Introduction to

python

with Application to Bioinformatics

- Day 4

Start by doing today's quiz

Go to Canvas, `Modules -> Day 4 -> Review Day 3`

~20 minutes

In what ways does the type of an object matter?

- Questions 1, 2 and 3

In [1]:
```
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!')
```

```
The price is a string!
```

In [2]:
```
print(sorted([ 2000,   30,   100 ]))
```

```
[30, 100, 2000]
```

In [3]:
```
print(sorted(['2000', '30', '100']))
```

```
['100', '2000', '30']
```

In what ways does the type of an object matter?

- Each type store a specific type of information
    - `int` for integers,
    - `float` for floating point values (decimals),
    - `str` for strings,
    - `list` for lists,
    - `dict` for dictionaries.
- Each type supports different operations, functions and methods.

- Each type supports different **operations**

In [4]:
```
30 > 2000
```

Out[4]:
```
False
```

In [5]:
```
'30' > '2000'
```

Out[5]:
```
True
```

- Each type supports different **functions**

In [6]:
```
max('2000')
```

Out[6]:
```
'2'
```

In [7]:
```
max(2000)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent
call last)
Cell In[7], line 1
----> 1 max(2000)

TypeError: 'int' object is not iterable
```

- Each type supports different **methods**

In [8]:
```
'ACTG'.lower()
```

Out[8]:
```
'actg'
```

In [9]:
```
[1, 2, 3].lower()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[9], line 1
----> 1 [1, 2, 3].lower()

AttributeError: 'list' object has no attribute 'lower'
```

- Each type supports different **methods**

In [8]:
```
'ACTG'.lower()
```

Out[8]:
```
'actg'
```

In [9]:
```
[1, 2, 3].lower()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent
call last)
Cell In[9], line 1
----> 1 [1, 2, 3].lower()

AttributeError: 'list' object has no attribute 'lower'
```

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

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

Convert string to number

- Questions 4, 5 and 6

In [10]:
```
float('2000')
```

Out[10]:
```
2000.0
```

In [11]:
```
float('0.5')
```

Out[11]:
```
0.5
```

In [12]:
```
float('1e9')
```

Out[12]:
```
1000000000.0
```

Convert to boolean: `1`, `0`, `'1'`, `'0'`, `''`, `{}`

- Question 7

In [13]:
```
values = [1, 0, '', '0', '1', [], [0]]
for x in values:
    if x:
        print(repr(x), 'is true!')
    else:
        print(repr(x), 'is false!')
```

```
1 is true!
0 is false!
'' is false!
'0' is true!
'1' is true!
[] is false!
[0] is true!
```

Convert to boolean: `1`, `0`, `'1'`, `'0'`, `''`, `{}`

- Question 7

In [13]:
```
values = [1, 0, '', '0', '1', [], [0]]
for x in values:
    if x:
        print(repr(x), 'is true!')
    else:
        print(repr(x), 'is false!')
```

```
1 is true!
0 is false!
'' is false!
'0' is true!
'1' is true!
[] is false!
[0] is true!
```

- `if x` is equivalent to `if bool(x)`

Container types, when should you use which? (Question 8)

- **lists**: when order is important
- **dictionaries**: to keep track of the relation between keys and values
- **sets**: to check for membership. No order, no duplicates.

In [14]:
```
genre_list = ["comedy", "drama", "drama", "sci-fi"]
genre_list
```

Out[14]:
```
['comedy', 'drama', 'drama', 'sci-fi']
```

In [15]:
```
genres = set(genre_list)
genres
```

Out[15]:
```
{'comedy', 'drama', 'sci-fi'}
```

In [17]:
```
genre_counts = {"comedy": 1, "drama": 2, "sci-fi": 1}
genre_counts
```

Out[17]:
```
{'comedy': 1, 'drama': 2, 'sci-fi': 1}
```

In [18]:
```
movie = {"rating": 10.0, "title": "Toy Story"}
movie
```

Out[18]:
```
{'rating': 10.0, 'title': 'Toy Story'}
```

Python syntax (Question 9)

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
```

Converting between strings and lists

- Question 10

In [19]:
```
list("hello")
```

Out[19]:
```
['h', 'e', 'l', 'l', 'o']
```

In [20]:
```
'_'.join('hello')
```

Out[20]:
```
'h_e_l_l_o'
```

TODAY

- More on functions:
    - scope of variables
    - positional arguments and keyword arguments
    - `return` statement
- Reusing code:
    - comments and documentation
    - importing modules: using libraries
- Pandas - explore your data!

More on functions: scope - global vs local variables

- Global variables can be accessed inside the function

In [21]:
```
HOST = 'global'

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

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

```
HOST inside the function = global
HOST outside the function = global
```

- Change in the function will not change the global variable

In [22]:
```
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()
```

```
HOST outside the function before change = global
HOST inside the function = local
HOST outside the function after change  = global
global
```

Will the global variable never to changed by function?

In [23]:
```
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}')
```

```
MOVIES outside the function before change = ['Toy story', 'Home
alone']
MOVIES inside the function = ['Toy story', 'Home alone', 'Fargo'
, 'The Usual Suspects']
MOVIES outside the function after change  = ['Toy story', 'Home
alone', 'Fargo', 'The Usual Suspects']
```

Will the global variable never to changed by function?

In [23]:
```
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}')
```

```
MOVIES outside the function before change = ['Toy story', 'Home
alone']
MOVIES inside the function = ['Toy story', 'Home alone', 'Fargo'
, 'The Usual Suspects']
MOVIES outside the function after change  = ['Toy story', 'Home
alone', 'Fargo', 'The Usual Suspects']
```

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

More on functions: `return` statement

A function that counts the number of occurences of `'C'` in the argument string.

