Introduction to

with Application to Bioinformatics

- Day 4

```
In [ ]:  row = 'sofa|2000|buy|Uppsala'
         fields = row.split('|')
         price = fields[1]
         if price == 2000:
             print('The price is a number!')
         if price == '2000':
             print('The price is a string!')
```

```
In [ ]:  print(sorted([ 2000,   30,   100 ]))
```

```
In [ ]:  print(sorted(['2000', '30', '100', '5', '301']))
```

```
In [ ]:  row = 'sofa|2000|buy|Uppsala'
         fields = row.split('|')
         price = fields[1]
         if price == 2000:
             print('The price is a number!')
         if price == '2000':
             print('The price is a string!')
```

```
In [ ]:  print(sorted([ 2000,   30,   100 ]))
```

```
In [ ]:  print(sorted(['2000', '30', '100', '5', '301']))
```

```
In [ ]:  ord('3')
```

```
In [ ]: 30 > 2000
```

```
In [ ]: '30' > '2000'
```

```
In [ ]: 30 > int('2000')
```

```
In [ ]: '12345'[2]
```

```
In [ ]: 12345[2]
```

```
In [ ]: max('2000')
```

```
In [ ]: max(2000)
```

```
In [ ]: math.cos(3.14)
```

```
In [ ]: math.cos('3.14')
```

```
In [ ]:  'ACTG'.lower()
```

```
In [ ]:  [1, 2, 3].lower()
```

```
In [ ]:  set([]).add('tiger')
```

```
In [ ]:  [].add('tiger')
```

```
In [ ]: 'ACTG'.lower()
```

```
In [ ]: [1, 2, 3].lower()
```

```
In [ ]: set([]).add('tiger')
```

```
In [ ]: [].add('tiger')
```

- How to find what methods are available: Python documentation, or `dir()`

```
In [ ]: dir('ACTG') # list all attributes
```

```python
float('2000')
```

```python
float('0.5')
```

```python
float('1e9')
```

```python
float('1e-2')
```

```python
int('2000')
```

```python
int('1.5')
```

```python
int('1e9')
```

```python
In [ ]: bool(1)
```

```python
In [ ]: bool(0)
```

```python
In [ ]: bool('1')
```

```python
In [ ]: bool('0')
```

```python
In [ ]: bool([])
```

```python
In [ ]: bool({})
```

```
In [ ]: values = [1, 0, '', '0', '1', [], [0]]
        for x in values:
            if x:
                print(repr(x), 'is true!')
            else:
                print(repr(x), 'is false!')
```

```
In [ ]: values = [1, 0, '', '0', '1', [], [0]]
        for x in values:
            if x:
                print(repr(x), 'is true!')
            else:
                print(repr(x), 'is false!')
```

- if x is equivalent to if bool(x)

```
In [ ]:  1 == True
```

```
In [ ]:  x = 1
         if x is True:
             print(repr(x), 'is true!')
         else:
             print(repr(x), 'is false!')
```

```
In [ ]:  x = 1
         if bool(x) is True:
             print(repr(x), 'is true!')
         else:
             print(repr(x), 'is false!')
```

```
In [ ]: 1 == True
```

```
In [ ]: x = 1
        if x is True:
            print(repr(x), 'is true!')
        else:
            print(repr(x), 'is false!')
```

```
In [ ]: x = 1
        if bool(x) is True:
            print(repr(x), 'is true!')
        else:
            print(repr(x), 'is false!')
```

