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 [ ]:

```
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 [36]:

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

In [ ]:

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

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 [ ]:

```
30 > 2000
```

In [ ]:

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

In [ ]:

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

In [ ]:

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

In [ ]:

```
12345[2]
```

- Each type supports different **functions**

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

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

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

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

- 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
```

Convert string to number

- Questions 4, 5 and 6

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

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

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

In [ ]:
```
float('1e-2')
```

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

In [ ]:
```
int('1.5')
```

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

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

- Question 7

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

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

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

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

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

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

- 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)`

- Is 1 equivalent to `True`?

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`

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 [43]:

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

In [44]:

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

In [45]:

```
'drama' in genre_list
'drama' in genres
# which operation is faster?
```

In [46]:

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

In [47]:

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

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 [ ]:

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

In [ ]:

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

In [ ]:

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

What is a function?

- A named piece of code that performs a specific task
- A relation (mapping) between inputs (arguments) and output (return value)

In [ ]:

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

print(increment_by_two(100))
```

TODAY

- More on functions:
    - scop 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 [ ]:

```
HOST = 'global'

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

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

- Change in the function will not change the global variable

In [ ]:

```
HOST = 'global'

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

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

- Pass global variable as argument

In [ ]:

```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}')
```

More on functions: scope - global vs local variables cont.

List as global variables

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}')
```

Will the global variable never to changed by function?

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.

More on functions: `return` statement

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

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, count2)
```

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

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')
```

- Functions without any `return` statement returns None

In [ ]:

```
def foo():
    do_nothing = 1

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

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

Keyword arguments

- A way to give a name explicitly to a function for clarity

In [ ]:

```
sorted('file', reverse=True)
```

In [ ]:

```
attribute = 'gene_id "unknown gene"'
attribute.split(sep=' ', maxsplit=1)
```

In [ ]:

```
# print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
print('x=', end='')
print('1')
```

Keyword arguments cont.

- Can be used in both ways, with or without keyword, if there is no ambiguity
- Arguments after * must be keyword arguments, e.g. sorted()

In [ ]:

```
open('files/recipes.txt', 'w', encoding='utf-8')
```

In [ ]:

```
open('files/recipes.txt',  mode='w', encoding='utf-8')
```

- The order of keyword arguments does not matter

In [ ]:

```
open('files/recipes.txt', mode='w', encoding='utf-8')
```

In [ ]:

```
open('files/recipes.txt', encoding='utf-8', mode='w')
```

- Positional arguments must be in front of keyword arguments

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

How to define functions taking keyword arguments

- Just define them as usual:

In [ ]:

```
def format_sentence(subject, value, end):
    return 'The ' + subject + ' is ' + value + end

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

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

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

Defining functions with default arguments

In [ ]:

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

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

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

Defining functions with optional arguments

- Convention: use the object None

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_value + end

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

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

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

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

In [ ]:

```
bool(None)
```

In [ ]:

```
None == False
```

In [ ]:

```
None == 0
```

In [ ]:

```
None == ''
```

In [ ]:

```
type(None)
```

Exercise 1

- Notebook Day_4_Exercise_1 (~30 minutes)
- Go to Canvas, `Modules -> Day 4 -> Exercise 1 - day 4`
- Quiz. Go to Canvas, `Modules -> Day 4 -> PyQuiz 4.1`
- Lunch break
- Extra reading:
    - https://realpython.com/python-kwargs-and-args/ (https://realpython.com/python-kwargs-and-args/)
    - https://able.bio/rhett/python-functions-and-best-practices--78aclaa (https://able.bio/rhett/python-functions-and-best-practices--78aclaa)

A short note on code structure

- functions
- modules (files)
- documentation

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

- 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 [ ]:

```
import sys

sys.argv[1]
```

In [ ]:

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

Python standard modules

Check out the module index (https://docs.python.org/3/py-modindex.html)

How to find the right module?

How to understand it?

How to find the right module?

- Look at the module index (https://docs.python.org/3/py-modindex.html)
- Search PyPI (http://pypi.org)
- Ask your colleagues
- Search the web!

- 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 [ ]:
```
text = 'Programming,is,cool'
```

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

- For slightly more complicated problems, e.g. how to download Python logo from internet with `urllib`
- URL: https://www.python.org/static/img/python-logo@2x.png (https://www.python.org/static/img/python-logo@2x.png)

In [ ]:

```
import urllib

help(urllib)
```

- Sometimes easier to find the answer by searching the web

In [ ]:

```
import urllib
url = 'https://www.python.org/static/img/python-logo@2x.png'
urllib.request.urlretrieve(url, 'files/python-logo.png')
```

Various ways of importing

In [ ]:

```
import math

math.sqrt(3)
```

In [ ]:

```
import math as m
m.sqrt(3)
```

In [ ]:

```
from math import sqrt
sqrt(3)
```

Documentation and commenting your code

Remember `help()` ?

In [ ]:

```

```

- This works because somebody else has documented their code!