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

```
2
 0
```

Functions that `return` are easier to repurpose than those that `print` their result

In [25]:
```
cytosine_count('catattac') + cytosine_count('tactactac')
```

Out[25]:
```
5
```

In [26]:
```
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')
```

```
2
0
```

In [27]:
```
print_cytosine_count('catattac') + print_cytosine_count('tactactac')
```

```
2
3
```

```
---------------------------------------------------------------
-----------
TypeError                                 Traceback (most recent
call last)
Cell In[27], line 1
----> 1 print_cytosine_count('catattac') + print_cytosine_count
('tactactac')

TypeError: unsupported operand type(s) for +: 'NoneType' and 'No
neType'
```

- Functions without any `return` statement returns `None`

In [28]:
```
def foo():
    do_nothing = 1

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

```
Return value of foo() = None
```

- Functions without any `return` statement returns `None`

In [28]:
```
def foo():
    do_nothing = 1

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

```
Return value of foo() = None
```

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

Small detour: Python's value for missing values: `None`

- Default value for optional arguments
- Implicit return value of functions without a `return` statement
- `None` is `None`, not anything else

```
In [29]: None == 0
```

```
Out[29]: False
```

```
In [30]: None == False
```

```
Out[30]: False
```

```
In [31]: None == ''
```

```
Out[31]: False
```

```
In [32]: bool(None)
```

```
Out[32]: False
```

```
In [33]: type(None)
```

```
Out[33]: NoneType
```

Keyword arguments

In [34]:
```
fh = open('../files/fruits.txt',  mode='w', encoding='utf-8'); fh.close(
```

In [35]:
```
sorted([1, 4, 100, 5, 6], reverse=True)
```

Out[35]:
```
[100, 6, 5, 4, 1]
```

Why do we use keyword arguments?

In [36]:
```
record = 'gene_id INSR "insulin receptor"'

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

Out[36]:
```
['gene_id', 'INSR', '"insulin receptor"']
```

Why do we use keyword arguments?

In [36]:
```
record = 'gene_id INSR "insulin receptor"'

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

Out[36]:
```
['gene_id', 'INSR', '"insulin receptor"']
```

In [37]:
```
record.split(sep=' ', maxsplit=2)
```

Out[37]:
```
['gene_id', 'INSR', '"insulin receptor"']
```

Why do we use keyword arguments?

In [36]:
```
record = 'gene_id INSR "insulin receptor"'

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

Out[36]:
```
['gene_id', 'INSR', '"insulin receptor"']
```

In [37]:
```
record.split(sep=' ', maxsplit=2)
```

Out[37]:
```
['gene_id', 'INSR', '"insulin receptor"']
```

- It increases the clarity and readability

The order of keyword arguments does not matter

In [38]:
```
fh = open('../files/fruits.txt', mode='w', encoding='utf-8'); fh.close()
```

In [39]:
```
fh = open('../files/fruits.txt', encoding='utf-8', mode='w'); fh.close()
```

Can be used in both ways, with or without keyword

- if there is no ambiguity

In [40]:
```
fh = open('../files/fruits.txt', 'w', encoding='utf-8'); fh.close()
```

In [41]:
```
fh = open('../files/fruits.txt',  mode='w', encoding='utf-8'); fh.close(
```

But there are some exceptions

In [42]:
```
fh = open('files/recipes.txt', encoding='utf-8', 'w'); fh.close()
```

```
  Cell In[42], line 1
    fh = open('files/recipes.txt', encoding='utf-8', 'w'); fh.cl
ose()
                                                           ^
SyntaxError: positional argument follows keyword argument
```

But there are some exceptions

In [42]:
```
fh = open('files/recipes.txt', encoding='utf-8', 'w'); fh.close()
```

```
  Cell In[42], line 1
    fh = open('files/recipes.txt', encoding='utf-8', 'w'); fh.cl
ose()
                                                           ^
SyntaxError: positional argument follows keyword argument
```

- Positional arguments must be in front of keyword arguments

Restrictions by purpose

In [43]:
```
sorted([1, 4, 100, 5, 6], reverse=True)
```

Out[43]:
```
[100, 6, 5, 4, 1]
```

In [44]:
```
sorted([1, 4, 100, 5, 6], True)
```

```
---------------------------------------------------------------------
-----------
TypeError                                 Traceback (most recent
call last)
Cell In[44], line 1
----> 1 sorted([1, 4, 100, 5, 6], True)

TypeError: sorted expected 1 argument, got 2
```

Restrictions by purpose

In [43]:
```
sorted([1, 4, 100, 5, 6], reverse=True)
```

Out[43]:
```
[100, 6, 5, 4, 1]
```

In [44]:
```
sorted([1, 4, 100, 5, 6], True)
```

```
---------------------------------------------------------------
-----------
TypeError                                 Traceback (most recent
call last)
Cell In[44], line 1
----> 1 sorted([1, 4, 100, 5, 6], True)

TypeError: sorted expected 1 argument, got 2
```

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

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

How to define functions taking keyword arguments

- Just define them as usual:

In [45]:
```
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='...'))
```

```
The lecture is ongoing.

---------------------------------------------------------------------
-----------
TypeError                                 Traceback (most recent
call last)
Cell In[45], line 6
      2     return 'The ' + subject + ' is ' + value + end
      4 print(format_sentence('lecture', 'ongoing', '.'))
----> 6 print(format_sentence('lecture', '!',  value='ongoing'))
      8 print(format_sentence(subject='lecture', value='ongoin
g', end='...'))

TypeError: format_sentence() got multiple values for argument 'v
alue'
```

Defining functions with default arguments

In [46]:
```
def format_sentence(subject, value, end='.'):
    return 'The ' + subject + ' is ' + value + end

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

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

```
The lecture is ongoing...
```

Defining functions with optional arguments

- Convention: use the object `None`

In [48]:
```
def format_sentence(subject, value, end='.', second_value=None):
    if second_value is None:
        return 'The ' + subject + ' is ' + value + end
    else:
        return 'The ' + subject + ' is ' + value + ' and ' + second_valu

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

print(format_sentence('lecture', 'ongoing', second_value='self-referenti
```

```
The lecture is ongoing.
The lecture is ongoing and self-referential!
```

Exercise 1

- Notebook Day_4_Exercise_1 (~30 minutes)
- Go to Canvas, `Modules -> Day 4 -> Exercise 1 - day 4`
- Extra reading:
    - https://realpython.com/python-kwargs-and-args/
    - https://able.bio/rhett/python-functions-and-best-practices--78aclaa

A short note on code structure

- Functions
    - e.g. sum(), print(), open()
- Modules
    - files containing a collection of functions and methods, e.g. string.py
- Documentation
    - docstring, comments

Why functions?

