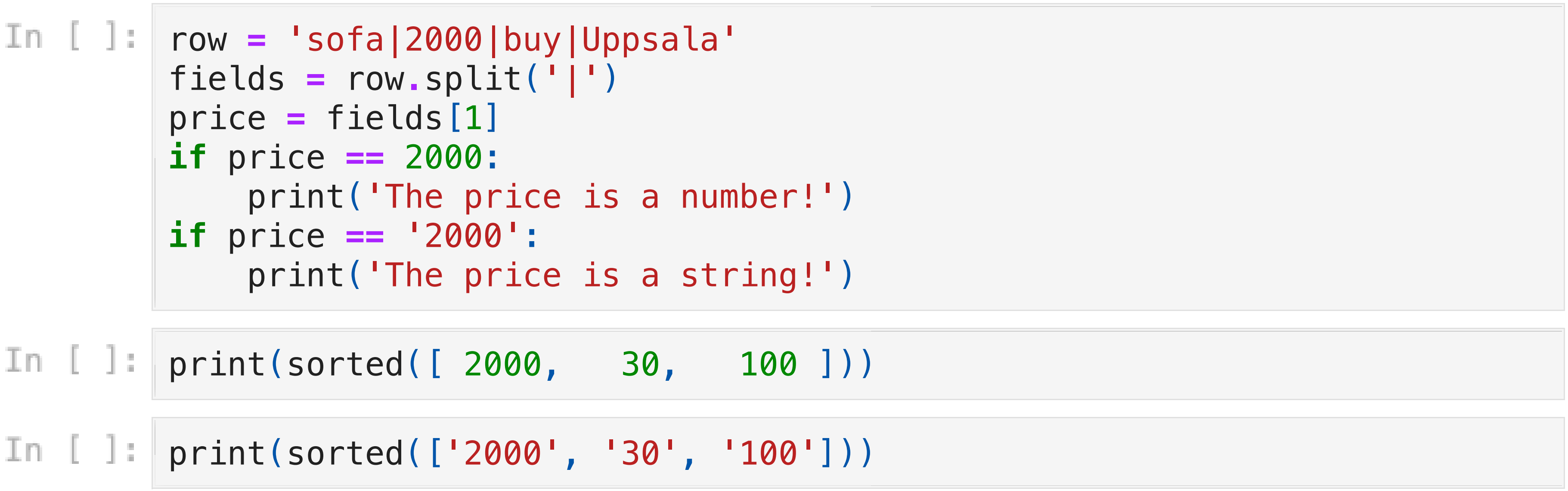
Introduction to

python

with Application to Bioinformatics

- Day 4



```python
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!')
```

```python
print(sorted([ 2000,   30,   100 ]))
```

```python
print(sorted(['2000', '30', '100']))
```

In [ ]:

```python
30 > 2000
```

In [ ]:

```python
'30' > '2000'
```

In [ ]:

```python
30 > int('2000')
```

In [ ]:

```python
'12345'[2]
```

In [ ]:

```python
12345[2]
```

In [ ]:

```
max('2000')
```

In [ ]:

```
max(2000)
```

In [ ]:

```
import math
math.cos(3.14)
```

In [ ]:

```
math.cos('3.14')
```

In [ ]:

```
'ACTG'.lower()
```

In [ ]:

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

In [ ]:

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

In [ ]:

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

- Each type supports different **methods**

```
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
```

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

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

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

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

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

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

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

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!')
```

- Python and the truth: true and false values

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 [ ]:

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

In [ ]:

```
genres = set(genre_list)
genres
```

In [ ]:

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

In [ ]:

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

In [ ]:

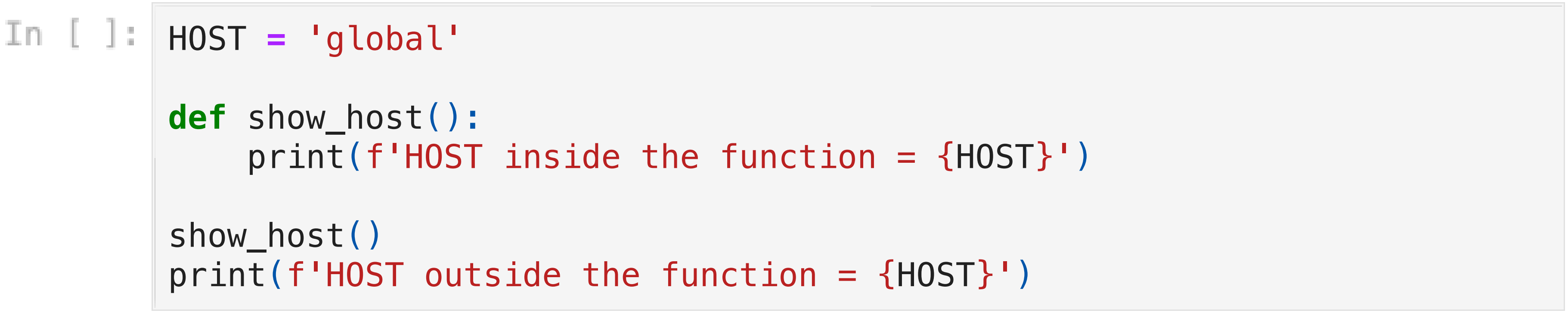
```
def echo(message): # starts a new function definition
    # this function echos the message
    print(message) # print state of the variable
    return message # return the value to end the function
```

In [ ]:

```
list("hello")
```

In [ ]:

```
'_'.join('hello')
```



In [ ]:

```
HOST = 'global'

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

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

In [ ]:

```
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()
```

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}')
```

Will the global variable never to changed by function?

In [ ]:

```python
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)
```

In [ ]:

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

In [ ]:

```
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')
```

In [ ]:

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

In [ ]:

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

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

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

```
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)
```

In [ ]:

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

In [ ]:

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

In [ ]:

```
record = 'gene_id INSR "insulin receptor"'

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

Why do we use keyword arguments?

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

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

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

Why do we use keyword arguments?

In [ ]:

```
record = 'gene_id INSR "insulin receptor"'

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

In [ ]:

```
record.split(sep=' ', maxsplit=2)
```

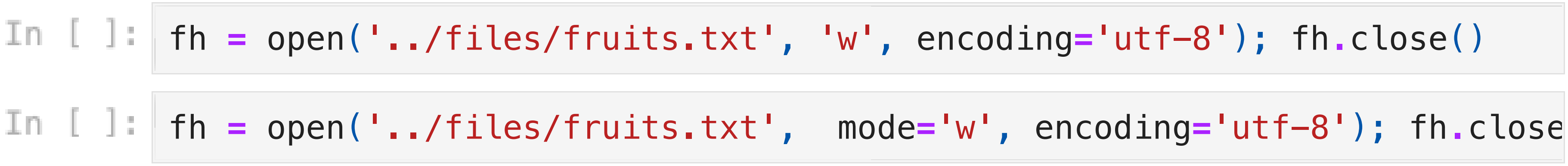
- It increases the clarity and readability

In [ ]:

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

In [ ]:

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



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

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

But there are some exceptions

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)
```