- Be careful: if x is True is **not** equivalent to if bool(x) is True

```
In [ ]:  genre_list = ["comedy", "drama", "drama", "sci-fi"]
         genre_list
```

```
In [ ]:  genres = set(genre_list)
         genres
```

```
In [ ]:  'drama' in genre_list
         'drama' in genres
         # which operation is faster?
```

```
In [ ]:  genre_counts = {"comedy": 1, "drama": 2, "sci-fi": 1}
         genre_counts
```

```
In [ ]:  movie = {"rating": 10.0, "title": "Toy Story"}
         movie
```

```
In [ ]: def echo(message): # starts a new function definition
            # this function echos the message
            print(message) # print state of the variable
            return message # return the value to end the function
```

```python
In [ ]: list("hello")
```

```python
In [ ]: str(['h', 'e', 'l', 'l', 'o'])
```

```python
In [ ]: '_'.join(['h', 'e', 'l', 'l', 'o'])
```

```python
def increment_by_two(number):
    number += 2
    return number

print(increment_by_two(100))
```

```
In [ ]:  HOST = 'global'

         def show_host():
             print('HOST inside the function = {HOST}')

         show_host()
         print(f'HOST outside the function = {HOST}')
```

```python
HOST = 'global'

def change_host():
    HOST = 'local'
    print(f'HOST inside the function = {HOST}')
def app2():
    print(HOST)
print(f'HOST outside the function before change = {HOST}')
change_host()
print(f'HOST outside the function after change  = {HOST}')
app2()
```

```python
HOST = 'global'

def change_host(HOST):
    HOST = 'local'
    print(f'HOST inside the function = {HOST}')

print(f'HOST outside the function before change = {HOST}')
change_host(HOST)
print(f'HOST outside the function after change  = {HOST}')
```

```
In [ ]:  MOVIES = ['Toy story', 'Home alone']

         def change_movie():
             MOVIES = ['Fargo', 'The Usual Suspects']
             print(f'MOVIES inside the function = {MOVIES}')

         print(f'MOVIES outside the function before change = {MOVIES}')
         change_movie()
         print(f'MOVIES outside the function after change  = {MOVIES}')
```

```
In [ ]: MOVIES = ['Toy story', 'Home alone']

def change_movie():
    MOVIES.extend(['Fargo', 'The Usual Suspects'])
    print(f'MOVIES inside the function = {MOVIES}')

print(f'MOVIES outside the function before change = {MOVIES}')
change_movie()
print(f'MOVIES outside the function after change  = {MOVIES}')
```

```
In [ ]:  MOVIES = ['Toy story', 'Home alone']

         def change_movie():
             MOVIES.extend(['Fargo', 'The Usual Suspects'])
             print(f'MOVIES inside the function = {MOVIES}')

         print(f'MOVIES outside the function before change = {MOVIES}')
         change_movie()
         print(f'MOVIES outside the function after change  = {MOVIES}')
```

Take away: be careful when using global variables. Do not use it unless you know what you are doing.

```
In [ ]: def cytosine_count(nucleotides):
            count = 0
            for x in nucleotides:
                if x == 'c' or x == 'C':
                    count += 1
            return count

        count1 = cytosine_count('CATATTAC')
        count2 = cytosine_count('tagtag')
        print(count1, "\n", count2)
```

```
cytosine_count('catattac') + cytosine_count('tactactac')
```

```
def print_cytosine_count(nucleotides):
    count = 0
    for x in nucleotides:
        if x == 'c' or x == 'C':
            count += 1
    print(count)

print_cytosine_count('CATATTAC')
print_cytosine_count('tagtag')
```

```
print_cytosine_count('catattac') + print_cytosine_count('tactactac')
```

```python
def foo():
    do_nothing = 1

result = foo()
print(f'Return value of foo() = {result}')
```

```
In [ ]:  def foo():
             do_nothing = 1

         result = foo()
         print(f'Return value of foo() = {result}')
```

- Use `return` for all values that you might want to use later in your program

```
In [ ]: None == 0
```

```
In [ ]: None == False
```

```
In [ ]: None == ''
```

```
In [ ]: bool(None)
```

```
In [ ]: type(None)
```

```python
In [ ]: fh = open('files/recipes.txt',  mode='w', encoding='utf-8')
```

```python
In [ ]: sorted([1, 4, 100, 5, 6], reverse=True)
```

```
In [ ]:  record = 'gene_id INSR "insulin receptor"'

         record.split(' ', 2)
```

```
In [ ]: record = 'gene_id INSR "insulin receptor"'

        record.split(' ', 2)
```

```
In [ ]: record.split(sep=' ', maxsplit=2)
```

```
In [ ]: record = 'gene_id INSR "insulin receptor"'

        record.split(' ', 2)
```

```
In [ ]: record.split(sep=' ', maxsplit=2)
```

- It increases the clarity and redability

```
In [ ]: fh = open('files/recipes.txt', mode='w', encoding='utf-8'); fh.close()
```

```
In [ ]: fh = open('files/recipes.txt', encoding='utf-8', mode='w'); fh.close()
```

```
In [ ]:  fh = open('files/recipes.txt', 'w', encoding='utf-8'); fh.close()
```

```
In [ ]:  fh = open('files/recipes.txt',  mode='w', encoding='utf-8'); fh.close()
```

```
In [ ]: fh = open('files/recipes.txt', encoding='utf-8', 'w'); fh.close()
```

```
In [ ]:  fh = open('files/recipes.txt', encoding='utf-8', 'w'); fh.close()
```