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

Why modules?

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

Why modules?

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

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

Example of modules

In [49]:
```
import sys

sys.argv[1]
```

Out[49]:
```
'-f'
```

In [50]:
```
from datetime import datetime
print(datetime.now())
```

```
2024-11-14 14:43:47.521944
```

In [ ]:
```
import os

os.system("ls")
```

How to find the right module and instructions?

- Look at the module index for Python standard modules
- Search PyPI
- Search https://www.w3schools.com/python/
- Ask your colleagues
- Search the web
- Use ChatGPT

How to find the right module and instructions?

- Look at the module index for Python standard modules
- Search PyPI
- Search https://www.w3schools.com/python/
- Ask your colleagues
- Search the web
- Use ChatGPT

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

How to understand it?

- E.g. I want to know how to split a string by the separator `,`

In [53]:
```
text = 'Programming,is,cool'
```

How to understand it?

- E.g. I want to know how to split a string by the separator `,`

In [53]:
```
text = 'Programming,is,cool'
```

In [54]:
```
help(text.split)
```

```
Help on built-in function split:

split(sep=None, maxsplit=-1) method of builtins.str instance
    Return a list of the words in the string, using sep as the d
elimiter string.

    sep
      The delimiter according which to split the string.
      None (the default value) means split according to any whit
espace,
      and discard empty strings from the result.
    maxsplit
      Maximum number of splits to do.
      -1 (the default value) means no limit.
```

How to understand it?

- E.g. I want to know how to split a string by the separator `,`

In [53]:
```
text = 'Programming,is,cool'
```

In [54]:
```
help(text.split)
```

```
Help on built-in function split:

split(sep=None, maxsplit=-1) method of builtins.str instance
    Return a list of the words in the string, using sep as the d
elimiter string.

    sep
      The delimiter according which to split the string.
      None (the default value) means split according to any whit
espace,
      and discard empty strings from the result.
    maxsplit
      Maximum number of splits to do.
      -1 (the default value) means no limit.
```

In [55]:
```
text.split(sep=',')
```

Out[55]:
```
['Programming', 'is', 'cool']
```

For slightly more complicated problems

- e.g. how to download Python logo from internet with `urllib` , given the URL https://www.python.org/static/img/python-logo@2x.png

In [56]:
```
import urllib

help(urllib)
```

```
Help on package urllib:

NAME
    urllib

MODULE REFERENCE
    https://docs.python.org/3.9/library/urllib

    The following documentation is automatically generated from
the Python
    source files.  It may be incomplete, incorrect or include fe
atures that
    are considered implementation detail and may vary between Py
thon
    implementations.  When in doubt, consult the module referenc
e at the
    location listed above.

PACKAGE CONTENTS
    error
    parse
    request
    response
    robotparser

FILE
    /Users/kostas/opt/miniconda3/envs/python-workshop-teacher/li
b/python3.9/urllib/__init__.py
```

For slightly more complicated problems

- e.g. how to download Python logo from internet with `urllib` , given the URL https://www.python.org/static/img/python-logo@2x.png

In [56]:
```
import urllib

help(urllib)
```

```
Help on package urllib:

NAME
    urllib

MODULE REFERENCE
    https://docs.python.org/3.9/library/urllib

    The following documentation is automatically generated from
the Python
    source files.  It may be incomplete, incorrect or include fe
atures that
    are considered implementation detail and may vary between Py
thon
    implementations.  When in doubt, consult the module referenc
e at the
    location listed above.

PACKAGE CONTENTS
    error
    parse
    request
    response
    robotparser

FILE
    /Users/kostas/opt/miniconda3/envs/python-workshop-teacher/li
b/python3.9/urllib/__init__.py
```

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

One minute exercise

- get help from ChatGPT (https://chat.openai.com/)

Using Python to download the Python logo from internet with urllib providing the url as https://www.python.org/static/img/python-logo@2x.png

One minute exercise

- get help from ChatGPT (https://chat.openai.com/)

Using Python to download the Python logo from internet with urllib providing the url as https://www.python.org/static/img/python-logo@2x.png

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 downloa

urllib.request.urlretrieve(url, filename)

print("Download completed.")
```

Documentation and commenting your code

In [57]:
```
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:])
```

In [58]:
```
help(process_file)
```

```
Help on function process_file in module __main__:

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.
```

In [58]:
```
help(process_file)
```

```
Help on function process_file in module __main__:

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.
```

- This works because somebody has documented the code!

Your code may have two types of users:

- library users
- maintainers (maybe yourself!)

Your code may have two types of users:

- library users
- maintainers (maybe yourself!)

Write documentation for both of them!

- library users (docstrings):

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

- maintainers (comments):