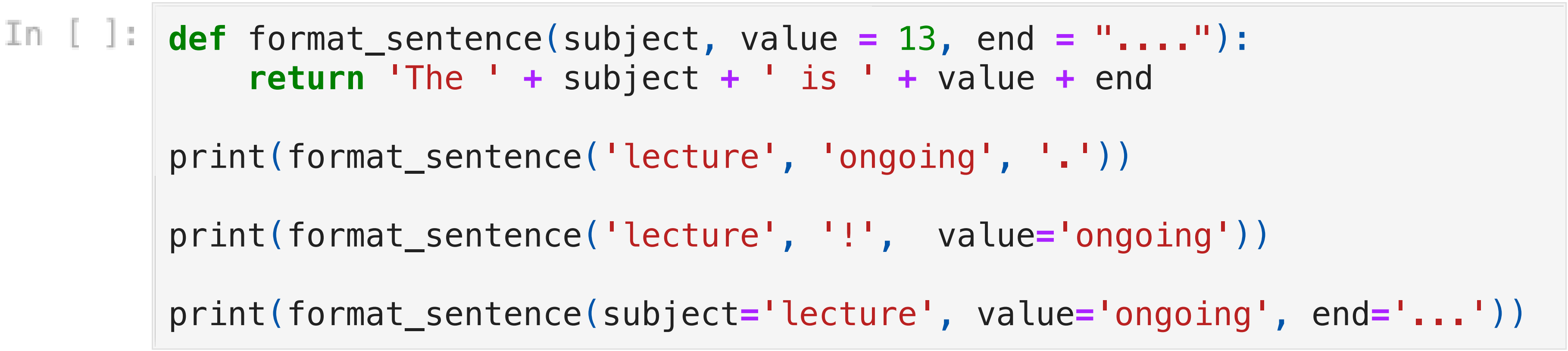
Restrictions by purpose

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

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

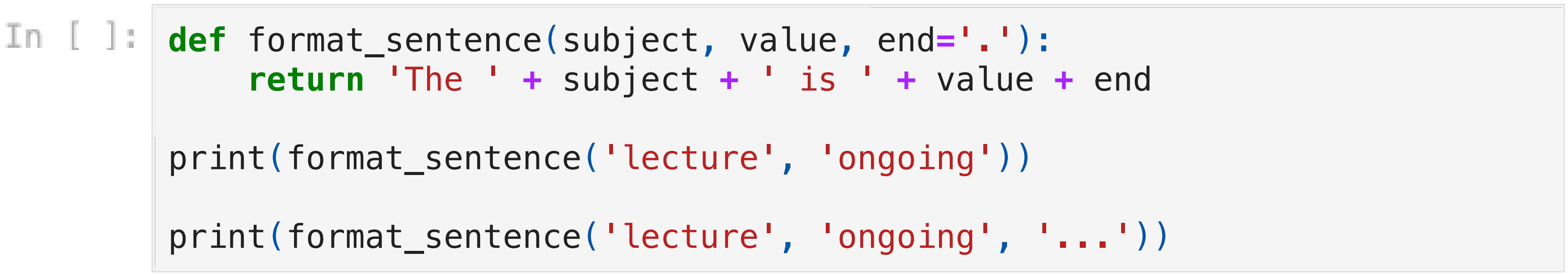
```
In [ ]: sorted(iterable, /, *, key=None, reverse=False)
```

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

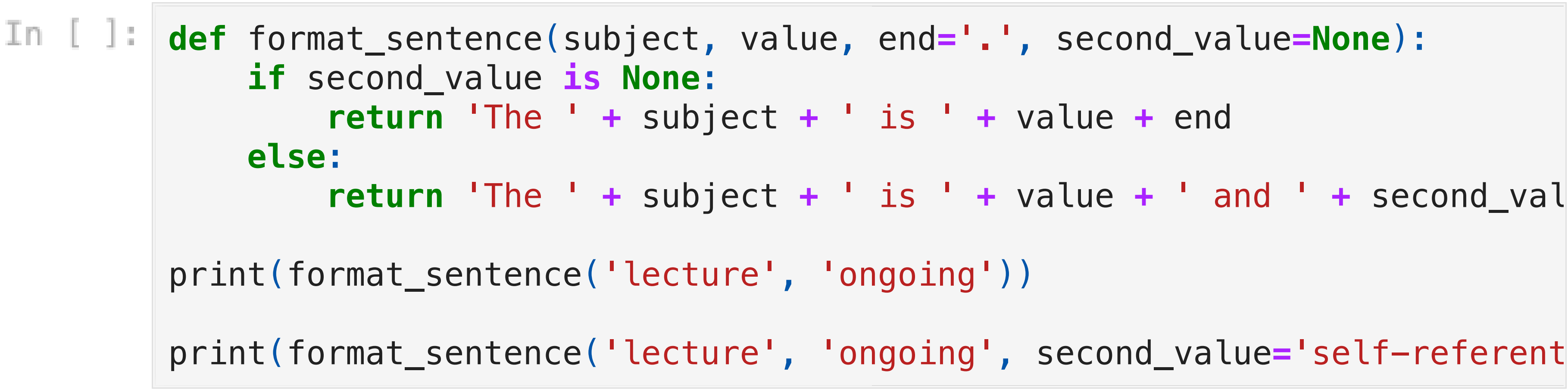


In [ ]:

```
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='...'))
```



```python
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=None):
            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-referent
```