- Positional arguments must be in front of keyword arguments

```
In [ ]: sorted([1, 4, 100, 5, 6], reverse=True)
```

```
In [ ]: sorted([1, 4, 100, 5, 6], True)
```

```
In [ ]:  sorted([1, 4, 100, 5, 6], reverse=True)
```

```
In [ ]:  sorted([1, 4, 100, 5, 6], True)
```

```
In [ ]:  sorted(iterable, /, *, key=None, reverse=False)
```

- arguments before / must be specified with position
- arguments after * must be specified with keyword

```
In [ ]:  def format_sentence(subject, value = 13, end = "...."):
             return 'The ' + subject + ' is ' + value + end

         print(format_sentence('lecture', 'ongoing', '.'))

         print(format_sentence('lecture', '!',  value='ongoing'))

         print(format_sentence(subject='lecture', value='ongoing', end='...'))
```

```
In [ ]:  def format_sentence(subject, value, end='.'):
             return 'The ' + subject + ' is ' + value + end

         print(format_sentence('lecture', 'ongoing'))

         print(format_sentence('lecture', 'ongoing', '...'))
```

```
In [ ]: def format_sentence(subject, value, end='.', second_value=0):
            if second_value is None:
                return 'The ' + subject + ' is ' + value + end
            else:
                return 'The ' + subject + ' is ' + value + ' and ' + second_val

        print(format_sentence('lecture', 'ongoing'))

        print(format_sentence('lecture', 'ongoing',
                              second_value='self-referential', end='!'))
```

- Collect all related functions in one file
- Import a module to use its functions
- Only need to understand what the functions do, not how

```python
import sys

sys.argv[1]
```

```python
from datetime import datetime
print(datetime.now())
```

```python
import os

os.system("ls")
```

- Standard modules: no installation needed
- Other libraries: install with `pip install` or `conda install`

```
In [ ]: text = 'Programming,is,cool'
        text.split(sep=',')
```

```
In [ ]:  text = 'Programming,is,cool'
         text.split(sep=',')
```

```
In [ ]:  help(text.split)
```

```
In [ ]:  text = 'Programming,is,cool'
         text.split(sep=',')
```

```
In [ ]:  help(text.split)
```

```
In [ ]:  text.split(sep=',')
```

```
In [ ]: import urllib

        help(urllib)
```

```
In [ ]:  import urllib

         help(urllib)
```

- Probably easier to find the answer by searching the web or using ChatGPT

```
In [ ]:  import urllib.request

         url = "https://www.python.org/static/img/python-logo@2x.png"
         filename = "python-logo.png"   # The name you want to give to the downlo

         urllib.request.urlretrieve(url, filename)

         print("Download completed.")
```

```
In [ ]: help(process_file)
```

```
In [ ]: help(process_file)
```

- This works because somebody else has documented their code!

```
In [ ]: help(process_file)
```

- This works because somebody else has documented their code!

```python
In [ ]: def process_file(filename, chrom, pos):
            """
            Read a very large vcf file, search for lines matching
            chromosome chrom and position pos.

            Print the genotypes of the matching lines.
            """
            for line in open(filename):
                if not line.startswith('#'):
                    col = line.split('\t')
                    if col[0] == chrom and int(col[1]) == pos:
                        print(col[9:])
```

Write documentation for both of them!

- library users (docstrings):

```
"""
What does this function do?
"""
```

- maintainers (comments):

```
# implementation details
```

- At the beginning of the file

```
"""

    This module provides functions for ...
"""
```