```
# implementation details
```

Places for documentation

- At the beginning of the file

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

- At every function definition

In [59]:
```
import random
def make_list(x):
    """Returns a random list of length x."""
    li = list(range(x))
    random.shuffle(li)
    return li
```

Comments

- Wherever the code is hard to understand

In [ ]:
```
my_list[5] += other_list[3]  # explain why you do this!
```

Read more:

https://realpython.com/documenting-python-code/

https://www.python.org/dev/peps/pep-0008/?#comments

Quiz time

Go to Canvas, `Modules -> Day 4 -> PyQuiz 4.1`

~10 min

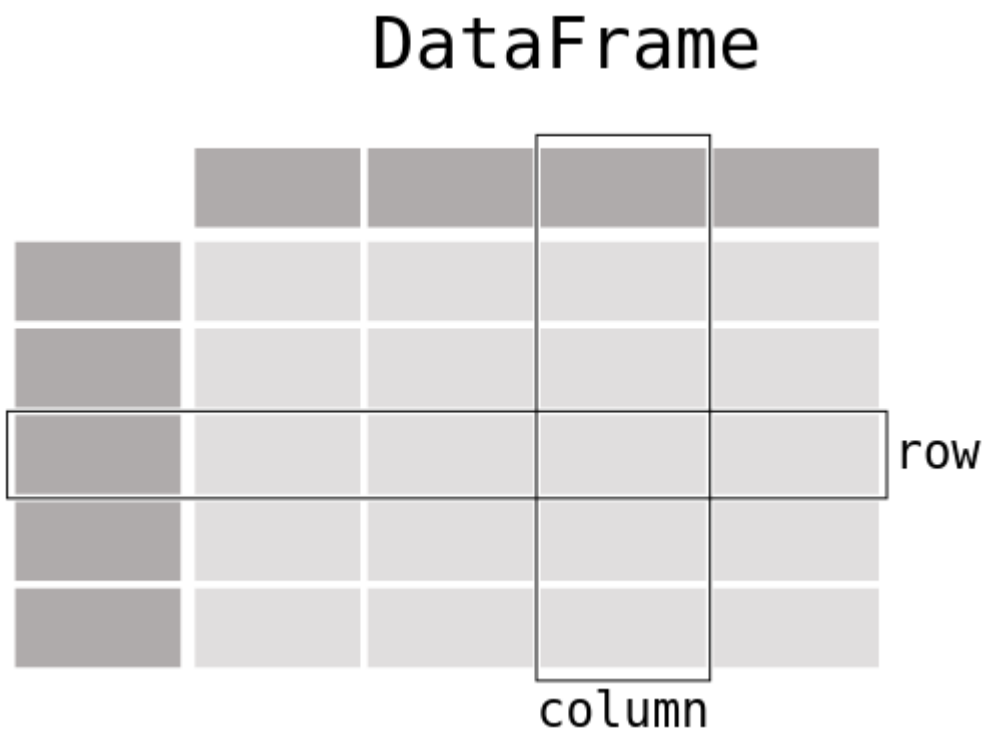
Lunch

Pandas!!!

Pandas

- Library for working with tabular data
- Data analysis:
    - filter
    - transform
    - aggregate
    - plot
- Main hero: the `DataFrame` type

DataFrame



Creating a small DataFrame

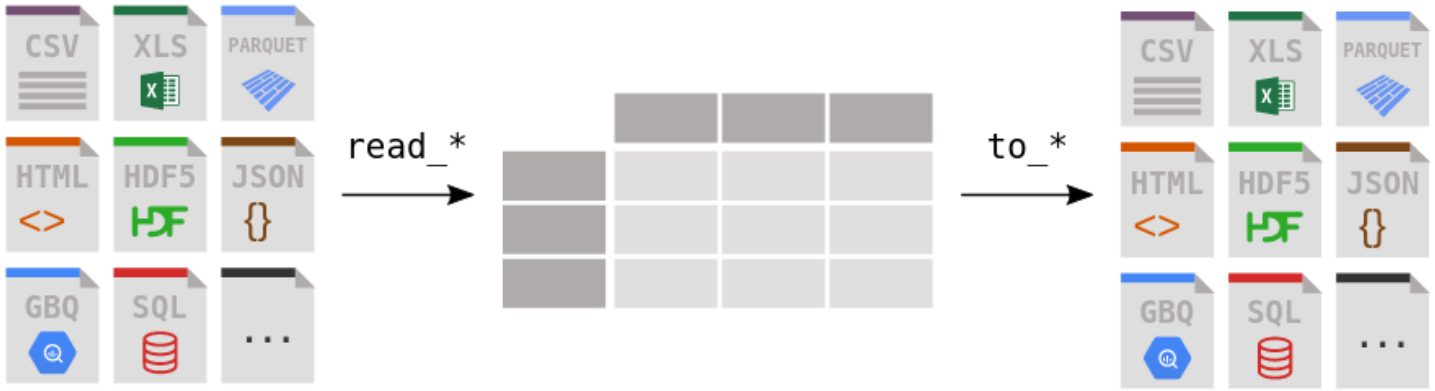
In [60]:
```
import pandas as pd
data = {
    'age': [1,2,3,4],
    'circumference': [2,3,5,10],
    'height': [30, 35, 40, 50]
}
df = pd.DataFrame(data)
df
```

Out[60]:

	age	circumference	height
0	1	2	30
1	2	3	35
2	3	5	40
3	4	10	50

Pandas can import data from many formats

- `pd.read_table` : tab separated values `.tsv`
- `pd.read_csv` : comma separated values `.csv`
- `pd.read_excel` : Excel spreadsheets `.xlsx`
- For a data frame `df` : `df.to_table()` , `df.to_csv()` , `df.to_excel()`



Orange tree data

In [61]:
```
df = pd.read_table('../downloads/Orange_1.tsv')
df
```

Out[61]:

	age	circumference	height
0	1	2	30
1	2	3	35
2	3	5	40
3	4	10	50

Orange tree data

In [61]:
```
df = pd.read_table('../downloads/Orange_1.tsv')
df
```

Out[61]:

	age	circumference	height
0	1	2	30
1	2	3	35
2	3	5	40
3	4	10	50

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

Read data from Excel file

In [62]:
```
df2 = pd.read_excel('../downloads/Orange_1.xlsx')
df2
```

Out[62]:

	age	circumference	height
0	1	2	30
1	2	3	35
2	3	5	40
3	4	10	50

Overview of your data, basic statistics

In [63]:
```
df
```

Out[63]:

	age	circumference	height
0	1	2	30
1	2	3	35
2	3	5	40
3	4	10	50

In [64]:
```
df.shape
```

Out[64]:
```
(4, 3)
```

In [65]:
```
df.describe()
```

Out[65]:

	age	circumference	height
count	4.000000	4.000000	4.000000
mean	2.500000	5.000000	38.750000
std	1.290994	3.559026	8.539126
min	1.000000	2.000000	30.000000
25%	1.750000	2.750000	33.750000
50%	2.500000	4.000000	37.500000
75%	3.250000	6.250000	42.500000
max	4.000000	10.000000	50.000000

In [66]:
```
df.max()
```

Out[66]:
```
age              4
circumference    10
height           50
dtype: int64
```

Selecting columns from a dataframe