In [ ]:

```python
def process_file(filename, chrom, pos):
    """
    Read a 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 [ ]:

```python
help(process_file)
```

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 ...
"""
```

In [ ]:

```
from files import timeit
```

- 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
```

Comments:

- Wherever the code is hard to understand

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

Read more:

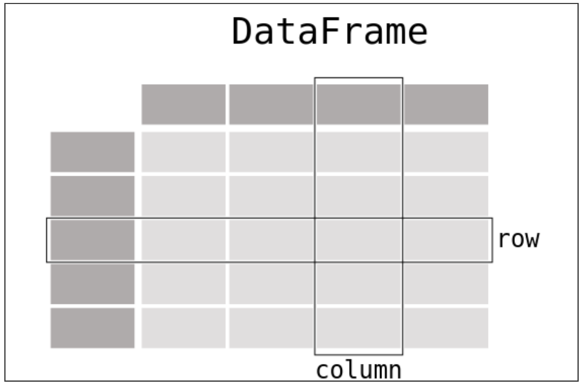
https://realpython.com/documenting-python-code/ (https://realpython.com/documenting-python-code/)

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

Pandas!!!

Pandas

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



Creating a small DataFrame

In [ ]:

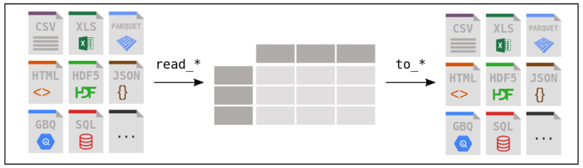
```
import pandas as pd
df = pd.DataFrame({
    'age': [1,2,3,4],
    'circumference': [2,3,5,10],
    'height': [30, 35, 40, 50]
})
df
```

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 [40]:

```
!cat ../downloads/Orange_1.tsv
```

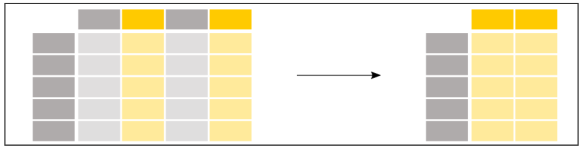
In [ ]:

```
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

Selecting columns from a dataframe

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



In [ ]:

```
df.columns
```

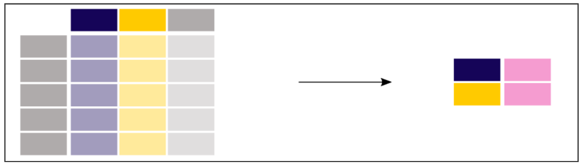
In [ ]:

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

In [ ]:

```
df.height
```

Calculating aggregated summary statistics



In [ ]:

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

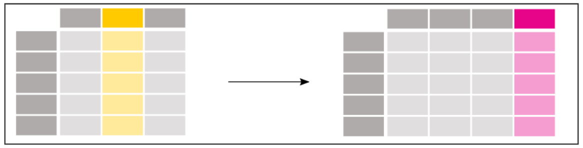
In [ ]:

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

In [ ]:

```
df['age'].max()
```

Creating new column derived from existing column

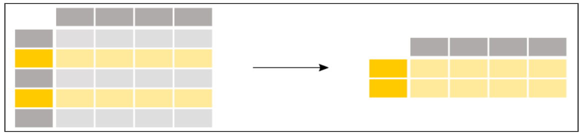


In [ ]:

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

Selecting rows from a dataframe by index

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



In [41]:

```
df.iloc[1:3]
```

Slightly bigger data frame of orange trees

In [ ]:

```
!head -n 10 ../downloads/Orange.tsv
```

In [ ]:

```
df = pd.read_table('../downloads/Orange.tsv')
df.iloc[0:5]  # can also use .head()
```

In [ ]:

```
df.Tree.unique()
```

In [ ]:

```
#young = df[df.age < 200]
#young
df[df.age < 1000]
```

Finding the maximal circumference and then filter the data frame by it

In [ ]:

```
df.head()
```

In [ ]:

```
max_c = df.circumference.max()
print(max_c)
```

In [ ]:

```
df[df.circumference == max_c]
```

Filter with multiple conditions

In [ ]:

```
df[(df.age > 100) & (df.age <= 250)]
```

Exercise

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 [27]:

```
import pandas as pd

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

df = pd.DataFrame(students)

df.grade.mean()
# df['grade'].mean()
```

Plotting

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

In [ ]:

```
small_df = pd.read_table('../downloads/Orange_1.tsv')
small_df.plot(x='age', y='height')
```

Plotting

What if no plot shows up?

In [ ]:

```
%pylab inline   # jupyter notebooks, run magic commands
```

In [ ]:

```
import matplotlib.pyplot as plt

plt.show()
```

Plotting - bars

- Plot a bar chart

In [ ]:

```
df[['circumference', 'age']].plot(kind='bar')
```

In [ ]:

```
df[['circumference', 'age']].plot(kind='bar', figsize=(12, 8), fontsize=16)
```

Scatterplot

```
df.plot(kind="scatter", x="column_name", y="other_column_name")
```

In [ ]:

```
df.plot(kind="scatter", x='age', y='circumference',
        figsize=(12, 8), fontsize=14)
```

Line plot

```
dataframe.plot(kind="line", x=..., y=...)
```

In [ ]:

```
tree1 = df[df['Tree'] == 1]
tree1.plot(kind="line", x='age', y='circumference',
           fontsize=14, figsize=(12,8))
```

Multiple graphs - grouping

In [ ]:

```
df.groupby('Tree')
```

In [ ]:

```
df.groupby('Tree').plot(kind="line", x='age', y='circumference')
```

In [ ]:

```
df.groupby('Tree').groups
```

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