- At every function definition

In [ ]:
```python
import random
def make_list(x):
    """Returns a random list of length x."""
    li = list(range(x))
    random.shuffle(li)
    return li
```
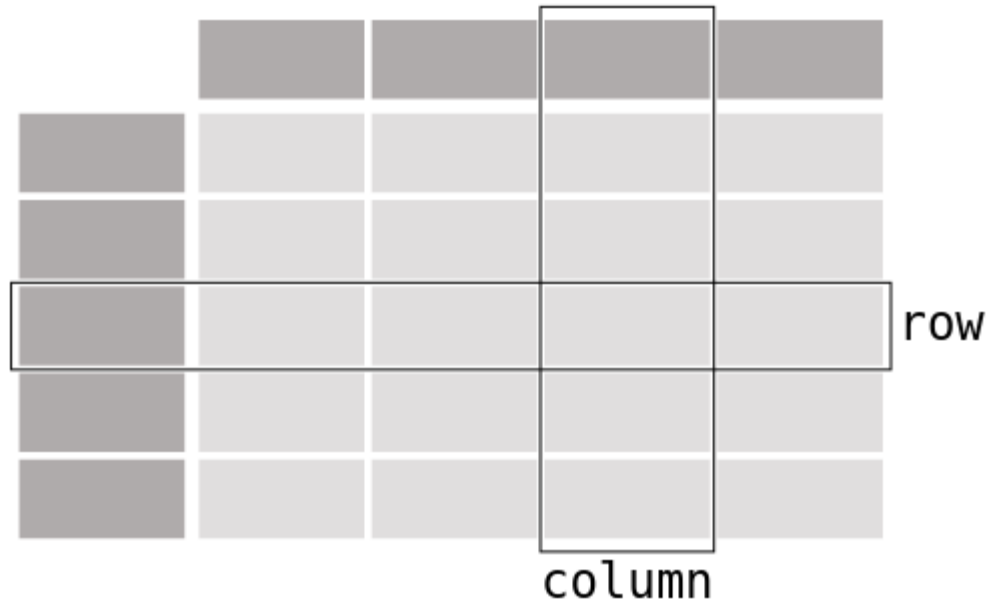
- Wherever the code is hard to understand

```
In [ ]: my_list[5] += other_list[3]   # explain why you do this!
```
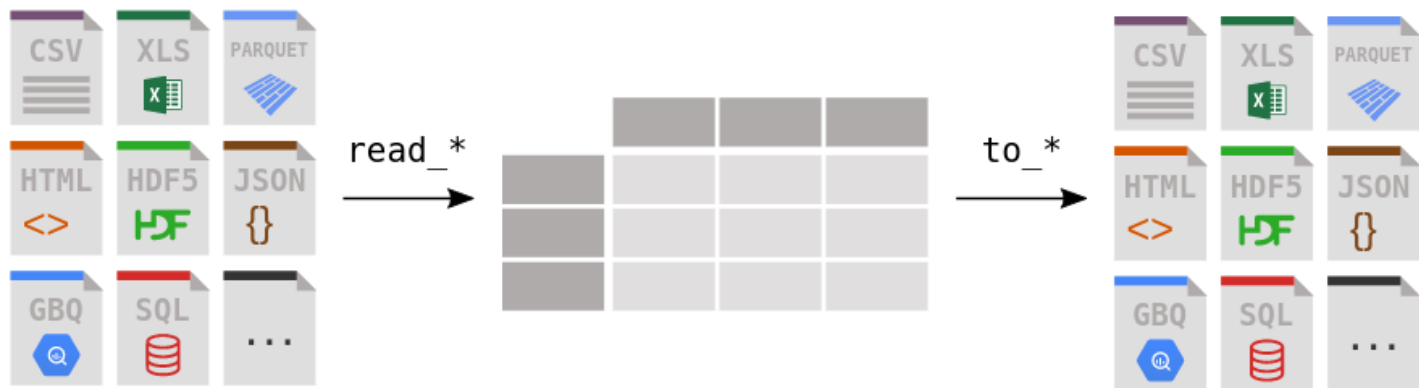
```
In [ ]:  from files import mywork
         mywork.pipeline(["accctt", "gaccct"])
```

Lunch

DataFrame

```
In [ ]:  import pandas as pd
         data = {
             'age': [1,2,3,4],
             'circumference': [2,3,5,10],
             'height': [30, 35, 40, 50]
         }
         df = pd.DataFrame(data)
         df
```

CSV	XLS	PARQUET
HTML	HDF5	JSON
GBQ	SQL	...

read_*

to_*

CSV	XLS	PARQUET
HTML	HDF5	JSON
GBQ	SQL	...

```
In [ ]:  df = pd.read_table('../downloads/Orange_1.tsv')
         df
```

```
In [ ]: df = pd.read_table('../downloads/Orange_1.tsv')
        df
```