```
dataframe.columnname
dataframe['columnname']
```

Selecting one column

In [67]:
```
df
```

Out[67]:

	age	circumference	height
0	1	2	30
1	2	3	35
2	3	5	40
3	4	10	50

In [68]:
```
df_new = df.age
df_new
```

Out[68]:
```
0    1
1    2
2    3
3    4
Name: age, dtype: int64
```

In [69]:
```
df['age']
```

Out[69]:
```
0    1
1    2
2    3
3    4
Name: age, dtype: int64
```

Selecting multiple columns

In [70]:
```
df
```

Out[70]:

	age	circumference	height
0	1	2	30
1	2	3	35
2	3	5	40
3	4	10	50

In [71]:
```
df[['age', 'height']]
```

Out[71]:

	age	height
0	1	30
1	2	35
2	3	40
3	4	50

Selecting multiple columns

In [70]:
```
df
```

Out[70]:

	age	circumference	height
0	1	2	30
1	2	3	35
2	3	5	40
3	4	10	50

In [71]:
```
df[['age', 'height']]
```

Out[71]:

	age	height
0	1	30
1	2	35
2	3	40
3	4	50

In [72]:
```
df[['height', 'age']]
```

Out[72]:

	height	age
0	30	1
1	35	2
2	40	3
3	50	4

Selecting rows from a dataframe

In [73]:
```
df
```

Out[73]:

	age	circumference	height
0	1	2	30
1	2	3	35
2	3	5	40
3	4	10	50

In [74]:
```
df.loc[0] # select the first row
```

Out[74]:
```
age               1
circumference     2
height           30
Name: 0, dtype: int64
```

In [75]:
```
df.loc[1:3] # select from row 2 to 4
```

Out[75]:

	age	circumference	height
1	2	3	35
2	3	5	40
3	4	10	50

In [76]:
```
df.loc[[1, 3, 0]] # select row 2, 4 and 1
```

Out[76]:

	age	circumference	height
1	2	3	35
3	4	10	50
0	1	2	30

Selecting cells from a dataframe

In [77]:
```
df
```

Out[77]:

	age	circumference	height
0	1	2	30
1	2	3	35
2	3	5	40
3	4	10	50

In [78]:
```
df.loc[[0], ['age']]
```

Out[78]:

	age
0	1

Run statistics on specific rows, columns, cells

In [79]:
```
df[['age', 'circumference']].describe()
```

Out[79]:

	age	circumference
count	4.000000	4.000000
mean	2.500000	5.000000
std	1.290994	3.559026
min	1.000000	2.000000
25%	1.750000	2.750000
50%	2.500000	4.000000
75%	3.250000	6.250000
max	4.000000	10.000000

In [80]:
```
df['age'].std()
```

Out[80]:
```
1.2909944487358056
```

Selecting data from a dataframe by index

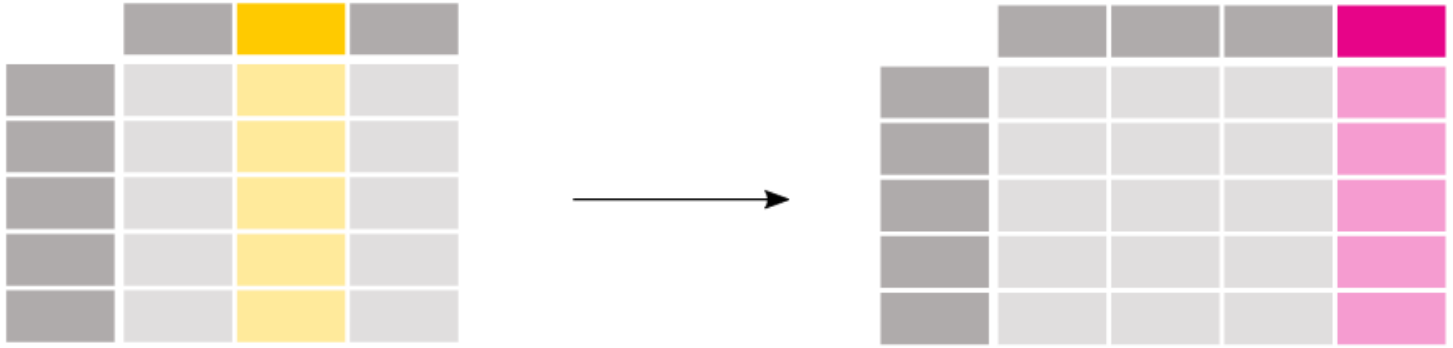
```
dataframe.iloc[index]
dataframe.iloc[start:stop]
```

Further reading from pandas documentation: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html

In [83]:
```
df
#df.iloc[:,0] # Show the first column
#df.iloc[1] # Show the second row
df.iloc[1,0] # Show the cell of the second row and the first column (you
```

Out[83]:
```
2
```

Creating new column derived from existing column



In [84]:
```
import math
df['radius'] = df['circumference'] / (2.0 * math.pi)
df
```

Out[84]:

	age	circumference	height	radius
0	1	2	30	0.318310
1	2	3	35	0.477465
2	3	5	40	0.795775
3	4	10	50	1.591549

Expand dataframe by concatenating

In [85]:
```
df1 = pd.DataFrame({
    'age': [1,2,3,4],
    'circumference': [2,3,5,10],
    'height': [30, 35, 40, 50]
})

df1
```

Out[85]:

	age	circumference	height
0	1	2	30
1	2	3	35
2	3	5	40
3	4	10	50

In [86]:
```
df2 = pd.DataFrame({
    'name': ['palm', 'ada', 'ek', 'olive'],
    'price': [1423, 2000, 102, 30]
})