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

In [ ]:

```
import sys

sys.argv[1]
```

In [ ]:

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

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

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

In [ ]:

```
text = 'Programming,is,cool'
text.split(sep=',')
```

How to understand it?

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

In [ ]:

```
text = 'Programming,is,cool'
text.split(sep=',')
```

In [ ]:

```
help(text.split)
```

How to understand it?

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

In [ ]:

```
text = 'Programming,is,cool'
text.split(sep=',')
```

In [ ]:

```
help(text.split)
```

In [ ]:

```
text.split(sep=',')
```

In [ ]:

```
import urllib

help(urllib)
```

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 [ ]:

```
import urllib

help(urllib)
```

- 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

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 [ ]:

```python
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 [ ]:

```
help(process_file)
```

- This works because somebody has documented the code!

Your code may have two types of users:

- library users
- maintainers (maybe yourself!)

Write documentation for both of them!

- library users (docstrings):

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

- maintainers (comments):

  ```python
  # implementation details
  ```

- At the beginning of the file

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

- At every function definition

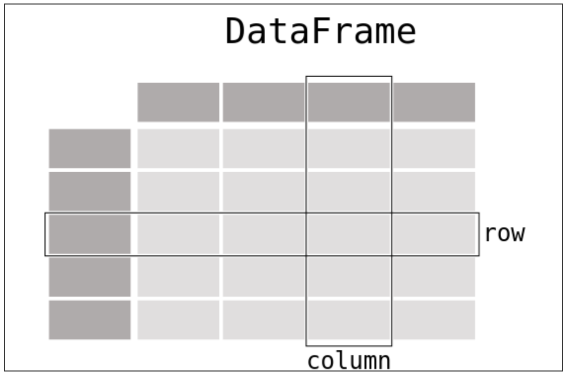
In [ ]:

```
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

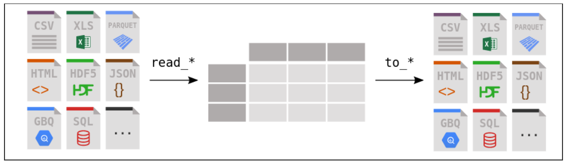
In [ ]:

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


DataFrame
row
column

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
```

Orange tree data

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 [ ]:

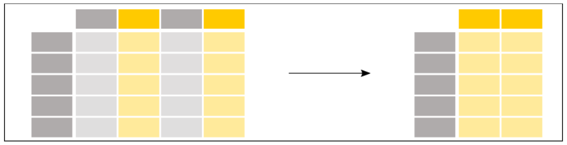
```
df2 = pd.read_excel('../downloads/Orange_1.xlsx')
df2
```

```
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']]
```

Selecting multiple columns

```
In [ ]: df
```

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

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

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 1, 3 and 0
```

In [ ]:

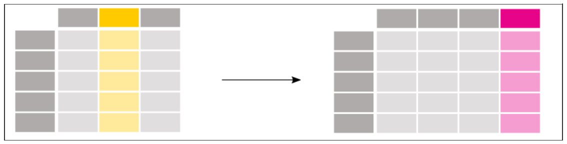
```
df
```

In [ ]:

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

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

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



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]
})

df1
```

In [ ]:

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

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 [ ]:

```
```

Plotting

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

In [ ]:

```
small_df = pd.read_table('../downloads/Orange_1.tsv')
small_df
```

In [ ]:

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

In [ ]:

```
%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')
```