- One implict index (0, 1, 2, 3)
- Columns: `age`, `circumference`, `height`
- Rows: one per data point, identified by their index

```
In [ ]:  df2 = pd.read_excel('../downloads/Orange_1.xlsx')
         df2
```

```
In [ ]: df
```

```
In [ ]: df.shape
```

```
In [ ]: df.describe()
```

```
In [ ]: df.max()
```

```
In [ ]: df
```

```
In [ ]: df_new = df.age
        df_new
```

```
In [ ]: df['age']
```

```
In [ ]: df
```

```
In [ ]: df[['age', 'height']]
```

```
In [ ]: df
```

```
In [ ]: df[['age', 'height']]
```

```
In [ ]: df[['height', 'age']] # what's the difference?
```

```
In [ ]: df
```

```
In [ ]: df.loc[0] # select the first row
```

```
In [ ]: df.loc[1:3] # select from row 2 to 4
```
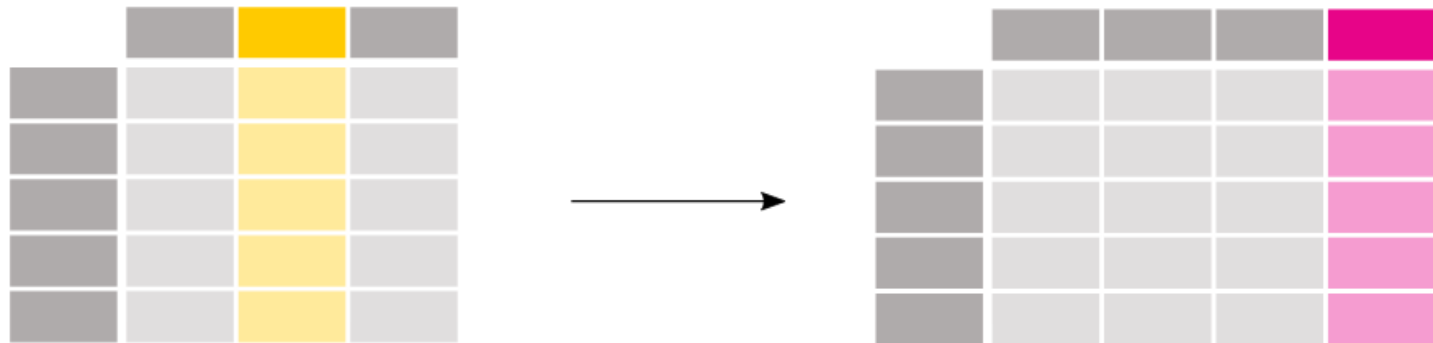
```
In [ ]: df.loc[[1, 3, 0]] # select row 2 and 4
```

```
In [ ]: df
```

```
In [ ]: df.loc[[0], ['age']]
```

```
In [ ]: df[['age', 'circumference']].describe()
```

```
In [ ]: df['age'].std()
```

```
In [ ]: import math
        df['radius'] = df['circumference'] / 2.0 / math.pi

        df
```

```
In [ ]:  df1 = pd.DataFrame({
             'age': [1,2,3,4],
             'circumference': [2,3,5,10],
             'height': [30, 35, 40, 50]
         })
```

```
In [ ]:  df2 = pd.DataFrame({
             'name': ['palm', 'ada', 'ek', 'olive'],
             'price': [1423, 2000, 102, 30]
         })
```

```
In [ ]: df2
```

```
In [ ]: df = pd.read_table('../downloads/Orange.tsv')
        df.head(3)  # can also use .head()
```

```
In [ ]: df.Tree.unique()
```

```
In [ ]: df[df['Tree'] == 1]
```

```
In [ ]: df[df.age > 500]
```

```
In [ ]: df[(df.age > 500) & (df.circumference < 100) ]
```

```
In [ ]: df
```

In [ ]:

```
In [ ]: small_df = pd.read_table('../downloads/Orange_1.tsv')
        small_df
```

```python
import matplotlib.pyplot as plt
plt.show()
```

```python
%matplotlib inline
```

```
In [ ]: small_df[['age']].plot(kind='bar')
```

```
In [ ]: small_df[['circumference', 'age']].plot(kind='bar')
```

```
In [ ]: small_df.plot(kind='hist', y = 'age', fontsize=18)
```

```
In [ ]: small_df.plot(kind='box', y = 'age')
```