df2
```

Out[86]:

	name	price
0	palm	1423
1	ada	2000
2	ek	102
3	olive	30

In [89]:
```
pd.concat([df2, df1], axis=0).reset_index(drop=True)
```

Out[89]:

	name	price	age	circumference	height
0	palm	1423.0	NaN	NaN	NaN
1	ada	2000.0	NaN	NaN	NaN
2	ek	102.0	NaN	NaN	NaN
3	olive	30.0	NaN	NaN	NaN
4	NaN	NaN	1.0	2.0	30.0
5	NaN	NaN	2.0	3.0	35.0
6	NaN	NaN	3.0	5.0	40.0
7	NaN	NaN	4.0	10.0	50.0

Selecting/filtering the dataframe by condition

e.g.

- Only trees with age larger than 100
- Only tree with circumference shorter than 20

Slightly bigger data frame of orange trees

In [90]:
```
df = pd.read_table('../downloads/Orange.tsv')
df.head(3)  # can also use .head()
```

Out[90]:

	Tree	age	circumference
0	1	118	30
1	1	484	58
2	1	664	87

In [91]:
```
df.Tree.unique()
```

Out[91]:
```
array([1, 2, 3])
```

Selecting with condition

In [92]:
```
df[df['Tree'] == 1]
```

Out[92]:

	Tree	age	circumference
0	1	118	30
1	1	484	58
2	1	664	87
3	1	1004	115
4	1	1231	120
5	1	1372	142
6	1	1582	145

In [93]:
```
df[df.age > 500]
```

Out[93]:

	Tree	age	circumference
2	1	664	87
3	1	1004	115
4	1	1231	120
5	1	1372	142
6	1	1582	145
9	2	664	111
10	2	1004	156
11	2	1231	172
12	2	1372	203
13	2	1582	203
16	3	664	75
17	3	1004	108
18	3	1231	115
19	3	1372	139
20	3	1582	140

In [94]:
```
df[(df.age > 500) & (df.circumference < 100) ]
```

Out[94]:

	Tree	age	circumference
2	1	664	87
16	3	664	75

Small exercise 1

- Find the maximal circumference and then filter the data frame by it

In [95]:
```
df
max_c=df.circumference.max()
max_c
df[df.circumference==max_c]
```

Out[95]:

	Tree	age	circumference
12	2	1372	203
13	2	1582	203

Small exercise 2

Here's a dictionary of students and their grades:

```
students = {'student': ['bob', 'sam', 'joe'], 'grade': [1, 3, 4]}
```

Use Pandas to:

- create a dataframe with this information
- get the mean value of the grades

In [96]:
```
students = {'student': ['bob', 'sam', 'joe'], 'grade': [1, 3, 4]}

ds=pd.DataFrame(students)

