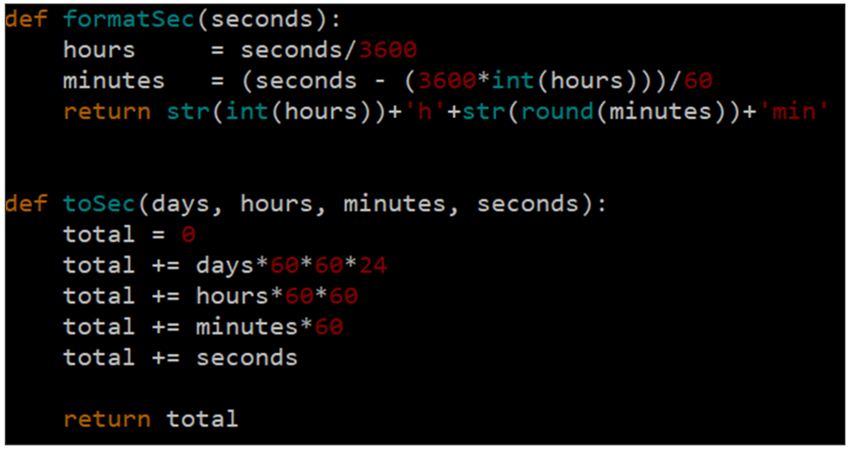
1. Create a file called myFunctions.py, located in the same folder as your script
2. Put a function called `formatSec()` in the file
3. Start writing your code in a separate file and `import` the function

```
In [32]: from myFunctions import formatSec

         seconds = 32154

         formatSec(seconds)

Out[32]: '8h56min'
```

In [33]:
```
from myFunctions import  formatSec, toSec

seconds = 21154
print(formatSec(seconds))

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

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

```
5h53min
79005s
```

myFunctions.py



```python
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

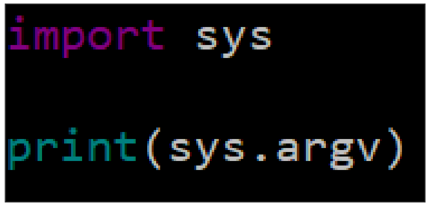
→ **Notebook Day_3_Exercise_1 (~30 minutes)**

NEW TOPIC: `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

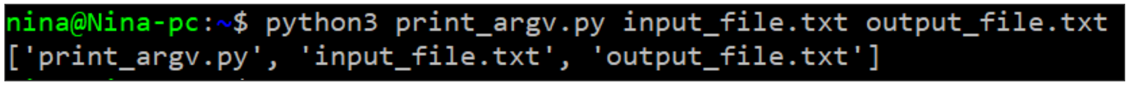
The `sys.argv` function

Python script called `print_argv.py` :



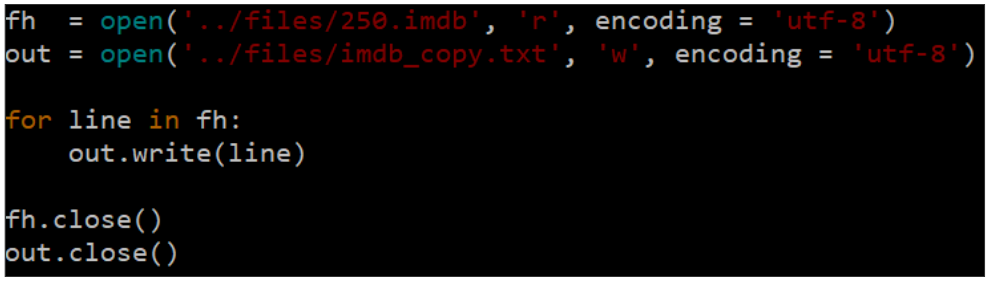
```
import sys

print(sys.argv)
```

Running the script with command line arguments as input:



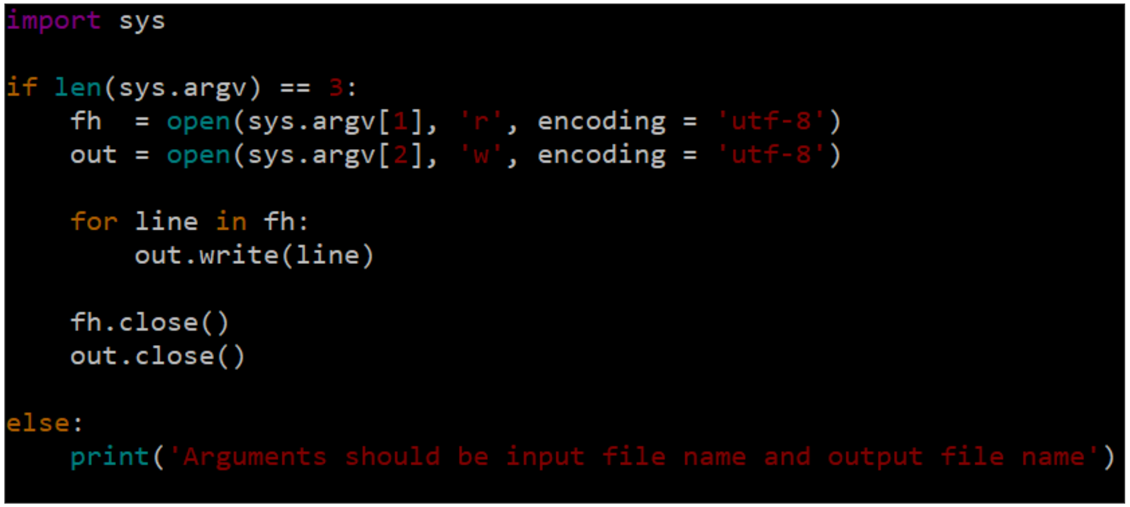
```
nina@Nina-pc:~$ python3 print_argv.py input_file.txt output_file.txt
['print_argv.py', 'input_file.txt', 'output_file.txt']
```

Instead of:



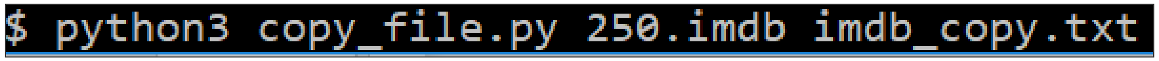
```python
fh  = open('../files/250.imdb', 'r', encoding = 'utf-8')
out = open('../files/imdb_copy.txt', 'w', encoding = 'utf-8')

for line in fh:
    out.write(line)

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

do:



```python
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
```

Formatting

Format text for printing or for writing to file.

What we have been doing so far:

In [34]:
```
title  = 'Toy Story'
rating = 10
print('The result is: ' + title + ' with rating: ' + str(rating))
```

```
The result is: Toy Story with rating: 10
```

Other (better) ways of formatting strings:

f-strings (since python 3.6)

In [35]:
```python
title  = 'Toy Story'
rating = 10
print(f'The result is: {title} with rating: {rating}')
```

```
The result is: Toy Story with rating: 10
```

format method

```
In [36]:
title  = 'Toy Story'
rating = 10
print('The result is: {} with rating: {}'.format(title, rating))
```

```
The result is: Toy Story with rating: 10
```

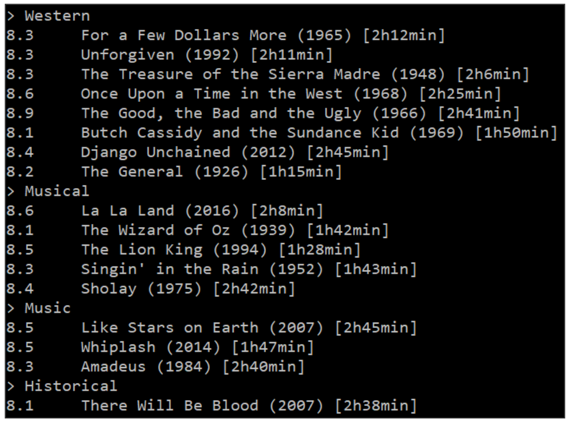
The ancient way (python 2)

In [37]:
```
title  = 'Toy Story'
rating = 10
print('The result is: %s with rating: %s' % (title, rating))
```

```
The result is: Toy Story with rating: 10
```

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