ds.grade.mean()
```

Out[96]:
```
2.6666666666666665
```

Plotting

```
df.columnname.plot()
```

Plotting

```
df.columnname.plot()
```

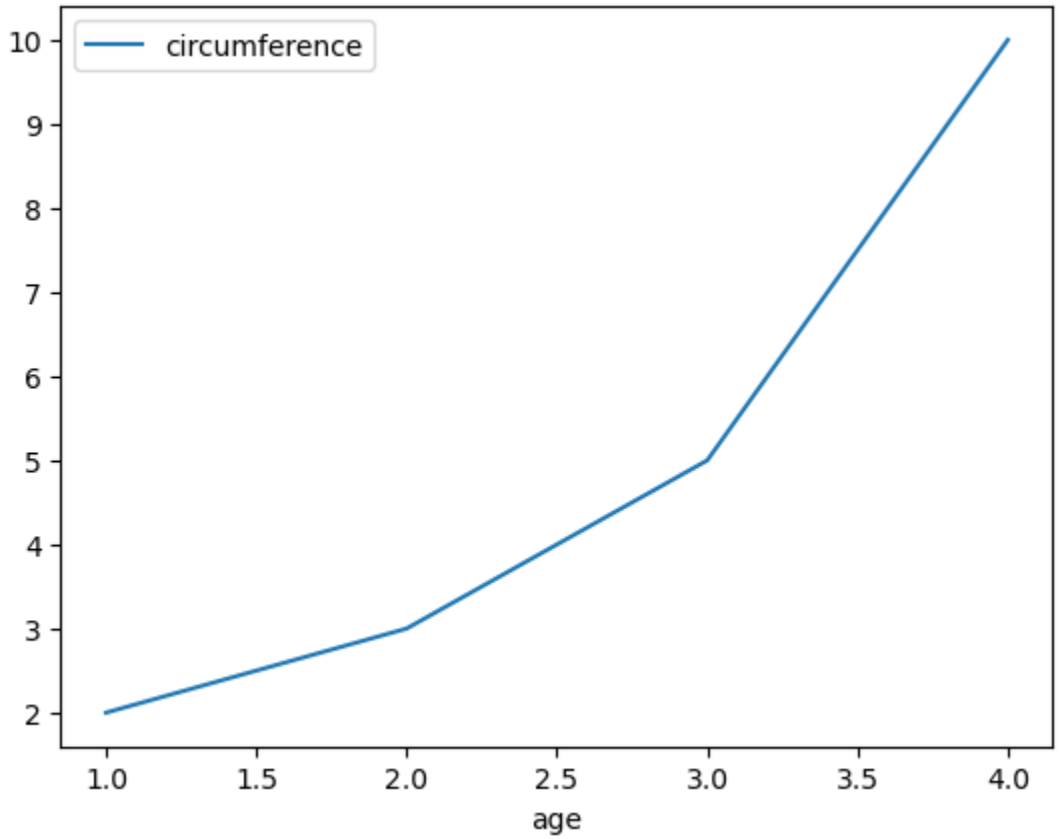
In [97]:
```
small_df = pd.read_table('../downloads/Orange_1.tsv')
small_df
```

Out[97]:

	age	circumference	height
0	1	2	30
1	2	3	35
2	3	5	40
3	4	10	50

In [98]:
```
small_df.plot(x='age', y='circumference', kind='line') # plot the relati
# try with other types of plots, e.g. scatter
```

Out[98]:
```
<AxesSubplot:xlabel='age'>
```



Tips: what if no plots shows up?

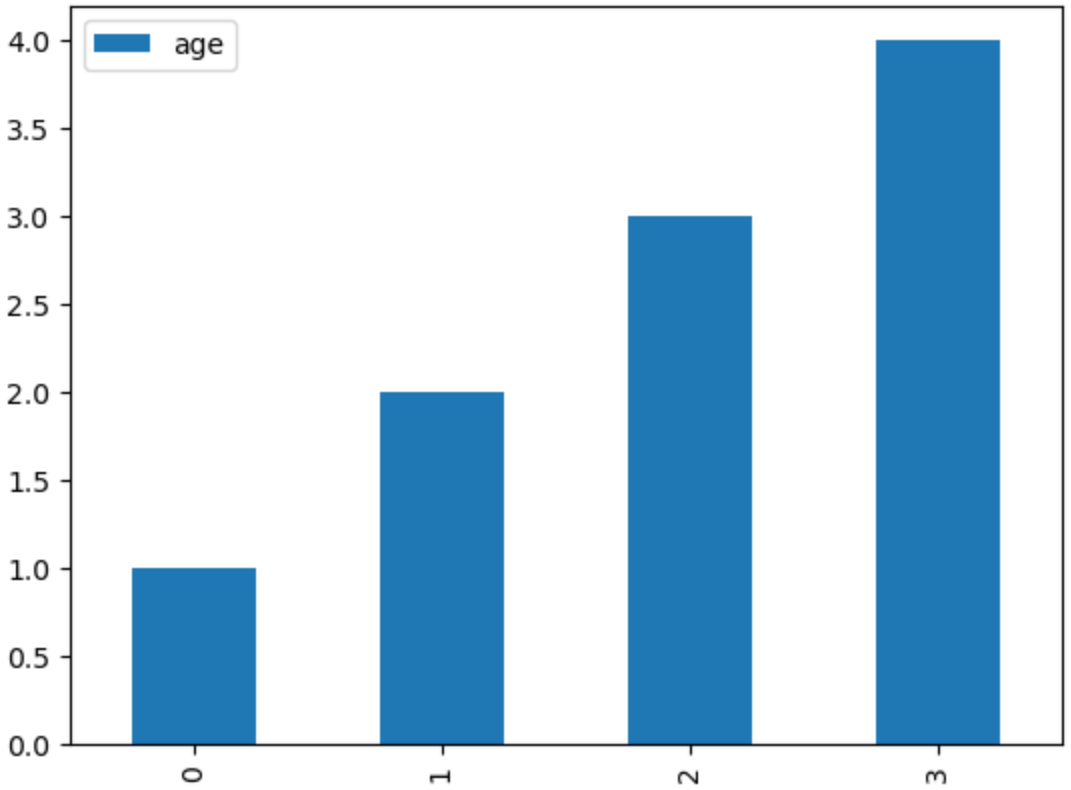
In [99]:
```
import matplotlib.pyplot as plt
plt.show()
```

In [100]:
```
%matplotlib inline
```

Plotting - bars

In [101]:
```
small_df[['age']].plot(kind='bar')
```

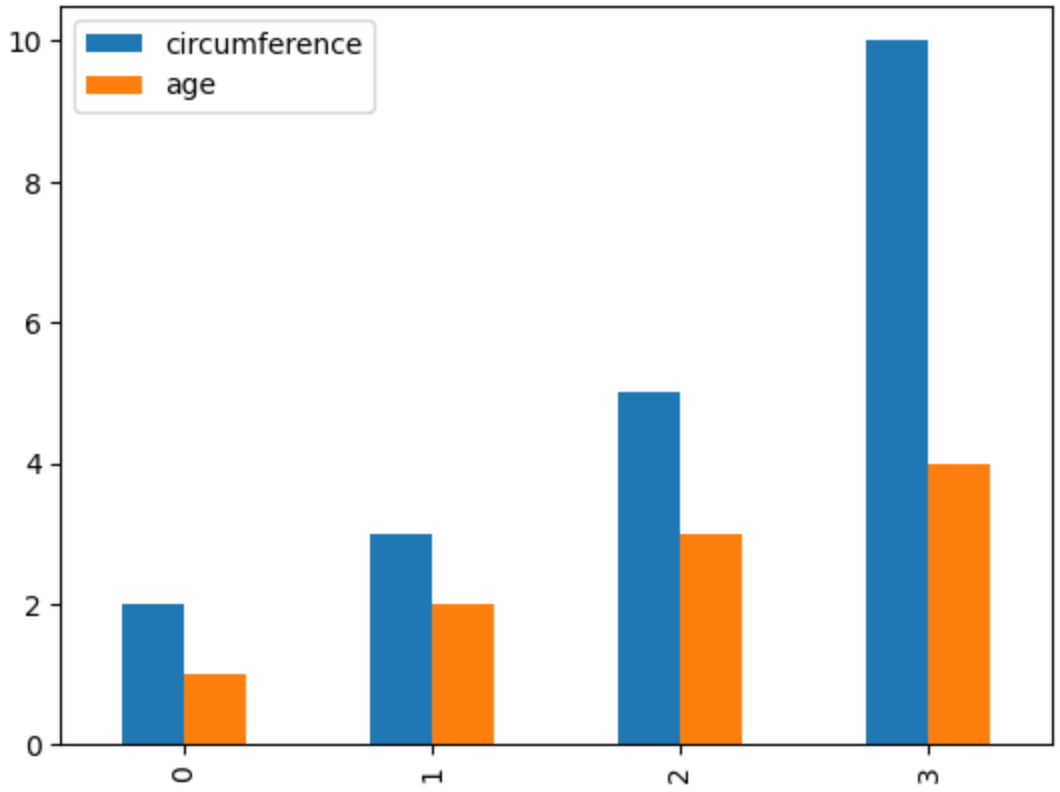
Out[101]:
```
<AxesSubplot:>
```



Plotting multiple columns

In [102]:
```
small_df[['circumference', 'age']].plot(kind='bar')
```

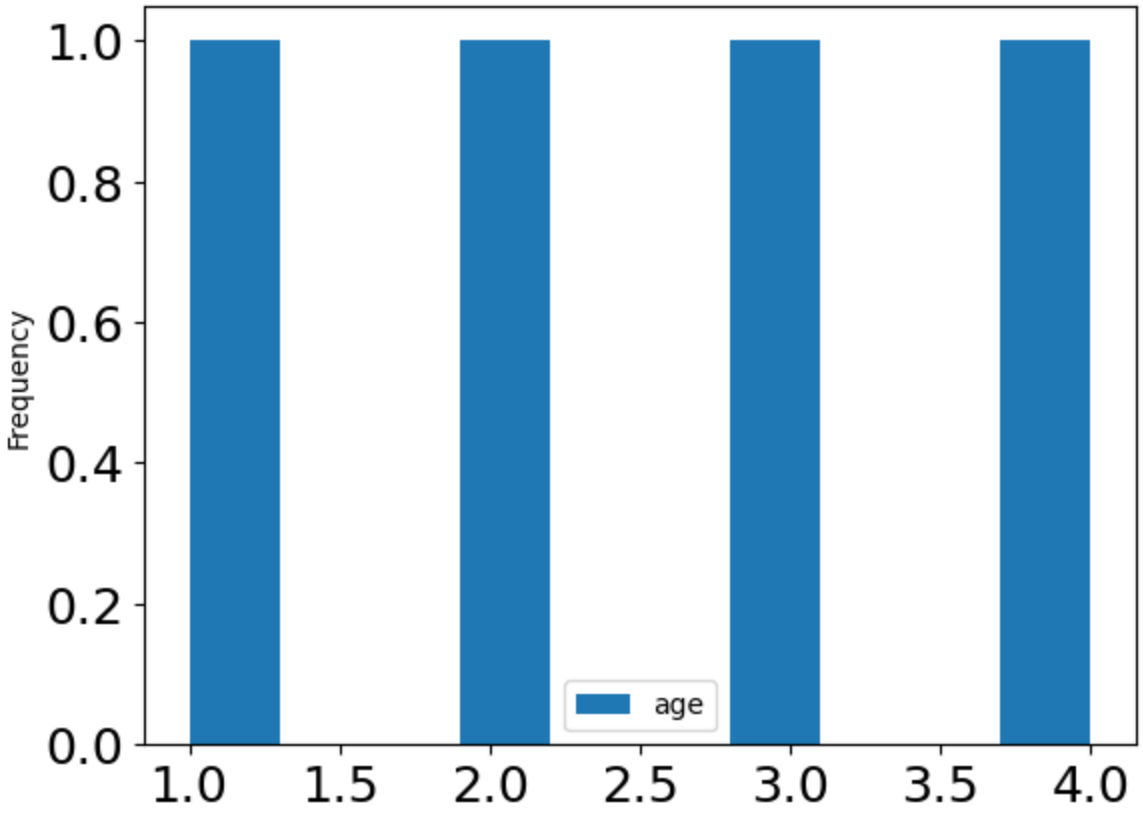
Out[102]:
```
<AxesSubplot:>
```



Plotting histogram

In [103]:
```
small_df.plot(kind='hist', y = 'age', fontsize=18)
```

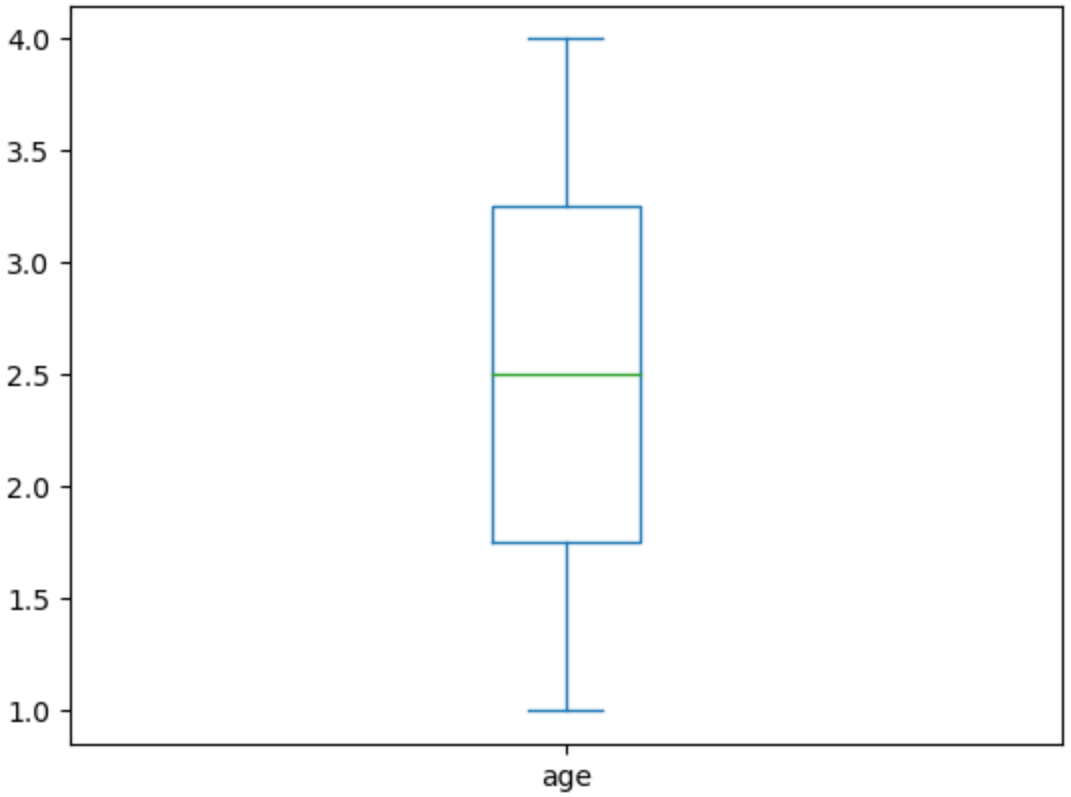
Out[103]:
```
<AxesSubplot:ylabel='Frequency'>
```



Plotting box

In [104]:
```
small_df.plot(kind='box', y = 'age')
```

Out[104]:
```
<AxesSubplot:>
```



Further reading: https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.plot.html

Exercise 2 (~30 minutes)

- Go to Canvas, `Modules -> Day 4 -> Exercise 2 - day 4`
- **Easy**:
    - Explore the `Orange_1.tsv`
- **Medium/hard**:
    - Use Pandas to read IMDB
    - Explore it by making graphs
- **Extra exercises**:
    - Read the pandas documentation :)
    - Start exploring your own data
- After exercise, do Quiz 4.2 and then take a break
- After break, working on the project